LABORATORI NAZIONALI DI FRASCATI
SIS-Pubblicazioni

LNF-96/006(NT)
6 Febbraio 1996

UNIX at LNF: Users Guide

Version 2.0

Massimo Carboni
INFN-Laboratori Nazionali di Frascati,
P.O.Box 13, I-00044 Frascati (ROMA) Italia

Informations above this Document

UNIX at LNF – Users Guide

Computing Service Support

This document is written using a local version of *cernman.sty* under $\LaTeX$.

This document will be frequently updated. The latest copy can be found under `http://www.lnf.infn.it/~carboni/unixguide/unixguide.ps.gz`

Requests for information should be addressed to:


```
Computing Service
Laboratori Nazionali di Frascati
Instituto Nazionale di Fisica Nucleare
Via Enrico Fermi, 40
Frascati ROMA Italy
Tel. +39 6 9403 2349
Fax. +39 6 9403 2372
DECnet: VAXLNF::CARBONI (node 39.5)
Internet: CARBONI@LNF.INFN.IT
```


Contact Person: Massimo Carboni/Computing Service

Table of Contents

Chapter 1: Introduction

This User's Guide describe the UNIX configuration available at LNF, introduced as a central facilities since December 1992. The original configuration was based only on HP 9000/7XX with **HP-UX** Operating Systems, now the new Digital with operating system **DEC-OSF/1** are also included.

Presently the LNF UNIX configuration is based on 11 Hewlett Packard Workstation with Batch Server functionality, one HP 9000/755 with 128 MB of Memory dedicated for INTERACTIVE use. Since June 1995 the Computing Service also support the DEC system with the new Alpha Processor and DEC-OSF/1 Operating System, this systems can be used for intensive I/O tasks, like data analysis.

This guide gives an overview of the LNF UNIX service and describes the procedure for job submission using *Network Queueing System*, NQS. More detailed description of the local commands are given in the online man pages. In addition, up to now only for HP-UX, a complete set of manuals is available on line via **lrom** command, which accesses a CDROM over the network.

In case of any trouble about the usage of the LNF UNIX system please contact:

carboni@lnf.infn.it

Chapter 2: How To Get Started

2.1 How to Obtain an Account

If you want to work with this **LNF UNIX** system, contact the system manager of this system to get an account:

Massimo Carboni,	Computing Center room 4
	carboni@hpserver.lnf.infn.it

2.2 Login Via Telnet

If you want to access an HP workstation, e.g. hpserver.lnf.infn.it enter the command **telnet**. Some messages appear, followed by the login prompt, enter your account name, and the password will be prompted.

Login on the LNF Hewlett Packard ROOTServer

```
% telnet hpserver.lnf.infn.it
Trying...  Connected to hpserver.lnf.infn.it.
Escape character is '^]'.

HP-UX hpserver A.09.05 A 9000/755 (ttysa)

login: carboni
password:
```

Be carefull when entering your account name and password, because UNIX is **case sensitive**! Upper and Lower case letters have a different meaning.
After your login, the system asks for your terminal type. With the terminal type you describe the terminal hardware or the emulator program you are using. The most important types of terminal used are:

vt100	Digital Terminal
vt220	Digital Terminal
xterm	X-Window Terminal Emulator
hpterm	HP Terminal Emulator

If you work in a UNIX environment with a wrong terminal type setting, you should keep in mind that not all keys of your keyboard may be available in the way you expect. Then you would correct your terminal type with the **export** command. For example, if you want to correct your terminal type to `vt100`, you have to enter:

export TERM=vt220	Bourne/Korn-Shell users
setenv TERM vt220	C-shell/T-C-shell users

2.3 Passwords

Your account is registered in a NIS (*Network Information System*) environment. You can change your password using the command **passwd** (a link to **/usr/bin/yppasswd**). The procedure to change the NIS password is the following:

Changing NIS user password.

```
% passwd
Changing NIS password for carboni...
Old NIS password:
New password:
Retype new password:
```

You will be prompted for your Old NIS password. Then you will be prompted to enter and re-enter your new password. The re-entered password must match the first entry.

There is not password ageing on this system but users are recomendend to use *non-trivial* passwords. A password must meet four criteria to be valid:

- It must contain at least six characters.
- At least two characters must be alphabetic.
- At least one character must be a number (0-9) or a special character (`/,?,!` or other punctuation mark.)
- It must differ from your previous password by at least three characters.

If you forget your password, please contact your system administrator for a new one.

2.4 Shell Environment

The shell interprets the text you type, and the keys you press, in order to direct the operating system to take the appropriate action. A shell can also be used as a programming language. You may customize your session through the shell using hidden files. The system executes these files at login time. The name of these files and the syntax depends on the shell that you use:

`.profile`	Used by sh,ksh at login time
`.kshrc`	Used by sh,ksh
`.login`	Used by csh,tcsh at login time
`.cshrc`	Used by csh,tcsh
`.tcshrc`	Used by tcsh

The default shell at LNF is the `tcsh` shell. At login time the system executes the following user's files if exist under their `$HOME` directory:
Here is an example of the standard LNF `.login` and `.tcshrc`[1] file:

[1] Please verify your configuration files, if differ modify it

UNIX LNF login file: .login

```
if ($?ENVIRONMENT)  then
   if ("$ENVIRONMENT" == "BATCH") exit
endif
#
#  Execute SYSTEM level commands if file exists and is readable
#
if (-r /usr/local/etc/system_login ) source /usr/local/etc/system_login
#
#  Execute GROUP level commands if a group file exists
#
if (-r /u1/$group/group_login) source /u1/$group/group_login
#
#  Execute USER level commands
#
```

The default terminal is: **TERM = (xterm)**, if you want a different default terminal like **vt220** you can modify your .login file as follow:

Login file: .login with different DEFTERM

```
                                   .
                                   .
setenv DEFTERM vt220
if (-r /usr/local/etc/system_login ) source /usr/local/etc/system_login
                                   .
```

UNIX LNF Example of .tcshrc file

```
if ( $?ENVIRONMENT ) then
   if ( "$ENVIRONMENT" == "BATCH" ) exit
endif
#
# Execute SYSTEM level commands if file exists and is readable
#
if ( -r /usr/local/etc/system_cshrc ) source /usr/local/etc/system_cshrc
#
# Execute GROUP level commands if a group file exists and is readable
#
if ( -r /u1/$group/group_cshrc ) source /u1/$group/group_cshrc
#
# Execute USER level commands
#
set path = ( $path /usr/local/bin/X11 .)
#
set savehist=50              # number to save across sessions.
set prompt="%m:%~%B%#%b "    # new tcsh prompt
set ignoreeof                # no logout with <Ctrl-D> set notify set filec
set autolist                 # completion function
#
```

Chapter 3: System Environment

3.1 Home Directory Space and File Backup

The user home directory is:

`/u1/`*username*

Disk quotas on home directories are enforced at user level. To display your current disk quota use (in 1024 byte blocks) this command:

quota *-v*

To change your disk quota space, contact the UNIX system administrator.

Home directories are backed up on DAT tapes at a regular intervals. Full backups are made weekly and daily incremental backup is under testing. Contact the UNIX System Administrator if you need to recover a file.

3.2 Message Of The Day

News and system announcements on the LNF UNIX cluster are made using the message of the day file */etc/motd* which is displayed at each login. The basic */etc/motd* file list a series of items and further information on each item can be obtained by typing:

less */etc/motd.details*

followed by '/-X-' where X is the letter attached to a specific item.

3.3 VAX and UNIX news

General UNIX and VAX infos can be read directly from UNIX system, using the local news readers. The most popular news reader are **pine** from terminal and **mxrn** from X-Window Terminal. Please read the local UNIX man pages to get more informations about those commands.

The relative UNIX folder are:

The LNF UNIX folders are:

```
lnf.hp: Information about HP at LNF at LNF
lnf.osf: Information about Digital Unix at LNF
lnf.unix: Information about Unix at LNF
```

You can read netnews within Emacs using the GNUS package. GNUS uses the NNTP protocol to communicate with a news server, which is a repository of news articles:

emacs *-f gnus*

3.4 Printing

This system doesn't have directly connected printers, but makes use of printers defined on VMS system. To print a file, type the command:

Print on HP-UX	Print on DEC-OSF/1
`% lp -d printer filename`	`% lpr -P printer filename`

On our systems is also installed a general purpose command **xprint**. This is a Bourne-shell script in `/usr/local/bin` and provides the access to the printers defined before, where the printer can be passed as a command line parameter:

```
xprint  [-q printer] [-s] [-n] [-2] [-l## -p##] [-v] [-h] [-H] files.
```

Options Description:

`-q printer` Postscript Printer Name

- `lps_post` points to 3 different postscript printers inside the printer room, near the computing room, A4 format, (One sheet per page).
  LPS - LPS2 - LPS17
- `lps_post2` points to 2 different postscript printers inside the printer room, near the computing room, A4 format, (Two sheet per page).
  LPS - LPS2
- `lps_post_a3` points to 2 different postscript printers inside the printer room, near the computing room, A3 format, (One sheet per page).
  LPS - LPS2
- `lps17_post` points to the 600 dpi resolution postscript printer inside the printer room, near the computing room, A4 format, (One sheet per page).
  LPS17
- `lnf_psc` points to the postscript color printer inside the computing room, A4 format, (One sheet per page).
  PSC

`-s`	Print One Side per Sheet
`-n`	Don't Print the Cover Page
`-2`	Two Pages per Side
`-p###`	Print in Portrait, ### Allowed is 80 or 132 Characters
`-l###`	Print in Landscape, ### Allowed is 80 or 132 Characters
`-h`	Print the Help Message
`-v`	Print the Xprint Program Version Number

`-H`	Give the Updated Printer List
`files`	Print any File Format Like: text, dvi, ps, etc.

The default printer is `lps_post2` Alternatively, the environment variable `XPRINTER` can be set to indicate your most commonly used printer. For example:

setenv *XPRINTER lps17_post*

This command line parameter overrides the `XPRINTER` value and it can be included in *.cshrc.*
For more information:

man *xprint*

or read the man pages at the end of this guide.

3.5 Scratch Disk Space

Temporary disk space has been re-organised. Now two scratch areas are available for end-users:

/scrtch1: 8.6 GBytes
/scrtch2: 8.6 GBytes

Everybody is able to create his/her login temporary top-level directory in:

/scrtch[1|2]/*$group*

where *'group'* is the user's corresponding group. This two areas don't have quotas, users are asked to delete unwanted files themselves. In addition garbage collection has been enforced on the two areas.

3.6 Backup your Data

There are many different ways to save data on UNIX system. The most common commands are:

dd, dump, cpio, tar, cp, ...

Using those commands it is possible to save and restore data from (to) any UNIX system you use.
The Computing Service had installed also on their VMS system the **tar** command. In this way everybody can exchange data from UNIX and VMS systems.
The Computing Service supports on each system available at LNF the **tar** command on magnetic tape 90m with capacity 2.0GB, without any hardware/software compression facilities.

Here is the list of the tapes available and systems where the tapes are connected:

UNIX: Tape device file

```
hpkloe01: /dev/rmt/0m (rewind)
  hpmac1: /dev/rmt/0m (rewind)
```

VMS: Tape device

```
VXLNFA: $1$MUA7:
        $1$MUA8:
        $1$MUA9:
AXLNF1: AXLNF1$MKC300:
```

The **Tar** command usage is very similar on UNIX and VAX. Infact you can create tar on disk and on tape, in the follow there are some examples that show the general **tar** usage:

UNIX: save scratch files

```
% cd /scrtch1/users/carboni
% tar cvf /dev/rmt/0m nutpla/
% mt -f /dev/rmt/0m offl
```

VMS: save scratch files

```
$ SET DEF SCRTCH1:[USERS.CARBONI]
$ MOUNT/FORE/RECO=512/BLOCK=10240 $1$MUA0:
$ TAR CVF $1$MUA0: [.NTUPLA...]
$ UMOUNT $1$MUA0:
```

In this example we are saving files from a scratch subdir named `ntupla/` on tape and dismount the tape.

In the next example we want to add another directory to the same archive:

UNIX: save directory

```
% cd /scrtch1/users/carboni
% tar rvf /dev/rmt/0m rawdata/
% mt -f /dev/rmt/0m offl
```

VMS: save directory

```
$ SET DEF SCRTCH1:[USERS.CARBONI]
$ MOUNT/FORE/RECO=512/BLOCK=10240 $1$MUA0:
$ TAR RVF $1$MUA0: [.RAWDATA...]
$ UMOUNT $1$MUA0:
```

To determine the table of contents, use:

UNIX: show tape contents

```
% tar tvf /dev/rmt/0m
```

VAX: show tape contents

```
$ MOUNT/FORE/RECO=512/BLOCK=10240 $1$MUA0:
$ TAR TVF $1$MUA0:
```

To extract file from the archive:

UNIX: Extract files

```
% cd /scrtch2/users/carboni
% tar xvf /dev/rmt/0m
```

VMS: Extract files

```
$ SET DEF SCRTCH2:[USERS.CARBONI]
$ MOUNT/FORE/RECO=512/BLOCK=10240 $1$MUA0:
$ TAR XVF $1$MUA0:
```

Using this last command you create two subdirectory: `ntupla/` and `rawdata/` if don't exist.

