

Netbatch 8.2.2

User Guide



Revision 4, June 29, 2015

Intel Confidential

CONFIDENTIAL

Revision History: June 29, 2015

Previous Version:

Page	Subjects (major changes since last revision)

We Listen to Your Comments

Please write to us about any information in this document that you feel is incorrect, unclear or missing.

Your feedback will help us to continuously improve the quality of this document.

Please send your proposal (including a reference to this document) to:

nb.dev@intel.com





Preface

This document explains how to use Netbatch 8.2.2 and associated tools.

About This Document

This User Guide is organized as follows:

- [**Section 1, Introduction**](#)
- [**Section 2, Basic Netbatch Concepts**](#)
- [**Section 3, How Netbatch Works**](#)
- [**Section 4, Quick Start Guide**](#)
- [**Section 5, Resource Allocation Concepts**](#)
- [**Section 6, Submitting Jobs to Netbatch**](#)
- [**Section 7, Using Netbatch Environment Variables**](#)
- [**Section 8, Tracking Job Status**](#)
- [**Section 9, Moving, Changing, and Cancelling Jobs**](#)
- [**Section 10, Submitting and Tracking Jobs with Netbatch Flow Manager**](#)
- [**Section 11, Using Netbatch Tracker**](#)
- [**Section 12, Increasing Throughput with Virtual Pools**](#)
- [**Appendix A:, User Command Reference**](#)
- [**Appendix B:, Precedence of Configuration Directives, Environment Variables, Job Profiles, and Command-Line Switches**](#)
- [**Appendix C:, The New Netbatch Command Suite**](#)
- [**Appendix D:, Log File Contents and Job Exit Codes**](#)
- [**Appendix E:, Environment Variables**](#)
- [**Appendix F:, Smart Classes Reference**](#)
- [**Appendix G:, Regex Replace Functions**](#)
- [**Appendix H:, Functions**](#)



Disclaimer

The dynamic nature of the Netbatch product means that the documentation is constantly being updated. The latest version of the documentation is always available here:

https://sharepoint.ger.ith.intel.com/sites/Netbatch/SiteAssets/HTMLPages/NB_docs_latest.html

The documents that are released with Netbatch are always Revision 1. The revision number is incremented with every documentation update.

Your Comments

We welcome your comments on this document. We are continuously trying to improve our documentation. Please send your remarks and suggestions by email to:

nb.dev@intel.com

In your email, please include the following:

In the subject line:

- Product Name—Netbatch v8.2.2

In the body of your email:

- Document type—User Guide
- Issue date—June 29, 2015
- Document revision number—Revision 4



Table of Contents

1	Introduction	1
1.1	Overview	1
1.2	Location of Netbatch v8.2.2 Executables	1
1.3	A Note about Notation	2
1.4	What's New in v8.2.2	2
2	Basic Netbatch Concepts	7
2.1	Jobs	7
2.1.1	Job	7
2.1.2	Batch Job	7
2.1.3	Express Job	7
2.1.4	Interactive Batch Job	7
2.1.5	Interactive Logon Session	8
2.1.6	Terminal Jobs	8
2.1.7	Parallel Job	8
2.1.8	Job Pre-Execution	9
2.1.9	Job Post-Execution	9
2.1.10	Job Starter	9
2.1.11	Job Profile	10
2.1.12	Job Dependencies	10
2.1.13	Job Priority	10
2.1.14	Job Cost	11
2.1.15	State Queues	11
2.2	Resources	12
2.2.1	Workstation/Compute Server	13
2.2.2	Idle Workstation	13
2.2.3	Workstation Auto-tuning	13
2.2.4	Pool	14
2.2.5	Resource Set	14
2.2.6	Resource Group	14
2.2.7	Physical Pool	15
2.2.8	Pool Master	16
2.2.9	Express Pool	17
2.2.10	Virtual Pool	17
2.2.11	Virtual Pool Master	17
2.2.12	Global Pool	18
2.3	Matching Jobs to Machines	18
2.3.1	Class	18
2.3.2	Smart Classes	19
2.3.3	Resource Reservation	20
2.3.4	Workstation Capabilities	20
2.3.5	External Probe	21
2.4	Resource Allocation	22



2.4.1	Overview	22
2.4.2	Queue	22
2.4.3	Default Queue	23
2.4.4	Logon Queue	23
2.4.5	Express Queue	23
2.4.6	Qslot	24
2.4.7	Default Qslot	24
2.4.8	Qslot Alias	25
2.4.9	What is the Difference between a Queue and a Qslot?	26
2.4.10	Preemption	26
2.4.11	Resource Allocation Methods	27
2.4.12	Resource Allocation Limits	29
2.5	Permissions	29
2.6	License Support	30
2.7	Netbatch Feeder	31
2.7.1	Netbatch Flow Manager	31
2.8	Netbatch Tracker	31
3	How Netbatch Works	33
3.1	Overview	33
3.2	How Netbatch Decides Which Job to Run Next	34
3.2.1	How Netbatch Checks Class Requirements	37
3.3	Job Life Cycle	39
3.3.1	Overview	39
3.3.2	Job Completion Detection	40
3.3.3	States of a Job in Netbatch	41
3.3.4	JobID	42
3.3.5	Suspended Jobs	43
3.4	Netbatch Pool Structures	44
3.4.1	Overview	44
3.4.2	Some Basic Pool Configurations	45
3.4.3	System Components	51
3.4.4	Master Failover and Load Balancing	62
4	Quick Start Guide	63
4.1	Netbatch Commands	63
4.1.1	Old Command Binaries	63
4.1.2	Elementary Commands	63
4.1.3	Common Netbatch Command Switches	64
4.2	Submitting Jobs	64
4.2.1	Before You Start	64
4.2.2	Updating Your Path	65
4.2.3	Specifying the Netbatch Configuration Directory	65
4.2.4	Selecting a Pool	66



4.2.5	Qslot Selection	67
4.2.6	Class Selection	71
4.2.7	Submitting the Job	74
4.3	Basic Troubleshooting	76
4.3.1	Job in Wait Queue too Long	77
4.3.2	Running Job Takes Too Long	79
4.3.3	Netbatch Causes a Workstation to Run Slowly	82
4.3.4	Jobs Run Through Netbatch Crash or Hang My Machine	83
4.3.5	Where Did My Job Go?	84
5	Resource Allocation Concepts	87
5.1	Where Should I Submit My Job?	87
5.2	Resource Allocation Overview	89
5.2.1	Job Submitted to a Queue	91
5.2.2	Job Submitted to a Qslot	91
5.3	How Netbatch Decides Which Job to Run Next	92
5.3.1	Job Precedence	95
5.3.2	Resource Allocation Methods	96
5.3.3	Preemption	97
5.4	Deciding Where to Submit a Job	98
5.4.1	Using nbstatus qslots to View Resource Allocation Structure Details	99
6	Submitting Jobs to Netbatch	107
6.1	Special Considerations	108
6.1.1	Pools that Include Windows Workstations	108
6.1.2	Process Authentication Groups	108
6.1.3	Jobs that Require LD_ASSUME_KERNEL to Have a Particular Value .	108
6.1.4	What to Do if you Belong to More than 16 Groups	109
6.2	Setting the Netbatch Configuration Directory	110
6.3	Running Jobs on or from Windows Workstations	110
6.3.1	Before You Start	110
6.3.2	Submitting Jobs	120
6.4	Enabling Your Jobs to Run on Windows Workstations	122
6.4.1	Setting the Environment Variables	122
6.5	Submitting a Job to Netbatch with nbjob run	124
6.5.1	Specifying a Pool	127
6.5.2	Specifying a Queue	129
6.5.3	Specifying a Qslot	129
6.5.4	Specifying a Virtual Pool	131
6.5.5	Specifying the Workstation Class	132
6.5.6	Email Options	133
6.5.7	Job Output (Log File) Location and Options	134
6.5.8	Suspended Job Options	136



6.5.9	Specifying a License	137
6.5.10	Kerberos Authentication	137
6.5.11	Job Environment Variables	138
6.5.12	Job Execution Limits	138
6.5.13	Job Hung Limits	139
6.5.14	Specifying a Job Pre-Execution Utility	140
6.5.15	Specifying a Job Post-Execution Utility	142
6.5.16	Specifying a Job Starter	143
6.5.17	Specifying the Job Priority	144
6.5.18	Specifying a Task Name	145
6.5.19	Submitting Groups of Jobs	146
6.5.20	Specifying Resource Limits	146
6.5.21	Protecting Jobs from Blackholes	149
6.5.22	Specifying Values for Required Properties	151
6.6	Working with Job Dependencies	151
6.6.1	Using nbjob modify --triggers to Set Job Dependencies	153
6.6.2	Using nbjob run --triggers to Set Job Dependencies	155
6.7	Jobs with Special Resource Requirements	157
6.7.1	Overview	157
6.7.2	Viewing Supported Workstation Attributes	159
6.7.3	Viewing Available Probe Parameters	164
6.7.4	Submitting a Job with Resource Requirements	164
6.8	Jobs That Require Licenses	168
6.8.1	Overview	168
6.8.2	Viewing Available Licenses	169
6.8.3	Submitting Jobs that Require Licenses	172
6.9	Kerberos Authentication	177
6.9.1	Overview	177
6.9.2	Acquiring Kerberos TGT	178
6.9.3	Renewing Kerberos TGT	178
6.10	Submitting a Job with Resource Reservation	178
6.10.1	Submitting a Job with Class Reservation	180
6.11	Submitting a Parallel Job	183
6.11.1	Creating a Master Job	186
6.11.2	Submitting a Master Job	191
6.11.3	Modifying a Parallel Job	195
6.11.4	Parallel Job Log Files	196
6.12	Submitting a Job Using VBatch	196
6.13	Using Vlon to Run a Virtual Machine through Netbatch	197
6.13.1	Submitting a Job to a VM Cluster	198
6.14	Using Job Profiles	199
6.14.1	Overview	199
6.14.2	Enabling Job Profiling	199



6.15	Interactive Batch Jobs	200
6.15.1	Overview	200
6.15.2	Submitting an Interactive Batch Job with nbjob run --mode interactive . 200	
6.15.3	Scheduling Interactive Batch Jobs	203
6.15.4	Accelerating gmake with Interactive Batch Jobs	203
6.15.5	Interactive Batch Job Messages and Exit Status	206
6.15.6	Special Aspects of Interactive Jobs	206
6.15.7	Netbatch Version	207
6.16	Interactive Logon Sessions	208
6.17	Terminal Jobs	209
6.17.1	Special Aspects of Terminal Jobs	209
6.18	ClearCase Jobs	210
6.19	Submitting a Job when Workstations Might not Have Access to Required Files 211	
6.20	Using Job Constraints	211
6.21	Logging Job Resource Usage Data	212
6.22	Specifying a Retry Timeout	212
6.23	Conditionally Resubmitting a Job on Finish	213
6.24	Enabling Checkpointing	214
7	Using Netbatch Environment Variables	215
7.1	Overview	215
7.2	Setting the Netbatch Job Execution Environment	215
7.2.1	Netbatch Configuration Directory	215
7.2.2	Default Job Output Directory	216
7.2.3	Project Name	217
7.2.4	Job File Permissions	217
7.2.5	Mail Notification of Failed Jobs	218
7.3	Setting Job Licensing Parameters	219
7.3.1	License Keyfile Location	219
7.3.2	Default License Requirements	219
7.4	Setting Default Command Parameters	220
7.4.1	Default Pool Name	220
7.4.2	Default Queue	220
7.4.3	Default Qslot	220
7.4.4	Default Physical Pools (for Jobs Submitted to a Virtual Pool)	221
7.4.5	Default Class	222
7.4.6	Default Re-queue Policies	222
7.4.7	Default Job Execution Limits	224
7.4.8	Default Job Hung Limits	225
7.4.9	Default Pre-execution Program	225



7.4.10	Default Post-execution Program	225
7.4.11	Default Job Starter Program	226
7.4.12	Setting the SIGKILL (Kill) Timeout	226
7.4.13	Setting the SIGSTOP (Suspend) Timeout	227
7.4.14	Default Task Name	227
7.4.15	Enabling Wash Groups	227
7.4.16	Setting Properties for Your Jobs	228
7.5	Environment Variables for Netbatch on Windows	228
7.6	Environment Variables Set by Netbatch when a Job Is Run	229
7.6.1	__NB_JOBID	229
7.6.2	NBWD	229
7.6.3	NB_PARALLEL_JOB_HOSTS	229
7.6.4	__NB_PARALLEL_POOL_HOST	230
7.6.5	__NB_SUBMIT_PWD	230
7.7	Environment Variables Set by the Netbatch Administrator	230
7.7.1	NBFEED_CONF	230
8	Tracking Job Status	231
8.1	Overview	231
8.2	Viewing Job Information	232
8.2.1	Finding out Why a Job Is Waiting Too Long	233
8.2.2	Finding Out Why a Job Is Taking a Long Time to Complete	235
8.2.3	Viewing Jobs in the Different State Queues	235
8.3	More Job Tracking Options	237
8.3.1	Job Totals	237
8.3.2	Parallel Job Details	238
8.3.3	Viewing a Job's Resource Usage	238
9	Moving, Changing, and Cancelling Jobs	241
9.1	Overview	241
9.2	Modifying One or More Jobs	241
9.2.1	Job Options	243
9.3	Removing One or More Jobs	250
9.4	Resubmitting One or More Running Jobs	252
9.5	Job Specification	253
10	Submitting and Tracking Jobs with Netbatch Flow Manager	257
10.1	Overview	257
10.2	Starting the Flow Manager	257
11	Using Netbatch Tracker	259
12	Increasing Throughput with Virtual Pools	261



12.1	Overview	261
12.2	Why Use a Virtual Pool?	261
12.3	Structure	262
12.3.1	Virtual Queues and Virtual Qslots	262
12.3.2	Migration Rules	262
12.3.3	Resource Allocation in Virtual Pools	262
12.4	Using Virtual Pools	263
12.4.1	Submitting a Job to a Virtual Pool	263
12.5	Commands	264
Appendix A User Command Reference		265
A.1	Existing Netbatch 4.x and 5.x Commands.....	265
A.2	nbjob	266
A.3	nbstatus	369
A.4	nbauth	646
A.5	nbdepend	647
A.6	nbexec	651
A.7	nbkrb renew	660
A.8	nblicense	660
A.9	nblicstat	665
A.10	nblog	670
A.11	nbpoolstat	673
A.12	nbprofile	680
A.13	nbq682
A.14	721
A.15	nbqrm	721
A.16	nbqslot	733
A.17	nbqstat	744
A.18	nbquery	763
A.19	nbstat	763
A.20	nbvm	779
Appendix B Precedence of Configuration Directives, Environment Variables, Job Profiles, and Command-Line Switches		785
Appendix C The New Netbatch Command Suite		799
Appendix D Log File Contents and Job Exit Codes		805
D.1	Log File Name and Directory	805
D.2	Exit Status	807
Appendix E Environment Variables		813
E.1	Environment Variables Set by the User	818
E.2	Environment Variables for Netbatch on Windows	826
E.3	Environment Variables Set by Netbatch when a Job Is Run ..	828



Appendix F Smart Classes Reference	835
F.1 Syntax	835
F.2 Attributes	836
F.3 Examples	839
Appendix G Regex Replace Functions	841
Appendix H Functions	845



List of Figures

Figure 1: Basic Netbatch Pool Structure	16
Figure 2: Netbatch Tracker Schematic.	32
Figure 3: Scheduling Example	36
Figure 4: Job Life Cycle Diagram.....	42
Figure 5: A Simple Netbatch Pool P1.....	45
Figure 6: Netbatch Pool P2 with Queues.....	46
Figure 7: Simple Netbatch Pool P3 with Resource Groups and Queues.....	48
Figure 8: Virtual Netbatch Pool Structure.....	49
Figure 9: A Global Pool	51
Figure 10: Netbatch Pool Components	53
Figure 11: Netbatch Virtual Pool Components.....	53
Figure 12: Example Pool Structure	65
Figure 13: nbstatus permissions Output.....	68
Figure 14: nbstatus classes Output	72
Figure 15: nbstatus workstations Output Showing Classes for a Specific Workstation	72
Figure 16: Successful Job Submission Message.....	76
Figure 17: Expected nbstatus jobs Output.....	78
Figure 18: A Simple Resource Allocation Structure.....	89
Figure 19: Resource Sets.....	90
Figure 20: Scheduling Example	94
Figure 21: Comparing Qslot Weights (or Priorities).....	97
Figure 22: Chained Job Dependencies	152
Figure 23: Viewing Supported Class Attributes	161
Figure 24: Output of nbjob run Command with Smart Class Specification	167
Figure 25: nbstatus licenses Output.....	171
Figure 26: Running tcsh as an Interactive Batch Job.....	202
Figure 27: Piping Input and Output from an Interactive Batch Job.....	203
Figure 28: Makefile with Interactive Batch Jobs.....	204
Figure 29: Example Interactive Batch Jobs	205
Figure 30: Output of gmake Running Interactive Batch Jobs.....	205
Figure 31: Version of Netbatch Pool Master	208
Figure 32: Version of Netbatch nbjob Client.....	208
Figure 1: nbstatus classes Output	383
Figure 2: nbstatus constants Output	389
Figure 3: nbstatus custom-fields Output.....	396
Figure 4: nbstatus denial-of-service Output	403
Figure 5: nbstatus distance-map Output	409
Figure 6: nbstatus dynamic-workstations Output	416
Figure 7: nbstatus feeders Output	424
Figure 8: nbstatus function Output.....	431
Figure 9: nbstatus groups Output.....	438
Figure 10: nbstatus jobs Output	464
Figure 11: nbstatus keyfiles Output	471
Figure 12: nbstatus license-allocation Output	480
Figure 13: nbstatus license-sessions Output	489
Figure 14: nbstatus licenses Output.....	497



Figure 15: nbstatus locks Output	504
Figure 16: nbstatus logical-licenses Output	512
Figure 17: nbstatus monitor-actions Output	519
Figure 18: nbstatus permissions Output	528
Figure 19: nbstatus policies Output	535
Figure 20: nbstatus pools Output	543
Figure 21: nbstatus probes Output	551
Figure 22: nbstatus qslots Output	564
Figure 23: nbstatus remote-qslots Output	574
Figure 24: nbstatus remote-availability Output	581
Figure 25: nbstatus resource-groups Output	588
Figure 26: nbstatus resources Output	595
Figure 27: nbstatus services Output	602
Figure 28: nbstatus virtualjobs Output	609
Figure 29: nbstatus vm-image-cache Output	616
Figure 30: nbstatus vms Output	624
Figure 31: nbstatus wan Output	630
Figure 32: nbstatus workstations Output	646
Figure 33: nbqslot Output	736
Figure 34: nbqstat Output	752
Figure 35: nbqstat -f -e Output	753
Figure 36: nbqstat --verbose Output	754
Figure 37: nbstat Output with the -e Option	768



List of Tables

Table 1:	Job Completion Scenarios	40
Table 2:	Sample Command Strings	76
Table 3:	Email Options	133
Table 4:	Suspended Job Option Options	136
Table 5:	Resource Limits	147
Table 6:	Attribute Value Types	159
Table 7:	Attributes that Can Be Specified in a Class Requirements Expression	162
Table 8:	NBMAILONERR <error type> Macros	218
Table 1:	Resource Limits	305
Table 2:	Resource Limits	361
Table 3:	Resource Limits	657
Table 4:	Resource Limits	714
Table 5:	Precedence of Configuration Directives, Environment Variables, Job Profiles, and Command-Line Switches	786
Table 6:	Precedence of Configuration Directives, Environment Variables, Job Profiles, and Command-Line Switches	791
Table 7:	nbjob	799
Table 8:	nbstatus	800
Table 9:	nbadmin	801
Table 10:	nbconfig	802
Table 11:	nbservice	803
Table 1:	Log File Contents, First Section	805
Table 2:	Log File Contents (Third Section)	806
Table 3:	Exit Status Error Codes	807
Table 1:	Environment Variables	813
Table 1:	Attributes that Can Be Specified in a Class Requirements Expression	838
Table 2:	Functions and Their Descriptions	846



CONFIDENTIAL

Netbatch
v8.2.2

Revision 4



1 Introduction

1.1 Overview

Netbatch makes use of available CPU time by running batch Jobs on idle or unused machines. This lets users:

- Run more than one Job at a time.
- Access Workstations running different operating systems.
- Access higher specification workstations (e.g. with more memory).

Netbatch v7.3.1 is fully backward compatible with NetStar, Netbatch 5.0, and Netbatch 4.x and incorporates a number of new features.

Note

When Netbatch is running Jobs on a Workstation, each Job shows up as one or more Java processes, each of which is shown as taking up a certain amount of memory.

However, these processes use a shared address space, so the actual memory usage is much less than it appears to be.

1.2 Location of Netbatch v8.2.2 Executables

The Netbatch executables are a collection of scripts and binary files that implement the Netbatch command interface.

For easy access to the Netbatch executables, ask your Netbatch administrator where the Netbatch v7.3.1 executables reside on your network file system, and append that path to your PATH environment variable.

Several generations of Netbatch executables may be resident on your site. Although Netbatch v7.3.1 is backward compatible with all previously released Netbatch executables, the Netbatch team recommends that you use only Netbatch v7.3.1 commands in order to make full use of the product's capabilities.



1.3 A Note about Notation

This section explains the notational conventions used in this manual.

bold, indented, monospace text	Command syntax and examples, configuration directive and block examples
<text_in_angle_brackets>	A placeholder that must be replaced with a real value
[text_in_square_brackets]	An optional item
...	The previous item may be repeated any number of times.
{item_1 item_2}	A choice—you must select one (and only one) of the options.
small, black, monospace text	Example configuration files and sample output
small, bold, monospace text	Mandatory items in example configuration files
small, grey, monospace text	Comments in example configuration files

For Example

```
nbjob run --target <pool_name>
[--mode {interactive|interactive-ls|
blocking|terminal}]
[--class-reservation <reservation_item>[,...]]
<command>
```

This means that:

- *The --target switch must be specified, and <pool_name> must be replaced by a real Pool name.*
- *The --mode switch is optional, but if it is used, it must be followed by interactive, interactive-ls, blocking, or terminal.*
- *The --class-reservation switch is optional, but if it is used, it must be followed by at least one reservation_item. You can specify any number of reservation_items, and they must be separated by commas.*
- *A command must be specified at the end of the line in place of the <command> placeholder.*

1.4 What's New in v8.2.2

The following new features have been introduced in Netbatch v8.2.2:

- **Netbatch virtualization**—Netbatch now supports two ways of working with virtual machines:



- Vbatch lets you run Jobs that require an operating system version that is not running on any of the Workstations in the Pool (for example, Windows in a Linux Pool, or an older version version of SLES). See the following:
 - ◆ [Submitting a Job Using VBatch](#)
 - ◆ The `--image` switch in [nbjob run](#) tells Netbatch which OS version the Job requires.
 - ◆ The new [nbstatus dynamic-workstations](#) command lists the virtual machines.
 - ◆ The new [nbstatus vm-image-cache](#) command lists the available OS images.
- Vlon lets you run virtual machines through Netbatch (for interactive use). You can boot a virtual machine (using the same images as mentioned above), and it is yours to use until you no longer need it, at which time you should shut it down. You can also create clusters of VMs, which can then accept Jobs, like a mini-Pool. See the following:
 - ◆ [Using Vlon to Run a Virtual Machine through Netbatch](#)
 - ◆ [Submitting a Job to a VM Cluster](#)
 - ◆ The [nbvm](#) command, which lets you boot and shut down VMs and create and remove clusters of VMs
 - ◆ The `--cluster-id` switch in [nbjob run](#), which allows you to submit a Job to a VM cluster,
- **Renting of reserved resources (HPC Pools only)**—if a Job reserves resources, Netbatch can now "rent out" these reserved (but currently unused) resources to smaller Jobs so that they can run until all the resources that the bigger Job has reserved are available. If the smaller Job has not finished running at this time, it will be suspended.