You can specify subsets of the original archive:

UNIX: Extract ∗.dat

```
% cd /scrtch2/users/carboni
% tar xvf /dev/rmt/0m 'rawdata/*.dat'
```

VMS: Extract ∗.dat

```
$ SET DEF SCRTCH2:[USERS.CARBONI]
$ MOUNT/FORE/RECO=512/BLOCK=10240 $1$MUA0:
$ TAR XVF $1$MUA0: [.RAWDATA...]*.DAT
```

Chapter 4: LNF Programming Environment

4.1 Fortran77 Compiler

The FORTRAN77 compiler on the HP and DEC machines looks very similar in terms of functionality as compared to other UNIX system. However, an awarness of the default compiler options is important. In some cases, a program will fail or produce wrong results if you have compiled it with an excessive level of optimization. When developing a code, it is recommended to use the default optmization level until the program is stable. Once this stage has been completed, you may experiment different optimization levels to improve the performance. Before starting any serious production ensure that the results obtained during the tests are consistent whith those obtained using different optimization levels.

Using two different platforms you must use different options. On each system you have different options, please refer to man pages to get more details:

man *f77*

Here is an example how to compile a simple program program on our UNIX systems, in the two cases HP and DEC:

Compile fortran on HP

```
fort77 +ppu -K -c myprog.f
```

Compile fortran on DEC

```
f77 -static -c myprog.f
```

HP Options Description:

`-O`	To increase optimization
`-g`	Generate additional information needed by the symbolic debugger xdb. This option is incompatible with optimization.
`+ppu`	To add a trailing to external routines. This is needed if you wish to make calls to the CERN program library; it must be used only on HP
`-K`	To generate static code. Static is a code where local variable are saved after routine invocation. Note that the VAX compilers automatically save local variables; this is not the default case with HP-UX

DEC Options Description:

`-O`	To increase optimization
`-g`	Generate additional information needed by the symbolic debugger xdb. This option is incompatible with optimization
`-static`	Causes all local variables to be statically allocated. Like `-K` on HP

4.2 C Compiler

cc is the standard UNIX C compiler. On HP compiler options are available to select ANSI C compiling or alternatively *Kernigan & Ritchie* mode; on DEC system this is done automatically. A full description of the C compiler and its options is available with the command:

man *cc*

4.3 CERNLIB

As with any other UNIX machine, the references to libraries may be in an absolute way (specifing the full file path) or in the POSIX way, using the options `-l` and `-L`. HP-UX did not adopt the POSIX standards in their implementation of f77, but instead they kept their previous interface fort77. A linking statment is performed in one of these two ways:

Linking with CERN Library:

```
% f77 -o myprog myprog.o /cern/pro/lib/libpacklib.a
% fort77 -o myprog myprog.o -L/cern/pro/lib -lpacklib
```

This approach is fine for a few libraries but for more complicated situations, `CERNLIB` variable simplifies command scripts. This variable is set locally with the **cernlib** command in two equivalent ways (Note the use of back quotes of this example):

setenv *CERNLIB 'cernlib geant pawlib graflib/X11'*

HP: CERNLIB Compilation

```
% fort77 +ppu -K -c myprog.f
% fort77 -o myprog myprog.o $CERNLIB
```

DEC: CERNLIB Compilation

```
% f77 -static -c myprog.f
% f77 -o myprog myprog.o $CERNLIB
```

4.4 File Input/Output

The file attachment in UNIX is rarely made using logical units as on VMS. FORTRAN programs should take care to issue the proper `OPEN` statement with the name of the file coded in. Standard input and output (units 5 and 6) can be handled in the normal UNIX way:

myprog < *mydata.inp* > *myresults.out*

One may also give data records after the invocation of the executable:

```
myprog << EoD
card1
card2
...
...
EoD
```

We have used `'EoD'` to stand for `'End of Data'`. Any other string may be used as long as it is not part of the data records themselves.

When using other FORTRAN units numbers, the **ln** (link) command must be used. This is equivalent to ASSIGN in VMS. As an example:

ln *-s mydata.out ftn10*

Note that, unlike VAX, such file linking is *permanent* across sessions. It should be removed right after execution completion with the **rm** command:

rm *ftn10*

The original file will not be destroyed; only the link. An alternative approach is to use the `OPEN` statement, for example:

OPEN(10, FILE='mydata.dat', ...)

4.5 Debugging Program

If your program does not execute properly, you may wish to use a debugger to locate and correct problems; **xdb** is the HP-UX symbolic debugger. Before invoking a symbolic debugger you should recompile your program with **-g** option and without any optimization `-O` flags. This ensure that the necessary debugging information is incorporated into the object code. The debugger has many commands for viewing and manipulating programs. You can:

Control execution with single step execution or use the breakpoints.

Look at data values.

Look at the content of your source files.

Look at the execution stack.

A sample of simple commands for the HP **xdb** debugger are:

HP debug options.

```
COMMAND       DESCRIPTION
-------       -----------
r             Run the program
b 82          Set breakpoint at line 82
c             Continue running until the next breakpoint
s             Single step trough the next source line
S             Step over a function or a subroutine
t             Print a trace at the current execution stack
v             View a window of lines
/string       Search forward in the source for occurency of string
?string       Search backward in the source for occurency of string
p abc         Print the value of the variable "abc"
p abc = 2.2 Assign a value to "abc"
q             Quit the debugger
```

Chapter 5: Batch Job Submission with NQS

5.1 NQS - An Overview

The Network Queueing System (NQS) batch system has been installed on LNF UNIX Environment and is used for batch job submission. A NQS job is a series of UNIX commands combined in a Shell script. NQS works by using the job queues and the NQS on LNF UNIX Cluster distinguishes beetwen two types of queue:

Batch queue	Batch queues are defined on each system in NQS domain. Each queue has different resources limits, in particular the CPU time.
Pipe queue	Pipe queues are a mechanism to distribute jobs and balance the workload evenly over the destination servers. Users submit to the pipe queue, and the load balancing software in NQS finds an empty server in which has to run the job. If all servers are full, then the job is put on wait state. If more than one job are on wait state, than the next job to be started is determined by an intelligent script which is aware of the current state of running and queued jobs. The following subsection describes the batch environment in more detail.

5.2 NQS Commands

NQS provides a total of nine user commands:

qcat	Display output files of NQS running requests
qcmplx	Display status of NQS queue complexe
qdel	Delete or signal NQS request
qhold	Hold NQS request
qjob	Display status of NQS networked queues
qlimit	Show supported limit and shell strategy
qrls	Release NQS request
qstat	Display status of NQS requests and queues
qsub	Submit NQS batch request

To get more information about a command, type:

man *command*

or refer to the man pages at the end of this guide.
Moreover some extra NQS commands specific to our UNIX environment are:

qwhere	Display location of running NQS job
qusage	Gives the cumulative execution time in seconds for all running NQS jobs. The execution time is evaluated twice per hour
qresources	Gives the available resources for the different batch queues

5.3 LNF UNIX Batch Queues

Users submit their jobs to these different queues:

cpqS cpqM cpqL cpqH

Where:

cpqS	Jobs of less 3600 native CPU seconds
cpqM	Jobs of less 86400 native CPU seconds
cpqL	Jobs of less 400000 native CPU seconds
cpqH	Jobs of less 500000 native CPU seconds

User that submits a NQS job must be connected on the NQS master (hpserver.lnf.infn.it). The queues are divided in two different classes: SHORT and MEDIUM,LONG,HEAVY.
The S queue point only to six headless batch server. M,L and H are member of a complex queue. On each batch server can run only one of this tree queues with different submission priority. The H queue can be pointed by memory consuming programs.
The queues act as pipes to the 10 destination batch servers, finding empty machines if any, or queueing jobs if all servers are full. Jobs are submitted with the command:

qsub *filename*

where *filename* is the pathname to a job script. An example of job script is shown in Section 5.7 appended to this guide. To see the status of the HP Batch Server, type:

qjob

To cancel the running request, you have to use (in conjunction with **qusage**):

qdel *-k -h batch-server request-id*

5.4 Working Space

When a NQS job starts up, an environmental variable called `WORKDIR` is set. This is the pathname of a unique directory created for the running job. The directory is located on the machine on which the job is running. It has at least 200 MBytes of free space. Users should include in their job submission script a line as:

cd *$WORKDIR*

and should write their output on the local disk. When the main executable finishes, users should copy (**cp**) produced files over NFS to their directories. When a user's job script ends, NQS will remove the (`WORKDIR`) directory.
If the copy procedure fails[1], it is possible to keep for a short period of time this working directory in order to (re)issue the copy command. For this purpose, the file:

`$WORKDIR`/nostage

should exist.

[1]The disk server is down or the stage disk is full.

5.5 Outputs

The stdout and stderr output from NQS jobs is returned to the directory from which the user has submitted the job. All other output created by a job must be copied with user commands in the job script.

5.6 Job Limits

The TIMEL (Time Left) routine from the CERN Program Library keeps track of how much CPU time is left for a job, starting from a default number of seconds (9999).

GEANT by default will properly close down a job when there is no time left. Therefore, users whishing to run for a time longer than the GEANT default must set the start time to a higher value:

CALL TIMEST(36000.)

This would allow a 10 CPU-hours job in native CPU units.

5.7 Example: NQS Job Script

The script could contain some NQS commands, this give to the users the capabilities to define regular submition options like queue name, total cputime, running name, kind of shell, etc.

All of the NQS flags that can be specified on the command line can also be specified within the first comment block inside the batch request script file as embedded default flags. Here is an example of the use of embedded flags within the script file.

For more options info read the nqs man pages:

man *qsub*

NQS script: an example

```
#   Batch request parameter
#@$-s /bin/csh  # Script shell / Begin of QSUB job description
#@$-r MyProgram # Request name
#@$-eo          # Merge error and standard output
#@$-me          # Send mail upon termination
#@$             # End of QSUB job description
#
#   Change working directory and print it
#
cd $WORKDIR
pwd
#
```

```
#   In order to be EFFICIENT (avoiding NFS traffic and swapping!)
#   main files (binaries and data) HAVE to be copied to the local
#   area
#
cp $HOME/work/myprog $WORKDIR
cp $HOME/work/mydata.dat $WORKDIR
#
#   Execute main program
#
time $WORKDIR/myprog < mydata.dat > myresults.lis
#
#   Copy LARGE output file to the scratch area
#
cp myprog.out /scrtch1/grp/username
if ( $status != 0 ) then
        touch $WORKDIR/nostage
endif
#
#   End of job
exit
```

Chapter 6: NQS++ Job Submission

With the introduction of the new DEC system is available a second BATCH Server based on DEC-OSF/1 Operating System. With NQS++ each user can submit and control the jobs on both BATCH Servers.

6.1 Setting the Preferred NQS Batch Server with qset

The command **qset** has been added in order to define dynamically the default NQS server node and if necessary the default UNIX/NQS username. Here is a typical sequence of commands which may be invoked :

Set HP BATCH server	Command Output
`% qset -h hpserver`	`Default NQS server node : hpserver.`

in case of the DEC BATCH Server.

Set DEC BATCH server	Command Output
`% qset -h axpals`	`Default NQS server node : axpals.`

HP Job Status Enquiry:

```
% qstat -p
============================================
NQS Version: 2.5  PIPE QUEUES on hpserver
============================================
QUEUE NAME      STATUS   TOTAL   RUNNING  QUEUED   HELD  TRANSITION
-------------------------------------------------------------------
cpqH            AVAILBL    0       0/1       0       0       0
cpqM            AVAILBL    0       0/1       0       0       0
cpqL            AVAILBL    0       0/1       0       0       0
cpqS            AVAILBL    0       0/1       0       0       0
```

To submit a job :

Job Submission	Command Output
`% qset -h axpals`	`1800 bytes transferred.`
`% qsub -o NQSscript.output NQSscript.job`	`Request 12.axpals submitted to queue: cpqL`

When NQS jobs end, an automatic procedure tries to make available NQS outputs files on the target machines. In case of failure, end-users receive mail messages on their originate host telling them where these files are and how to get them back on their local hosts.

6.2 Write NQS script.job machine dependent.

If your code is compiled for both systems (DEC & HP) you can submit your jobs independently from the architecture. Following the usual scheme shown in the chapter5 you can modify the script as follow:

NQS script architecture dependent: an example

```
#   Batch request parameter
#@$-s /bin/csh  # Script shell / Begin of QSUB job description
#@$-r MyProgram # Request name
#@$-eo          # Merge error and standard output
#@$-me          # Send mail upon termination
#@$             # End of QSUB job description
#
#   Change working directory and print it
#
cd $WORKDIR; pwd
#
#   Select the architecture dependent code.
#
if ( "`uname`" == "HP-UX" ) then
      setenv MYPROG myprog_hpux.exe
else if ( "`uname`" == "OSF1" ) then
      setenv MYPROG myprog_osf1.exe
endif
#
#   In order to be EFFICIENT (avoiding NFS traffic and swapping!)
#   main files (binaries and data) HAVE to be copied to the local
#   area
#
cp /u1/username/work/$MYPROG $WORKDIR/myprog
cp /u1/username/work/mydata.dat $WORKDIR
#
#   Execute main program
#
time $WORKDIR/myprog < mydata.dat > myresults.lis
#
#   Copy LARGE output file to the scratch area
#
cp myprog.out /scrtch1/grp/username
if ( $status != 0 ) then
      touch $WORKDIR/nostage
endif
#
#   End of job
```

Appendix A: Other Informations

A.1 Hardware Configuration

The Hewlett Packard cluster available at LNF includes 12 HP 9000/7xx, with hardware configuration shown in Tab.A.1, and 2 DEC System DEC-4000/610 and DEC-2100 4/275, with alpha processor, with the relative hardware characteristics shown in Tab.A.2.

Name	IP Address	Model	Configuration			Performance	
			RAM	Disks	Tape	SPECint92	SPECfp92
hpserver	192.84.127.75	755/99	128 MB	5x1.3 GB	1.3 GB	80	150
hpcalc	192.84.127.10	710/33	32 MB	420 MB		24	45
hpmac1	192.84.127.57	735/99	48 MB	420 MB	4.0 GB	80	150
hpcal2	192.84.127.107	715/75	32 MB	1.05 GB		61	113
hpcal3	192.84.127.108	715/75	32 MB	1.05 GB		61	113
hpmac2	192.84.127.106	715/75	32 MB	1.05 GB		61	113
hpalp1	192.84.127.109	715/75	32 MB	1.05 GB		61	113
hpalp2	192.84.127.176	715/80	64 MB	1.05 GB		85	125
hpcad	192.84.127.111	735/99	80 MB	2x1.05 GB	2.0 GB	80	150
hpcad1	192.84.127.112	715/50	64 MB	1.05 GB		36	72
hpcad2	192.84.127.113	715/50	64 MB	1.05 GB		36	72
hpkloe01	192.84.127.206	735/125	80 MB	2x1.05 GB	4.0 GB	136	201
hpkloe01-f	192.84.130.17						
						801	1407

Table A.1: Hewlett Packard Cluster - Hardware Configuration.

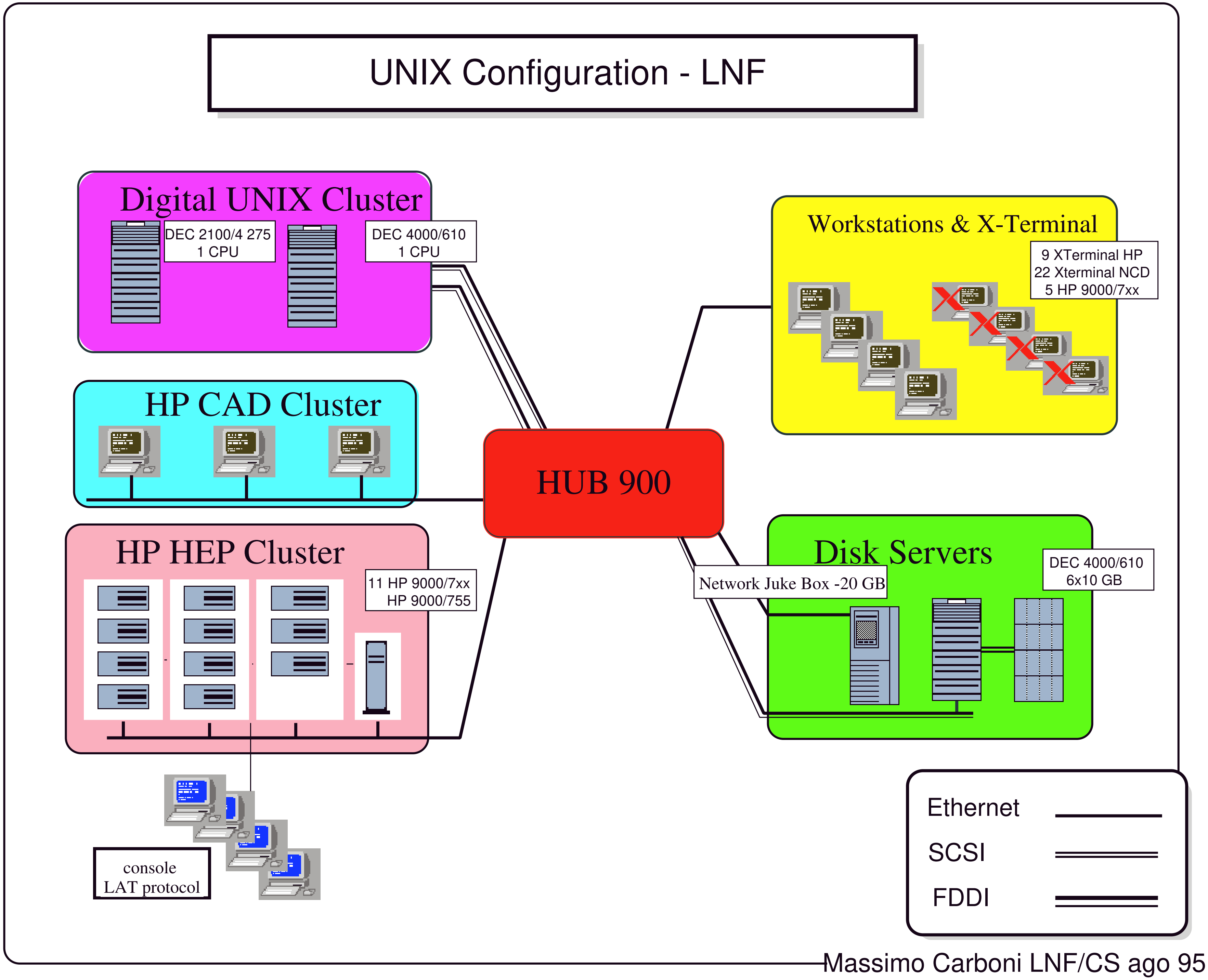
Name	IP Address	Model	Configuration			Performance	
			RAM	Disks	Tape	SPECint92	SPECfp92
axpals	192.84.127.55	4000/610	128 MB	2x2.0 GB	2.6 GB	131	161
axpals-f	192.84.130.7			6x9.1 GB			
kloe01	192.84.127.232	2100 4/275	128 MB	2.1 GB	10 GB	200	291
kloe01-f	192.84.130.22			1x9.1GB			
						331	452

Table A.2: Digital Cluster - Hardware Configuration.

The workstations HP9000/7XX are in the *Edificio Alte Energie*, and are located in different places inside the building; for this reason the Network connection to the LAN (Local Area Network) uses two different **Thin Wire Ethernet** that are connected to the **LNFEthernet Backbone** by a **DEC-Dempr**; no bridge solution is adopted.

Only two workstations have the monitor, the remaining five are installed inside the computing center structure, making use of the centralized UPS and cooling system in order to increase the system availability. Four of these workstations are head-less and the console for each system is obtained using the serial line connected with a **Terminal Server** port. On each port of the Terminal Server a remote service is defined, that allows the connection from a generic terminal to each workstation console using the **Local Area Trasport** protocol (LAT).

In figure the hardware configuration including the remote console system is shown.


UNIX Configuration - LNF
Digital UNIX Cluster
DEC 2100/4 275
1 CPU
DEC 4000/610
1 CPU
HP CAD Cluster
HP HEP Cluster
11 HP 9000/7xx
HP 9000/755
console
LAT protocol
HUB 900
Workstations & X-Terminal
9 XTerminal HP
22 Xterminal NCD
5 HP 9000/7xx
Disk Servers
Network Juke Box -20 GB
DEC 4000/610
6x10 GB
Ethernet
SCSI
FDDI
Massimo Carboni LNF/CS ago 95

Appendix B: Selection of man pages

NAME

xprint − print files on VAX/VMS printers

SYNOPSIS

xprint [**−q** printer] [**−s**] [**−n**] [**−2**] [**−l** number] [**−p** number] [**−man unix-manual**] [**−H**] [**−h**] [**--**] [files]

DESCRIPTION

Xprint prints the specified file on a printer. Only VAX/VMS printer are supported:

VAX/VMS

VAX/VMS postscript printer are accessible from UNIX systems. For such printers, the user's file is routed to the VAX/VMS which print the specified files. The transfer to VAX/VMS is made using the TCP/IP <---> DECNET gateway software running on a VAX/VMS system. *Xprint* check the printer selected and the hardware characteristics. Use the standard UNIX command like: *lp, a2ps, pstops* and *dvips. Xprint* recognize standard postscript file, higz postscript file, text file, TeX files. It's possible print at the same time TeX and PAW files just giving the complete file name.

Printing formats are selected with command options

OPTIONS

−q printer Select the printing device: *lps_post lps_post2 lps_post_a3 lps17_post aen1a1_post aen2a1_post lnf_psc.*

−s print one side per sheet only, the default value is two per page [rectoverso].

−n Don't print the *cover page.* Use this options only when you are near the printer and your output is short.

−2 Print two pages per physical page (twinpage mode), side by side or up-down, depending on the printing mode (landscape or portrait). By default print only one page per physical page (single page mode).

−p −l number Print files in portrait (landscape) mode (vertical or horizontal pages). In text mode you can select beetwen 80 or 132 characters per page. The default is : **−p 80.**

−man manual Print in postscript the usual UNIX manuals.

−H The *help* option lists the names of known postscript printers and the place where the printers are located.

−h Print usage information.

−- Read from standard input and print the output on the selected printer. This is a positional option use it as last option.

PRINTERS

lps_post point to 3 different postscript printer one sheet per page inside che printer room, near the computing room, A4 format: *LPS - LPS2 - LPS17.*

lps_post2 point to 2 different postscript printer two sheet per page inside che printer room, near the computing room, A4 format: *LPS - LPS2.*

lps_post_a3 point to 2 different postscript printer one sheet per page inside che printer room, near the computing room, A3 format: *LPS - LPS2.*

lps17_post point to the 600 dpi resolution postscript printer one sheet per page inside che printer room, near the computing room, A4 format: *LPS17.*

aen1a1_post point to the postscript printer inside the New Alte Energie Build 1 Floor A Side, A4 format: *AEN1A1.*

aen2a1_post point to the postscript printer inside the New Alte Energie Build 2 Floor A Side, A4 format: *AEN2A1.*

NEW FEATURES

Postscript print of Unix man pages, see **−man** option for more details.

ENVIRONMENT VARIABLES

PRINTER Specifies the default printer when **xprint** is invoked without the **−q** option.

XPRINTER Same as PRINTER, but has higher priority.

SEE ALSO

lp(1), a2ps(1), pstops(1), dvips(1)

DIAGNOSTICS

A print request is confirmed by returning the spool id of the print job.

HISTORY

Origin: Laboratori Nazionali di Frascati

Febrary 1993 − Massimo Carboni, LNF/CS
Original release.

November 1995
Last release.

NAME

qcat − display error, input, or output text files of NQS running requests.

SYNOPSIS

qcat [**−e**] [**−i**] [**−o**] [**−t** number] [**−h** target-host] request-id ...

DESCRIPTION

Qcat displays the contents of error, input, or output text files of Network Queueing System (NQS) running requests.

Qcat finds and if it exists, reads each file in sequence and displays it on the standard output.

An NQS request is always uniquely identified by its *request-id*, no matter where it is in the network of the machines. A *request-id* is always of the form: *seqno* or *seqno.hostname* where *hostname* identifies the machine from whence the request was originally submitted, and *seqno* identifies the sequence number assigned to the request on the originating host. If the *hostname* portion of a *request-id* is omitted, then the local host is always assumed.

The following flags are available:

-e Displays the error file if it exists.

-i Displays the input file (script file).

-o Displays the output file if it exists.

-t *number*
Begins copying at distance number from the end of the file. number is counted in units of lines.

-h *target-name*
Specifies the target NQS host from which display information is to be obtained.

If no option is specified the input file tries to be displayed.

CAVEATS

NQS is not finished, and continues to undergo development.

SEE ALSO

qstat(1)

NPSN HISTORY

Origin: CERN

April 1992 − Christian Boissat, CERN
Original release.

NAME

qcmplx − display status of NQS complex(es)

SYNOPSIS

qcmplx [**−h** host-name] [**−n**] [**−Q**]
[complex-name] [complex-name@host-name]

DESCRIPTION

Qcmplx displays the Network Queueing System (NQS) complexes.

In the absence of a *-h host-name* specifier, the local host is assumed.

Each entry displays the complexes on a given host. The *-Q* option displays the queues within the complex. The *-n* option eliminates the qcmplx header display.

CAVEATS

NQS is not finished, and continues to undergo development. This command may or may not be supported on all of your machines in the network.

SEE ALSO

qdel(1), qdev(1), qlimit(1), qpr(1), qstat(1), qsub(1), qmgr(1M)

NPSN HISTORY

Origin: Sterling Software Incorporated

August 1985 - Brent Kingsbury, Sterling Software
Original release.

Feb. 1990 − Terrie Carver, Computer Science Corperation
Second release.

NAME

qdel − delete or signal NQS request(s).

SYNOPSIS

qdel [**−k**] [**−s**] [**−c**] [**−h hostname**] [−signo] [**−u** username] request-id ...

DESCRIPTION

Qdel deletes all queued NQS requests whose respective *request-id* is listed on the command line. Additionally, if the flag **-k** is specified, then the default signal of **SIGINT** (-2) is sent to any running request whose *request-id* is listed on the command line. This will cause the receiving request to exit and be deleted. If the flag **-s** is specified, then the default signal of **SIGSTOP** is sent to any running request whose *request-id* is listed on the command line. This will cause the receiving request to be stopped. If the flag **-c** is specified, then the default signal of **SIGCONT** is sent to any running request whose *request-id* is listed on the command line. This will cause the receiving request to continue after being stopped.
If the flag **−h** hostname is requested then the action will be taken on the given host. If the flag −signo is present, then the specified signal is sent instead of the **SIGINT** signal to any running request whose *request-id* is listed on the command line. In the absence of the **-k** and −signo flags, *qdel* will **not** delete a *running* NQS request.