Caution

This feature is only intended for use in HPC Pools. It is not available in design or tapeout Pools.

See the following:

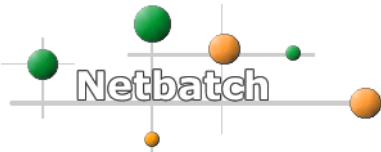
- [nbjob run](#) has new `--renting-condition` and `--expected-runtime` switches. Use these when submitting a Job that can use these unused reserved resources. (Expected run time can also be used in a `--job-constraints` expression.)
- [nbstatus jobs](#) has the following new fields: `RentingCondition`, `Lenders`, `Renters`.



- The new `canRent()` function can be used in `nbstatus jobs` to see whether a Job can borrow resources. See [Appendix H: Functions](#).
- **Other enhancements:**
 - Parallel Job enhancements—the new `--master-class` switch in `nbjob run` lets you specify the required class for the master Job. In addition, the `--parallel-slave-class` switch is now deprecated. See also the new `masterClass` field in `nbstatus jobs`.
 - For parallel Jobs that use MPI, there is a new Job starter script that sets up the MPI environment for you (which saves you having to set environment variables, start the MPI daemon, dispatch the slave Jobs, and so on). See [Section 6.11.1.1, MPI](#).
 - By default, Netbatch Jobs now run with a umask of 027, instead of 022 as they did previously. This can be changed by setting the `NBUMASK` environment variable.
 - In `nbstatus resources`, the `On Since` and `Last Select` fields are now called `OnSince` and `LastSelect`.
 - New switches in `nbjob run` let you specify different directories for the log files of the pre- and/or post-execution scripts. See `--log-file-dir-pre-exec` and `--log-file-dir-post-exec` in `nbjob run`.
 - All sub-commands, switches, and field names are now listed in alphabetical order in the help (`--help` or `-h`) output.
 - Wash groups now support LDAP.
 - When a Job runs, Netbatch now automatically creates a directory with 700 permissions for you (to do whatever you need to with). By default, this directory is located in `/netbatch/` (a symlink to `/tmp/netbatch`), and is named `<username>_<jobid>`.

The location of this directory is available to the Job in the `__NB_SANDBOX_DIR` environment variable.

- Netbatch now supports checkpointing (using tools such as DMTCP). See:
 - **nbjob run:** the `--checkpoint-*` and `--num-of-checkpoints` switches
 - **nbjob checkpoint:** creates a checkpoint manually for one or more Jobs
 - **nbjob recall:** the `--restore-from-checkpoint` and `--restore-from-last-checkpoint` switches allow you to rerun a Job from a particular checkpoint
 - **nbjob run --on-job-finish:** resubmit the Job when it ends to run it again from the last checkpoint



- ◆ **nbstatus jobs:** the `Checkpoint*` fields
- ◆ There is a new exit status (-321) for Jobs where the checkpoint failed (see [Appendix D](#):



CONFIDENTIAL

Netbatch
v8.2.2

Revision 4

Introduction



2 Basic Netbatch Concepts

2.1 Jobs

2.1.1 Job

A **Job** can be any executable script or tool, including ones that:

- Require user interaction
- Run on multiple machines simultaneously

From Netbatch's point of view, a Job is the script/tool, plus a set of attributes that define where, how, and when it will be dispatched for execution.

2.1.2 Batch Job

A **batch Job** is a Job that requires no user interaction while it is running.

2.1.3 Express Job

Express Jobs are short Jobs that need immediate and fast execution. They are typically Jobs with low resource, low memory, and low disk usage requirements.

2.1.4 Interactive Batch Job

An **Interactive Batch Job** is a Job that is dispatched for execution through Netbatch in a special way. When a Job is dispatched for interactive execution, the standard input, output and error streams are redirected between the process from which the Job is submitted, and the process which executes the Job (usually on a remote machine). After an interactive Job completes execution, the submitting process will exit and assume the exit status of the completed Job.

Interactive Batch Jobs are especially suited for batch execution of complex sequences of interdependent Jobs as defined in a Makefile.

Interactive Batch Jobs should not be confused with interactive users. An interactive user is a user that is logged on to workstation and has an active terminal process.

For more details, see [Section 6.15, Interactive Batch Jobs](#).



2.1.5 Interactive Logon Session

An **interactive logon session** is where a user logs on to a Workstation through Netbatch. The user must submit this type of Job to a special **Logon Queue**.

See [Section 6.16, Interactive Logon Sessions](#).

2.1.6 Terminal Jobs

A **Terminal Job** is when an application that requires interactive user input and manipulates user input and screen display through the terminal is run through Netbatch. Examples include vi, less, emacs, and tcsh.

For more details, see [Section 6.17, Terminal Jobs](#).

2.1.7 Parallel Job

A **Parallel Job** is a Job that requires more than one **Execution Slot** (that is, a processor or a host), OR a Job that consists of several Jobs, each of which requires an execution slot.

The actual Job that is submitted to Netbatch is known as a **Master Job**. This Master Job submits the **Slave Jobs** that make up the Parallel Job.

A Parallel Job usually spends more time in the Wait State Queue than other Jobs, while it waits for Netbatch to accumulate the execution slots that it needs.

For a more detailed explanation of Parallel Jobs, see [Section 6.11, Submitting a Parallel Job](#).

2.1.8 Job Pre-Execution

Before a Job is run, Netbatch can perform a **Pre-Execution** phase. During Pre-Execution, a utility program is run. The purpose of this utility is to verify the existence of certain preliminary conditions that are required for the Job's execution. If the Pre-Execution utility indicates that it has failed, the Job is sent back to the Wait Queue. This prevents the Job from starting to execute on a machine where it would fail.

For more information, see [Section 6.5.14, Specifying a Job Pre-Execution Utility](#).

2.1.9 Job Post-Execution

After a Job runs, Netbatch can perform a **Post-Execution** phase. During Post-Execution a utility program is run. Typically the Post-Execution utility will perform cleanup tasks.



A Post-Execution phase will also take place if Pre-Execution fails, and the Job is not run.

For more information, see [Section 6.5.15, Specifying a Job Post-Execution Utility](#).

2.1.10 Job Starter

A **Job Starter** is a wrapper utility that launches the Job. Job Starters allows easy integration of third party tools with Jobs that are run through Netbatch. Typically a Job Starter will set environment variables that are required for the correct execution of the Job.

For more information, see [Section 6.5.16, Specifying a Job Starter](#).

2.1.11 Job Profile

A **Job Profile** is a set of default Netbatch options that are assumed when specific Jobs are submitted to Netbatch. Different Job Profiles are attributed to the different Jobs according to the names of the applications that are submitted to Netbatch.

For more information, see [Section 6.14, Using Job Profiles](#).

2.1.12 Job Dependencies

Job Dependencies define a definite order of execution between Jobs. Execution dependencies can be specified at or after Job submission time so that Jobs are dispatched for execution in the specified order. Each subsequent **Dependent Job** in the execution order is dispatched after the preceding **Trigger Job** has completed. A dependent Job may have several triggering Jobs.

A Dependent Job will be submitted for execution only upon successful completion of its Trigger Job(s). A Job with a positive exit status is considered successful. An exit status above 128 is considered negative.

If a Trigger Job fails, or is removed from Netbatch, prior to successful completion, its Dependent Jobs are suspended.

In order to dispatch such sequences of interdependent Jobs we recommend constructing a Makefile with interactive batch Jobs, and then submitting the sequence to Netbatch using `gmake`.

For more details, see [Section 6.6, Working with Job Dependencies](#).



2.1.13 Job Priority

A user can assign priorities to Jobs that are submitted to Netbatch.

Two identical Jobs that are submitted by the same user to the same **Queue** and **Qslot**, will be scheduled for execution according to the following rules, which are considered in the following order:

- Jobs with higher assigned priority will be scheduled before Jobs with lower assigned priority.
- Jobs will be scheduled for execution according to the order in which they were submitted to the pool (FIFO).

For more information, see [Section 6.5.17, Specifying the Job Priority](#).

2.1.14 Job Cost

The priority of a Job within a Qslot can be calculated dynamically based on a cost formula (when the **Cost Fair-share Resource Allocation** scheduling method is used).

Cost can be influenced by the time the Job is waiting for execution, by its resource requirements, and by the machine that it ends up running on (according to the machine's configured cost).

Two Jobs that are submitted by the same user to the same Queue and Qslot, will be scheduled for execution according to the following rules, which are considered in the following order:

- Jobs with higher assigned priority are dispatched before Jobs with lower assigned priority.
- Jobs with a higher calculated dynamic priority are dispatched before Jobs with a lower calculated dynamic priority.
- Jobs will be scheduled for execution according to the order in which they were submitted to the Pool (FIFO).

For more information, see [Section 5.3.1, Job Precedence](#).

2.1.15 State Queues

Every Job submitted to a Pool has a specific Netbatch state associated with it. The Netbatch state indicates whether the Job is waiting for execution, is running, or has already run. The Netbatch state can also indicate other special conditions of a Job.



Conceptually each Queue in the Pool can be seen as being subdivided into six **State Queues**, each containing all of the Jobs in the Queue grouped according to their Netbatch state.

The six State Queues are:

- **Run State Queue**—contains all the Jobs that are running, or that have started running and have been suspended.
- **Wait State Queue**—contains all the Jobs that are waiting to be run.
- **Moved State Queue**—contains all the Jobs that have been moved to a different Pool, and which have not yet completed.
- **Notify State Queue**—contains all the completed Jobs that were originally moved from another Pool or Queue. These Jobs must notify the original Pool (where they were submitted) that they have completed. After the original Pool is notified, these Jobs move to the History State Queue.
- **History State Queue**—contains all the Jobs that have recently exited the Netbatch system.
- **Dirty State Queue**—no longer used.

The State Queue that a Job is located in gives an indication of where the Job is in its life cycle.

For more information, see [Section 3.3.3, States of a Job in Netbatch](#).

2.2 Resources

2.2.1 Workstation/Compute Server

Workstations and **Compute Servers** are resources used to run Jobs. The only difference between the two is their computing power and usage policy (determined by the Netbatch system administrator). Typically Workstations are configured to give interactive users priority over Netbatch Jobs for optimal resource utilization.

Note

In this document both Workstations and Compute Servers are referred to as Workstations.



2.2.2 Idle Workstation

An **Idle Workstation** is a Workstation with CPU load, memory usage, and number of interactive users that are below the thresholds set by the administrator. Only Idle Workstations can accept Jobs.

2.2.3 Workstation Auto-tuning

The **WSM auto-tuning** feature uses machine learning to establish the optimum number of Jobs that a Workstation should be running, given its current state (load, free real and virtual memory, etc.).

2.2.4 Pool

The basic Netbatch unit is the **Pool**. A Pool is a set of resources that will be used to run the Jobs. Each Pool is controlled by one **Pool Master**, which maintains a number of Job Queues. The resources in a Pool can be accessed and managed as a single super set (the default Resource Set), or as a collection of subsets called **Resource Groups**.

2.2.5 Resource Set

A **Resource Set** is a collection of resources in a Pool that have been grouped together by a Netbatch administrator according to a set of special characteristics, for example:

- Operating system
- Available memory
- Project ownership

Note

Multiple Resource Sets have been deprecated. Each Pool has a default Resource Set that includes all the Workstations in the Pool. To define sub-groups of Workstations, use Resource Groups. See [Section 2.2.6, Resource Group](#), below.

2.2.6 Resource Group

A **Resource Group** is a collection of resources (Workstations) that have been grouped together. Resource Groups do not assign any attributes to their members.

They are only used for scheduling (that is, Jobs sent to a Queue or Qslot will be dispatched to Workstations that belong to the Resource Group that belongs to the Queue/Qslot).

2.2.7 Physical Pool

The term **Physical Pool** is used to differentiate Pools that are collections of resources and queues—Physical Pools, from Pools that are collections of other Physical Pools—Virtual Pools.

Note

*In this document the term **Pool**, unless specified otherwise, refers to a Physical Pool.*

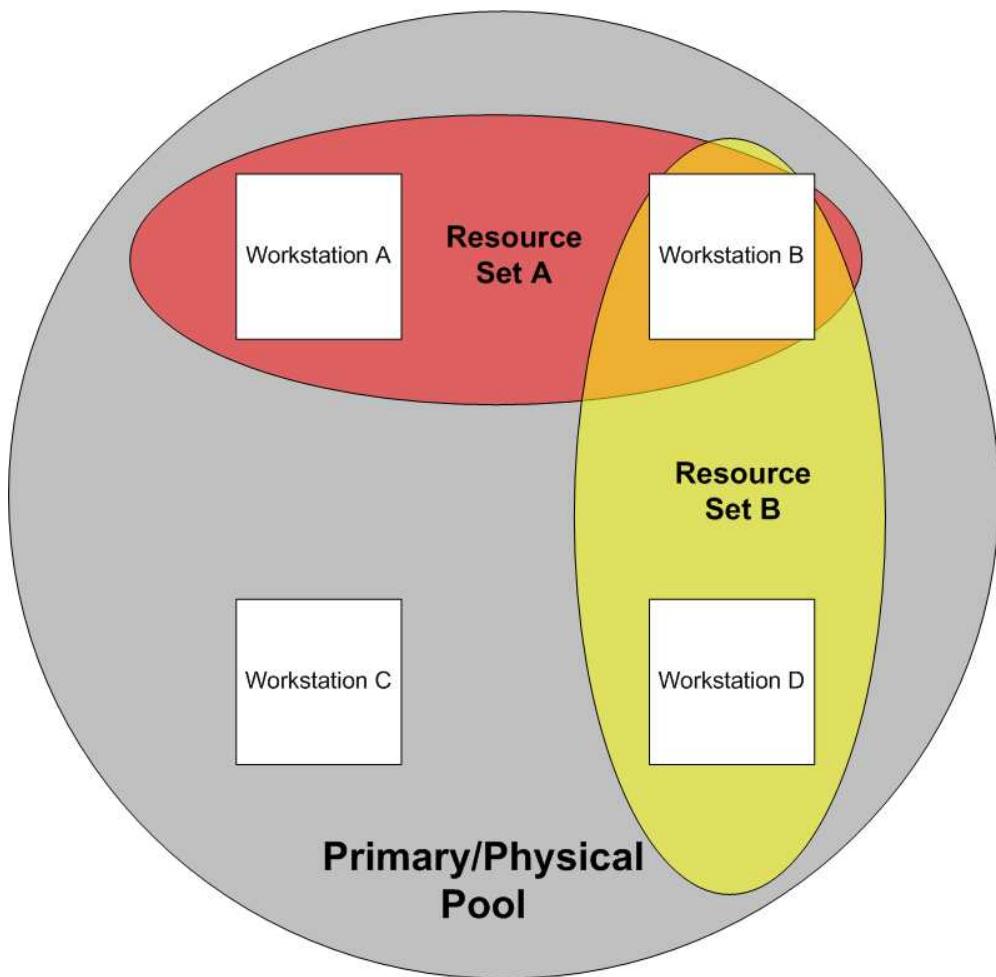


Figure 1: Basic Netbatch Pool Structure



2.2.8 Pool Master

The **Pool Master** is the machine that runs the services that control the flow of Jobs in the Pool. The Pool Master's scheduler service matches Jobs that are waiting in the Queues with idle Workstations in the different Resource Groups so that Jobs run:

- On the type of resources that they require
- According to configured resource allocation limits
- Ordered in accordance with the Queues' Resource Allocation methods

The machine on which the Pool Master runs can also serve as a Workstation in the Pool and accept Jobs.

2.2.9 Express Pool

In older releases of Netbatch (4.x and 5.x), an **Express Pool** was a Pool that was installed/configured on top of another Pool. It was intended to provide fast scheduling and fast turnaround for Express Jobs (that is, short Jobs with low resource requirements).

In Netbatch v8.2.2, instead of an Express Pool, a Pool can be configured to handle Express Jobs in one of two ways:

- By designating an Express Queue
- By using an Express Pool emulation that simulates the old implementation

2.2.10 Virtual Pool

A **Virtual Pool** is a Pool of **virtual resources**.

Each virtual resource is a Queue in a physical Netbatch Pool.

The Virtual Pool has Queues of its own that accept Jobs like a regular Pool and schedules them for execution on the best available virtual resource.

The Virtual Pool is constantly updated with regard to the status of the Jobs on its virtual resources. To Netbatch users it appears as though their Jobs were scheduled and dispatched on Queues that belonged to the Virtual Pool itself.

For more details, see [Section 12, Increasing Throughput with Virtual Pools](#).



2.3 Matching Jobs to Machines

2.3.1 Class

A Pool may contain different types of Workstation machines. Each machine may be designated as belonging to one or more **Classes**.

A Class is a name that represents Workstation characteristics such as platform, operating system type, memory size, or CPU power. A Class may apply to more than one Workstation and a Workstation may be tagged with more than one Class.

Class "Linux_4G", for example, can represent a Linux machine that has 4GB of RAM.

When a Job is submitted to Netbatch, any number of Classes may be specified, thus ensuring that the Job runs only on Workstations that have all of the desired characteristics.

Note

To find out which classes are supported by a Pool, use the [nbstatus classes](#) command.

2.3.2 Smart Classes

Similar to a Class, a **Smart Class** relates to Workstation characteristics but provides a broader and more dynamic representation of its attributes. A Smart Class is a boolean expression which defines the desired relation between the attributes of a Workstation—such as load, number of CPUs or supported classes—and the values of the attributes that are required by the Job. This mechanism is used to match a Job to one or more Workstations that best fit the Job's requirements.

As well as attributes, you can also specify **external probe** parameter names. The Netbatch Administrator can set up an external probe to measure parameters that are not measured and available as Workstation (or other) attributes. (See [Section 2.3.5, External Probe](#), below.)

These boolean expressions are composed of conditions and operators. In fact, Netbatch automatically converts simple Class specifications (<class>) into a simple Smart Class expression of the following form:

<class> is true

For Example

The expression:



```
"OSName is 'Linux' && (load <1.5 | CPUCount > 1)"
```

will restrict the Job assignment to Linux workstations, and that have either a load factor that is lower than 1.5 or more than 1 CPU.

Another example. The expression:

```
"Linux_4G && (fVM > 2000)"
```

requires that the Job execute on a workstation that supports the class "Linux_4G" and has at least 2GB of free virtual memory.

For a more detailed explanation, see [Section 6.7, Jobs with Special Resource Requirements](#).

2.3.3 Resource Reservation

The resources used by a Job when it starts running (for example, memory or disk space) are often less than those that it needs at later stages of execution. This can lead to a situation where two (or more) Jobs are dispatched to a Workstation and successfully begin execution, but hang because their increasing resource requirements exceed the capabilities of the machine.

Netbatch's **Resource Reservation** feature prevents such problems. With Resource Reservation, the user can specify the resources that a Job requires when it is submitted. Netbatch uses this information to ensure that the Job is assigned the necessary resources for the required duration.

Netbatch allows the following resource types to be reserved:

- Memory (real and virtual)
- Dedicated hosts

If a Job needs one or more licenses, the required licenses are automatically reserved.

A Job that reserves resources usually spends more time in the Wait State Queue than other Jobs, while it waits for Netbatch to accumulate the resources that it needs.

For more details, see [Section 6.10, Submitting a Job with Resource Reservation](#).



2.3.4 Workstation Capabilities

The Workstations in a Pool can have custom capabilities or attributes defined for them. Queues (see below) can have requirements associated with them for these Workstation capabilities. Users can specify these capabilities as part of a class reservation requirement when submitting a Job.

This allows the Netbatch Administrator or Queue manager to allocate a larger share of these capabilities to a specific Queue, as compared to other Queues.

This means that when a Job from this Queue is running on an individual Workstation, Jobs from the other Queues that can dispatch Jobs to it may be partly or completely excluded.

See [Section 6.10.1, Submitting a Job with Class Reservation](#).

2.3.5 External Probe

The Netbatch Administrator can set up an External Probe to measure parameters that are not measured and available as Workstation (or other) attributes. External Probes fall into three categories:

- Workstation probes monitor parameters for one or more Workstations.
- Scheduler probes monitor parameters at the scheduler (Pool) level.
- Job probes monitor parameters for Jobs

Having this additional information can lead to:

- Better matching between Jobs and virtual resources
- System flexibility and better usage of virtual resources
- Lower administrative overhead, as users can be easily prevented from using overloaded resources

You can use the information gathered by an External Probe when submitting Jobs in a class expression (using `nbjob run`'s `--class` switch).

2.4 Resource Allocation

2.4.1 Overview

The term **Resource Allocation** refers to the process of determining which Netbatch Jobs get to run on which of a Pool's Workstations and when.



Resource allocation in Netbatch is configured through a hierarchical structure of Queues, Qslots, and User Slots, referred to as a **Resource Allocation Structure**.

The Resource Allocation Structure determines resource allocation—that is, which of the Jobs submitted to a Pool is most eligible for execution on the Pool's resources at any given moment.

It is up to the Netbatch Administrator and/or Queue manager to design and configure the Resource Allocation Structure in a way that:

- Suits the requirements of the organization that will be using Netbatch
- Best utilizes the resources available to the Netbatch Pool

For detailed instructions on configuring Resource Allocation, see .

2.4.2 Queue

A **Queue** is a holding place for Jobs submitted to the Pool Master.

Jobs are submitted to a queue via an `nbjob run` command, and are lined up in the Queue in a defined order, as they wait to be executed on the Pool's resources.

Different Queues are typically configured to handle Jobs with specific workflows. Queues can define the default requirements for the Jobs submitted to them, thus ensuring their correct execution.