To delete or signal an NQS request, the invoking user *must* be the owner; namely the submitter of the request. The only exception to this rule occurs when the invoking user is the *superuser*, or has NQS operator privileges as defined in the NQS manager database. Under these conditions, the invoker may specify the **−u** username flag which allows the invoker to delete or signal requests owned by the user whose account name is *username*. When this form of the command is used, **all** *request-ids* listed on the command line are presumed to refer to requests owned by the specified user.

An NQS request is always uniquely identified by its *request-id*, no matter where it is in the network of the machines. A *request-id* is always of the form: *seqno* or *seqno.hostname* where *hostname* identifies the machine from whence the request was originally submitted, and *seqno* identifies the sequence number assigned to the request on the originating host. If the *hostname* portion of a *request-id* is omitted, then the local host is always assumed.

The *request-id* of any NQS request is displayed when the request is first submitted (unless the *silent* mode of operation for the given NQS command was specified). The user can also obtain the *request-id* of any request through the use of the *qstat*(1) command.

CAVEATS

When an NQS request is signalled by the methods discussed above, the proper signal is sent to *all* processes comprising the NQS *request* that are in the same *process group*. Whenever an NQS request is spawned, a new *process group* is established for all processes in the request. However, should one or more processes of the request successfully execute a *setpgrp*() system call, then such processes will **not** receive any signals sent by the *qdel*(1) command. This can lead to "rogue" request processes which must be killed by other means such as the *kill*(1) command. For the UNIX implementations that support the ability to "lock" a process, and all of its progeny into a *process-group*, NQS will exploit this capability to prevent processes from "escaping" in this manner.

SEE ALSO

qcmplx(1), qdev(1), qlimit(1), qpr(1), qstat(1), qsub(1), qmgr(1M),
kill(2), setpgrp(2), signal(2)

NPSN HISTORY

Origin: Sterling Software Incorporated

August 1985 − Brent Kingsbury, Sterling Software
Original release.

May 1986
Second release.

Feb. 1990 − Terrie Carver, Computer Sciences Corporation

Third release.

NAME

qdev − display status of NQS devices

SYNOPSIS

qdev [device-name] [device-name@host-name ...]

DESCRIPTION

Qdev displays the status of devices known to the Network Queueing System (NQS).

If no devices are specified, then the current state of each NQS device on the local host is displayed. Otherwise, the response is limited to the devices specified. Devices may be specified either as *device-name* or *device-name@host-name*. In the absence of a *host-name* specifier, the local host is assumed.

A *device header* with several headings is displayed for each of the selected devices. The first heading in a device header appears as **Device**:, and is followed by the name of the device formatted as *device-name@host-name*. The second heading of **Fullname**: is followed by the full path name of the special file associated with the device. The third heading of **Server**: is followed by the command line which will be used to *execve(2)* the device server. The fourth heading of **Forms**: is followed by the forms configured for the device.

The final heading of **Status**: prefaces a display of the general device state. The general state of a device is defined by two principal properties of the device.

The first property concerns whether or not the device is willing to continue accepting queued requests. If it is, the device is said to be **ENABLED**. If the device is unwilling to continue accepting queued requests, and is idle, its state is **DISABLED**. A third state of **ENABLED/CLOSED** is used to describe a device that is unwilling to continue accepting queued requests, but is not yet idle.

The second principal property of a device concerns whether or not the device is busy. There are three cases. If the device is busy, it is said to be **ACTIVE**. If the device is idle and not known to be out of service, it is said to be **INACTIVE**. Finally, if the device is idle and known to be out of service, it is said to be **FAILED**. **FAILED** covers both hardware and software failures.

If a device is busy, information about the active request follows the device header. The *request-name*, *request-id*, and the name of the user who submitted the request are all displayed.

SEE ALSO

qdel(1), qlimit(1), qpr(1), qstat(1), qsub(1), qmgr(1M)

NPSN HISTORY

Origin: Sterling Software Incorporated

May 1986 − Robert Sandstrom, Sterling Software
Original release.

NAME

qhold − hold NQS request(s).

SYNOPSIS

qhold [**−u** username] request-id ...

DESCRIPTION

Qhold holds all queued or waiting NQS requests whose respective *request-id* is listed on the command line. *Qhold* will **not** hold a *running* NQS request.

To hold an NQS request, the invoking user *must* be the owner; namely the submitter of the request. The only exception to this rule occurs when the invoking user is the *superuser*, or has NQS operator privileges as defined in the NQS manager database. Under these conditions, the invoker may specify the **−u** username flag which allows the invoker to hold requests owned by the user whose account name is *username*. When this form of the command is used, **all** *request-ids* listed on the command line are presumed to refer to requests owned by the specified user.

An NQS request is always uniquely identified by its *request-id*, no matter where it is in the network of the machines comprising the NPSN. A *request-id* is always of the form: *seqno* or *seqno.hostname* where *hostname* identifies the machine from whence the request was originally submitted, and *seqno* identifies the sequence number assigned to the request on the originating host. If the *hostname* portion of a *request-id* is omitted, then the local host is always assumed.

The *request-id* of any NQS request is displayed when the request is first submitted (unless the *silent* mode of operation for the given NQS command was specified). The user can also obtain the *request-id* of any request through the use of the *qstat*(1) command.

SEE ALSO

qdel(1), qrls(1), qstat(1).
qmgr(1M) in the *NPSN UNIX System Administrator Reference Manual*.

NPSN HISTORY

Origin: CERN

January 1992 − Christian Boissat, CERN
Original release.

NAME

qjob − display status of NQS requests in a networked environment.

SYNOPSIS

qjob [**−a**]

DESCRIPTION

Qjob displays the status of requests known to the Network Queueing System (NQS) as remote ones. The */etc/batchservers* file should contain host names for machines *currently* in connection with the local host via pipe queues. Each entry consists of a line specifying the name of the machine. This file is normally created and maintained by NQS queue managers.

The current state of each NQS request on the remote hosts is displayed. Each entry displays information about a given request. Ordinarily, *qjob* shows only those requests belonging to the invoker. Nevertheless the following flag is available:

-a Displays all requests.

REQUEST STATE

The state of a request may be *arriving*, *holding*, *waiting*, *queued*, *staging*, *routing*, *running*, *departing*, or *exiting*. A request is said to be *arriving* if it is being enqueued from a remote host. *Holding* indicates that the request is presently prevented from entering any other state (including the *running* state), because a *hold* has been placed on the request. A request is said to be *waiting* if it was submitted with the constraint that it not run before a certain date and time, and that date and time have not yet arrived. *Queued* requests are eligible to proceed (by *routing* or *running*). When a request reaches the head of a pipe queue and receives service there, it is *routing*. A request is *departing* from the time the pipe queue turns to other work until the request has arrived intact at its destination. *Staging* denotes a *batch* request that has not yet begun execution, but for which input files are being brought on to the execution machine. A *running* request has reached its final destination queue, and is actually executing. Finally, *exiting* describes a batch request that has completed execution, and will exit from the system after the required output files have been returned (to possibly remote machines).

Imagine a batch request originating on a workstation, destined for the batch queue of a computation engine, to be run immediately. That request would first go through the states *queued*, *routing*, and *departing* in a local pipe queue. Then it would disappear from the pipe queue. From the point of view of a queue on the computation engine, the request would first be *arriving*, then *queued*, *staging* (if required by the batch request), *running*, and finally *exiting*. Upon completion of the *exiting* phase of execution, the batch request would disappear from the batch queue.

CAVEATS

NQS is not finished, and continues to undergo development. Some of the request states shown above may or may not be supported in your version of NQS.

SEE ALSO

qcmplx(1), qdel(1), qdev(1), qlimit(1), qpr(1), qstat(1), qsub(1), qmgr(1M)

NPSN HISTORY

Origin: CERN

August 1991 − Christian Boissat, CERN
Original release.

April 1992
Second release.

NAME

qlimit − show supported batch limits, and shell strategy for the local host.

SYNOPSIS

qlimit

DESCRIPTION

Qlimit displays the set of batch request resource limit types that *can* be directly enforced, and also the *batch request shell strategy* defined for the implied local host.

NQS supports many batch request resource limit types that can be applied to an NQS batch request. However, not all UNIX implementations are capable of supporting the rather extensive set of limit types that NQS provides.

The set of limits applied to a batch request, is always restricted to the set of limits that can be directly supported by the underlying UNIX implementation. If a batch request specifies a limit that cannot be enforced by the underlying UNIX implementation, then the limit is simply ignored, and the batch request will operate as though there were no limit (other than the obvious physical maximums), placed upon that resource type.

When an attempt is made to queue a batch request, each *limit-value* specified by the request (that can also be supported by the local UNIX implementation), is compared against the corresponding *limit-value* as configured for the destination batch queue. If the corresponding batch queue *limit-value* for all batch request *limit-values is defined as unlimited*, or is *greater than* or *equal to* the corresponding batch request *limit-value*, then the request can be successfully queued, provided that no other anomalous conditions occur. For request *infinity limit-values*, the corresponding queue *limit-value* must also be configured as infinity.

These resource limit checks are performed irrespective of the batch request arrival mechanism, either by a direct use of the *qsub*(1) command, or by the indirect placement of a batch request into a batch queue via a *pipe* queue. It is impossible for a batch request to be queued in an NQS batch queue if *any* of these resource limit checks fail.

Finally, if a request fails to specify a *limit-value* for a resource limit type that is supported on the execution machine, then the corresponding *limit-value* as configured for the destination queue, becomes the *limit-value* for the unspecified request limit.

Upon the successful queueing of a request in a batch queue, the set of limits under which the request will execute is frozen, and will not be modified by subsequent *qmgr*(1M) commands that alter the limits of the containing batch queue.

As mentioned above, this command also displays the *shell strategy* as configured for the implied local host, or named hosts. In the absence of a *shell specification* for a batch request, NQS must choose which shell should be used to execute that batch request. NQS supports three different algorithms, or *strategies* to solve this problem that can be configured for each system by a system administrator, depending on the needs of the user's involved, and upon system performance criterion.

The three possible shell strategies are called:

fixed,
free, and
login.

These shell strategies respectively cause the configured *fixed* shell to be exec'd to interpret all batch requests, cause the user's login shell as defined in the password file to be exec'd which in turn chooses and spawns the appropriate shell for running the batch shell script, or cause only the user's login shell to be exec'd to interpret the script.

A shell strategy of *fixed* means that the same shell as chosen by the system administrator, will be used to execute **all** batch requests.

A shell strategy of *free* will run the batch request script *exactly* as would an interactive invocation of the script, and is the default NQS shell strategy.

The strategies of *fixed*, and *login* exist for host systems that are short on available free processes. In these two strategies, a single shell is exec'd, and that same shell is the shell that executes all of the commands in the batch request shell script.

When a shell strategy of *fixed* has been configured for a particular NQS system, then the "fixed" shell that will be used to run **all** batch requests at that host is displayed.

SEE ALSO

qdel(1), qdev(1), qpr(1), qstat(1), qsub(1), qmgr(1M)

NPSN HISTORY

Origin: Sterling Software Incorporated

May 1986 – Brent Kingsbury, Sterling Software
Original release.

NAME

qpr − submit a hardcopy print request to NQS

SYNOPSIS

qpr [**−a** date-time] [**−f** form-name] [**−mb**] [**−me**]
[**−mu** user-name] [**−n** number-of-copies] [**−p** priority]
[**−q** queue-name] [**−r** request-name] [**−z**] [files]

DESCRIPTION

Qpr places the named files in a *Network Queueing System* (NQS) queue to be printed by a device such as a line printer or laser printer. If no files are specified, *qpr* will read from the standard input.

In the absence of the **−z** flag, *qpr* will print a *request-id* on the standard output, upon successful queueing of a request. This *request-id* can be compared with what is reported by *qdev*(1) and *qstat*(1) to find out what happened to a request, and given as an argument to *qdel*(1) to delete a request. A *request-id* is always of the form: *seqno.hostname* where *seqno* refers to the sequence number assigned to the request by NQS, and *hostname* refers to the name of originating local machine. This identifier is used throughout NQS to uniquely identify the request, no matter where it is in the network.

The following options to *qpr* may appear in any order and may be intermixed with file names.

−a *date-time*

Submit at the specified date and/or time. In the absence of this flag, *qpr* will submit the request immediately.

If a *date-time* specification is comprised of two or more tokens separated by whitespace characters, then the *date-time* specification must be placed within double quotes as in: **−a** *"July, 4, 2026 12:31-EDT"*, or otherwise escaped such that the shell will interpret the entire *date-time* specification as a single lexical token.

The syntax accepted for the *date-time* parameter is relatively flexible. Unspecified date and time values default to an appropriate value (e.g. if no date is specified, then the current month, day, and year are assumed).

A date can be specified as a month and day (current year assumed). The year can also be explicitly specified. It is also possible to specify the date as a weekday name (e.g. "Tues"), or as one of the strings "today" or "tomorrow". Weekday names and month names can be abbreviated by any three character (or longer) prefix of the actual name. An optional period can follow an abbreviated month or day name.

Time of day specifications can be given using a twenty-four hour clock, or "am" and "pm" specifications may be used alternatively. In the absence of a meridian specification, a twenty-four hour clock is assumed.

It should be noted that the time of day specification is interpreted using the precise meridian definitions whereby "12am" refers to the twenty-four hour clock time of 0:00:00, "12m" refers to noon, and "12-pm" refers to 24:00:00. Alternatively, the phrases "midnight" and "noon" are accepted as time of day specifications, where "midnight" refers to the time of 24:00:00.