Queues may implement different scheduling methods, such as "priority" or "fairshare" scheduling, which determine resource utilization.

A Queue can be served by one or more Resource Groups within the Pool.

The precedence of different Queues in the same Pool is determined based on:

- The Queues' configured Resource Allocation method
- The Queues' configured parameters (weight, priority, etc.)

For a detailed explanation about Queues and how Jobs are processed in the Pool, see [Section 5, Resource Allocation Concepts](#).

2.4.3 Default Queue

The **Default Queue** is a Queue that is always created by Netbatch regardless of the configuration, and is assigned the name of the Pool.



When a Job is submitted to a Pool without a Queue specification, it is submitted to the Default Queue.

2.4.4 Logon Queue

A **logon Queue** is one that has been set up for the purpose of accepting [Interactive Logon Sessions](#).

Interactive logon sessions (Jobs) submitted to such a Queue are not affected by defined Netbatch interactive user limits. This means that Netbatch will not kill, suspend, renice, or resubmit them if the interactive user limit is reached.

2.4.5 Express Queue

An **Express Queue** is one that has been configured so that Jobs submitted to it are dispatched and executed more quickly than Jobs submitted to other Queues. When a Job from an Express Queue is dispatched to a Workstation, any running Jobs will be preempted (i.e., suspended, resubmitted, or killed), and the Workstation will not accept any other new Jobs.

2.4.6 Qslot

Qslots are the building blocks of the **Resource Allocation Structure** that is maintained by the Queue.

The Qslots form a flat or hierarchical structure that is based on the project/business organizational structure and support the resource allocation decisions made by the resource allocation managers of the Netbatch customers.

Each Qslot has a specific **priority** or **weight** assigned to it according to the resource allocator's decision. These parameters are used to calculate the order in which the Qslots are scheduled to utilize the available resources in the pool.

Access to Qslots may be restricted to specific users and groups of users. Only users that have access to a Qslot may submit Jobs to it.

Qslots may also have special **Submission Profiles** that affect Jobs that are submitted to them, as well as **Resource Allocation Limits** that affect the Qslots' maximum possible resource utilization.

Qslots may be configured to support scheduling methods that override the scheduling method of the Queues to which they belong.

For a detailed explanation about Qslots, default Qslots, and the Resource Allocation Structure, see [Section 5, Resource Allocation Concepts](#).



2.4.7 Default Qslot

Each Qslot in the resource allocation hierarchy has two roles:

- As a holding place for Jobs that are submitted to it
- As a root for its sub-Qslots

Jobs that are queued in a Qslot compete for the same resources as the Jobs that are queued in that Qslot's sub-Qslots.

To distinguish between the different roles, each Qslot is automatically created with a default sub-Qslot during system initialization. Jobs submitted to the Qslot will automatically be placed in the default Qslot. The default Qslot may be configured like a regular Qslot.

When a limit (e.g., max running limit) is defined for a Qslot, it is applied to the entire sub-tree of Qslots. When a limit is defined for the default Qslot, it applies only to the Jobs that are submitted to the Qslot itself.

Note

In nbstatus qslots output and in the configuration files, the default Qslots are identified by appending the keyword OTHERS to the Qslot name: <Qslot>/OTHERS.

2.4.8 Qslot Alias

The hierarchical structure of Qslots maintained by Netbatch 5.0, 6, and 7 requires path names for Qslots. These path names can be long, and using the full path in the command line can be cumbersome. To enable easier access with Netbatch commands to hierarchical Qslots, the Netbatch administrator or Queue manager can assign **aliases** (i.e., shorter, more convenient names) to Qslots.

Hierarchical Qslots may be referred to either by their hierarchical name, or by their alias. If a Qslot is referred to by its full hierarchical name in a Netbatch command, the output of the command will also refer to this name. Likewise if a Qslot is referred to by its alias in a Netbatch command, the output of the command will also refer to the alias.

2.4.9 What is the Difference between a Queue and a Qslot?

A Queue and a Qslot share many similar characteristics, but they are not the same.



The main difference is that a Queue is a namespace that applies to all the Jobs in its internal Qslots, whereas individual Qslots do not have a unique namespace. Hence, all the Jobs that are waiting in Qslots within the Queue are also regarded as queued in the Queue.

The key reason for using multiple Queues is the usage model. When it is necessary to separate sets of Jobs according to flows and Job characteristics, it is wise to create a separate Queue for each such set.

On the other hand, when the main difference between two sets of Jobs is the resources that should be allocated to them, it is a good idea to break a Queue into Qslots.

For Example

A Pool will be split into Queues when there is one flow for Interactive Jobs and another for regression jobs.

A Queue will be split up into Qslots when resources must be allocated to two different projects.

2.4.10 Preemption

Netbatch provides two different types of preemption:

- Preemption between Queues and Qslots (recommended method)
- Preemption between Pools (for Workstations that belong to multiple Pools)

In Queue/Qslot preemption, when no resources are available to process a Job that is waiting in a "preemptive" Queue or Qslot, then this Job can preempt a running Job from a "preemptable" Queue or Qslot. (There are a few other conditions that must also be satisfied, but this is basically what happens.)

Note

The preemption values of the Job are inherited from the Queue preemption values.

The preempted Job can be **killed**, **resubmitted**, or **suspended** according to the configuration of the Queues and Qslots.

**Note**

If the Netbatch Administrator has configured custom-defined Workstation capabilities and Queue weighting, a Job from a preemptive Queue may require multiple Jobs to be preempted (that is, killed, suspended, or resubmitted).

2.4.11 Resource Allocation Methods

The Netbatch Administrator can define methods that determine resource allocation.

The resource allocation methods can be applied to:

- A Queue
- A Qslot

Netbatch supports three general types of resource allocation, which are explained in the following subsections.

See [Section 5, Resource Allocation Concepts](#) for a more detailed explanation of resource allocation.

2.4.11.1 Priority Resource Allocation

In a **prioritized** scheme, all Jobs submitted to Queues and Qslots that have a high priority will be executed before Jobs submitted to Queues and Qslots with a lower priority.

An exception to this rule is when a high-priority Queue or Qslot only contains Jobs that cannot run (for lack of appropriate resources for instance). In this case Jobs from lower-priority Queues and Qslots will be allowed to run before the Jobs in the high-priority Queues and Qslots.

Queues and Qslots are assigned a **priority value**. A higher-priority value means that the Queue or Qslot has a higher precedence. If two Queues or Qslots have an identical priority, Netbatch will dispatch their Jobs for execution on a first-come first-served basis.

2.4.11.2 Fair-Share Resource Allocation

In a **fair-share** scheme, Netbatch will try to ensure that at any given moment, the Pool's resources are allocated to the different Queues and Qslots in proportion to their relative assigned **weight value**.



For example, if Queue A and Queue B are the only contenders for a Pool's resources, and Queue A is assigned a weight value of 40 and Queue B is assigned a weight value of 60, then Netbatch will strive to allocate 40% of the Pool's resources to Jobs from Queue A and 60% to Jobs from Queue B.

2.4.11.3 FIFO Resource Allocation

In a **FIFO** scheme, Jobs in the sub-tree of the Queue or Qslot where FIFO is specified as the scheduling method are dispatched for execution on a first-come first-serve basis.

2.5 Permissions

For the purposes of permissions, Netbatch is considered to be made up of a number of entities, each of which has a number of operations that can be performed on it. For each such operation, permissions can be granted or denied to one or more users or groups (Netbatch groups or NIS groups).

The Netbatch entities are:

- The Pool
- Queues and Qslots
- Jobs

The operations for which permissions can be granted or denied are as follows:

- Submit Jobs to a Queue/Qslot.
- Modify Jobs that other users have submitted to a Queue/Qslot¹.
- Remove Jobs that other users have submitted to a Queue/Qslot¹.
- Open a Queue/Qslot.
- Close a Queue/Qslot.
- Make a Queue/Qslot active.
- Make a Queue/Qslot inactive.

Permissions for logical groups of operations can also be granted/denied:

- Job administration—all of the above Job operations
- Queue/Qslot administration—all of the above Queue/Qslot operations

¹⁾Users can modify and remove their own Jobs. No special permissions need to be granted.



- Pool administrative operations—all of the above Job and Queue/Qslot operations

2.6 License Support

Some Jobs use other software tools while they are running. A license is a software vendor's permission to use a pre-determined number of instances of a software tool.

Netbatch supports license-based scheduling. It is important to ensure that the total number of Jobs using a tool does not exceed the number of licenses that Intel owns for the tool.

Netbatch tracks the number of licenses available for each tool. Netbatch does not dispatch a Job for execution if the licenses it requires are not available.

2.7 Netbatch Feeder

Netbatch Feeder is a utility that allows users to feed a large number of Jobs to Netbatch automatically, and then track them easily. The user needs to create text files that contain parameters relating to Job submission via the feeder, and a list of Jobs (a **task**) to be fed to Netbatch. The Jobs are submitted to the specified Pool in a regulated way ("trickle mode") based on the Pool's availability. The submitted Jobs are constantly monitored and enable the user to track the progress of Jobs until completion.

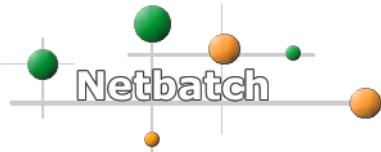
2.7.1 Netbatch Flow Manager

The Netbatch Flow Manager is a GUI application that allows users to:

- Start and stop Feeders
- Create, edit, and submit tasks to a Feeder
- Create task dependencies
- Monitor and manipulate Feeders, tasks, and Jobs

To run the Flow Manger, run the `nbfflow` command. For detailed usage instructions, select **Help | Contents** in the Flow Manager.

See [Section 10, Submitting and Tracking Jobs with Netbatch Feeder](#).



2.8 Netbatch Tracker

Netbatch Tracker collects and displays data about Netbatch Jobs. Pool Masters and Virtual Pool Masters report Job data to Netbatch Tracker when each Job ends. Netbatch Tracker records the activity information, which may be queried later for analysis and troubleshooting. Netbatch Tracker data is stored in a relational database located on a machine designated as the database server.

The following figure shows a schematic view of how Netbatch Tracker gathers and stores information from the Netbatch system.

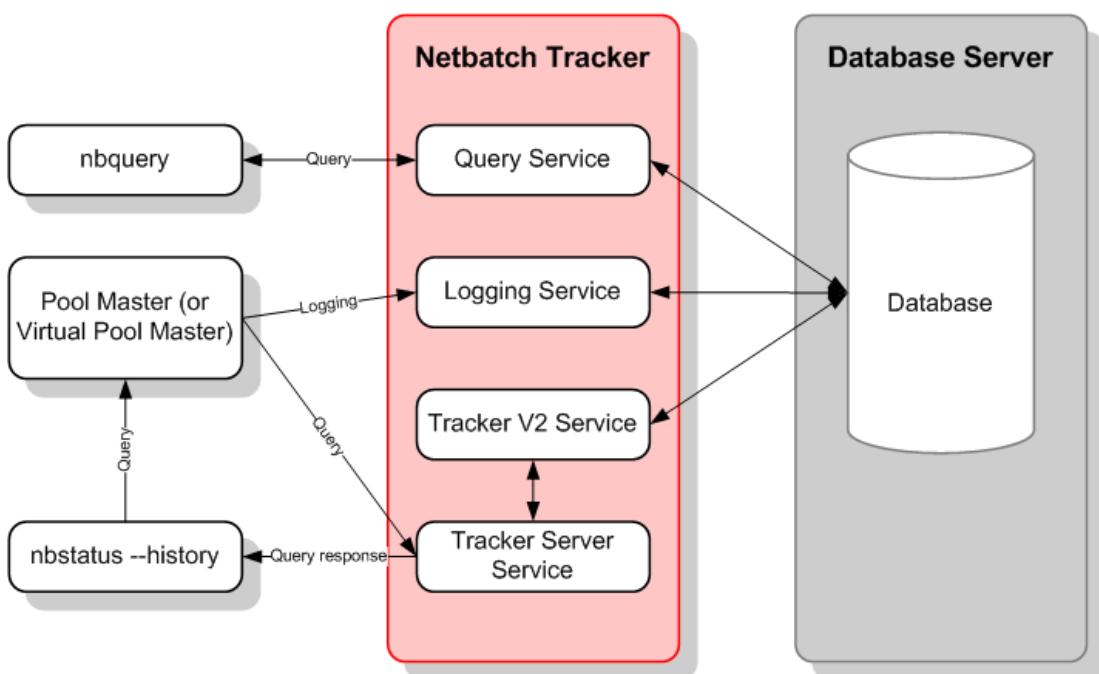


Figure 2: Netbatch Tracker Schematic

See [Section 11, Using Netbatch Tracker](#).



CONFIDENTIAL

Netbatch
v8.2.2

Revision 4

Basic Netbatch Concepts



3 How Netbatch Works

3.1 Overview

You usually have a choice of Pools that you can send Jobs to. Each Pool uses a system of **Queues** and **Qslots** to determine which Job is run when, and on which Workstation.

You can submit Jobs to any Queues and Qslots that are allocated to your group.

You can also choose the **Class** of Workstation that your Job needs. The Class describes:

- The architecture and/or hardware specification of a Workstation
- Dynamic information relating to the current status of a Workstation

If you understand how this works, you can choose the best Pool, Queue, Qslot and Class available to you so that your Job runs as quickly as possible.

The Netbatch Pool Master's scheduler service decides which Job to process next based on the following factors:

- Workstation availability
- Queue Priority/Weight
- Qslot Priority/Weight
- Matching of the requested Class requirements to available Workstations
- Queues' capability requirements and defined Workstation capabilities

The following sections explain this in detail.

When your Job ends, Netbatch sends the output file to the directory where you submitted the Job (unless you specify otherwise).

The output file contains log information as well as the Job's output.

3.2 How Netbatch Decides Which Job to Run Next

Netbatch selects the next Job to be executed as follows:

1. Find the most eligible Queue.



Netbatch decides which Queue is most eligible, according to the rules defined in the resource allocation structure (see [Section 5.3.2, Resource Allocation Methods](#), below).

2. Find the most eligible Qslot or User Slot.

Netbatch examines the hierarchy of Qslots and User Slots in the selected Queue, and decides which "leaf" Qslot or User Slot is most eligible according to the rules defined in the resource allocation structure (see [Section 5.3.2, Resource Allocation Methods](#), below).

3. Find the most eligible Job.

Netbatch decides which Job in the selected Qslot/User Slot is most eligible according to the rules of Job Precedence (see [Section 5.3.1, Job Precedence](#), below).

If this Job cannot run because no suitable Workstation is available **and** it is a preemptive Job (and there is a running Job that it can preempt), Netbatch preempts the running Job so that this Job can run.

This marks the end of this scheduler cycle.

4. For the most preferred Resource Set/Group of the most eligible Qslot (or of the User Slot's parent Qslot if this is a User Slot), find the most available Workstation. The most available Workstation is with one with the lowest load (that is, the one with the fewest processes competing for its CPU time at a given moment) or WSRank (see [Section 11.2.27, Configuring Scheduling Based on Availability of Multiple Resources](#)).
5. If the most available Workstation can accept a Job, check if it can run the most eligible Job:
 - a) Check if the Workstation can ever run the Job. (That is, check whether the Workstations matches the Job's static class requirements and whether the Job's dynamic requirements are within the Workstation's limits. For example, if the Job requires 3GB of free virtual memory, check that the Workstation has at least 3GB *total* virtual memory.)
 - b) Perform resource reservation (if any). (Even if the Job cannot run on the Workstation at the moment, it may be able to reserve resources on it.) See [Section 2.3.3, Resource Reservation](#) for more information.

Note

*Starting with Netbatch 8.0.0, Jobs no longer hold reservations on a particular Workstation for more than one scheduler cycle. In the new **Stateless Reservation** scheme, at the end of each cycle, all reservations are released and reserved anew in the next cycle.*



This gives new, more important Jobs access to the resources that they need and also stops Jobs that reserve resources from waiting longer than they have to.

- c) Check if the Job can run on the Workstation right now. (That is, check against the Job's dynamic class requirements.)

If the Workstation cannot satisfy the Job's class requirements, Netbatch tries the next most available Workstation, then the next, and so on until it finds one that does. If none of the available Workstations can satisfy the Job's class requirements, it moves on to the next most eligible Job.

- d) Make sure the Job has any licenses that it needs.

If all of these conditions are met, dispatch the Job we found in Step 3 to the Workstation.

If the most available Workstation in the Qslot's most preferred Resource Set/Group cannot accept any Jobs, find the most available in *the Qslot's next preferred Resource Set/Group*, and so on.

If no Workstation from any of the Resource Sets/Groups that the Qslot can use can accept the Job, find the most eligible Job in the *next most eligible Qslot in the Queue*, and so on.

If none of the Workstations from any of the Resource Sets/Groups that the Qslots in the Queue can use can accept Jobs, find the most eligible Job in the most eligible Qslot in the *next most eligible Queue*, and so on.

For Example

In Figure 3, below, if there are Jobs waiting in both QS1 and QS2, and QS1 is more eligible, then Netbatch will first look for the most available Workstation in Resource Group RG1 to run the most eligible Job in QS1.

If the most available Workstation in RG1 cannot accept Jobs, it then looks in RG2. If the most available Workstation in RG2 cannot accept Jobs, it looks for the most available Workstation in the Pool to run the most eligible Job in QS2.

If the most available Workstation in the Pool cannot accept Jobs, Netbatch starts the process all over again.

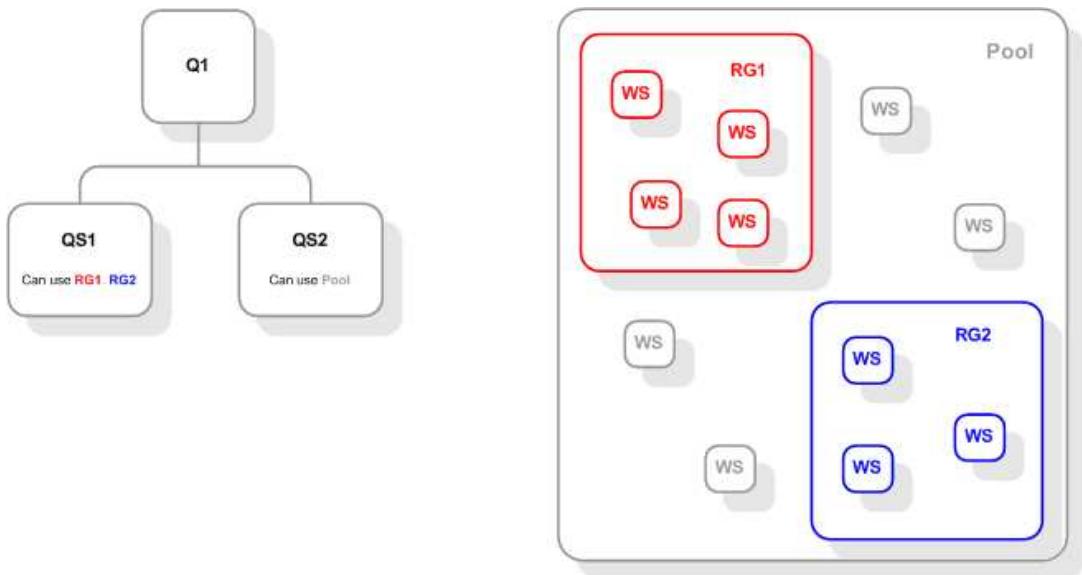


Figure 3: Scheduling Example

Note

Netbatch decides which resource (Workstation) is most available by comparing Workstation loads (that is, the number of processes competing for CPU time).

However, this can give a misleading picture, as different Workstations have different capabilities (i.e., processor speed, number of processors), so the load on its own is not a good basis for comparison.

Netbatch's Workstation Load Normalization feature (which must be configured by the administrator) takes these different capabilities into account, producing normalized load data that are more meaningful.

(The Netbatch Administrator can also configure Netbatch to take the availability of other resources into account when deciding which Workstation is most available.)

Note

If the most eligible Job was submitted to a Queue that has Workstation capability requirements defined for it, then in certain situations, the most available Workstation cannot accept the most eligible Job; instead it will accept a less eligible Job.



For example, if the Queue that the less eligible Job was submitted to has a memory requirement of 1, and the Queue that the most eligible Job was submitted to has a memory requirement of 4, but the Workstation only has 2 units of memory available, then the Workstation cannot accept the most eligible Job, so it will accept the less eligible Job.

3.2.1 How Netbatch Checks Class Requirements

There are two kinds of numeric parameter/attribute that can be used in a Job's class requirements:

- Ones whose values change¹
- Ones whose values do not change²

When a user submits a Job with a class requirement, Netbatch makes two checks:

1. It checks whether the requirement can ever be met:
 - If the parameter/attribute is one whose value **does not** change, Netbatch compares the requirement with the value. Netbatch only queues the Job if the **value** meets the requirement.
 - If the parameter/attribute is one whose value **does** change, Netbatch compares the requirement with the **limit**³ for the parameter/attribute. Netbatch only queues the Job if the **limit** meets the requirement.

Note

When there are remote Workstations to which Netbatch can send the Job, and none of the local Workstations meet the requirement, Netbatch accepts the Job while it queries the remote Workstations. If one (or more) of the remote Workstations meets the requirement, the Job will eventually be dispatched for execution.

If no remote Workstation meets the requirement, the Job will remain in the wait state queue until it is removed—it can never be dispatched for execution.

Once Netbatch has performed the query, it caches the answer. Subsequent Jobs with the same requirement will either be accepted (and later dispatched) or rejected immediately.

¹⁾ Examples include load, free disk space, and free real memory.

²⁾ Examples include number of processors, total virtual memory, and total real memory.

³⁾ Every smart class attribute whose value changes has a corresponding limit (though this limit is hidden from users). A probe parameter only has a limit if you configure it to have one.



2. The Netbatch scheduler checks whether the specified requirement can **currently** be satisfied when it is matching Jobs to Workstations. If not, it checks again during its next cycle, then the next, and so on, until the requirement can be satisfied.

For Example

A Job is submitted with a smart class requirement that **free** real memory on the Workstation must be greater than the specified value. Netbatch makes two checks:

- *If there are any Workstations among those to which the Job may be dispatched that have **total** real memory that is higher than the specified value, Netbatch queues the Job.*
- *When the scheduler is trying to match a Workstation to the Job, it compares the value with the **free** real memory of each Workstation that becomes available. It only dispatches the Job to a Workstation if its **free** real memory is greater than the specified value (and if any other requirements are also satisfied).*

For Example

A Job is submitted with a smart class requirement that **total** real memory on the Workstation must be greater than the specified value. Netbatch makes two checks:

- *If there are any Workstations among those to which the Job may be dispatched that have **total** real memory that is higher than the specified value, Netbatch queues the Job.*
- *When the scheduler is trying to match a Workstation to the Job, it compares the value with the **total** real memory of each Workstation that becomes available. It only dispatches the Job to a Workstation if its **total** real memory is greater than the specified value (and if any other requirements are also satisfied).*

3.3 Job Life Cycle

3.3.1 Overview

Every Job that is successfully submitted to Netbatch follows a life cycle that is made up of a number of characteristic stages.

A typical Job will pass through the following stages in the Netbatch system:

1. A Job ID is assigned to it.
2. The Job will wait some time until a Workstation and any reserved resources are allocated to it.



3. The Job is sent to the allocated Workstation for execution.
4. The Job starts executing on the allocated Workstation.
5. The Job finishes executing. (See [Section 3.3.2, Job Completion Detection](#).)
6. The Workstation is freed and the Job's termination is recorded.

Other life cycles are possible depending on the type of Job that is submitted, the policies that apply to it, and which actions were taken after it was submitted.

At any point in time, it is possible to see where a Job is in its life cycle according to the State Queue it appears in. See [Section 2.1.15, State Queues](#).

3.3.2 Job Completion Detection

There are several different Job completion scenarios that Netbatch must handle. These are summarized in [Table 1](#), below:

Table 1: Job Completion Scenarios

Type of Job/ How Job Runs	Child Processes ¹⁾	How Netbatch Determines that a Job Has Completed
<p><u>Batch/blocking/ interactive Job</u></p> <p><u>Batch Job:</u></p> <ul style="list-style-type: none">• Job is executed asynchronously.• Job's stdin file descriptor is not written to.• Job's stdout file descriptor is dumped to a file. <p><u>Blocking Job:</u> As batch Job, but executed synchronously.</p> <p><u>Interactive Job:</u></p> <ul style="list-style-type: none">• Job is executed synchronously.• Job's stdin file descriptor receives input redirected from stdin of the submitting nbjob run process.• Output from Job's stdout file descriptor is redirected to stdout of the submitting nbjob run process.	No	When the Job's file descriptors (stdin, stdout, & stderr) are closed, Job is considered ended.

**Table 1: Job Completion Scenarios**

Type of Job/ How Job Runs	Child Processes ¹⁾	How Netbatch Determines that a Job Has Completed
Terminal Job²⁾: <ul style="list-style-type: none">• Job is executed synchronously.• The Job's stdin file descriptor receives input redirected through a terminal emulation from stdin of the submitting nbjob run process.• Output from Job's stdout file descriptor is redirected through a terminal emulation to stdout of the submitting nbjob run process.	No ³⁾ Yes	When the Workstation receives a SIGCHLD signal from the Job. When the Workstation receives a SIGCHLD from the Job, the Job is no longer considered by Netbatch to be a terminal session. Any remaining processes are handled as belonging to an ordinary batch Job (see above). The nbjob run client exits and notifies the user that these processes continue running "in the background." Netbatch Process Authentication Groups (PAGs) are used to track child processes. This means that even those that have different PGIDs than the parent can be tracked (unless PAGs are disabled).

¹⁾ If use of PAGs is enabled, Netbatch uses the PAG to determine whether any child processes are still running. If use of PAGs is not enabled, Netbatch uses the Process Group ID (PGID) to determine whether any child processes are still running.

²⁾ As of version 6.3.1, Netbatch automatically senses the context in which an interactive or terminal Job is being executed, and runs it as either an interactive or terminal Job, whichever is appropriate. Thus, there is no need to use --terminal. Whenever you want the Job to run as if it is running locally, use -i.

³⁾ For example, if running vi as a Terminal Job

See also:

3.3.3 States of a Job in Netbatch

Figure 4 shows the State Queues and sub-states within the State Queues that a Job can occupy, from the time the user submits it to the Pool until the time it exits the Pool.

The arrows between the states in **Figure 4** show the path of a typical Job in Netbatch. It is not important that you understand all of the intricacies of the Job's life-cycle at this stage. You can refer back to this section once you have a broader understanding of Netbatch, and want to understand the indicators Netbatch provides about the progress of your Job in the system.

A Job passes through the State Queues and sub-states depicted in **Figure 4** in the following way:

1. When a Job enters Netbatch it is placed in the Wait State Queue in an Alive state.
2. The Job can move to the suspend (susp) state if it is suspended by the user or a superuser (with the nbjob suspend command). The Job returns to the Alive sub-state once it is resumed (with the nbjob resume command).



3. When an appropriate Workstation becomes available, the Job enters the Run State Queue in a Send sub-state., If a Job's pre-execution script is running (as specified with `nbjob run`'s `--pre-exec` switch), it is also assigned the Send state.
4. Once the Job actually starts running, its sub-state changes to Run.
5. Here too the Job can move to a suspend (susp) sub-state if it is suspended by a user or a super user (with `nbjob suspend`). The Job will return to the Run sub-state once it is resumed (with `nbjob resume`).
6. The Job will enter the Del sub-state if the user has recalled it (with `nbjob recall`). The Job remains in that sub-state pending approval that the Job has been recalled successfully.
7. The disconnected (Disc) sub-state indicates that the Workstation the Job was running on has disconnected from the network.
8. A Job can return from the Run State Queue to the Wait State Queue if it is resubmitted (with `nbjob recall`, or according to a resubmission policy).
9. When the Job finishes (whether successfully or unsuccessfully), it enters the History State Queue.
10. If the Job is moved to another Pool (with `nbqrm -M`), it enters the Move State Queue. (When the Job finishes in the other Pool, it is moved to the History State queue.)
11. When a Job from another Pool (which was moved to this Pool) finishes executing, it enters the Notify State Queue pending confirmation from the origin Pool (that is, the Pool that the Job was moved from) that the Job has finished. When the confirmation is received, the Job enters the History State Queue.

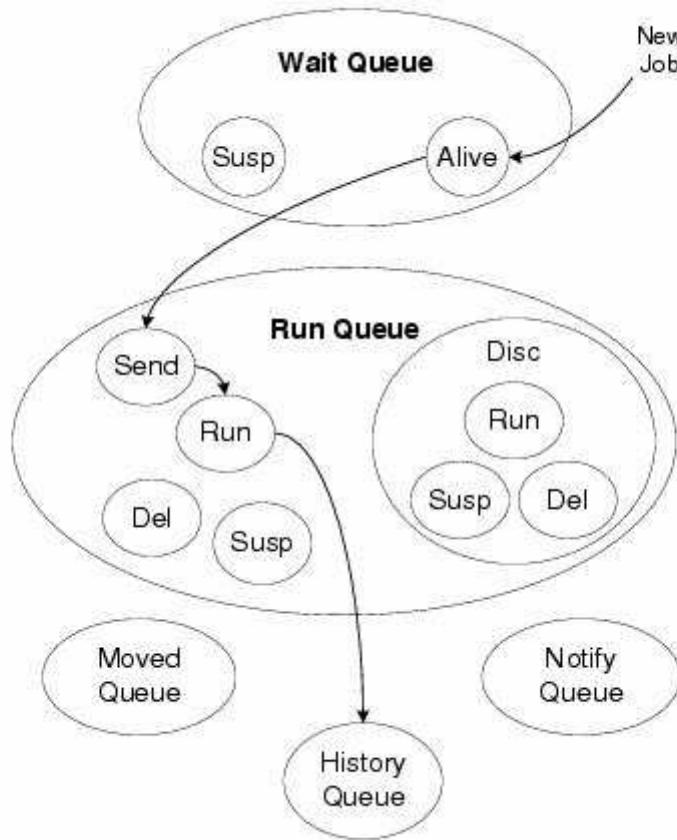


Figure 4: Job Life Cycle Diagram

3.3.4 JobID

Once your Job is submitted to Netbatch, it is assigned a unique JobID. The JobID is used to identify the Job in Netbatch.

There are several ways to refer to a Job's JobID:

- Short notation—This is a number that is assigned to the Job when it is submitted to Netbatch. For example:, 11
- Full notation —This includes the name of the Pool to which the Job was submitted and the number assigned to the Job. For example:, myPool.11
- Parallel Job extension—Parallel Slave Jobs are referred to by the JobID of their Master Job plus an index, which is different for each Slave Job. These can be expressed in short or full notation, as shown above. For example:

13:5

myPool.13:5



All these forms of JobID may be used. When Jobs are moved from one Pool to another, it is advisable to use the full JobID notation to avoid confusion.

3.3.5 Suspended Jobs

Netbatch suspends a Job if:

- The Job's owner or a superuser has requested that it be suspended.
- The Job depends on another Job which has failed or has been cancelled.
- The Job has started executing on a Workstation, and is then preempted by another Job from a higher priority Qslot.
- The Job has started executing on a Workstation, and the load on that Workstation has exceeded a set threshold.
- The Job has started executing on a Workstation, and the number of interactive users on that Workstation has exceeded the set limit.

Note

When the number of interactive users on a Workstation exceeds the set limit for that Workstation, a Job running on it may also be "niced" instead of suspended.

"Nicing" a Job means modifying its UNIX priority. The effect is that your Job is "nice" to other UNIX processes, and allows them to use more CPU time.

The Qslot priorities, load thresholds, interactive user limits, and whether Jobs are niced or suspended, are all configured by the Netbatch Administrator. You cannot control these aspects of Netbatch behavior.

Special command line switches can be specified by users during Job submission to determine what happens if a Job is suspended after it has started executing on a Workstation. The possible options are:

- Do nothing.
- Cancel the Job.
- Resubmit the Job a predetermined number of times, and then cancel it, if it is suspended more than the predetermined number of times.

In addition users may instruct Netbatch to notify them by email when these events occur.



For more information about Suspended Job options, see [Section 6.5.8, Suspended Job Options](#).

3.4 Netbatch Pool Structures

3.4.1 Overview

The basic Netbatch unit is the **Pool**. A Pool consists of a collection of **Workstations** and **Queues**. The Workstations may be grouped into **Resource Groups**. Individual Resource Groups can be assigned to the different Queues.

Netbatch supports a variety of Pool structures. The structure of the Netbatch Pool is determined in the Netbatch configuration files.

The Netbatch configuration files may be manipulated directly with a text editor, or indirectly through the provided installation scripts. Editing Netbatch configuration files directly offers maximum flexibility. Editing the Netbatch configuration files indirectly with the installation scripts offers maximum convenience.

The diagrams in the following sections illustrate some of the elementary Pool architectures.



3.4.2 Some Basic Pool Configurations

3.4.2.1 The Simple Netbatch Pool

Figure 5 Illustrates the structure of a simple Netbatch Pool called P1.

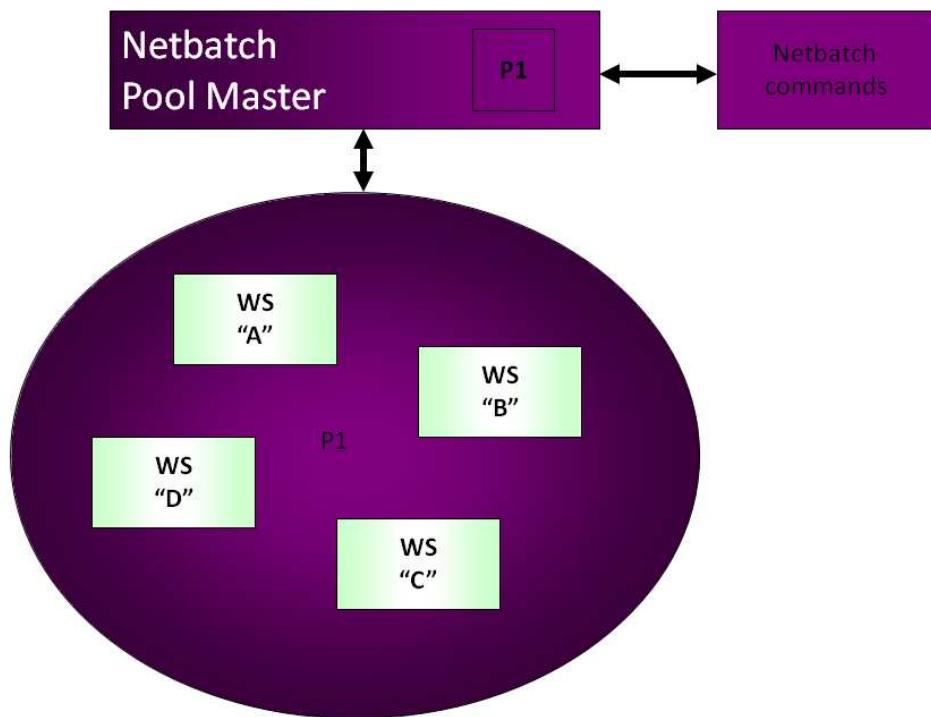


Figure 5: A Simple Netbatch Pool P1

Jobs submitted to the Pool P1 are scheduled in the Pool's default Queue. The default Queue has the same name as the Pool—P1.

The Jobs in the default Queue P1 are distributed by the Pool Master among the Workstations of the default Resource Set of the Pool which also has the same name as the Pool—P1.

The default Resource Set contains all of the Workstations of the Pool, marked A to D.



3.4.2.2 A Simple Pool with Queues

Figure 6 shows a Pool P2 with four Queues—three configured Queues called Q1 to Q3, and one default Queue called P2.

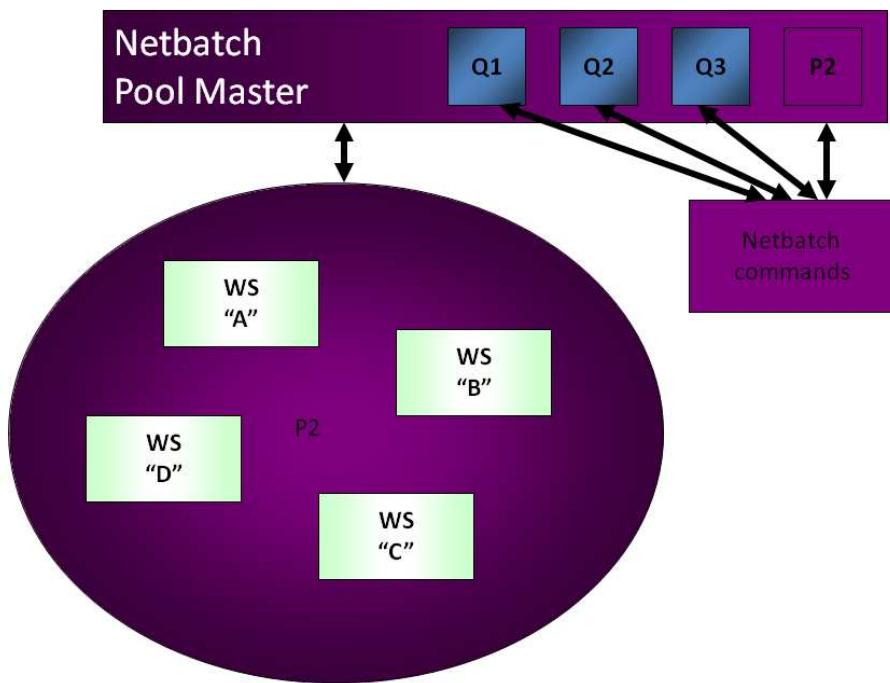


Figure 6: Netbatch Pool P2 with Queues

Jobs that are submitted to Queues Q1, Q2, Q3, or to the default Queue are distributed by the Pool Master among the Workstations of the default Resource Set P2 according to the resource allocation method and other settings configured for the Pool.

For Example

If the Queues of P2 are configured to use a prioritized resource allocation method, and Queues 1 to 3 are assigned the priorities 1 to 3 respectively, then Jobs submitted to Q3 will be dispatched for execution prior to Jobs submitted from Queue Q2 and Q1.

Each Queue (including the default Queue) can maintain a complete hierarchy of Qslots that utilize the Pool according to varying resource allocation methods.

For a complete discussion of Netbatch resource allocation methods, please refer to [Chapter 5.1, Resource Allocation Concepts](#).

3.4.2.3 A Simple Pool with Queues and Resource Groups

Figure 7 illustrates a simple Pool with three Resource Groups and four Queues.

The Resource Groups are RS_1, RS_2, and the default Resource Group P3.

The Queues are Q1, Q2, Q3, and the default Queue P3.

The Resource Groups RS_1 and RS_2 contain the Workstations A and D, and C and D respectively. The default Resource Group P3 contains all the Workstations in the Pool.

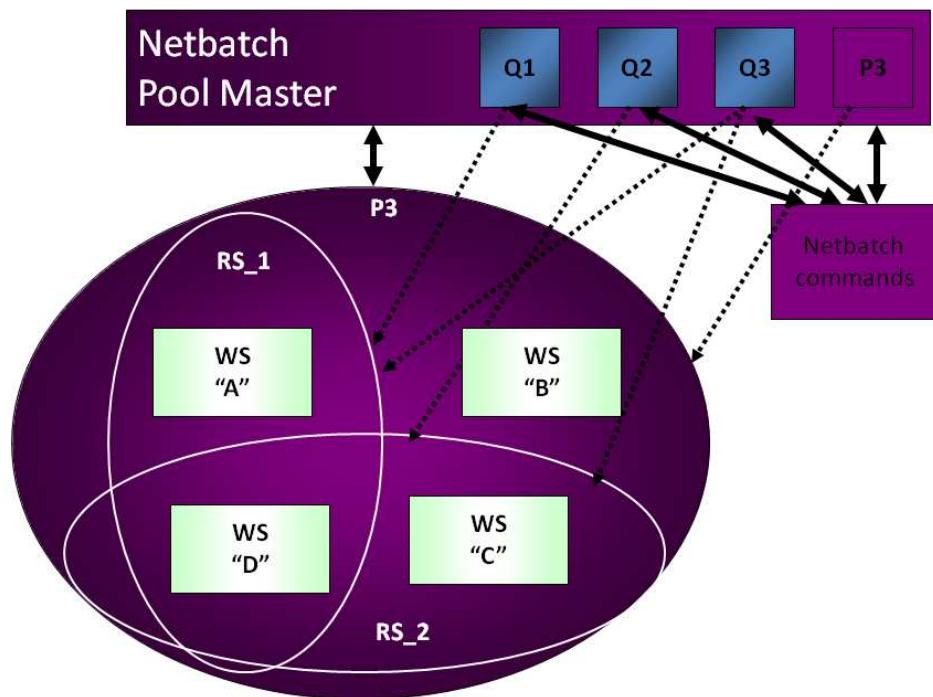


Figure 7: Simple Netbatch Pool P3 with Resource Groups and Queues

In P3:

- Queue Q1 can send Jobs to Resource Group RS_1.
- Queue Q2 can send Jobs to Resource Group RS_2.
- Queue Q3 can send Jobs to Resource Group RS_1 and Resource Group RS_2.
- The default Queue (i.e. Queue P3) can send Jobs to the default Resource Group P3.

Thus, Workstation B will process Jobs submitted to the default Queue only. Workstation C may process Jobs submitted to Q1, Q2, or the default Queue. Workstation D may process Jobs submitted to any of the Queues.

The precedence between the Jobs submitted to the Pool is determined by the precedence of the specific Queues they were submitted to.

The precedence between Jobs within the same Queue is determined by the resource allocation method that applies to that Queue.

If a Queue can send Jobs to multiple Resource Groups, the precedence is defined by the Resource Group order on the Queue's `member_resources` directive. See [Appendix E.5.22, member_resources](#).

3.4.2.4 Virtual Pool

A Virtual Pool has a structure similar to [The Simple Netbatch Pool](#) of [Section 3.4.2.1](#), but instead of workstations, the Pool dispatches Jobs to its virtual resources.

The virtual resources of a Virtual Pool may be Resource Groups or Queues of different Physical Pools.

Figure 8 illustrates the structure of a Virtual Pool.

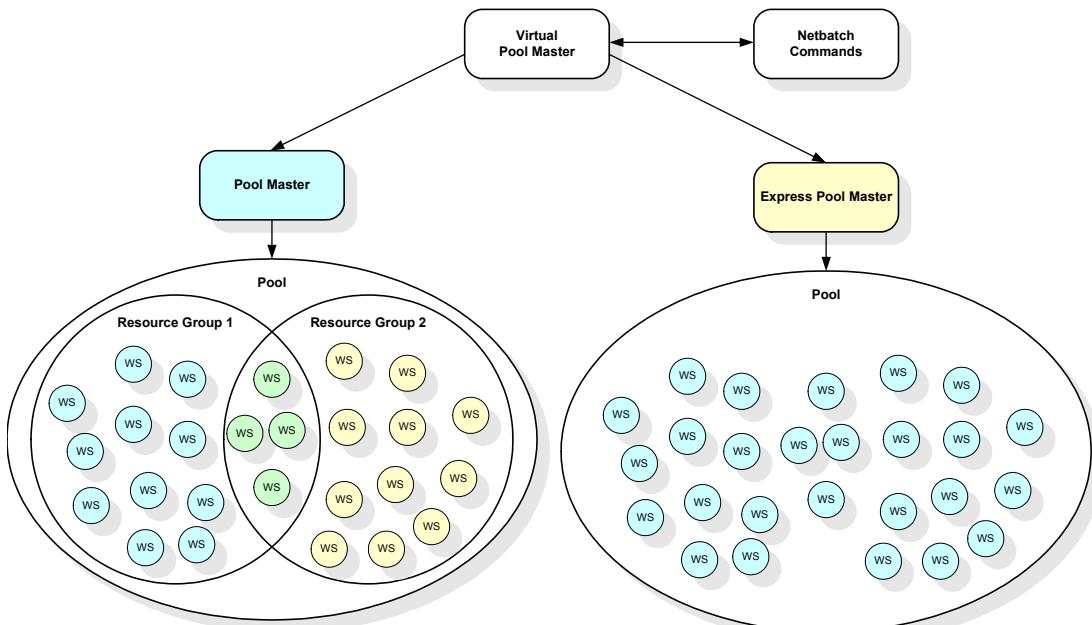


Figure 8: Virtual Netbatch Pool Structure



Virtual Pools are typically used to implement load balancing between different Physical Pools within the same site or at different sites.

To learn more about Virtual Pools, see [Section 13, Virtual Pools](#).

3.4.3 System Components

Netbatch is made up of a number of services that implement various Pool structures:

- **Pool Master Services**, see [Section 3.4.3.1](#)
The Pool Master runs a number of separate services.
- **Workstation Services**, see [Section 3.4.3.2](#)
Each Workstation runs the **Workstation Manager** (WSM) service (which has a built-in **Load Monitor** service). It can also include the WSM auto-tuning service.
- **Directory Service**, see [Section 3.4.3.3](#)
The **Directory Service** acts as a registry for Pools, Virtual Pools, and Netbatch Trackers.
- **Virtual Pool Master (VPM)**, see [Section 3.4.3.4](#)
- **Netbatch Tracker**, see [Section 3.4.3.5](#)
Netbatch Tracker runs three services—the query service, the reporting service, and the GUI service.
- **External Probes**, which allow Netbatch to use existing tools to monitor resources—see [Section 3.4.3.6](#)
- **Master Authentication Service (Windows Only)**, see [Section 3.4.3.7](#)
- **Configuration Service (Windows Only)**, see [Section 3.4.3.8](#)

Figure 9 illustrates the relationships between the Netbatch components in a Physical Pool. (The Netbatch Tracker database is not a Netbatch component, but is shown for clarity.)