A timezone may also appear at any point in the *date-time* specification. Thus, it is legal to say: "April 1, 1987 13:01-PDT". In the absence of a timezone specification, the local timezone is assumed, with daylight savings time being inferred when appropriate, based on the date specified.

All alphabetic comparisons are performed in a case insensitive fashion such that both "WeD" and "weD" refer to the day of Wednesday.

Some valid *date-time* examples are:

```
01-Jan-1986 12am, PDT
Tuesday, 23:00:00
11pm tues.
```

tomorrow 23-MST

−f *form-name*

Limit the set of acceptable devices to those devices which are loaded with the forms: *form-name*. In the absence of this flag, *qpr* will submit the request only to a device that is loaded with the *default* forms. If there is no *default* forms defined, the request will be submitted to the appropriate output device without regard to the forms configured for the device.

In any case, only those devices associated with the chosen queue will be considered.

−mb Send mail to the user on the originating machine when the request begins execution. If the **−mu** flag is also present, then mail is sent to the user specified for the **−mu** flag instead of to the invoking user.

−me Send mail to the invoker on the originating machine when the request has ended execution. If the **−mu** flag is also present, then mail is sent to the user specified for the **−mu** flag instead of to the invoking user.

−mu *user-name*

Specify that any mail concerning the request should be delivered to the user *user-name*. *User-name* may be formatted either as *user* (containing no '@' characters), or as *user@machine*. In the absence of this flag, any mail concerning the request will be sent to the invoker on the originating machine.

−n *number-of-copies*

Print *number-of-copies* copies. The default is one.

−p *priority*

Assign an intra-queue priority to this request. The specified *priority* must be an integer, and must be in the range [0..63], inclusive. A value of 63 defines the highest *intra-queue* request priority, while a value of 0 defines the lowest. This priority does **not** determine the execution priority of the request. This priority is only used to determine the relative ordering of requests within a queue.

When a request is added to a queue, it is placed at a specific position within the queue such that it appears ahead of all existing requests whose priority is less than the priority of the new request. Similarly, all requests with a higher priority will remain ahead of the new request when the queueing process is complete. When the priority of the new request is equal to the priority of an existing request, the existing request takes precedence over the new request.

If no *intra-queue* priority is chosen by the user, then NQS assigns a default value.

−q *queue-name*

Specify the queue to which the device request is to be submitted. If no **−q** *queue-name* specification is given, then the user's environment variable set is searched for the variable: **QPR_QUEUE**. If this environment variable is found, then the character string value for **QPR_QUEUE** is presumed to name the queue to which the request should be submitted. If the **QPR_QUEUE** environment variable is not found, then the request will be submitted to the default device request queue, *if* defined by the local system administrator. Otherwise, the request cannot be queued, and an appropriate error message is displayed to this effect.

−r *request-name*

Assign a name to this request. In the absence of an explict **−r** *request-name* specification, the *request-name* defaults to the name of the first print file (leading path name removed) specified on the command line. If no print files were specified, then the default *request-name* assigned to the request is **STDIN**.

In all cases, if the *request-name* is found to begin with a digit, then the character 'R' is prepended to prevent a *request-name* from beginning with a digit. All *request-names* are truncated to a maximum length of 15 characters.

Be sure not to confuse *request-name* with *request-id*.

−z Submit the request silently. If the request is submitted successfully, nothing will be written to stdout or stderr.

QUEUE ACCESS

NQS supports queue access restrictions. For each queue of queue type other than *network*, access may be either *unrestricted* or *restricted*. If access is *unrestricted*, any request may enter the queue. If access is *restricted*, a request can only enter the queue if the requester or the requester's login group has been given access to that queue (see *qmgr*(1M)). Requests submitted by root are an exception; they are always queued, even if root has not explicitly been given access.

Use *qstat*(1) to determine who has access to a particular queue.

SEE ALSO

mail(1), qdel(1), qdev(1), qlimit(1), qstat(1), qsub(1), qmgr(1M)

NPSN HISTORY

Origin: Sterling Software Incorporated

May 1986 − Robert Sandstrom, Sterling Software
Original release.

NAME

qrls − release NQS request(s).

SYNOPSIS

qrls [**−u** username] request-id ...

DESCRIPTION

Qrls releases all held NQS requests whose respective *request-id* is listed on the command line.

To release an NQS request, the invoking user *must* be the owner; namely the submitter of the request. The only exception to this rule occurs when the invoking user is the *superuser*, or has NQS operator privileges as defined in the NQS manager database. Under these conditions, the invoker may specify the **−u** username flag which allows the invoker to release requests owned by the user whose account name is *username*. When this form of the command is used, **all** *request-ids* listed on the command line are presumed to refer to requests owned by the specified user.

An NQS request is always uniquely identified by its *request-id*, no matter where it is in the network of the machines comprising the NPSN. A *request-id* is always of the form: *seqno* or *seqno.hostname* where *hostname* identifies the machine from whence the request was originally submitted, and *seqno* identifies the sequence number assigned to the request on the originating host. If the *hostname* portion of a *request-id* is omitted, then the local host is always assumed.

The *request-id* of any NQS request is displayed when the request is first submitted (unless the *silent* mode of operation for the given NQS command was specified). The user can also obtain the *request-id* of any request through the use of the *qstat*(1) command.

SEE ALSO

qdel(1), qhold(1), qstat(1).
qmgr(1M) in the *NPSN UNIX System Administrator Reference Manual*.

NPSN HISTORY

Origin: CERN

January 1992 − Christian Boissat, CERN
Original release.

NAME

qstat − display status of NQS requests and queues.

SYNOPSIS

qstat [**−a**] [**−U**] [**−b**] [**−d**] [**−p**] [**−f**] [**−l**] [**−n**] [**−s** state -rqht-] [**−h** target-host] [**−u** user-name] [**−A** user-target-name]
[queue-name ...] [queue-name@host-name ...]
[request-id ...] [request-id.hostname ...]

DESCRIPTION

Qstat displays the status of Network Queueing System (NQS) requests and queues.

If no objects are specified, then the current state of each NQS request on the local host is displayed. Otherwise, information is displayed for the specified object only. Each entry displays information about a given request. Ordinarily, *qstat* shows only those requests belonging to the invoker.

An NQS request is always uniquely identified by its *request-id*, no matter where it is in the network of the machines. A *request-id* is always of the form: *seqno* or *seqno.hostname* where *hostname* identifies the machine from whence the request was originally submitted, and *seqno* identifies the sequence number assigned to the request on the originating host. If the *hostname* portion of a *request-id* is omitted, then the local host is always assumed.

If information about the queues is requested with the **-b**, **-d** or **-p** options, but no queues are specified, then the current state of each NQS queue on the local host is displayed. Otherwise, information is displayed for the specified queues only. Queues may be specified either as *queue-name* or *queue-name@host-name*. In the absence of a *host-name* specifier, the local host is assumed. You must have an account on the host specfied in order for qstat to work. Also, *root* use of qstat is limited to the local machine.

For each selected queue, *qstat* displays information about the queue itself. The following flags are available:

-a Displays all requests. The **-U** (unrestricted) option is synonymous.

-b Displays batch queues.

-d Displays device queues.

-p Displays pipe queues.

-f Queues are shown in a full format. The **-l** (long) option displays in the same format.

-n The queue header and trailer are not displayed.

-s Displays requests which are in a particular state: running, queued, held, or transiting (r, q, h, or t).

-h *target-host*
Specifies the target NQS host from which display information is to be obtained.

-u *user-name*
Shows only those requests belonging to *user-name*. The -A (account) option is synonymous.

When a queue is being examined, the queue name, host machine, priority, number of requests in a given state, resource limits, and access are displayed.

QUEUE STATE

The general state of a queue is defined by two principal properties of the queue.

The first property determines whether or not requests can be submitted to the queue. If they can, then the queue is said to be *enabled*. Otherwise the queue is said to be *disabled*.

The second principal property of a queue determines if requests which are ready to run, but are not now presently running, will be allowed to run upon the completion of any currently running requests, and whether any requests are presently running in the queue.

If queued requests not already running are blocked from running, and no requests are presently executing in the queue, then the queue is said to be *stopped*. If the same situation exists with the difference that at least one request is running, then the queue is said to be *stopping*, where the requests presently executing will be allowed to complete execution, but no new requests will be spawned.

One of the words *AVAILABLE*, *STOPPED*, *DISABLED*, *UNAVAIL*, or *NQS DOWN* will appear in the queue status field to indicate the respective queue states of:

AVAILABLE= enabled and started,
STOPPED = enabled and stopped,
DISABLED = disabled and running or
UNAVAIL = disabled and stopped.

Requests can only be submitted to the queue if the queue is enabled, and the local NQS daemon is present.

If the NQS daemon for the local host upon which the queue resides is not running, the status displays *NQS DOWN*.

REQUEST STATE

The state of a request may be *arriving*, *holding*, *waiting*, *queued*, *staging*, *routing*, *running*, *departing*, or *exiting*. A request is said to be *arriving* if it is being enqueued from a remote host. *Holding* indicates that the request is presently prevented from entering any other state (including the *running* state), because a *hold* has been placed on the request. A request is said to be *waiting* if it was submitted with the constraint that it not run before a certain date and time, and that date and time have not yet arrived. *Queued* requests are eligible to proceed (by *routing* or *running*). When a request reaches the head of a pipe queue and receives service there, it is *routing*. A request is *departing* from the time the pipe queue turns to other work until the request has arrived intact at its destination. *Staging* denotes a *batch* request that has not yet begun execution, but for which input files are being brought on to the execution machine. A *running* request has reached its final destination queue, and is actually executing. Finally, *exiting* describes a batch request that has completed execution, and will exit from the system after the required output files have been returned (to possibly remote machines).

Imagine a batch request originating on a workstation, destined for the batch queue of a computation engine, to be run immediately. That request would first go through the states *queued*, *routing*, and *departing* in a local pipe queue. Then it would disappear from the pipe queue. From the point of view of a queue on the computation engine, the request would first be *arriving*, then *queued*, *staging* (if required by the batch request), *running*, and finally *exiting*. Upon completion of the *exiting* phase of execution, the batch request would disappear from the batch queue.

CAVEATS

NQS is not finished, and continues to undergo development. Some of the request states shown above may or may not be supported in your version of NQS.

SEE ALSO

qcmplx(1), qdel(1), qdev(1), qlimit(1), qpr(1), qsub(1), qmgr(1M)

NPSN HISTORY

Origin: Sterling Software Incorporated

August 1985 − Brent Kingsbury, Sterling Software
Original release.

May 1986
Second release.

Feb. 1990 − Terrie Carver, Computer Sciences Corporation
Third release.

NAME

qsub − submit an NQS batch request.

SYNOPSIS

qsub [flags] [script-file]

DESCRIPTION

Qsub submits a batch request to the Network Queueing System (NQS).

If no *script-file* is specified, then the set of commands to be executed as a batch request is taken directly from the standard input file (*stdin*). In all cases however, the *script file* is spooled, so that later changes will **not** affect previously queued batch requests.

All of the flags that can be specified on the command line can also be specified within the first comment block inside the batch request *script file* as *embedded default flags*. Such flags appearing in the batch request *script file* set default characteristics for the batch request. If the same flag is specified on the command line, then the command line flag (and any associated value) takes precedence over the *embedded* flag. See the section entitled: **LONG DESCRIPTION** for more information on *embedded default flags*.

What follows is a terse definition of the flags implemented by the *Qsub* command (see the section: **LONG DESCRIPTION** for the complete definition and syntax used for each of these flags).

−a − run request after stated time
−e − direct stderr output to stated destination
−eo − direct stderr output to the stdout destination
−ke − keep stderr output on the execution machine
−ko − keep stdout output on the execution machine
−lc − establish per-process corefile size limit
−ld − establish per-process data-segment size limits
−lf − establish per-process permanent-file size limits
−lF − establish per-request permanent-file space limits
−lm − establish per-process memory size limits
−lM − establish per-request memory space limits
−ln − establish per-process nice execution value limit
−ls − establish per-process stack-segment size limits
−lt − establish per-process CPU time limits
−lT − establish per-request CPU time limits
−lv − establish per-process temporary-file size limits
−lV − establish per-request temporary-file space limits
−lw − establish per-process working set limit
−mb − send mail when the request begins execution
−me − send mail when the request ends execution
−mu − send mail for the request to the stated user
−nr − declare that batch request is not restartable
−o − direct stdout output to the stated destination
−p − specify intra-queue request priority
−q − queue request in the stated queue
−r − assign stated request name to the request
−re − remotely access the stderr output file
−ro − remotely access the stdout output file
−s − specify shell to interpret the batch request script
−x − export all environment variables with request
−z − submit the request silently

LONG DESCRIPTION

As described above, it is possible to specify *default* flags within the batch request *script file* that configure the default behavior of the batch request. The algorithm used to scan for such *embedded default flags* within an NQS batch request script file is as follows:

1. Read the first line of the *script file*.
2. If the current line contains only whitespace characters, or the first non-whitespace character of the line is ":", then goto step 7.
3. If the first non-whitespace character of the current line is not a "#" character, then goto step 8.
4. If the second non-whitespace character in the current line is *not* the "@" character, or the character immediately following the second non-whitespace character in the current line is *not* a "$"

   OR

   If the second non-whitespace character is not a "Q" followed immediately by the string "SUB", then goto step 7.
5. If no "-" is present as the first non-whitespace character *immediately* following the "@$" sequence or the "QSUB" sequence, then goto step 8.
6. Process the *embedded* flag, stopping the parsing process upon reaching the end of the line, or upon reaching the first unquoted "#" character.
7. Read the next *script file* line. Goto step 2.
8. End. No more *embedded* flags will be recognized.