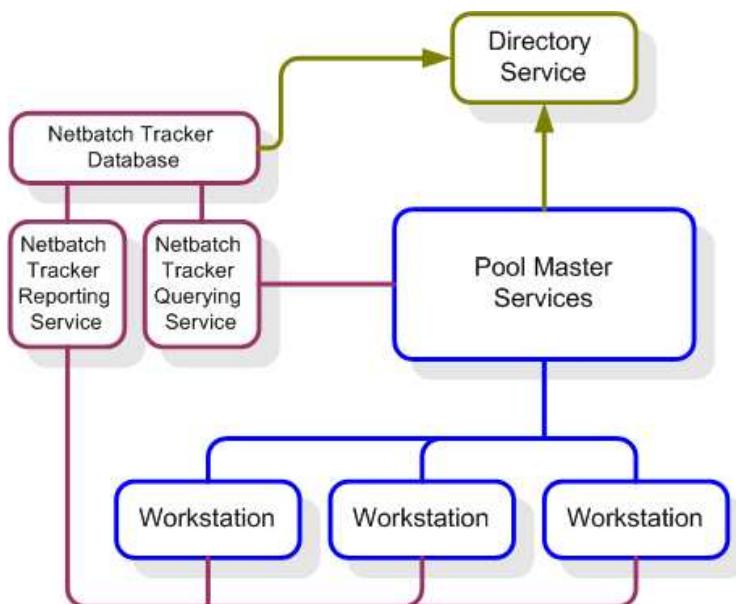
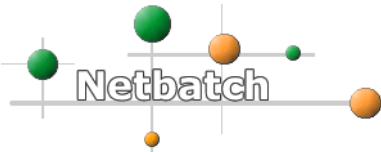


Figure 9: Netbatch Pool Components

Figure 10 illustrates the relationships between the Netbatch components in a Virtual Pool.

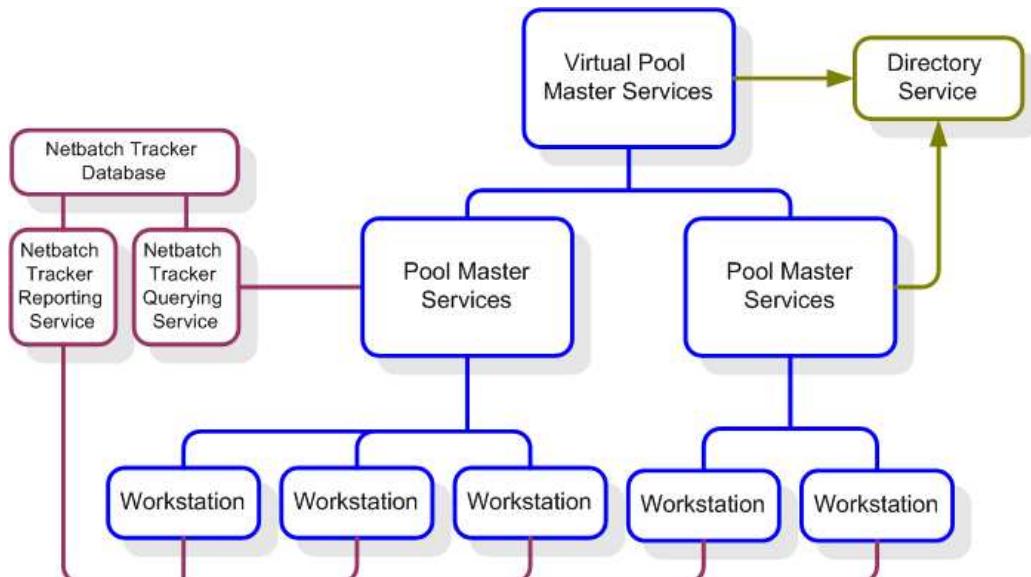


Figure 10: Netbatch Virtual Pool Components



3.4.3.1 Pool Master Services

The Pool Master runs a number of different services, each of which is responsible for a different area of functionality, as follows¹ (* means a built-in service):

- ◆ **Authorization** - holds details of all permissions that are set for the actions that can be performed on the various Netbatch entities (that is, the Pool, Qslots, Workstations, Resource Sets. and Jobs).

When a user runs a command to perform an action, this service checks whether the user has permissions to perform the action.
- ◆ **Debug*** - allows the Netbatch development team to perform debugging operations.
- ◆ **Groups** - caches information about which users belong to each NIS group and provides this information to services than require it, and when requested through the `nbstatus groups` command.
- ◆ **Licensing** - holds information about available features. The scheduler consults this service when it needs to dispatch a Job that has license requirements to check whether the feature(s) that the Job requires are available.
- ◆ **LocalDBReporter*** - handles querying of and reporting to Netbatch Tracker.
- ◆ **LocalMail*** - handles mail delivery and consolidation of mail messages.
- ◆ **MatbatchLogFile** - responsible for writing to the Matbatch log file.
- ◆ **NetbatchConfiguration** - reads and caches the configuration files. Other services request configuration information from this service instead of reading the files themselves.
- ◆ **PeriodicJobs** - allows certain Jobs to be automatically submitted according to a schedule. Specifically, it is used to run the WSM auto-tuning feature's learner process automatically.
- ◆ **Persistency*** - services that need to write to persistency use this service.

¹⁾The names of the services are as they appear in the output of the `nbservice status` command.



- ◆ **RegistrationService** - acts as a registry for Netbatch feeders and is also responsible for registering the Pool with the central Directory Services.

Each Feeder has an expiration date/time. If it does not unregister itself by this date/time, the Directory Service unregisters it. This means that a Feeder that fails before unregistering itself does not stay registered indefinitely.

The Registration Service keeps a list of all the central Directory Services (there are a number of such services, which stay synchronized with each other). When it re-registers (once a day) and when it receives an `nbstatus pools` request, it tries the same Directory Service that it used last time. If the Directory Service does not respond, it tries the next one in the list, and so on, until it finds one that does respond.
- ◆ **RequestService** - handles all incoming client requests.
- ◆ **ResourceAllocation** - maintains a view of the resource allocation structure (Queues, Qslots, and User Slots), and handles charging, checking permissions, and so on.
- ◆ **ResourcesDirectory** - maintains a list of the Workstations in the Pool and each one's status.
- ◆ **ResourcesUpdate** - collects updates sent by the Workstations.
- ◆ **Scheduler** - decides which Job to run next and on which Workstation.
- ◆ **ServiceCommunication*** - allows services to communicate with each other.
- ◆ **ServiceConfiguration*** - this service uses the Service Manager service to start first the built-in services and then the services listed in the service configuration file (`NBServices.conf`).
- ◆ **ServiceManager*** - this service handles all start, stop, hup, and status requests for services. (You can hup individual services.)
- ◆ **SessionDirectory** - maintains a list of running, waiting, and recently completed (history queue) Jobs.
- ◆ **SessionsUpdate** - collects Job updates.
- ◆ **Topology*** - maintains a list of services. For each service, it knows its status, the machine it is running on, the port it listens on, and so on.



The services are started in the following order:

1. Bootstrap stage—the Topology, Service Manager, and Service Configuration services start (in that order).
2. The Service Configuration service uses the Service Manager service to start all the built-in services.
3. The Service Configuration service uses the Service Manager service to start the non-built-in services (that is, the ones listed in `NBServices.conf`).

3.4.3.2 Workstation Services

3.4.3.2.1 Workstation Manager (WSM)

The Workstation Manager controls Job execution on the Workstation on the basis of information that it receives from the Load Monitor.

Note

The WSM replaces the Netbatch 4.x and 5.0 garcon server.

WSM Lite is a lightweight implementation of the Workstation Manager that does not require Java. This allows it to run on machines with limited memory, such as Windows CE-based devices. See [Section 7, Netbatch on Non-Java Platforms](#).

The following subsections describe the other services that run on the Workstation.

3.4.3.2.2 Load Monitor (LM)

The Load Monitor runs on the Workstation and collects information about:

- Workstation hardware and operating system
- Workstation utilization
- Number of interactive users
- Idle time since last interactive use
- CPU load
- Used real memory
- Used virtual memory
- Disk utilization

The Load Monitor process can run jointly with the WSM process (this is the default) or separately.



3.4.3.2.3 nbresserv

nbresserv is part of the WSM process. It collects the following X-display user information, which it reports to the Load Monitor:

- Interactive X-users using the same system as the X-display
- Interactive X-users using a different system from the X-display
- Remote X-displays—interactive X-users using a different system from the display, with the X-display being remote and managed by a local XDM (X-display manager); for example, Exceed.

Note

*Tcl applications are not ICCCM (Inter-Client Communications Conventions Manual) compliant. This means that **nbresserv** cannot detect them.*

The result is that instead of being suspended (or killed, resubmitted, or niced) when a user starts using a Tcl application, a Netbatch Job that is running on the same machine will continue to run.

If a user complains that this is happening, tell the user to add the following command to the Tcl application to remedy the situation:

```
wm client . $env(HOST)
```

3.4.3.2.4 nbserviced

The **nbserviced** daemon is an optional service. If enabled, it runs on the Workstation along with the WSM. When the WSM starts running, it registers itself with **nbserviced** (which includes telling **nbserviced** the command used to start it).

If the PPM detects that the WSM has failed, it instructs **nbserviced** to restart it. (It does this using the correct command, to ensure that it starts the correct version.)

3.4.3.2.5 WSMAutoTuningService

The WSM auto-tuning service (if enabled) uses machine learning to establish the optimum number of Jobs that a Workstation should be running, given its current state (load, free real and virtual memory, etc.).



3.4.3.3 Directory Service

The Directory Service acts as a registry for Pool Masters, Virtual Pool Masters, and Netbatch Trackers.

Each Pool Master, Virtual Pool Master, and Tracker registers with the Directory Service:

- At startup
- Once each day
- Whenever it is hopped
- When it is forced to re-register

Note

When a Pool Master is successfully moved from one machine to another (that is, its persistency area is copied from the old machine to the new one), the change is automatically detected by the Directory Service.

You can use the **nbstatus pools** command to view registered Pools.

If a registered component fails to re-register for three days, it is removed from the registry.

3.4.3.4 Virtual Pool Master (VPM)

The Virtual Pool Master manages the Virtual Pool. Jobs and requests submitted to the Virtual Pool Master are distributed among its constituent Pools and Queues.

3.4.3.5 Netbatch Tracker

Netbatch Tracker records information about Jobs, Workstations, Qslots. By default, it keeps this information for two months. You can query Tracker for information about:

- Any Job that was submitted in the last two months
- The status of a Qslot at any time in the last two months

See **Section 11, Using Netbatch Tracker** for detailed information about using Netbatch Tracker.



3.4.3.6 External Probes

An External Probe is a generic tool that lets you define a resource that can be monitored using existing tools. External Probes fall into two categories:

- Workstation probes monitor parameters for one or more Workstations.
- Scheduler probes monitor parameters at the scheduler (Pool) level.

Having this additional information can lead to:

- Better matching between Jobs and virtual resources
- System flexibility and better usage of virtual resources
- Lower administrative overhead, as you can easily limit users from using overloaded resources

External Probes can be used together with Workstation policies to implement specific desirable behaviors. For example, you can specify that if paging on a Workstation exceeds the specified threshold, no new Jobs should be sent to it. See [Section 13.5, Defining Workstation Policies](#).

The information gathered can also be used by users when submitting Jobs in a class expression (using `nbjob run`'s `--class` switch).

3.4.3.7 Master Authentication Service (Windows Only)

In Pools that include Windows Workstations, the Master Authentication Service (MAS) provides a secure way of communicating passwords in cleartext from a user to the WSM to which the user's Job has been dispatched.

3.4.3.8 Configuration Service (Windows Only)

In Pools that include Windows Workstations, the configuration service supplies Pool information to the Windows Workstations at runtime.



4 Quick Start Guide

The Quick Start Guide provides the following information:

- [Netbatch Commands](#)
- [Submitting Jobs](#)
- [Basic Troubleshooting](#)

4.1 Netbatch Commands

4.1.1 Old Command Binaries

The old Netbatch 4.x and 5.0 commands are deprecated:

- | | |
|--|---|
| <ul style="list-style-type: none">• <code>nbdepend</code>• <code>nblicstat</code>• <code>nbq</code>• <code>nbqrm</code> | <ul style="list-style-type: none">• <code>nbqslot</code>• <code>nbqstat</code>• <code>nbstat</code> |
|--|---|

4.1.2 Elementary Commands

Netbatch's user functionality is implemented as the following five commands:

- `nbjob run`—submits Jobs for execution by Netbatch. See [Section 6, Submitting Jobs to Netbatch](#) for a detailed explanation of this command.
- `nbstatus jobs`—monitors Jobs in a Pool. See [Section 8, Tracking Job Status](#) for a detailed explanation of this command.
- `nbjob modify`—modifies Jobs that have been submitted to a Netbatch Pool. See [Section 9, Moving, Changing, and Cancelling Jobs](#) for a detailed explanation of this command.
- `nbjob remove, suspend, resume, recall`—remove and suspend Jobs, resume suspended Jobs, send running Jobs back to the wait state queue.
- `nbstatus workstations`—monitors Workstations in a Pool. See [Section 4.3.2, Running Job Takes Too Long](#).
- `nbstatus qslots`—displays information about the Queue and Qslot structure of a Pool. See [Section 5.4.1, Using nbstatus qslots to View Resource Allocation Structure Details](#).



There are additional commands for controlling certain other aspects of Netbatch. See [Appendix A: User Command Reference](#).

4.1.3 Common Netbatch Command Switches

The following command line switches are shared by all of the above Netbatch commands:

--version

Print version number and release date of the Netbatch command binary.

--help

Display help information.

--target

Direct the Netbatch command to the specified Pool (which can be specified by Pool name or Pool Master hostname).

--debug

Show debug information.

4.2 Submitting Jobs

4.2.1 Before You Start

Before you can start submitting Jobs you must:

1. Make sure the directory where the Netbatch commands are located is in your UNIX path. See [Section 4.2.2](#).
2. Find out the location of the **Netbatch configuration directory**, and set the **NBCONF** environment variable. See [Section 4.2.3](#).
3. Gather information about the **Pools** that are available to run your Job. See [Section 4.2.4](#).
4. Gather information about the **Queues** that are available to run your Job. See [Section 4.2.5](#).
5. Gather information about the **Qslots** that are available to run your Job. See [Section 4.2.5](#).
6. Gather information about the **Classes** that are available to run your Job. See [Section 4.2.6](#).

The examples in the following sections refer to a hypothetical Pool called *myPool* which has the structure shown in [Figure 12](#) below.

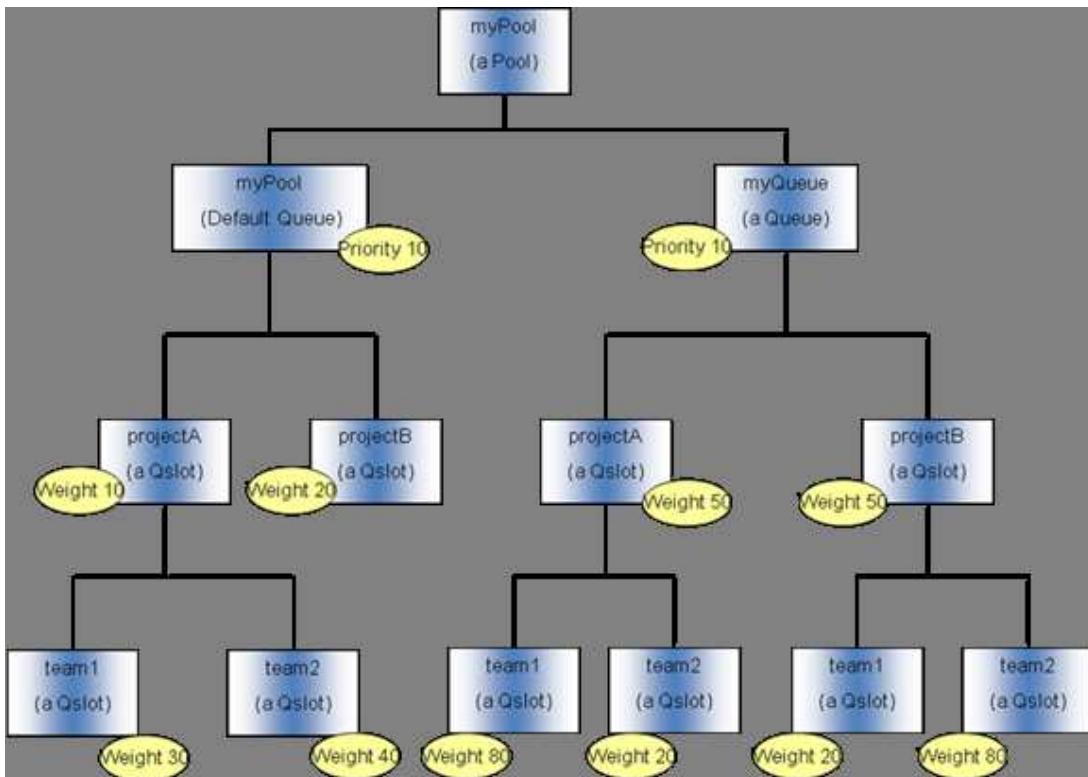


Figure 12: Example Pool Structure

4.2.2 Updating Your Path

The Netbatch commands should be available in common tools (`/usr/intel/bin`) at all sites.

If so, you should be able to run Netbatch commands. Try this by typing:

```
nbstatus jobs
```

If you receive a `Command not found` error:

1. Ask your Netbatch Administrator where the Netbatch binaries are located.
2. Add this location to your path.

4.2.3 Specifying the Netbatch Configuration Directory

To specify the Netbatch configuration directory:

1. Ask your Netbatch Administrator where the Netbatch configuration directory is located.



2. Set the value of the **NBCONF** environment variable to this location.

If you do not set **NBCONF**, Netbatch assumes that the Netbatch configuration directory is at its default location of `/usr/local/lib/netbatch`.

For Example

If you are using tcsh:

```
setenv NBCONF <path_to_config_directory>
```

Note

*As the **NBCONF** directory rarely changes, you may want to have the **NBCONF** environment variable set in your UNIX logon script.*

4.2.4 Selecting a Pool

To find out which Pools exist on your site and which Pools are available for your use, either:

- Ask your Netbatch Administrator.
- Locate the file called `pools` in the **NBCONF** directory. The `pools` file contains a list of all of the Pools and Queues that are configured at your site, along with the hostnames of the Netbatch Pool Masters that manage them.

When you run a Netbatch command, you must specify the Pool Master to which the command is directed. The best way to do this is to specifying the name of the Pool as the argument to the `--target` switch.

For Example

```
nbjob run --target myPool echo hello
```

This command submits the Job `echo hello` for execution by the Netbatch Pool named `myPool`.

If you set the **NBPOOL** environment variable, you will not have to use the `--target` switch for every Netbatch command that you run.

For Example

```
setenv NBPOOL myPool
```

**Note**

*The examples below assume that **NBPOOL** has been set.*

For Example

```
nbjob run echo hello
```

This command also submits the echo hello Job for execution by the Netbatch Pool named myPool.

Note

*Specifying a Pool name explicitly in a Netbatch command overrides the Pool specification in the **NBPOOL** variable.*

For more details on how Netbatch commands select the Pool Master, see [Section 6.5.1, Specifying a Pool](#).

4.2.5 Qslot Selection

Once you have decided which Pool you intend to use ([Section 4.2.4](#)) and have set **NBPOOL**, you must select a Qslot, as follows:

1. Find out which Qslots you can submit Jobs to. See [Section 4.2.5.1](#).
2. If there is more than one that you can use, select the one with the highest priority/allocation. See [Section 4.2.5.1](#).

4.2.5.1 Finding Out which Qslots You Can Use

To find out which Queues and Qslots in the Pool you can use, type:

```
nbstatus permissions
"access=='grant'&&PermissionType=='JobSubmit'&&
(UserGroup=='<username>' ||
UserGroup=='<group_name>' ||
UserGroup=='*'")
```

where:

- **username** is your username.
- **group_name** is the name of a group that you belong to.



This command displays a list of all the Queues and Qslots for which Job submission permissions have been granted to `username`, `group_name`, or to everybody (*).

For Example

Figure 13 below shows the output of the following command:

```
nbstatus permissions
"access=='grant'&&PermissionType=='JobSubmit'&&
(UserGroup=='martinxpx'|||
UserGroup=='iec_sws'|||
UserGroup=='*'")
```

The Queue queue2 and the Qslots /1, /2a, /a, and /b accept Jobs from anyone.

The Qslot /c accepts Jobs from users that belong to UNIX group iec_sws.

The user slot /2a.martinxpx accepts Jobs from the user martinxpx only.

AuthorizationService on itstl139						
Version 7.1.0_0156_00						
On since 10/27/2005 14:25:52						
Time now 10/30/2005 08:31:23						
EntityType	EntityName	UserGroup	PermissionType	Access	Recursive	
Qslot	/1	*	JobSubmit	grant	false	
Qslot	/2a	*	JobSubmit	grant	false	
Qslot	/2a.martinxpx	martinxpx	JobSubmit	grant	false	
Qslot	/a	*	JobSubmit	grant	false	
Qslot	/b	*	JobSubmit	grant	false	
Qslot	/c	iec_sws	JobSubmit	grant	false	
Queue	queue2	*	JobSubmit	grant	true	

Figure 13: nbstatus permissions Output

To find out which UNIX groups you belong to, type:

```
id -Gn
```



4.2.5.2 Selecting the qslot According to Priority/Allocation

You should usually have permissions for Qslots that correspond to the organizational unit to which you belong. In other words, if you belong to Team 1 and are working on Project A, than you should have permission to submit Jobs to the Qslot `/projectA/team1`. If not, talk to your Netbatch Administrator.

Note

Often, a group is assigned more than one Qslot—a high-priority Qslot for short Jobs, and a low-priority Qslot for longer Jobs. Make sure you follow your group guidelines.

If there is more than one Qslot that you are allowed to use, choose the one with the highest weight (**Allocation**) or **Priority** factor

Note

*Qslots that compete for resources according to a fair-share scheduling policy have a **weight** value assigned to them.*

*Qslots that compete for resources according to a priority scheduling policy have a **priority** value assigned to them.*

See [Section 2.4.11, Resource Allocation Methods](#).

Commonly, a Qslot with a higher numeric value (or alias) has higher priority or weight than a Qslot with a lower numeric value.

4.2.5.3 Job Submission Switches

When you submit your Job, you must specify both the Queue and the Qslot that you want to use:

```
nbjob run --queue <queue_name>
--qslot <qslot_name> [<other options>] <command>
```

For Example

```
nbjob run --queue myQueue --qslot 2a echo hello
```

This command submits the Job `echo hello` to the Qslot 2a in the Queue `myQueue`.



*Note that this command assumes that the **NBPOOL** environment variable has been set. See [Section 4.2.4](#).*

Note

*You can submit Jobs to a Queue's default Qslot by omitting the **--qslot** switch (assuming you have permissions to submit Jobs to the Queue).*

*However, submitting Jobs to the default Qslot is **not recommended**, as this Qslot usually has a very low priority—that is, Jobs submitted to this Qslot spend a lot of time waiting for execution.*

If you plan to submit all or most of your Jobs to a specific Queue, you can avoid typing the name of the Queue for each Netbatch command you invoke by setting the **NBQUEUE** environment variable.

For Example

```
setenv NBQUEUE myQueue
```

Once the **NBQUEUE** environment variable is set, you may omit the name of the Queue from all Netbatch commands.

For Example

```
nbjob run echo hello
```

This command also submits the `echo hello` command to the Queue `myQueue`.

Specifying a Queue name explicitly in a Netbatch command overrides the Queue specification in the **NBQUEUE** variable.



4.2.6 Class Selection

Sometimes it is imperative that your Job run on a particular Workstation in the Pool, or on one of a group of Workstations in the Pool that have special characteristics. If your Job does not require a specific type of Workstation, then you are ready to submit it to the Pool, and can skip straight to [Section 4.2.7, Submitting the Job](#).

Note

The resources used by a Job when it starts running (for example, memory or disk space) are often less than those that it needs at later stages of execution. This can lead to a situation where two (or more) Jobs are dispatched to a Workstation and successfully begin execution, but hang because their increasing resource requirements exceed the capabilities of the machine.

You can avoid such problems by using Netbatch's resource reservation feature. See [Section 6.10, Submitting a Job with Resource Reservation](#).

The names and types of Classes in the Pool you use are determined by your Netbatch Administrator.

You can use the `nbstatus classes` command (see [Appendix A.3.1](#)) to find out which classes are supported by a Pool, Resource Set, or Queue, and to see how many Workstations support each class (and which Workstations they are):

- To see the classes supported by the Workstations in the Pool, type:
`nbstatus classes`
- To see the classes supported by the Workstations in a Resource Set, type:
`nbstatus classes "resouceset='<resource_set>'"`
where `resource_set` is the name of a Resource Set.
- To see the classes supported by the Workstations available to a Queue, type:
`nbstatus classes "queue='<queue_name>'"`
where `queue_name` is the name of a Queue.

For Example

Figure 14 illustrates the output of the `nbstatus classes` command.



```
PPM on itstl008
Version 7.2.0_0178_05
On since 11/06/2006 11:55:35
Time now 11/08/2006 13:45:47
```

Class	Configured	Connected	Available
@	1	1	1
burton	1	1	1

Figure 14: nbstatus classes Output

Class names usually indicate the Workstation attribute they refer to. For example, in **Figure 14**, the class **LINUX_256M** signifies that all Workstations that belong to this class run Linux and have at least 256 Megabytes of RAM.

To find out which Classes are supported by a specific Workstation in a Pool, type:

```
nbstatus workstations --fields 'server,classes'
"server=='<hostname>'"
```

For Example

Figure 15 shows the output of the following command:

```
nbstatus workstations --target sles_idc
--fields 'server,classes' --format block
"server=='icsl6755'"
```

The items after **Classes =** are the class names (comma-delimited).

```
Pool sles_idc on inbm006
Version 6.5_0134_19
On since 08/14/2005 10:07:45
Time now 09/01/2005 15:36:44
{
    Server = icsl6755
    Classes = @, 512M, 1G, 2G, 3G, 4G, scratch12G, scratch10G,
    scratch8G, scratch4G, scratch3G, scratch2G, scratch1G, scratch512M,
    tmpp26G, tmpp14G, SMP, CP3_1g, CP4_4g, CP4, 2400, CP_2400, 2G_SMP,
    x86_64
}
```

Figure 15: nbstatus workstations Output Showing Classes for a Specific



Workstation

Choose a Class or combination of Classes that best fits the kind of Job you intend to run.

Note

When submitting a Job, you specify a combination of Classes as a simple boolean expression of the form <class1> && <class2>.

For example:

```
nbjob run --class myClass echo hello
```

or

```
nbjob run --class "myClass1 && myClass2" echo hello
```

It is possible to specify an arbitrarily complex boolean expression as well as non-static Class requirements (for example, CPU utilization) and **external probe**¹ parameter values. See [Section 6.7.4, Submitting a Job with Resource Requirements](#).

If you do not specify Class requirements when submitting a Job, Netbatch assumes that your Job can run on any Workstation. The Class of such a Job is marked with the "@" symbol.

You are now ready to submit your first Job.

Note

Before submitting your Job, you can check whether the resources that you are requesting are available in the Pool without submitting the Job for execution:

```
nbjob run --class <class_expression> --validate  
<command>
```

With this switch, nbjob run displays Job not queued (validate mode), followed by one of the following:

- Class '<class_name>' is served by this pool.
- Class '<class_name>' is not served by this pool.

¹⁾An external probe is something that the Netbatch Administrator sets up to measure parameters that are not measured and available as Workstation (or other) attributes.



4.2.7 Submitting the Job

Use the `nbjob run` command to submit Jobs to Netbatch.

The `nbjob run` command has the following general syntax:

```
nbjob run [options] <command> [parameters]
```

where:

- `[options]` are the `nbjob run` options.
- `command` is the Job to be executed (as you would type it at the command line).
- `[parameters]` are the parameters passed to `command`.

The following options can be specified:

- Pool specification, see [Section 4.2.4, Selecting a Pool](#)
- Queue specification, see [Section 4.2.5, Qslot Selection](#).
- Qslot specification, see [Section 4.2.5, Qslot Selection](#)
- Class specification, see [Section 4.2.6, Class Selection](#)
- Email options, see [Section 6.5.6, Email Options](#)
- Job output location, see [Section 6.5.7, Job Output \(Log File\) Location and Options](#)
- Suspended Job options, see [Section 6.5.8, Suspended Job Options](#)
- License requirements, see [Section 6.5.9, Specifying a License](#)
- Environment variables, see [Section 6.5.11, Job Environment Variables](#)
- Job execution limits, see [Section 6.5.12, Job Execution Limits](#)
- Hung and runaway Job options, see [Section 6.5.13, Job Hung Limits](#)
- Pre-execution utility, see [Section 6.5.14, Specifying a Job Pre-Execution Utility](#)
- Post-execution utility, see [Section 6.5.15, Specifying a Job Post-Execution Utility](#)
- Job starter, see [Section 6.5.16, Specifying a Job Starter](#)
- Priority specification, see [Section 6.5.17, Specifying the Job Priority](#)
- Job task name, see [Section 6.5.18, Specifying a Task Name](#)
- Groups of Jobs, see [Section 6.5.19, Submitting Groups of Jobs](#)



- Resource limits, see [Section 6.5.20, Specifying Resource Limits](#)
- Protection from blackholes, see [Section 6.5.21, Protecting Jobs from Blackholes](#)
- Job dependencies, see [Section 6.6, Working with Job Dependencies](#)
- Reserving resources, see [Section 6.10, Submitting a Job with Resource Reservation](#)
- Parallel Jobs, see [Section 6.11, Submitting a Parallel Job](#)
- Interactivity, see [Section 6.15, Interactive Batch Jobs](#)
- File copying, see [Section 6.19, Submitting a Job when Workstations Might not Have Access to Required Files](#)
- Resource limits/wait time limit, see [Section 6.20, Using Job Constraints](#)
- How Process Authentication Groups are added to the Job, see [Section 6.1.2, Process Authentication Groups](#)

To submit your Job, type:

```
nbjob run --target <pool_name> [--queue <Queue>]  
[--qslot <qslot>] [--class <class>] <command>  
[parameters]
```

Parameters that are optional appear in square brackets ([]). Parameters that must be filled with real values appear in angled brackets (< >).

Fill in <pool name>, <Queue>, <qslot>, and <class> with the information you gathered above, replace <command> and [parameters] with the actual Job command (including any parameters), and press **Enter**.

Note

*If you have set the **NBPOOL** and **NBQUEUE** environment variables as explained in [Section 4.2.4](#), and [Section 4.2.5](#), then you do not have to specify the Pool and Queue parameters.*

Output

If your Job is successfully submitted to Netbatch, it is assigned a JobID, and you receive a message indicating successful submission of the Job, as in [Figure 16](#) below. The JobID is used to identify your Job in Netbatch.



```
Your Job has been queued (JobID 11, Class @, Queue myQueue, slot
projectB/team1)
```

Figure 16: Successful Job Submission Message

The Job's output is written to a file in:

- Your current working directory,
- The directory you specified with the **--log-file** switch,
- The directory you specified with the **--log-file-dir** switch, or
- One of the directories specified in the **NBWD** environment variable¹.

Netbatch names this file **##Date-Time#.<Workstation>** (unless you specify a different name with the **--log-file** switch).

Note

*If both **--log-file-dir** and **--log-file** are used and
--log-file's argument includes a path, **--log-file-dir** is ignored.*

More Information

See [Section 6, Submitting Jobs to Netbatch](#), for a complete discussion of **nbjob run**. [Table 2](#) lists some sample command strings.

Table 2: Sample Command Strings

Command String	Description
<code>nbjob run --target linux2 --qslot 10 myJob</code>	Submits <code>myJob</code> to Qslot 10 of the default Queue in Pool <code>linux2</code> .
<code>nbjob run --target linux2 --qslot project1 myJob</code>	Submits <code>myJob</code> to Qslot <code>project1</code> of the default Queue in Pool <code>linux2</code> .
<code>nbjob run --target dvpool --qslot /project2/design --class HP3000 myJob</code>	Submits <code>myJob</code> to Qslot <code>project/design</code> of the default Queue in the Pool <code>dvpool</code> , and requests an <code>HP3000</code> class machine to run the Job.
<code>nbjob run --target linux1 --queue fast --qslot 5 --priority 12 myJob</code>	Submits <code>myJob</code> to Qslot 5 of the Queue <code>fast</code> in the Pool <code>linux1</code> with a priority of 12.

4.3 Basic Troubleshooting

If you have a problem, use the following procedure:

¹⁾If NBWD contains multiple directories, and Netbatch cannot write to the first one, it tries the second, and so on, until it finds a directory that it can write to.



1. Check the questions and answers below.
2. If this does not help, call front-line support for your site.
3. If they cannot solve the problem, they pass it on to the Netbatch Administrator. If needed, the Netbatch Administrator consults with the NB SET team.
4. If the Administrator cannot solve the problem, the Netbatch development team intervenes.

The following sections describe some common problems.

4.3.1 Job in Wait Queue too Long

Your Job may be in the **Wait Queue** for the following reasons:

- There are no machines available of the Class that you specified.
- The maximum number of concurrent Jobs defined to run in the Qslot has been exceeded.
- The maximum number of concurrent Jobs defined to run on a specific class of Workstations has been exceeded.
- The Qslot state is inactive.
- The load in the designated Pool is high and your Job is in a low priority Qslot.
- The required licenses are not available for your Job.
- Your Job is a Parallel Job and Netbatch is accumulating the execution slots that the Job requires.
- Your Job needs certain resources to be reserved and Netbatch is accumulating the resources that the Job requires.

Check if your Job is in the Wait State Queue as follows:

1. Type:

```
nbstatus jobs --target <poolName>
"user='<username>'&&status=='wait'"
```

where:

- <poolName> is the name of the Pool you submitted the Job to,
- <username> is your UNIX system logon user name.

Figure 17 illustrates the expected nbstatus jobs output.



```
Pool newpoll on itstl139
version 7.0.0_0153_03
on since 09/01/2005 12:58:25
Time now 09/04/2005 10:22:49
```

Status	Jobid	Class	Qslot	User	Cmdline	workstation
<hr/>						
wait	28	@	/	martinpx	sleep 10	
wait	29	@	/	martinpx	sleep 1	
wait	30	@	/	martinpx	sleep 1	

Figure 17: Expected nbstatus jobs Output

2. If your Job is in the list, it is still waiting.

Note

If you submitted your Job with Resource Reservation and/or License Reservation, you can use the --fields switch to display the following fields so you can see both the requested resource/license reservation and the resources/licenses that Netbatch has already accumulated for the Job:

- **ClassReservation**—requested resource reservation requirements
- **ClassImplicitReservation**—implicit reservation defined at Queue/Qslot level

(The Netbatch Administrator can specify that Jobs submitted to a particular Queue or Qslot are automatically assigned real or virtual memory requirements, or requirements that correspond to defined custom Workstation attributes.)

- **LicenseReservationRequest**—requested license reservation requirements
- **ReservedClasses**—resources that have already been reserved for the Job
- **ReservedLicenses**—licenses that have already been reserved for the Job

For example, to display the resource reservation requirements that were requested and the resources that have already been reserved for each of your waiting Jobs, type:



```
nbstatus jobs --target <poolName> --fields
'* ,ClassReservation,ReservedClasses'
"user='<username>'&&status=='wait'"
```

3. Use **nbstatus workstations** to list the Workstations and supported Classes in the Pools available to you:

```
nbstatus workstations --target <pool_name> --fields
'* ,classes'
```

Check if there are any that match the Class that your Job requires.

4. If there are, use **nbstatus qslots** to see if you could be using a better Qslot. For details of how to do this, see [Section 4.2.5, Qslot Selection](#).
5. If not, check whether you can specify a different Class for your Job. If you know that the Job runs on your machine and your machine belongs to a Netbatch Pool, you can specify the same Class for your Job.

Check your machine Class as follows:

- a) Type:

```
nbstatus --target <poolName> --fields '* ,classes'
"server=='<hostname>'"
```

where **poolName** is the name of the Pool that your machine belongs to and **hostname** is your machine's hostname.

- b) Locate your machine in the list and see which classes it supports.

Note

For details of how to resubmit your Job to a different Qslot or Class, see [Chapter 9, Moving, Changing, and Cancelling Jobs](#).

4.3.2 Running Job Takes Too Long

A Job in process may take a long time for the following reasons:

- Netbatch has suspended your Job because of interactive use or high load on the Workstation running the Job.
- Netbatch has niced your Job (because of interactive use or high load on the Workstation it is running on).
- There is a problem with the Workstation that Netbatch sent your Job to.
- The Job is running on a low-specification Workstation.
- Your Job is hung due to unavailability of a required license.
- Your Job is faulty.



This section describes how to use **nbstatus jobs** and **nbstatus workstations** to track Jobs.

1. Use **nbstatus jobs** to generate a list of your Jobs in a specific Pool. Type:

```
nbstatus jobs --target <poolname> --fields
'*',ExitStatus' user=<username>
```

where:

- **poolname** is the name of the Pool you submitted the Job to.
- **username** is your UNIX system user name.

2. Find your Job in the list. Look at the Job status and note the Workstation name. The Job status is one of the following:

- **Comp**—the Job has completed (i.e., it ended or was killed).
 - **Wait**—the Job is in the Wait State Queue.
 - **Run**—the Job is currently running.
 - **Send**—the Job is being dispatched to a Workstation for execution. The Job is also assigned this state if a pre-execution script is running (as specified with **nbjob run's --pre-exec** switch).
 - **Fail**—failed to send the Job to the Workstation.
 - **DEL**—the Job is being deleted. Waiting for confirmation from Workstation.
 - **Disc-D**—disconnected delete - the delete request has been sent but there is a communication problem between the Pool Master and the Workstation. The Job will be deleted when communication is restored.
 - **Disc-R**—disconnected running - the Job is running on a Workstation, but there is a communication problem between the Pool manager and the workstation. The Job status will change to **Run** when communication is restored.
 - **Move**—the Job is being moved to a different Pool.
3. If your Job does not appear in the list, it could be because it was sent to a Blackhole.

Note

A *Blackhole* is a machine that takes Jobs but fails to execute them.



4. If the Jobs status is **Comp** (completed), the **ExitStatus** column shows its exit code:

- If the exit code is 0, this indicates that your Job has completed successfully.
- If the exit code is -8, this indicates that someone has killed it.

For other exit codes check [Appendix D, Log File Contents and Job Exit Codes](#).

Note

The maximum number of Jobs that are maintained at any given time in the History State Queue (and that are displayed with a status of Comp in the output of nbstatus jobs) is limited. Terminated Jobs are deleted from the end of the history queue as newly-completed Jobs are added to it.

You can also use nbstatus jobs --history to retrieve information about terminated Jobs from the Netbatch Tracker database. See [Appendix 22.31.10, nbstatus jobs](#) and [Section 11, Using Netbatch Tracker](#).

5. If your Job is shown as running, but is taking too long, make a note of the name of the Workstation it is running on (the hostname in the **Workstation** column).

Use **nbstatus workstations** to generate a Job status report for that Workstation as follows:

- a) Type:

```
nbstatus workstations --target <pool_name>
--fields all --format block "server=='<hostname>'"
```

where:

- **pool_name** is the name of the Pool
- **hostname** is the hostname of the Workstation on which your Job is running.

- b) Look at the values of the following fields (the meaning of each one is given below):

- **Load**—load—current load (average of the number of running processes 1 minute, 5 minutes, and 15 minutes ago)/lower load threshold (if exceeded, no new Jobs are accepted)/upper load threshold (if exceeded, running Jobs are suspended/niced)



- ◆ **InteractiveUsers**—number of users—active users/active user threshold (if exceeded, no new Jobs are accepted and running Jobs are suspended)
- ◆ **RunningJobsCount**—number of Jobs currently running on the Workstation
- ◆ **SuspendedJobsCount**—number of suspended Jobs currently on the Workstation
- ◆ **NicedJobsCount**—number of niced Jobs currently on the Workstation
- ◆ **HungJobsCount**—number of hung Jobs currently on the Workstation
- ◆ **RunAwayJobsCount**—number of runaway Jobs currently on the Workstation
- ◆ **ReservedJobsCount**—number of reserved Jobs currently on the Workstation
- ◆ **CanRun**—the number of Jobs that the Workstation can currently run
- ◆ **NotAcceptingReason**—the reason why the Workstation cannot accept any Jobs

These will give you a useful picture of the state of the Workstation.

- c) If it looks as though high load, interactive use, or some other factor is causing Netbatch to repeatedly nice or suspend Jobs on the Workstation, you may want to resubmit your Job. See [Chapter 9, Moving, Changing, and Cancelling Jobs](#).
6. Unless your Job has successfully completed, check that the command line submission is correct. If it is not, submit your Job again. See [Chapter 6, Submitting Jobs to Netbatch](#).

4.3.3 Netbatch Causes a Workstation to Run Slowly

If you think that a Netbatch Job is making your Workstation run slowly, check if Netbatch is running a Job on your Workstation as follows:

1. Type:

```
nbstatus workstations --target <pool_name>
"server=='<hostname>'"
```

where:

- **pool_name** is the name of the Pool
- **hostname** is the hostname of the Workstation.



Check if there is a data line for the Workstation and whether there are any Jobs running on it (i.e., the **Jobs** field has a value other than zero).

2. If there are, type one of the following:
 - in Linux:
`ps -efww | grep WSM`
 - in Solaris:
`ps -eax | grep WSM`
 - in HP-UX:
`ps -efx | grep WSM`
 - on machines running older versions of HP-UX:
`ps -ef | grep WSM`
3. Check how much CPU time and memory the WSM process and its child processes are using.
4. If a Netbatch Job is using resources on your machine, ask your Netbatch Administrator to do one of the following:
 - Kill the Job and resubmit it to a different Workstation.
 - Change the thresholds on your Workstation so that Netbatch automatically suspends Jobs when you need to use it.

Note

On Workstations, Netbatch runs as a daemon and does not use any resources. Only the Jobs Netbatch runs use resources.

4.3.4 Jobs Run Through Netbatch Crash or Hang My Machine

If your Netbatch Job unexpectedly crashes or hangs, check whether any of the following are true:

- Your Job requires user input. If it does, try submitting it with the **-i** switch. See [Section 6.15, Interactive Batch Jobs](#).
- Your Job is immune to signals **SIGTSTP**, **SIGSTOP**, or **SIGCONT**.
- The file system your Job requires is not mounted on the machine where Netbatch is trying to run your Job.
- You do not have permission to access the files that your Job needs.
- You do not have permission to create and write to the user log file. Netbatch creates this file in:



- The directory you specified with the **-J** switch,
- Your current working directory, or
- The directory specified in the **NBWD** environment variable¹.
- You did not define the right class for your Job.
- Your Job requires that the **LD_ASSUME_KERNEL** environment variable have a specific value, but Netbatch removes this environment variable.

To remedy this problem, use a Job Starter to reset **LD_ASSUME_KERNEL**.

4.3.5 Where Did My Job Go?

If your Job seems to have disappeared:

1. Type:

```
nbstatus jobs --target <poolname> --fields
'*',ExitStatus' "user='<username>'"
```

and check whether your Job appears in the output.

2. If your Job's status is **Comp**, it either finished running or was killed. Check the **ExitStatus** field. A value of 0 means that it finished running successfully. A value of -8 means that it was killed.

Note

If a Job's exit status is -31 or -50, it means that there was a problem opening or writing to the Job log file. This can happen for a number of different reasons.

If this happens repeatedly, look at the /tmp/netbatch_internal_error.log file on the Workstation to which the Job was dispatched. If this does not help, consult the Netbatch Administrator.

3. Check whether the output log file exists in the directory where you submitted the Job.
4. If the output log file is not in the directory where you submitted the Job, look for the output log file in one of the directories specified by your **NBWD** environment variable.
5. If you still cannot find the output log file, type:

¹⁾If NBWD contains multiple directories, and Netbatch cannot write to the first one, it tries the second, and so on.



```
nbstatus jobs --target <poolname> --fields
'* ,ExitStatus' --history 1D "user='<username>'"
```

Again, check whether the Job has completed, and what its exit status was.

If your Job appears to have completed successfully, and you still cannot find the output log file, call front-line support or your Netbatch Administrator.



CONFIDENTIAL

Netbatch
v8.2.2

Revision 4

Quick Start Guide



5 Resource Allocation Concepts

This section explains how Netbatch allocates resources to Jobs—that is, how Netbatch determines which Job to run on which of a Pool's Workstations and when.