Here is an example of the use of *embedded* flags within the *script file*.

```
#
# Batch request script example:
#
# @$-a "11:30pm EDT" -lt "21:10, 20:00"
#         # Run request after 11:30 EDT by default,
#         # and set a maximum per-process CPU time
#         # limit of 21 minutes and ten seconds.
#         # Send a warning signal when any process
#         # of the running batch request consumes
#         # more than 20 minutes of CPU time.
# QSUB -lT 1:45:00
#         # Set a maximum per-request CPU time limit
#         # of one hour, and 45 minutes. (The
#         # implementation of CPU time limits is
#         # completely dependent upon the UNIX
#         # implementation at the execution
#         # machine.)
# QSUB-mb -me # Send mail at beginning and end of
#         # request execution.
# @$-q batch1 # Queue request to queue: batch1 by
#         # default.
# @$       # No more embedded flags.
#
make all
```

The following paragraphs give the detailed descriptions of the *flags* supported by the *Qsub* command.

−a *date-time* Do not run the batch request before the specified date and/or time. If a *date-time* specification is comprised of two or more tokens separated by whitespace characters, then the *date-time* specification must be placed within double quotes as in: **−a** *"July, 4, 2026 12:31-EDT"*, or otherwise escaped such that *Qsub* and the shell will interpret the entire *date-time* specification as a single character string. This restriction also applies when an embedded default **−a** flag with accompanying *date-time* specification appears within the batch request *script file*.

The syntax accepted for the *date-time* parameter is relatively flexible. Unspecified date and time values default to an appropriate value (e.g. if no date is specified, then the current month, day, and year are assumed).

A date may be specified as a month and day (current year assumed), or the year can also be explicitly specified. It is also possible to specify the date as a weekday name (e.g. "Tues"), or as one of the strings: "today", or "tomorrow". Weekday names and month names can be abbreviated by any three character (or longer) prefix of the actual name. An optional period can follow an abbreviated month or day name.

Time of day specifications can be given using a twenty-four hour clock, or "am" and "pm" specifications may be used alternatively. In the absence of a meridian specification, a twenty-four hour clock is assumed.

It should be noted that the time of day specification is interpreted using the precise meridian definitions whereby "12am" refers to the twenty-four hour clock time of 0:00:00, "12m" refers to noon, and "12-pm" refers to 24:00:00. Alternatively, the phrases "midnight" and "noon" are accepted as time of day specifications, where "midnight" refers to the time of 24:00:00.

A timezone may also appear at any point in the *date-time* specification. Thus, it is legal to say: "April 1, 1987 13:01-PDT". In the absence of a timezone specification, the local timezone is assumed, with daylight savings time being inferred when appropriate, based on the date specified.

All alphabetic comparisons are performed in a case insensitive fashion such that both "WeD" and "weD" refer to the day of Wednesday.

Some valid *date-time* examples are:

01-Jan-1986 12am, PDT
Tuesday, 23:00:00
11pm tues.
tomorrow 23-MST

−e *[machine:][[/]path/] stderr-filename*
Direct output generated by the batch request which is sent to the *stderr* file to the named *[machine:][[/]path/] stderr-filename*.

The brackets "[" and "]" enclose optional portions of the *stderr* destination *machine*, *path*, and *stderr-filename*.

If no explicit *machine* destination is specified, then the destination machine defaults to the machine that originated the batch request, or to the machine that will eventually run the request, depending on the respective absence, or presence of the **−ke** flag.

If no *machine* destination is specified, and the path/filename does not begin with a "/", then the current working directory is prepended to create a fully qualified path name, provided that the **−ke** (keep stderr) flag is also absent. In all other cases, any partial path/filename is interpreted relative to the user's home directory on the *stderr* destination machine.

This flag cannot be specified when the **−eo** flag option is also present.

If the **−eo** and **−e** *[machine:][[/]path/] stderr-filename* flag options are not present, then all *stderr* output for the batch request is sent to the file whose name consists of the first seven characters of the *request-name* followed by the characters: ".e", followed by the request sequence number portion of the *request-id* discussed below. In the absence of the **−ke** flag, this default *stderr* output file will be placed on the machine that originated the batch request in the current working directory, as defined when the batch request was first submitted. Otherwise, the file will be placed in the user's home directory on the execution machine.

−eo Direct all output that would normally be sent to the *stderr* file to the *stdout* file for the batch request. This flag cannot be specified when the **−e** *[machine:][[/]path/] stderr-filename* flag option is also present.

−ke In the absence of an explicit *machine* destination for the *stderr* file produced by a batch request, the *machine* destination chosen for the *stderr* output file is the machine that originated the batch request. In some cases however, this behavior may be undesirable, and so the **−ke** flag can be specified which instructs NQS to leave any *stderr* output file produced by the request on the machine that actually *executed* the batch request.

This flag is meaningless if the **−eo** flag is specified, and cannot be specified if an explicit *machine* destination is given for the *stderr* parameter of the **−e** flag.

−ko In the absence of an explicit *machine* destination for the *stdout* file produced by a batch request, the *machine* destination chosen for the *stdout* output file is the machine that originated the batch request. In some cases however, this behavior may be undesirable, and so the **−ko** flag can be specified which instructs NQS to leave any *stdout* output file produced by the request on the machine that actually *executed* the batch request.

This flag cannot be specified if an explicit *machine* destination is given for the *stdout* parameter of the **−o** flag.

−lc *per-process corefile size limit*

Set a *per-process* maximum *core file size limit* for all processes that constitute the running batch request. If any process comprising the running request attempts to exit creating a core file whose size would exceed the maximum *per-process core file size limit* for the request, then the core file image of the aborting process will be reduced to the necessary size by an algorithm dependent upon the underlying UNIX implementation.

Not all UNIX implementations support *per-process corefile size limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-process corefile size limit*.

−ld *per-process data-segment size limit [, warn-limit]*

Set a *per-process* maximum and an optional warning *data-segment size limit* for all processes that constitute the running batch request. If any process comprising the running request exceeds the maximum *per-process data-segment size-limit* for the request, then that process is terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-process data-segment warning size limits*. When a warning limit is exceeded, a signal as determined by the underlying UNIX implementation is delivered to the offending process.

If a maximum limit (and optional warning limit) specification is comprised of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes, or otherwise escaped such that *Qsub* and the shell will interpret the entire specification as a single character string token. This caveat also applies when an embedded default **−ld** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-process data-segment size limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-process data-segment size limit*.

–lf *per-process permanent-file size limit [, warn-limit]*

Set a *per-process* maximum and an optional warning *permanent-file size limit* for all processes that constitute the running batch request. If any process comprising the running request attempts to write to a permanent file such that the file size would increase beyond the maximum *per-process permanent-file size limit* for the request, then that process is terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-process warning permanent-file size limits*. When a warning limit is exceeded, a signal as determined by the underlying UNIX implementation is delivered to the offending process.

If a maximum limit (and optional warning limit) specification is comprised of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes, or otherwise escaped such that *Qsub* and the shell will interpret the entire specification as a single character string token. This caveat also applies when an embedded default **–lf** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-process permanent-file size limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

At the time of this writing, the author was unaware of any UNIX implementation that made a distinction at the **kernel** level, between *permanent*, and *temporary* files. While it is certainly possible to construct a *pseudo-temporary* file by first creating it, and then unlinking its pathname, the disk space allocated for such a file will be allocated from the same pool of disk space that all other UNIX files are allocated from. Furthermore, such a file will be subject to the same quota enforcement mechanisms, if any are provided by the underlying UNIX implementation, that all other UNIX files are created under.

For all UNIX implementations that do not support a distinction between *permanent*, and *temporary* files at the **kernel** level, this limit is interpreted as a *per-process file size limit*, with the word *permanent* removed from the definition.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-process permanent-file size limit*.

–lF *per-request permanent-file space limit [, warn-limit]*

Set a *per-request* maximum and an optional warning cumulative *permanent-file space limit* for all processes that constitute the running batch request. If any process comprising the running request attempts to write to a permanent file such that the file space consumed by all permanent files opened for writing by all of the processes in the batch request, would increase beyond the maximum *per-request permanent-file space limit* for the request, then all of the processes in the request will be terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-request warning permanent-file space limits*. When such a warning limit is exceeded, a signal is delivered to one or more of the processes comprising the running request, depending upon the underlying UNIX implementation.

If a maximum limit (and optional warning limit) specification is comprised of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes, or otherwise escaped such that *Qsub* and the shell will interpret the entire specification as a single character string token. This caveat also applies when an embedded default **–lF** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-request permanent-file space limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

At the time of this writing, the author was unaware of any UNIX implementation that made a distinction at the **kernel** level, between *permanent*, and *temporary* files. While it is certainly possible to construct a *pseudo-temporary* file by first creating it, and then unlinking its pathname, the disk space allocated for such a file will be allocated from the same pool of disk space that all other UNIX files are allocated from. Furthermore, such a file will be subject to the same quota enforcement mechanisms, if any are provided by the underlying UNIX implementation, that all other UNIX files are created under.

For all UNIX implementations that do not support a distinction between *permanent*, and *temporary* files at the **kernel** level, this limit is interpreted as a *per-request file space limit*, with the word *permanent* removed from the definition.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-request permanent-file space limit*.

–lm *per-process memory size limit [, warn-limit]*

Set a *per-process* maximum and an optional warning *memory size limit* for all processes that constitute the running batch request. If any process comprising the running request exceeds the maximum *per-process memory size limit* for the request, then that process is terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-process warning memory size limits*. When a warning limit is exceeded, a signal as determined by the underlying UNIX implementation is delivered to the offending process.

If a maximum limit (and optional warning limit) specification is comprised of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes, or otherwise escaped such that *Qsub* and the shell will interpret the entire specification as a single character string token. This caveat also applies when an embedded default **–lm** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-process memory size limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-process memory size limit*.

–lM *per-request memory space limit [, warn-limit]*

Set a *per-request* maximum and an optional warning cumulative *memory space limit* for all processes that constitute the running batch request. If the sum of all memory consumed by all of the processes comprising the running request exceeds the maximum *per-request memory space limit* for the request, then all of the processes in the request will be terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-request warning memory size limits*. When such a warning limit is exceeded, a signal is delivered to one or more of the processes comprising the running request, depending upon the underlying UNIX implementation.

If a maximum limit (and optional warning limit) specification is comprised of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes, or otherwise escaped such that *Qsub* and the shell will interpret the entire specification as a single character string token. This caveat also applies when an embedded default **–lM** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-request memory space limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-request memory space limit*.

–ln *per-process nice value limit*

Set a *per-process nice value* for all processes comprising the running batch request.

At present, all UNIX implementations support the use of an integer called the *nice* value, which determines the *execution-time* priority of a process relative to all other processes in the system. By letting the user set a limit on the *nice* value for all processes comprising the running request, a user can cause a request to consume less (or more) of the CPU resource presented by the execution machine.

This is particularly useful when a user wishes to execute a CPU intensive batch request on a machine running interactive processes. By setting a low *execution-time priority*, a user can make a long running batch request give way to more interactive processes during the daytime, while the coming of the nighttime hours with typically smaller process loads will allow such a request to gain more and more of the CPU resource. In this way, long running batch requests can be polite to their more transient, interactive neighbor processes.

The only quirk associated with this flag results from the peculiar choice of *nice* values, implemented by the standard UNIX implementations. In general, increasingly *negative* nice values cause the relative execution priority of a process to *increase*, while increasingly *positive* nice values causes the relative priority to *decrease*! Thus, a *nice value* limit specification of: "-ln -10" is greedier than a *nice value* limit specification of: "-ln 0".

Since varying UNIX implementations often support a different finite range of *nice values*, NQS allows the specification of *nice values* that can eventually turn out to be outside the limits for the UNIX implementation running at the *execution* machine. In such cases, NQS will simply bind the specified *nice value* limit to within the necessary range as appropriate.

Lastly, any *nice value* specified by the use of this flag must be acceptable to the batch queue in which the request is ultimately placed (see the section entitled **LIMITS** for more information).

–ls *per-process stack-segment size limit [, warn-limit]*

Set a *per-process* maximum and an optional warning *stack-segment size limit* for all processes that constitute the running batch request. If any process comprising the running request exceeds the maximum *per-process stack-segment size limit* for the request, then that process is terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-process warning stack-segment size limits*. When a warning limit is exceeded, a signal as determined by the underlying UNIX implementation is delivered to the offending process.

If a maximum limit (and optional warning limit) specification is comprised of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes, or otherwise escaped such that *Qsub* and the shell will interpret the entire specification as a single character string token. This caveat also applies when an embedded default **–ls** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-process stack-segment size limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-process stack-segment size limit*.

–lt *per-process CPU time limit [, warn-limit]*

Set a *per-process* maximum and an optional warning *CPU time limit* for all processes that constitute the running batch request. If any process comprising the running request exceeds the maximum *per-process CPU time limit* for the request, then that process is terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-process CPU warning time limits*. When a warning limit is exceeded, a signal as determined by the underlying UNIX implementation is delivered to the offending process.

If a maximum limit (and optional warning limit) specification is comprised of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes, or otherwise escaped such that *Qsub* and the shell will interpret the entire specification as a single character string token. This caveat also applies when an embedded default **–lt** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-process CPU time limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-process CPU time limit*.

–lT *per-request CPU time limit [, warn-limit]*

Set a *per-request* maximum and an optional warning cumulative *CPU time limit* for all of the processes that constitute the running batch request. If the sum of the CPU times consumed by all of the processes in the batch request exceeds the maximum *per-request CPU time limit* for the request, then all of the processes in the request will be terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-request CPU warning time limits*. When such a warning limit is exceeded, a signal is delivered to one or more of the processes comprising the running request, depending upon the underlying UNIX implementation.

If a maximum limit (and optional warning limit) specification is comprised of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes, or otherwise escaped such that *Qsub* and the shell will interpret the entire specification as a single character string token. This caveat also applies when an embedded default **–lT** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-request CPU time limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-request CPU time limit*.

–lv *per-process temporary file size limit [, warn-limit]*

Set a *per-process* maximum and an optional warning *temporary (volatile) file size limit* for all processes that constitute the running batch request. If any process comprising the running request attempts to write to a *temporary* file such that the file size would increase beyond the maximum *per-process temporary-file size limit* for the request, then that process is terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-process warning temporary-file size limits*. When a warning limit is exceeded, a signal as determined by the underlying UNIX implementation is delivered to the offending process.

If a maximum limit (and optional warning limit) specification is comprised of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes, or otherwise escaped such that *Qsub* and the shell will interpret the entire specification as a single character string token. This caveat also applies when an embedded default **–lv** flag with its associated limit value(s) appears within the batch request *script file*.

At the time of this writing, no UNIX operating system known to the author supported a distinction at the **kernel** level between *permanent* and *temporary files*. Certainly, a *pseudo-temporary* file can be constructed by creating it, and then unlinking its pathname. However, the file space allocated for such a file will be allocated from the same pool of disk space that all other UNIX files are allocated from.

Until a mechanism is implemented in the **kernel** that knows about *permanent* and *temporary* files, this limit cannot be supported in the sense most useful for batch requests, namely the strict enforcement of disk quotas for *permanent* versus *temporary* files.

Until such a time, this limit will simply be ignored.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-process temporary-file size limit*.

–lV *per-request temporary file space limit [, warn-limit]*

Set a *per-request* maximum and an optional warning cumulative *temporary (volatile) file space limit* for all processes that constitute the running batch request. If any process comprising the running request attempts to write to a *temporary* file such that the file space consumed by all *temporary* files opened for writing by all of the processes in the batch request would increase beyond the maximum *per-request temporary-file space limit* for the request, then all of the processes in the request will be terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-request warning temporary-file space limits*. When such a warning limit is exceeded, a signal is delivered to one or more of the processes comprising the running request, depending upon the underlying UNIX implementation.

If a maximum limit (and optional warning limit) specification is comprised of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes, or otherwise escaped such that *Qsub* and the shell will interpret the entire specification as a single character string token. This caveat also applies when an embedded default **–lV** flag with its associated limit value(s) appears within the batch request *script file*.

At the time of this writing, no UNIX operating system known to the author supported a distinction at the **kernel** level between *permanent* and *temporary files*. Certainly, a *pseudo-temporary* file can be constructed by creating it, and then unlinking its pathname. However, the file space allocated for such a file will be allocated from the same pool of disk space that all other UNIX files are allocated from.

Until a mechanism is implemented in the **kernel** that knows about *permanent* and *temporary* files, this limit cannot be supported in the sense most useful for batch requests, namely the strict enforcement of disk quotas for *permanent* versus *temporary* files.

Until such a time, this limit will simply be ignored.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *temporary-file space limit*.

−lw *per-process working set size limit*

Set a *per-process* maximum *working set size limit* for all processes that constitute the running batch request.

Not all UNIX implementations support *per-process working set size limits*, and such a limit only makes sense in the context of a paged virtual memory machine. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-process working set size limit*.

−mb Send mail to the user on the originating machine when the request begins execution. If the **−mu** flag is also present, then mail is sent to the user specified for the **−mu** flag instead of to the invoking user.

−me Send mail to the user on the originating machine when the request has ended execution. If the **−mu** flag is also present, then mail is sent to the user specified for the **−mu** flag instead of to the invoking user.

−mu *user-name*

Specify that any mail concerning the request should be delivered to the user *user-name*. *User-name* may be formatted either as *user* (containing no '@' characters), or as *user@machine*. In the absence of this flag, any mail concerning the request will be sent to the invoker on the originating machine.

−nr Declare that the request is non-restartable. If this flag is specified, then the request will not be restarted by NQS upon system boot if the request was running at the time of an NQS shutdown or system crash.

By default, NQS assumes that all requests are restartable, with the caveat that it is the responsibility of the user to ensure that the request will execute correctly if restarted, by the use of appropriate programming techniques.

Requests that are not running are always preserved across host crashes and NQS shutdowns for later requeueing, with or without this flag.

When NQS is shutdown via an operator command to the *qmgr*(1M) NQS control program, a **SIGTERM** signal is sent to all processes comprising all running NQS requests on the local host, and all queued NQS requests are barred from beginning execution. After a finite number of seconds have elapsed (typically sixty, but this value can be overridden by the operator), all remaining processes comprising all remaining running NQS requests are killed by the signal: **SIGKILL**.

For an NQS request to be properly restarted after an NQS shutdown, the **−nr** flag must not be specified, and the spawned batch request shell must ignore **SIGTERM** signals (which is done by default). The spawned batch request shell must also not exit before the final **SIGKILL** arrives. Since the batch request shell is simply spawning commands and programs, waiting for their completion, this implies that the commands and programs being executed by the batch request shell must also be immune to **SIGTERM** signals, saving state as appropriate before being killed by the final **SIGKILL** signal.

See the **CAVEATS** section below for more discussion concerning the restartability of batch requests.

−o *[machine:][[/]path/] stdout-filename*

Direct output generated by the batch request which is sent to the *stdout* file to the named *[machine:][[/]path/] stdout-filename*.

The brackets "[" and "]" enclose optional portions of the *stdout* destination *machine*, *path*, and *stdout-filename*.

If no explicit *machine* destination is specified, then the destination machine defaults to the machine that originated the batch request, or to the machine that will eventually run the request, depending on the respective absence, or presence of the **−ko** flag.

If no *machine* destination is specified, and the path/filename does not begin with a "/", then the current working directory is prepended to create a fully qualified path name, provided that the **−ko** (keep stdout) flag is also absent. In all other cases, any partial path/filename is interpreted relative to the user's home directory on the *stdout* destination machine.

If no **−o** *[machine:][[/]path/] stdout-filename* flag is specified, then all *stdout* output for the batch request is sent to the file whose name consists of the first seven characters of the *request-name* followed by the characters: ".o", followed by the request sequence number portion of the *request-id* discussed below. In the absence of the **−ko** flag, this default *stdout* output file will be placed on the machine that originated the batch request in the current working directory, as defined when the batch request was first submitted. Otherwise, the file will be placed in the user's home directory on the execution machine.

−p *priority* Explicitly assign an *intra-queue* priority to the request. The specified *priority* must be an integer, and must be in the range [0..63], inclusive. A value of 63 defines the highest *intra-queue* request priority, while a value of 0 defines the lowest. This priority does **not** determine the execution priority of the request. This priority is only used to determine the relative ordering of requests within a queue.

When a request is added to a queue, it is placed at a specific position within the queue such that it appears ahead of all existing requests whose priority is less than the priority of the new request. Similarly, all requests with a higher priority will remain ahead of the new request when the queueing process is complete. When the priority of the new request is equal to the priority of an existing request, the existing request takes precedence over the new request.

If no *intra-queue* priority is chosen by the user, then NQS assigns a default value.

−q *queue-name*

Specify the queue to which the batch request is to be submitted. If no **−q** *queue-name* specification is given, then the user's environment variable set is searched for the variable: **QSUB_QUEUE**. If this environment variable is found, then the character string value for **QSUB_QUEUE** is presumed to name the queue to which the request should be submitted. If the **QSUB_QUEUE** environment variable is not found, then the request will be submitted to the default batch request queue, *if* defined by the local system administrator. Otherwise, the request cannot be queued, and an appropriate error message is displayed to this effect.

−r *request-name*

Assign the specified *request-name* to the request. In the absence of an explict **−r** *request-name* specification, the *request-name* defaults to the name of the *script file* (leading path name removed) given on the command line. If no *script file* was given, then the default *request-name* assigned to the request is **STDIN**.

In all cases, if the *request-name* is found to begin with a digit, then the character 'R' is prepended to prevent a *request-name* from beginning with a digit. All *request-names* are truncated to a maximum length of 15 characters.

−re By default, all output generated by a batch request sent to the *stderr* file is temporarily into a file residing in a protected directory on the machine that executes the request. When the batch request completes execution, this file is then spooled to its final destination, possibly on a remote machine.

This default spooling of the *stderr* output file is done to reduce the network traffic costs incurred by the submitter (owner) of a batch request which is to return its *stderr* output to a remote machine upon completion. In some cases, this behavior is not desired. If it is necessary to override this behavior, then the **−re** flag can be specified which says that *stderr* output produced by the request is to be *immediately* written to the final destination file, as output is generated, no matter what the networking cost.

Circumstances may not allow a given NQS implementation to support this flag, in which case it will be ignored, and the *stderr* output file will simply be spooled as it ordinarily would without this flag.

−ro By default, all output generated by a batch request sent to the *stdout* file is temporarily spooled into a file residing in a protected directory on the machine that executes the request. When the batch request completes execution, this file is then spooled to its final destination, possibly on a remote machine.

This default spooling of the *stdout* output file is done to reduce the network traffic costs incurred by the submitter (owner) of a batch request which is to return its *stdout* output to a remote machine upon completion. In some cases, this behavior is not desired. If it is necessary to override this behavior, then the **−ro** flag can be specified which says that *stdout* output produced by the request is to be *immediately* written to the final destination file, as output is generated, no matter what the networking cost.

Circumstances may not allow a given NQS implementation to support this flag, in which case it will be ignored, and the *stdout* output file will simply be spooled as it ordinarily would without this flag.

−s *shell-name*

Specify the absolute path name of the shell which will be used to interpret the batch request script. This flag unconditionally overrides any *shell strategy* configured on the execution machine for selecting which shell to spawn in order to interpret the batch request script.

In the absence of this flag, the NQS system at the execution machine will use one of three (3) distinct *shell choice strategies* for the execution of the batch request. Any one of the three strategies can be configured by a system administrator for each NQS machine.

The three shell strategies are called:

fixed,
free, and
login.

These shell strategies respectively cause the configured *fixed* shell to be exec'd to interpret all batch requests, cause the user's login shell as defined in the password file to be exec'd which in turn chooses and spawns the appropriate shell for interpreting the batch request script, or cause only the user's login shell to be exec'd to interpret the script.

A shell strategy of *fixed* means that the same shell (as configured by the system administrator), will be used to execute **all** batch requests.

A shell strategy of *free* will run the batch request script *exactly* as would an interactive invocation of the script, and is the default NQS shell strategy.

The strategies of *fixed* and *login* exist for host systems that are short on available free processes. In these two strategies, a single shell is exec'd, and that same shell is the shell that executes all of the commands in the batch request script.

The *shell strategy* configured for a particular NQS system can be determined by the *qlimit*(1) command.

−x Export all environment variables. When a batch request is submitted, the current values of the environment variables: **HOME**, **SHELL**, **PATH**, **LOGNAME** (not all systems), **USER** (not all systems), **MAIL**, and **TZ** are saved for later recreation when the batch request is spawned, as the respective environment variables: **QSUB_HOME**, **QSUB_SHELL**, **QSUB_PATH**, **QSUB_LOGNAME**, **QSUB_USER**, **QSUB_MAIL**, and **QSUB_TZ**. Unless the **−x** flag is specified, no other environment variables will be exported from the originating host for the batch request. If the **−x** flag option is specified, then all remaining environment variables whose names do not conflict with the automatically exported variables, are also exported with the request. These additional environment variables will be recreated under the same name when the batch request is spawned.

−z Submit the batch request silently. If the request is submitted successfully, then no messages are displayed indicating this fact. Error messages will, however, always be displayed.

If the batch request is successfully submitted and the **−z** flag has not been specified, the *request-id* of the request is displayed to the user. A *request-id* is always of the form: *seqno.hostname* where *seqno* refers to the sequence number assigned to the request by NQS, and *hostname* refers to the name of originating local machine. This identifier is used throughout NQS to uniquely identify the request, no matter where it is in the network.

The following events take place in the following order when an NQS *batch* request is spawned:

The process that will become the head of the *process group* for all processes comprising the batch request is created by NQS.

Resource limits are enforced.

The real and effective group-id of the process is set to the group-id as defined in the local password file for the request owner.

The real and effective user-id of the process is set to the real user-id of the batch request owner.

The user file creation mask is set to the value that the user had on the originating machine when the batch request was first submitted.

It the user explicitly specified a shell by use of the **−s** flag (discussed above), then that user-specified shell is chosen as the shell that will be used to execute the batch request script. Otherwise, a shell is chosen based upon the *shell strategy* as configured for the local NQS system (see the earlier discussion of the **−s** flag for a description of the possible *shell strategies* that can be configured for an NQS system).