If you do not have time to read this whole section, [Section 5.1](#), below, explains how to work out where to submit your Job in just a few easy steps.

The remaining sections explain resource allocation in greater detail.

5.1 Where Should I Submit My Job?

To find out which is the best Qslot that you can submit your Job to:

1. Find out which Queues you can submit Jobs to.

Type:

```
nbstatus qslots --target <pool_name> --fields
'*',permissions' "type=='queue'"
```

where <pool_name> is the name of the Pool that you are using.

In the output, look for Queues that either:

- Have your username or the name of a group that you belong to in their **Permissions** fields
- Have no names in their **Permissions** fields

2. Find out which Qslot(s) you can submit Jobs to.

For each of the Queues that you can submit Jobs to, type:

```
nbstatus qslots --target <pool_name>
--fields '*',permissions' "queue=='<queue_name>'"
&&type=='qslot'"
```

where <pool_name> is the name of the Pool that you are using and <queue_name> is the name of the Queue.

In the output, look for Qslots that either:

- Have your username or the name of a group that you belong to in their **Permissions** fields
- Have no names in their **Permissions** fields

3. Figure out which Qslot is most appropriate for your Job.



As a user, you usually only have one Qslot that you are allowed to submit Jobs to for each project that you are part of. Qslots are usually named according to the projects and/or teams that they belong to.

If there is more than one Qslot that you can submit your job to, figure out which one receives a greater share of available resources by comparing their weights and/or priorities (and the weight/priority of each one's parent Queue/Qslot, and so on up to the root of the Resource Allocation Structure).

Note

To be able to figure which Qslot receives a greater share of available resources, you will need to understand more about the principles of resource allocation. These are explained in detail in the following subsections.

The contents of the **Name** field is the full Qslot path that you will specify when you submit Jobs.

4. Submit your Job.

Type:

```
nbjob run --target <pool_name> --queue <queue_name>
--qslot <full_Qslot_path> <command> <arguments>
```

where:

- <**pool_name**> is the name of the Pool that you are using,
- <**queue_name**> is the name of the Queue,
- <**full_Qslot_path**> is the Qslot's full path,
- <**command**> is the name of the Job, and
- <**arguments**> are the command's arguments.

Note

There are many other options that you can use when submitting your Job. See [Section 6, Submitting Jobs to Netbatch](#), for more details.

5.2 Resource Allocation Overview

A Pool contains one or more Queues, each of which contains one or more Qslots. A Qslot may contain sub-Qslots or User Slots (but not both). This arrangement of Queues, Qslots, and User Slots is known as the **Resource Allocation Structure**.

The Resource Allocation Structure is configured so that some parts of the hierarchy receive a larger share of available resources, while others receive a smaller share. This allows it to closely reflect the actual requirements of each team and project, and so best serve the needs of the organization as a whole.

A simple Resource Allocation Structure is shown in [Figure 18](#), below.

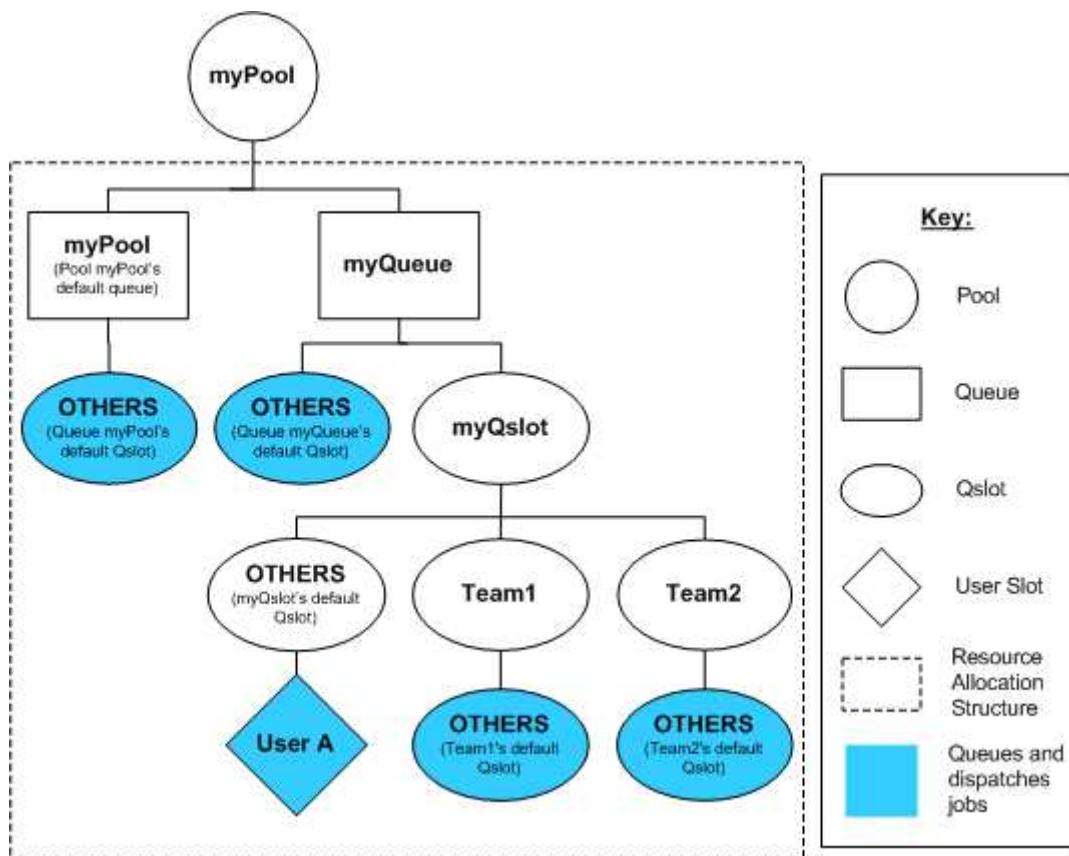


Figure 18: A Simple Resource Allocation Structure

In physical terms, a Netbatch Pool is a group of resources (Workstations), which are grouped into **Resource Sets**. Each of the Pool's Queues is assigned one or more Resource Sets.

Resource Sets can also be allocated to individual Qslots.

A Resource Set can be assigned to more than one Queue or Qslot.

Note

A resource (Workstation) may belong to more than one Resource Set, as shown in [Figure 19](#), below. (This diagram is the same as [Figure 1](#), and is reproduced here for clarity.)

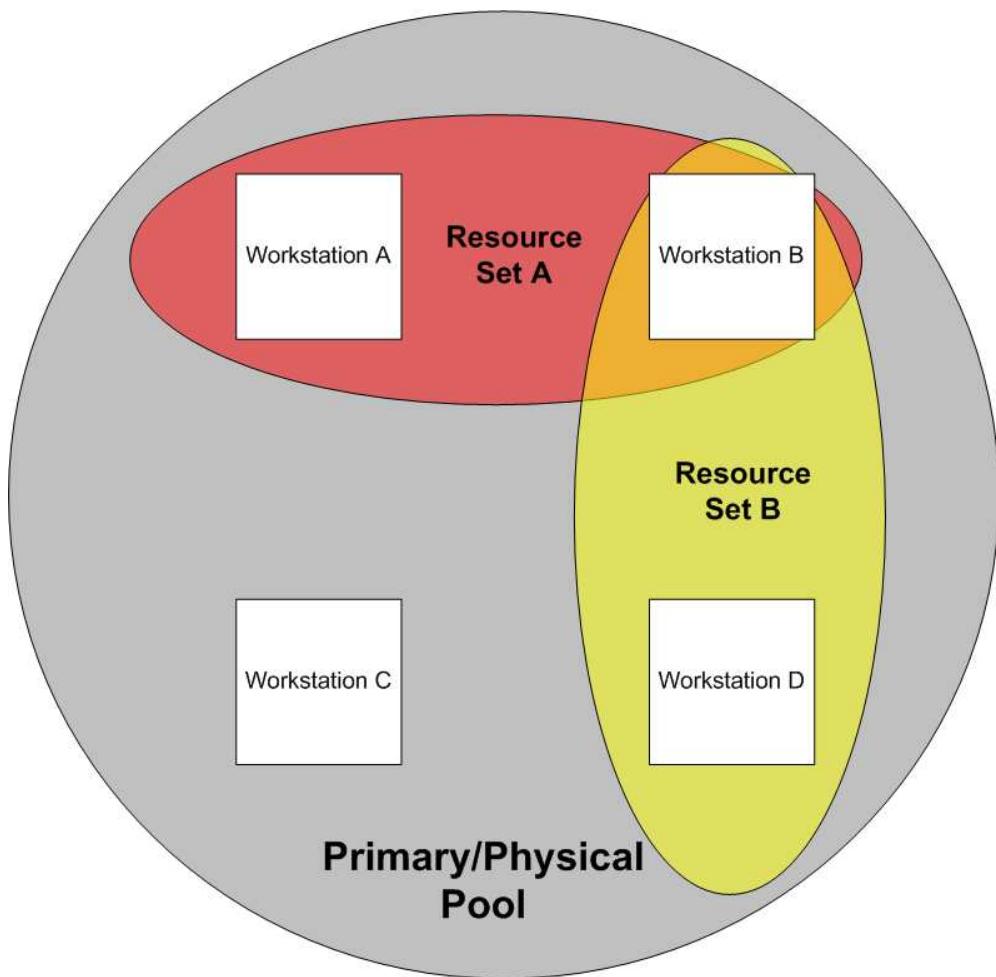


Figure 19: Resource Sets

A Job may be submitted to either:

- A Queue, or
- A Qslot

Note

Qslots have two functions:



- As a place for queuing and dispatching Jobs
- As a resource allocation unit

This means that Jobs are not only queued, but are also dispatched according to the priorities defined in the Resource Allocation Structure.

Note

Jobs are only queued in and dispatched from User Slots and default Qslots that have no User Slots—that is, the lowest elements, or "leaves," of the resource allocation structure (as shown in blue in [Figure 18](#), above).

The other Qslots (for example, myQslot in [Figure 18](#)) determine the allocation of resources to the Jobs in the leaves of the part of the tree below it (in this case, in Qslots Team1 and Team2).

Note

A User Slot is a special kind of Qslot that allows Netbatch to allocate resources on a per-user basis without needing a Resource Allocation Structure that reflects every single user.

5.2.1 Job Submitted to a Queue

If you submit a Job to a Queue *without specifying a Qslot*, the Job is queued in the Queue's default Qslot.

For Example

If you submit a Job to myQueue (see [Figure 18](#), above) without specifying a Qslot or User Slot, the Job is queued in OTHERS (myQueue's default Qslot).

5.2.2 Job Submitted to a Qslot

If you submit a Job to a Qslot that contains sub-Qslots *without specifying a sub-Qslot*, the Job is queued in the Qslot's default sub-Qslot or in the appropriate User Slot, if configured.

For Example

If you submit a Job to myQslot (see [Figure 18](#), above) without specifying a sub-Qslot or User Slot, the Job is queued in one of the User Slots in OTHERS (myQueue's default Qslot).



If you are User A (that is, your username matches the name of the User Slot), the Job is then queued in User A.

If you are NOT User A:

- *An ad-hoc User Slot is created (and the Job is queued there), or*
- *Your Job is queued in a single, shared User Slot, along with Jobs submitted by other users who are allowed to submit Jobs to this Qslot, but who are not User A.*

This depends on the way the Netbatch Administrator or Netbatch Resource Allocation Manager has set up the Resource Allocation Structure.

5.3 How Netbatch Decides Which Job to Run Next

Netbatch selects the next Job to be executed as follows:

1. Find the most eligible Queue.

Netbatch decides which Queue is most eligible, according to the rules defined in the resource allocation structure (see [Section 5.3.2, Resource Allocation Methods](#), below).

2. Find the most eligible Qslot or User Slot.

Netbatch examines the hierarchy of Qslots and User Slots in the selected Queue, and decides which "leaf" Qslot or User Slot is most eligible according to the rules defined in the resource allocation structure (see [Section 5.3.2, Resource Allocation Methods](#), below).

3. Find the most eligible Job.

Netbatch decides which Job in the selected Qslot/User Slot is most eligible according to the rules of Job Precedence (see [Section 5.3.1, Job Precedence](#), below).

If this Job cannot run because no suitable Workstation is available **and** it is a preemptive Job (and there is a running Job that it can preempt), Netbatch preempts the running Job so that this Job can run.

This marks the end of this scheduler cycle.

4. For the most preferred Resource Set/Group of the most eligible Qslot (or of the User Slot's parent Qslot if this is a User Slot), find the most available Workstation. The most available Workstation is with one with the lowest load (that is, the one with the fewest processes competing for its CPU time at a given moment) or WSRank (see [Section 11.2.27, Configuring Scheduling Based on Availability of Multiple Resources](#)).



5. If the most available Workstation can accept a Job, check if it can run the most eligible Job:
 - a) Check if the Workstation can ever run the Job. (That is, check whether the Workstations matches the Job's static class requirements and whether the Job's dynamic requirements are within the Workstation's limits. For example, if the Job requires 3GB of free virtual memory, check that the Workstation has at least 3GB *total* virtual memory.)
 - b) Perform resource reservation (if any). (Even if the Job cannot run on the Workstation at the moment, it may be able to reserve resources on it.) See [Section 2.3.3, Resource Reservation](#) for more information.

Note

*Starting with Netbatch 8.0.0, Jobs no longer hold reservations on a particular Workstation for more than one scheduler cycle. In the new **Stateless Reservation** scheme, at the end of each cycle, all reservations are released and reserved anew in the next cycle.*

This gives new, more important Jobs access to the resources that they need and also stops Jobs that reserve resources from waiting longer than they have to.

- c) Check if the Job can run on the Workstation right now. (That is, check against the Job's dynamic class requirements.)
If the Workstation cannot satisfy the Job's class requirements, Netbatch tries the next most available Workstation, then the next, and so on until it finds one that does. If none of the available Workstations can satisfy the Job's class requirements, it moves on to the next most eligible Job.
- d) Make sure the Job has any licenses that it needs.

If all of these conditions are met, dispatch the Job we found in Step 3 to the Workstation.

If the most available Workstation in the Qslot's most preferred Resource Set/Group cannot accept any Jobs, find the most available in the *Qslot's next preferred Resource Set/Group*, and so on.

If no Workstation from any of the Resource Sets/Groups that the Qslot can use can accept the Job, find the most eligible Job in the *next most eligible Qslot in the Queue*, and so on.

If none of the Workstations from any of the Resource Sets/Groups that the Qslots in the Queue can use can accept Jobs, find the most eligible Job in the most eligible Qslot in the *next most eligible Queue*, and so on.

For Example

In **Figure 20**, below, if there are Jobs waiting in both QS1 and QS2, and QS1 is more eligible, then Netbatch will first look for the most available Workstation in Resource Group RG1 to run the most eligible Job in QS1.

If the most available Workstation in RG1 cannot accept Jobs, it then looks in RG2. If the most available Workstation in RG2 cannot accept Jobs, it looks for the most available Workstation in the Pool to run the most eligible Job in QS2.

If the most available Workstation in the Pool cannot accept Jobs, Netbatch starts the process all over again.

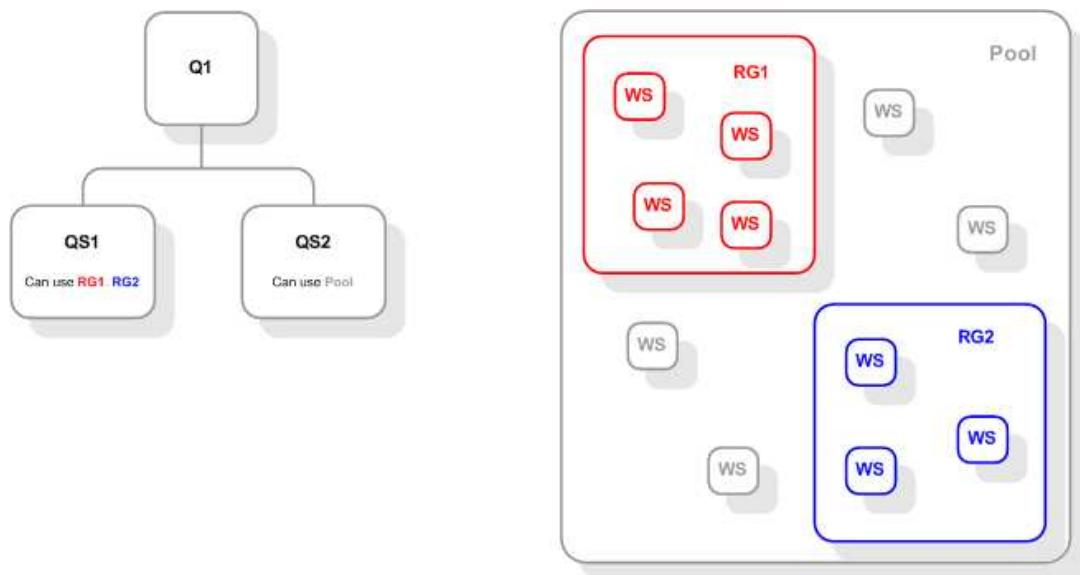


Figure 20: Scheduling Example

Note

Netbatch decides which resource (Workstation) is most available by comparing Workstation loads (that is, the number of processes competing for CPU time).

However, this can give a misleading picture, as different Workstations have different capabilities (i.e., processor speed, number of processors), so the load on its own is not a good basis for comparison.

Netbatch's Workstation Load Normalization feature (which must be configured by the administrator) takes these different capabilities into account, producing normalized load data that are more meaningful.



(The Netbatch Administrator can also configure Netbatch to take the availability of other resources into account when deciding which Workstation is most available.)

Note

If the most eligible Job was submitted to a Queue that has Workstation capability requirements defined for it, then in certain situations, the most available Workstation cannot accept the most eligible Job; instead it will accept a less eligible Job.

For example, if the Queue that the less eligible Job was submitted to has a memory requirement of 1, and the Queue that the most eligible Job was submitted to has a memory requirement of 4, but the Workstation only has 2 units of memory available, then the Workstation cannot accept the most eligible Job, so it will accept the less eligible Job.

5.3.1 Job Precedence

Within each User Slot and "leaf" Qslot (that is, a Qslot that does not have any sub-Qslots), Netbatch decides which Job is most eligible for execution based on the following factors:

- **Which Job was submitted first**—That is, Qslots and User Slots work on a FIFO basis.
- **Job Priority**—You specify this when you submit a Job using the `nbjob run` command.
- **Job Cost**—A dynamically-calculated value based on:
 - ◆ How much time has passed since the Job was submitted
 - ◆ The type of resource that the Job requires
 - ◆ The license(s) (if any) that the Job requires (and the configured cost of each license)

Higher-priority Jobs are more eligible for execution than lower-priority Jobs.

Higher-cost Jobs are more eligible for execution than lower-cost Jobs.

If two Jobs have the same cost and the same priority, the one that was submitted first is considered more eligible for execution.



5.3.2 Resource Allocation Methods

Netbatch supports three different methods of allocating resources:

- **Priority Allocation**—Jobs in a higher-Priority Queue or Qslot are always dispatched for processing before Jobs from a lower-priority Queue or Qslot (unless the Jobs in the higher-priority Queue/Qslot require resources that are unavailable).
- **Fair-Share Allocation**—Each Queue and Qslot receives a share of the available resources according to its **Weight**.

For example, if a Queue has two Qslots, one (Q1) with a weight of 10 and one (Q2) with a weight of 100, then ten Jobs will be dispatched for processing from Q2 for every Job dispatched from Q1.

- **FIFO (first-in first-out) Allocation**—Jobs in the sub-tree (below the Queue or Qslot where FIFO is specified as the scheduling method) are dispatched for execution on a first-come first-serve basis.

Note

The weight or priority of a Qslot (or Queue or User Slot) is only meaningful when compared to the weight or priority of other Qslots (or Queues or User Slots) at the same level and under the same parent (that is, "sibling" Qslots).

For Example

In Figure 21, below, a common mistake would be to think that Qslot3 would get more resources than any of the other Qslots, as it seems to have the highest weight. (This example uses Fair-Share Allocation, but the same principle applies to Priority Allocation.)

However, both Qslot1 and Qslot2 would get more resources than Qslot3, because they are in a Queue that has a higher weight.

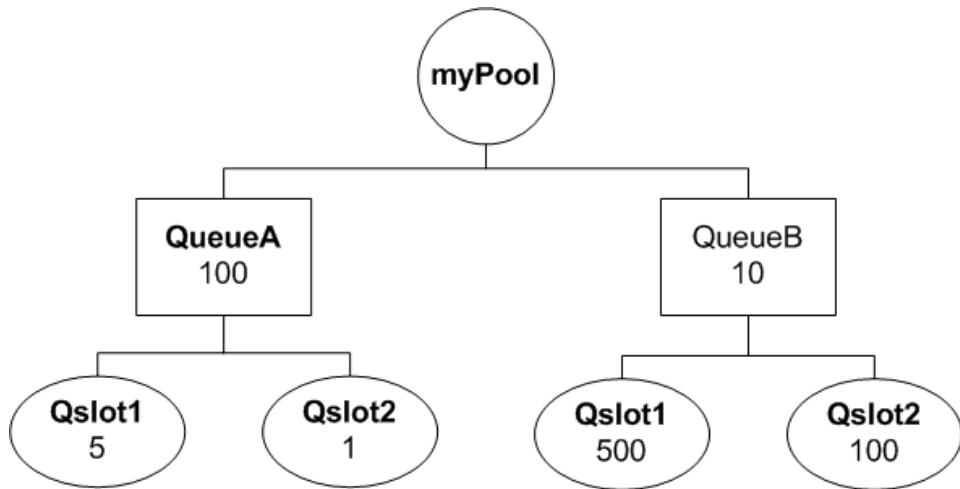


Figure 21: Comparing Qslot Weights (or Priorities)

Note

Different parts of the resource allocation structure may use different resource allocation methods.

For Example

In Figure 21, above, QueueA might use Priority Allocation and QueueB might use Fair-Share Allocation.

5.3.3 Preemption

Netbatch allows Queues and Qslots to be designated as **Preemptable** or **Preemptive**.

What this means is that Netbatch will suspend a Job that was dispatched from a preemptable Queue or Qslot so that the Workstation can run a Job from a preemptive Queue or Qslot.

Note

This only happens if:

- *The preemptive Queue or Qslot is the most eligible in the Resource Allocation Structure, AND*
- *No other resource is available to process the preempting Job.*

**Note**

If the Netbatch Administrator has configured custom-defined Workstation capabilities and Queue weighting, a Job from a preemptive Queue may require multiple Jobs to be preempted (that is, killed, suspended, or resubmitted).

Note

If a parallel Job is preempted, the appropriate action is applied to the whole parallel Job; that is, the master Job and all its slave Jobs.