The environment variables of **HOME**, **SHELL**, **PATH**, **LOGNAME** (not all systems), **USER** (not all systems), and **MAIL** are set from the user's password file entry, as though the user had logged directly into the execution machine.

The environment string: **ENVIRONMENT=BATCH** is added to the environment so that shell scripts (and the user's **.profile** (*Bourne shell*) or **.cshrc** and **.login** (*C-shell*) scripts), can test for batch request execution when appropriate, and not (for example) perform any setting of terminal characteristics, since a batch request is not connected to an input terminal.

The environment variables of **QSUB_WORKDIR**, **QSUB_HOST**, **QSUB_REQNAME**, and **QSUB_REQID** are added to the environment. These environment variables equate to the obvious respective strings of the working directory at the time that the request was submitted, the name of the originating host, the name of the request, and the request *request-id*.

All of the remaining environment variables saved for recreation when the batch request is spawned are added at this point to the environment. When a batch request is initially submitted, the current values of the environment variables: **HOME**, **SHELL**, **PATH**, **LOGNAME** (not all systems), **USER** (not all systems), **MAIL**, and **TZ** are saved for later recreation when the batch request is spawned. When recreated however, these variables are added to the environment under the respective names: **QSUB_HOME**, **QSUB_SHELL**, **QSUB_PATH**, **QSUB_LOGNAME**, **QSUB_USER**, **QSUB_MAIL**, and **QSUB_TZ**, to avoid the obvious conflict with the local version of these environment variables. Additionally, all environment variables exported from the originating host by the **−x** option are added to the environment at this time.

The current working directory is then set to the user's home directory on the execution machine, and the chosen shell is exec'd to execute the batch request script with the environment as constructed in the steps outlined above.

In all cases, the chosen shell is exec'd as though it were the *login* shell. If the *Bourne* shell is chosen to execute the script, then the **.profile** file is read. If the *C-shell* is chosen, then the **.cshrc** and **.login** scripts are read.

If the user did not specify a specific shell for the batch request, then NQS chooses which shell should be used to execute the shell script, based on the *shell strategy* as configured by the system administrator (see the earlier discussion of the **–s** flag).

In such a case, a *free* shell strategy instructs NQS to execute the login shell for the user (as configured in the password file). The login shell is in turn instructed to examine the shell script file, and fork another shell *of the appropriate type* to interpret the shell script, behaving *exactly* as an interactive invocation of the script.

Otherwise no additional shell is spawned, and the chosen *fixed* or *login* shell sequentially executes the commands contained in the shell script file until completion of the batch request.

QUEUE TYPES

NQS supports four different queue types that serve to provide four very different functions. These four queue types are known as *batch*, *device*, *pipe*, and *network*.

The queue type of *batch* can only be used to execute NQS *batch requests*. Only NQS *batch requests* created by the *qsub*(1) command can be placed in a *batch queue*.

The queue type of *device* can only be used to execute NQS *device requests*. Only NQS *device requests* created by the *qpr*(1) command can be placed in a *device queue*.

Queues of type *pipe* are used to send NQS requests to other *pipe* queues, or to request destination queues of type *batch* or *device*, as appropriate for the request type. In general, *pipe queues*, in combination with *network queues*, act as the mechanism that NQS uses to transport both *batch* and *device* requests to distant queues on other remote machines. It is also perfectly legal for a *pipe queue* to transport requests to queues on the *same* machine.

When a *pipe queue* is defined, it is given a *destination set* which defines the set of possible destination queues for requests entered in that *pipe queue*. In this manner, it is possible for a *batch* or *device* request to pass through many pipe queues on its way to its ultimate destination, which must eventually be a queue of type *batch* or *device* (matching the request type).

Each *pipe queue* has an associated *server*. For each request handled by a *pipe queue*, the associated server is spawned which must select a queue destination for the request being handled, based on the characteristics of the request, and upon the characteristics of each queue in the *destination set* defined for the pipe queue.

Since a different server can be configured for each pipe queue, and *batch* and *device* queues can be endowed with the *pipeonly* attribute that will only admit requests queued via another *pipe queue*, it is possible for respective NQS installations to use *pipe queues* as a *request class* mechanism, placing requests that ask for different resource allocations in different queues, each of which can have different associated limits and priorities.

It is also completely possible for a *pipe queue server*, when handling a request, to discover that no *destination queue* will accept the request, for various reasons which can include insufficient resource limits to execute the request, or a lack of a corresponding account or privilege for queueing at a remote queue. In such circumstances, the request will be deleted, and the user will be notified by mail (see *mail*(1)).

The queue type of *network*, as alluded to earlier, is implicitly used by *pipe* queues to pass NQS requests between machines, and is also used to handle queued file transfer operations.

QUEUE ACCESS

NQS supports queue access restrictions. For each queue of queue type other than *network*, access may be either *unrestricted* or *restricted*. If access is *unrestricted*, any request may enter the queue. If access is *restricted*, a request can only enter the queue if the requester or the requester's login group has been given access to that queue (see *qmgr*(1M)). Requests submitted by root are an exception; they are always queued, even if root has not explicitly been given access.

Use *qstat*(1) to determine who has access to a particular queue.

LIMITS

NQS supports many batch request resource limit types that can be applied to an NQS batch request. The existence of configurable resource limits allows an NQS user to set resource limits within which his or her request must execute. In many instances, smaller limit values can result in a more favorable scheduling policy for a batch request.

The syntax used to specify a *limit-value* for one of the *limit-flags* (**−l***limit-letter-type*), is quite flexible, and describes values for two general limit categories. These two general categories respectively deal with time related limits, and those limits are not time related.

For *finite* CPU time limits, the *limit-value* is expressed in the reasonably obvious format:

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

Whitespace can appear anywhere between the principal tokens, with the exception that no whitespace can appear around the decimal point.

Example time *limit-values are:*

1234 : 58 : 21.29	− 1234 hrs 58 mins 21.290 secs
12345	− 12345 seconds
121.1	− 121.100 seconds
59:01	− 59 minutes and 1 second

For all other *finite* limits (with the exclusion of the *nice limit-value* **−l***n*), the acceptable syntax is:

.fraction [units]

or

integer [.fraction] [units]

where the *integer* and *fraction* tokens represent strings of up to eight decimal digits, denoting the obvious values. In both cases, the *units* of allocation may also be specified as one of the case insensitive strings:

b	− bytes
w	− words
kb	− kilobytes (2^10 bytes)
kw	− kilowords (2^10 words)
mb	− megabytes (2^20 bytes)
mw	− megawords (2^20 words)
gb	− gigabytes (2^30 bytes)
gw	− gigawords (2^30 words)

In the absence of any *units* specification, the units of *bytes* are assumed.

For all limit types with the exception of the *nice limit-value* (**−l***n*), it is possible to state that no limit should be applied. This is done by specifying a *limit-value* of "unlimited", or any initial substring thereof. Whenever an *infinite limit-value* is specified for a particular resource type, then the batch request operates as though no explicit limits have been placed upon the corresponding resource, other than by the limitations of the physical hardware involved.

The complications caused by *batch request* resource limits first show up when queueing a *batch request* in a *batch queue*. This operation is described in the following paragraphs.

If a batch request specifies a limit that cannot be enforced by the underlying UNIX implementation, then the limit is simply ignored, and the batch request will operate as though there were no limit (other than the obvious physical maximums), placed upon that resource type. (See the *qlimit*(1) command to find out what limits are supported by a given machine.)

For each remaining *finite* limit that can be supported by the underlying UNIX implementation that is **not** a CPU *time-limit* or UNIX *execution-time nice-value-limit*, the *limit-value* is internally converted to the units of *bytes* or *words*, whichever is more appropriate for the underlying machine architecture.

As an example, a *per-process memory size limit value* of 321 megabytes would be interpreted as 321 x 2^20 bytes, provided that the underlying machine architecture was capable of directly addressing single bytes. Thus the original limit *coefficient* of 321 would become 321 x 2^20. On a machine that was only capable of addressing words, the appropriate conversion of 321 x 2^20 *bytes / #of-bytes-per-word* would be performed.

If the result of such a conversion would cause overflow when the coefficient was represented as a *signed-long integer* on the supporting hardware, then the coefficient is replaced with the coefficient of: of 2^*N*-1 where *N* is equal to the number of bits of precision in a signed long integer. For typical 32-bit machines, this *default extreme limit* would therefore be 2^*31*-1 bytes. For word addressable machines in the super-computer class supporting 64-bit long integers, the *default extreme limit* would be 2^*63*-1 words.

Lastly, some implementations of UNIX reserve coefficients of the form: 2^*N*-1 as synonymous with infinity, meaning no limit is to be applied. For such UNIX implementations, NQS further decrements the *default extreme limit* so as not to imply infinity.

The identical internal conversion process as described in the preceding paragraphs is also performed for each *finite limit-value* configured for a particular batch queue using the *qmgr*(1M) program.

After all of the applicable *limit-values* have been converted as described above, each such resulting *limit-value* is then compared against the corresponding *limit-value* as configured for the destination batch queue. If, for every type of limit, the batch queue *limit-value* is *greater than* or *equal to* the corresponding batch request *limit-value*, then the request can be successfully queued, provided that no other anomalous conditions occur. For request *infinity limit-values*, the corresponding queue *limit-value* must also be configured as infinity.

These resource limit checks are performed irrespective of the batch request arrival mechanism, either by a direct use of the *qsub(1)* command, or by the indirect placement of a batch request into a batch queue via a *pipe* queue. It is impossible for a batch request to be queued in an NQS batch queue if *any* of these resource limit checks fail.

Finally, if a request fails to specify a *limit-value* for a resource limit type that is supported on the execution machine, then the corresponding *limit-value* configured for the destination queue becomes the *limit-value* for the unspecified request limit.

Upon the successful queueing of a request in a batch queue, the set of limits under which the request will execute is frozen, and will not be modified by subsequent *qmgr*(1M) commands that alter the limits of the containing batch queue.

CAVEATS

When an NQS batch request is spawned, a new *process-group* is established such that all processes of the request exist in the same *process-group*. If the *qdel*(1) command is used to send a signal to an NQS batch request, the signal is sent to all processes of the request in the created *process-group*. However, should one or more processes of the request choose to successfully execute a *setpgrp*(2) system call, then such processes will **not** receive any signals sent by the *qdel*(1) command. This can lead to "rogue" requests whose constituent processes must be killed by other means such as the *kill*(1) command. However, NQS takes advantage of any UNIX implementations that provide a mechanism of "locking" a process, and all of its subsequent children in a particular *process-group*. For such UNIX implementations, this problem does not occur.

It is extremely wise for all processes of an NQS request to catch any **SIGTERM** signals. By default, the receipt of a **SIGTERM** signal causes the receiving process to die. NQS sends a **SIGTERM** signal to all processes in the established *process-group* for a batch request as a notification that the request should be prepared to be killed, either because of an *abort queue* command issued by an operator using the *qmgr*(1M) program, or because it is necessary to shutdown NQS and all running requests as part of a general shutdown procedure of the local host.

It must be understood that the spawned *shell* ignores **SIGTERM** signals. If the current immediate child of the *shell* does not ignore or catch **SIGTERM** signals, then it will be killed by the receipt of such, and the shell will go on to execute the next command from the script (if there is one). In any case, the shell will not be killed by the **SIGTERM** signal, though the executing command will have been killed.

After receiving a **SIGTERM** signal delivered from NQS, a process of a batch request typically has sixty seconds to get its "house in order" before receiving a **SIGKILL** signal (though the sixty second duration can be changed by the operator).

All batch requests terminated because of an operator *NQS shutdown request* that did not specify the **−nr** flag are considered restartable by NQS, and are requeued (provided that the batch request shell process is still present at the time of the **SIGKILL** signal broadcast as discussed above), so that when NQS is rebooted, such batch requests will be respawned to continue execution. It is however, up to the user to make the request restartable by the appropriate programming techniques. NQS simply spawns the request again as though it were being spawned for the first time.

Upon completion of a batch request, a mail message can be sent to the submitter (see the discussion of the **−me** flag above). In many instances, the completion code of the spawned *Bourne* or *C-Shell* will be displayed. This is merely the value returned by the shell through the *exit*(2) system call.

Lastly, there is no good way to echo commands executed by unmodified versions of the *Bourne* and *C* shells. While the *C-shell* can be spawned in such a fashion as to echo the commands it executes, it is often very difficult to tell an echoed command from genuine output produced by the batch request, because no "magic" character such as a '$' is displayed in front of the echoed command. The *Bourne* shell does not support any echo option whatsoever.

Thus, one of the better ways to write the shell script for a batch request is to place appropriate lines in the shell script of the form:

```
echo "explanatory-message"
```

where the echoed message should be a meaningful message chosen by the user.

LIMITATIONS AND IMPLEMENTATION NOTES

Network queues have not yet been implemented.

In the present implementation, it is **not** possible to see the *stderr* or *stdout* files produced by the batch request while the request is **running**, unless the **-re** and **-ro** flags have been respectively specified.

Lastly, the strange "@$" syntax used to introduce *embedded argument* flags was chosen because it rarely conflicts with anything else present in a shell script file. NQS users with better minds will (rightly) suggest improved alternatives to this convention.

SEE ALSO

qdel(1), qdev(1), qlimit(1), qpr(1), qstat(1), qmgr(1M), plus mail(1), kill(2), setpgrp(2), signal(2)

NPSN HISTORY

Origin: Sterling Software Incorporated

August 1985 − Brent Kingsbury, Sterling Software
Original release.

May 1986
Second release.