5.4 Deciding Where to Submit a Job

The Resource Allocation Structure is usually set up to reflect the needs of projects and teams. As a user, you usually only have one Qslot that you are allowed to submit Jobs to for each project that you are part of.

For Example

If you are working on two projects—Project A and Project B—you might be allowed to submit Jobs to:

- *Qslot Team1 in Queue ProjectA*
- *Qslot Team6 in Queue ProjectB.*

You should submit all your Jobs that relate to Project A to /ProjectA/Team1, and all your Jobs that relate to Project B to /ProjectB/Team6.

However, it could be that for a particular project, there is more than one Qslot that you are allowed to submit Jobs to. In this case, to ensure that your Job is executed as quickly as possible, you should submit it to the Qslot that is most eligible, according to the rules explained above.

Before you can figure out which Qslot is most eligible, you must first find out:

- Which Pool(s) you can use
- Which Queue(s) you can use
- Which Qslot(s) you can use

Once you have found out this information, you can find out which Qslot is most eligible by comparing their weights and/or priorities (and the weight/priority of each one's parent Queue/Qslot, and so on up to the root of the Resource Allocation Structure).

**Note**

A Queue, Qslot, or User Slot can be either **Closed** or **Open**. If it is closed, you cannot submit Jobs to it.

If a Queue, Qslot, or User Slot is closed, all its sub-Qslots and User Slots are also closed. (Note that a Queue or Qslot's default Qslot can be closed, while its sub-Qslots and User Slots remain open.)

A Queue, Qslot, or User Slot can be either **Active** or **Inactive**.

If a Queue, Qslot, or User Slot is inactive, all its sub-Qslots and User Slots are also inactive. (Note that a Queue or Qslot's default Qslot can be inactive, while its sub-Qslots and User Slots remain active.)

Inactive Queues, Qslots, and User Slots can still accept Jobs!

Use `nbstatus qslots` to check states. (See [Section 5.4.1, Using nbstatus qslots to View Resource Allocation Structure Details, below](#).)

The following section explains in greater detail how to use the `nbstatus qslots` command to:

- Understand how resource allocation is set up on the Pool that you intend to use
- Get the information that you need to figure out which Queue and which Qslot is most eligible
- Check the status of the Queue/Qslot that you want to use (to make sure that it is open and active)

5.4.1 Using `nbstatus qslots` to View Resource Allocation Structure Details

The `nbstatus qslots` command is used to view details of the Queues, Qslots, and User Slots in a Pool. This information can be viewed in a number of different ways (depending on the options specified).

[Appendix A.3.22](#) lists all the possible `nbstatus qslots` options. The following subsections explain how to specify display options and filters to get the information that you need.

**Note**

To view nbstatus qslots output for a specific Qslot that is not a direct child of a Queue, you must specify the Qslot's full path (separated by slashes).

5.4.1.1 Specifying the Fields to Be Displayed

By default, **nbstatus qslots** displays the following fields:

- **Name**—Qslot name
- **Alias**—Qslot alias
- **Priority**—priority (for priority scheduling)
- **Allocation**—allocation (for fairshare scheduling)
- **ShouldGet**—amount of resources the Qslot should be getting
- **GettingNow**—amount of resources the Qslot is getting now

Use the **--fields** switch to specify the fields that you want to display:

```
nbstatus qslots --fields "[*,]<field>[,...]"|all]
```

Here:

- * means all the default fields (as listed above).
- **field** is one of the available fields (see below).
- **all** means all available fields.
- The available fields are as follows:
 - ◆ **Name*** - Qslot name
 - ◆ **FullPath** - Full Qslot path
 - ◆ **Queue** - The name of the Queue to which the Qslot belongs
 - ◆ **Alias*** - Qslot alias
 - ◆ **Priority*** - Priority (for priority scheduling)
 - ◆ **Allocation*** - Allocation (for fairshare scheduling)
 - ◆ **ShouldGet*** - Amount of resources the Qslot should be getting
 - ◆ **GettingNow*** - Amount of resources the Qslot is getting now
 - ◆ **MaxRunning** - Maximum permitted number of running Jobs



- ◆ **MaxWaiting** - Maximum permitted number of waiting Jobs
- ◆ **ShouldGetPerLevel** - The resources that the Qslot should be receiving as a percentage of the total resources that it and its "sibling" Qslots should be receiving
- ◆ **GettingNowPerLevel** - The resources that the Qslot is receiving as a percentage of the total resources that it and its "sibling" Qslots are receiving
- ◆ **Running** - Number of running Jobs
- ◆ **Waiting** - Number of waiting Jobs
- ◆ **Dispatched** - Number of Jobs dispatched
- ◆ **LastDispatched** - The date/time the last Job was dispatched
- ◆ **LastSubmitted** - The date/time the last Job was submitted
- ◆ **Status** - Qslot state (**open** or **closed**, **active** or **inactive**)
- ◆ **Type** - Qslot type:
 - ◆ **root** - The root of the resource allocation structure
 - ◆ **queue** - Queue
 - ◆ **qslot** - regular Qslot
 - ◆ **user** - User Slot
 - ◆ **/others** - default Qslot
- ◆ **InactiveUsers** - List of users for whom this Qslot has been locked by the Monitoring Service
- ◆ **Index** - Qslot index (for Netbatch 5 compatibility)
- ◆ **Level** - Qslot level (that is, how "deep" in the hierarchy the Qslot is). For example, **Level** for a Qslot with one "parent" Qslot and one "grandparent" Qslot is 4 (the four levels being the Queue, the grandparent, the parent, and the Qslot itself).
- ◆ **SchedulingMethod** - The scheduling method used:
 - ◆ **Priority**
 - ◆ **Fairshare**
 - ◆ **Complement fairshare**
 - ◆ **Summary fairshare**
 - ◆ **FIFO**
- ◆ **Permissions** - Users and groups allowed to submit Jobs to the Qslot



This is a space-delimited list of items of the format
`{<user>|<group>}:<type>:{grant|deny}:`
`{Recursive|nonRecursive}.`

`{<user>|<group>}` is the user/group to whom the permissions are granted/denied.

`type` is one of the following:

- ◆ `JobSubmit`
- ◆ `JobModify`
- ◆ `JobRemove`
- ◆ `JobAdmin`
- ◆ `QueueOpen`
- ◆ `QueueClose`
- ◆ `QueueActivate`
- ◆ `QueueInactivate`
- ◆ `QueueAdmin`
- ◆ `{grant|deny}` specifies whether the permissions are granted or denied.
- ◆ `{Recursive|nonRecursive}` specifies whether the permissions apply only to this Queue/Qslot (`nonRecursive`), or to this Queue/Qslot and all the Qslots under it (`Recursive`).
- ◆ **MemberResources** - The Resource Set(s)/Group(s) that are assigned to the Qslot
- ◆ **JobsSlots** - total number of Jobs from this Qslot that ran during the configured timeframe
Only Jobs that **started** running in the current timeframe are counted.
- ◆ **JobsSlotsShare** - number of Jobs run, expressed as a percentage of Jobs run in the specified context Qslot (if no context Qslot is specified, the Queue that the Qslot belongs to)
Only Jobs that **started** running in the current timeframe are counted.
- ◆ **WTime** - wall clock time (in seconds) accrued by the Jobs from this Qslot that ran during the configured timeframe
- ◆ **WTimeShare** - wall clock time (in seconds) accrued by the Jobs from this Qslot, expressed as a percentage of the wall clock time accrued by Jobs run in the specified context Qslot



(if no context Qslot is specified, the Queue that the Qslot belongs to)

- ◆ **CPUTime** - CPU time (in seconds) accrued by the Jobs from this Qslot that ran during the configured timeframe
- ◆ **CPUTimeShare** - CPU time (in seconds) accrued by the Jobs from this Qslot, expressed as a percentage of the CPU time accrued by Jobs run in the specified context Qslot (if no context Qslot is specified, the Queue that the Qslot belongs to)
- ◆ **AverageWaitTime** - average wait time (in minutes) for the Jobs from this Qslot that ran during the configured timeframe
- ◆ **ResourceUsageHistory** - the configured timespan for the Qslot in hours (if not configured, the default is 168 hours—seven days)

5.4.1.2 Specifying the Display Format

There are two ways to modify how the `nbstatus qslots` output is displayed:

- Add modifiers to specify column widths
- Add modifiers to perform mathematical functions on field values
- Specify a different output format

Adding Column Width Modifiers

If you specify a particular field in the argument to the `--fields` switch, you can specify how wide the column for that field should be:

```
nbstatus qslots --fields "<field>:[<func>]:[<width>]"
```

where:

- **field** is the name of the field.
- **func** specifies a mathematical function. This is explained in the next section.
- **width** specifies the display width.

For Example

```
nbstatus qslots --fields "name::30,permissions::80"
```

This displays the Name and Permissions fields for each Qslot in the default Queue.



*Note that we are not using any mathematical functions here, so **func** is omitted.*

Adding Modifiers to Perform Mathematical Operations

You can also add modifiers to the arguments to the **--fields** switch to perform mathematical operations on the values of a field:

```
nbstatus qslots --fields "<field>:[<func>]:[<width>]"
```

where:

- **field** is the name of the field.
- **func** specifies a mathematical function. It can be one of the following:
 - ◆ **sum** - Sum of all values
 - ◆ **min** - Minimum value
 - ◆ **max** - Maximum value
 - ◆ **avg** - Average of all values
 - ◆ **stdev** - Statistical standard deviation of all values
- **width** specifies the display width (as described in the previous sections).

Output consists of a single line of data

For Example

```
nbstatus qslots --fields "Running:sum" all
```

This displays the total number of Jobs in all Qslots and Queues. This is quite a useless example, because the number of running Jobs displayed for a Queue includes all the Jobs in all its Qslots.

A more useful example would be:

```
nbstatus qslots --fields "Running:sum"  
"Queue=='myQueue'"
```

This calculates and displays the total number of running Jobs for the Queue called myQueue.

Specifying an Output Format

By default, **nbstatus qslots** displays its output in columns. There are two other options:

- Block format—useful when you want to display a large number of fields



- Comma-separated values (CSV)—useful for exporting to Excel

To display output in one of these formats, add the `--format` switch:

```
nbstatus qslots --format {block|csv}  
[other options] [<specification>]
```

5.4.1.3 Specifying a Filter

You can add an arbitrarily complex boolean expression (or `all`) as the final argument of an `nbstatus qslots` command to filter the Queues, Qslots, and User Slots that are displayed:

```
nbstatus qslots [other switches] [<specification>]
```

`all` specifies that all Queues, Qslots, and User Slots should be displayed.

A boolean expression can consist of `<attribute>`s, `<operator>`s, and `<value>`s, where:

- `attribute` is a valid field (see the list of fields in [Section 5.4.1.1](#), above).
- `operator` defines the relationship between the attribute and the value. `operator` can be one of the following:
 - An arithmetic operator: `+`, `-`, `*`, `/`, `%`, unary `+` (e.g., `"+10"`), or unary `-` (e.g., `"-10"`)
 - A logical operator: `&&`, `|`, or `!`
 - A comparison operator: `<`, `<=`, `>`, `>=`, `!=`, or `==`
 - A non-strict operator: `is` or `isnt`
 - A bitwise operator: `&`, `|`, `~`, `^`, `<<`, `>>`, or `>>>`
 - A conditional operator: `?` or `:`
- `<value>` is a number, a string, or another boolean expression.

You can group expressions together using parentheses.

You can also use the `startswith` operator. It matches fields whose value starts with the specified string. Its syntax is as follows:

```
"startswith(<field_name>,'<string>')"
```

For Example

```
nbstatus qslots all
```

This displays the default set of fields for all the Queues, Qslots, and User Slots in the Pool.

**For Example**

```
nbstatus qslots "name=='/myQslot'&&queue=='myQueue'"
```

This displays the default set of fields for the Qslot called `myQslot`, which is an immediate child of the Queue called `myQueue`.

*Note that if you omit the `&&queue=='myQueue'` part, the **Qslot is not displayed**. (Unless the specification is all or you explicitly specify the Queue, only Qslots and User Slots in the default Queue are displayed.)*

For Example

```
nbstatus qslots "waiting>0&&queue=='myQueue'"
```

This displays the default set of fields for all the Qslots and User Slots in the Queue called `myQueue` that contain waiting Jobs.

For Example

```
nbstatus qslots --fields "*,level"  
"level==2&&queue=='myQueue'"
```

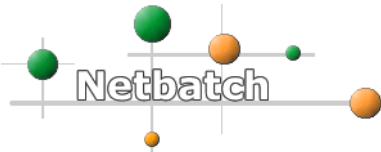
This displays the default set of fields plus the `Level` field for all the top-level Qslots in the Queue called `myQueue`.

(`Level` is 1 for Queues, 2 for top-level Qslots, 3 for the immediate children of top-level Qslots, and so on.)

For Example

```
nbstatus qslots --fields "*,status"  
"status=='open:active'&&(type=='queue' ||  
type=='qslot')&&queue=='myQueue'"
```

This displays the default set of fields plus the `status` field for the Queue called `myQueue` and all its Qslots. (Note the use of parentheses.)



6 Submitting Jobs to Netbatch

This chapter covers the following topics:

- [Special Considerations](#)
- [Setting the Netbatch Configuration Directory](#)
- [Running Jobs on or from Windows Workstations](#)
- [Submitting a Job to Netbatch with nbjob run](#)
- [Working with Job Dependencies](#)
- [Jobs with Special Resource Requirements](#)
- [Jobs That Require Licenses](#)
- [Kerberos Authentication](#)
- [Submitting a Job with Resource Reservation](#)
- [Submitting a Parallel Job](#)
- [Submitting a Job Using VBatch](#)
- [Using Vlon to Run a Virtual Machine through Netbatch](#)
- [Using Job Profiles](#)
- [Interactive Batch Jobs](#)
- [Interactive Logon Sessions](#)
- [Terminal Jobs](#)
- [ClearCase Jobs](#)
- [Submitting a Job when Workstations Might not Have Access to Required Files](#)
- [Using Job Constraints](#)
- [Logging Job Resource Usage Data](#)
- [Specifying a Retry Timeout](#)
- [Conditionally Resubmitting a Job on Finish](#)
-



6.1 Special Considerations

6.1.1 Pools that Include Windows Workstations

Netbatch Pools can include Windows Workstations. When submitting a Job from a Windows Workstation or that will run on a Windows Workstation, there are a few things that you must do differently.

See [Section 6.3, Running Jobs on or from Windows Workstations](#).

6.1.2 Process Authentication Groups

Netbatch uses a Process Authentication Group (PAG) to track Jobs and the processes that they spawn. It adds a PAG to each Job that you submit (which is inherited by any processes that it spawns).

Some Jobs (such as `vsim`) require that Netbatch not suspend its child processes that change their process group ID (though it can still use its PAG to track them for accounting purposes).

For such Jobs, add a `--pag nosuspend` switch:

```
nbjob run [other options] --pag nosuspend <command>
<command arguments>
```

6.1.3 Jobs that Require LD_ASSUME_KERNEL to Have a Particular Value

When you submit a Job to Netbatch, it removes the `LD_ASSUME_KERNEL` environment variable (if set). If your Job requires that `LD_ASSUME_KERNEL` have a particular value, it will fail.

In this case, use a Job Starter to reset the value of `LD_ASSUME_KERNEL`—see [Appendix 6.4.16](#).



6.1.4 What to Do if you Belong to More than 16 Groups

If you belong to more than 16 NIS groups, you must specify the group(s) that will be applied to the Jobs that you submit to Netbatch.

To do this:

1. Set the **NB_WASH_GROUPS** environment variable so that it contains the groups to be applied to your Jobs (up to 15 groups may be specified; your primary group will automatically be applied):

```
setenv NB_WASH_GROUPS <group_name>[ , . . . ]
```

2. Do one of the following:

- Add the **--wash** switch to **nbjob run** every time you submit a Job.
- Set the **NB_WASH_ENABLED** environment variable:

```
setenv NB_WASH_ENABLED
```

Note

A Job's owner must belong to all of its wash groups according to the group map.

6.2 Setting the Netbatch Configuration Directory

For correct operation of all Netbatch commands, set the **NBCONF** environment variable to the location of the Netbatch configuration directory.

To specify the Netbatch configuration directory:

1. Ask your Netbatch Administrator where the Netbatch configuration directory is located.
2. Set the value of the **NBCONF** environment variable to this location.

If you do not set **NBCONF**, Netbatch assumes that the Netbatch configuration directory is at its default location of **/usr/local/lib/netbatch/conf**.

For Example

If you are using tcsh, and the Netbatch configuration directory is /usr/local/nb/conf, type:

```
setenv NBCONF /usr/local/nb/conf
```

**Note**

As the **NBCONF** directory rarely changes, you may want to have the **NBCONF** variable set in your Unix logon script.

6.3 Running Jobs on or from Windows Workstations

6.3.1 Before You Start

Before you can submit Jobs from a Windows machine or that will run on a Windows Workstation you must:

1. Install Netbatch on your Windows machine (if it is not already installed). See [Section 6.3.1.1](#).
2. Run **nbauth**. See [Section 6.3.1.2](#).
3. If necessary, set up a mapping file so that your Jobs have access to the files that they need to read from or write to while they are running. See [Section 6.3.1.3](#).
4. If necessary, disable mapping of some drives. See [Section 6.3.1.4](#).
5. If necessary, modify your scripts so that they do not use hard-coded drive letters or paths. See [Section 6.3.1.5](#).

Once you have carried out these steps, you can start submitting Jobs, as explained in See [Section 6.3.2](#).

6.3.1.1 Ensuring that the Netbatch Binaries Are Installed

Check whether the Netbatch is installed on your Windows machine:

1. Open a command shell. (Click **Start | Run**, type **cmd**, and press **Enter**.)
2. Type **nbjob run**. If the following is returned, it means that the binaries are not installed:
**'nbjob' is not recognized as an internal or external
command, operable program or batch file.**

If Netbatch is not installed, install it as follows:

1. Download the latest installer (**WindowsWSM_<version>_<build>.exe**) from:
<http://www.iil.intel.com/swiss/netbatch/dist/netbatch/windows/>
2. Double-click the downloaded file to extract it. (It is a self-extracting zip file.)



3. Open a Command Prompt **as an administrator**.
4. Navigate to the folder containing the extracted files.
5. Run the installer by typing **cliInstaller.exe** with any required switches.
(Run it with the **--help** switch to list the available switches.)

Note

You can usually install Netbatch for Windows by running this command with just the one mandatory switch to specify the configuration server (which is usually the Pool Master):

```
cliInstaller.exe --confserver <hostname>
```

*where **hostname** is the fully-qualified host name of the configuration server.*

Switches are as follows:

- **--add_path {yes|no}**

Specifies whether to add the Netbatch executables to the system PATH.

(Default: **yes**)

- **--add_registry {yes|no}**

Specifies whether to add Netbatch entries to the registry.

(Default: **yes**)

- **--authserver <hostname>**

Specifies the host name of the machine running the Authentication Service.

(Default: <**confserver**>)

Note

You can change this later by editing the following line in <targetdir>\conf\WSMService.conf:

```
java.jvm.arg=-  
Dnetbatch.authservice.host=<hostname>
```



- **--conf <path>**

Specifies the full path (folder and filename) where the service configuration file will be located.

(Default: <confdir>\WSMService.conf)

- **--confdir <path>**

Specifies the folder where the Netbatch configuration will be located.

(Default: <targetdir>\conf)

- **--confserver <hostname>**

Mandatory. Specifies the hostname of the configuration server (mandatory).

(Default: none)

Note

You can change this later by editing the following line in <targetdir>\conf\WSMService.conf:

```
java.jvm.args=
Dnetbatch.confservice.host=<hostname>
```

- **--displayname <name>**

Specifies the display name for the Netbatch service (as listed together with <servicename> in Windows Services).

(Default: Netbatch Windows WSM)

- **--help**

Displays the Help, which lists the available switches with their descriptions.

- **--install {yes|no}**

Specifies what to install:

- **yes** installs the Netbatch WSM and the Netbatch Authentication Service and adds the binaries to the PATH and to the firewall.
- **no** installs just the binaries but not the WSM and adds the binaries to the PATH and to the firewall. This lets you run Jobs from the machine, but Jobs cannot run on it.

(Default: yes)



- **--javahome <path>**

Specifies the location of the folder where Java is installed. Only use this if Java is already installed on the machine.

(Default: Netbatch uses its own, local version of Java, which the installer puts in <targetdir>\jre.)

- **--local_dir <path>**

Specifies the local directory that the installer copies build files to. By default, persistency and directories are also put in here.

(Default: C:\Netstar.)

- **--persistencydir <path>**

Specifies the folder where Netbatch will store its persistency data.

(Default: <local_dir>\persist)

- **--service_logfile <path>**

Specifies the full path (including folder and file name) of the WSM log file.

(Default: <local_dir>\wsm.service.log)

- **--servicename <name>**

Specifies the name of the Netbatch service (as listed together with <displayname> in Windows Services).

(Default: netbatch)

- **--start_nbauthservice {yes|no}**

Specifies whether the Netbatch service should be started when the installation completes.

(Default: yes)

- **--startup {auto|demand}**

Specifies whether the WSM service is started automatically or on-demand.

(Default: auto)

- **--targetdir <path>**

Specifies the folder into which Netbatch will be installed.

(Default: the folder where the installer is located.)

- **--uninstall**

Uninstalls the WSM.



- **--verbose**

Enables verbose output.

This installs the WSM and the Netbatch binaries. It installs the WSM as a service that starts automatically, and starts it. It can be started and stopped using the Control Panel's Services applet (or on the command line using the `net.exe` tool).

To remove the WSM from the machine, type:

```
cliInstaller.exe --uninstall
```

Note

If the Windows WSM has problems connecting to the Pool Master (the Workstation is shown as Disc (disconnected) in `nbstatus workstations`), you may have to modify the Windows hosts file. To do this:

- i) Open `C:\Windows\System32\Drivers\etc\hosts` in Notepad as administrator. (Press the Windows key, search for Notepad, then right-click Notepad and select Run as administrator. Then open the file.)
- ii) Add a line in the following format:

```
<IP address> <hostname>
```

For example:

```
10.184.190.239 ivnc0099
```

(You can find out the Pool Master's IP address and host name from the `IP` and `Host` fields in `nbstatus pools`.)

- iii) Save the file.
- iv) Close Notepad.

6.3.1.2 Running nbauth

Before you can submit Jobs that will run on Windows Workstations, you must run `nbauth` (once initially and then again **every time you change your Windows password**).

This stores your Windows username and password so that it will be available to the Windows Workstations on which your Jobs will run.



1. In either Windows or UNIX, type:
nbauth
2. When prompted, enter your Windows username and password.
3. To enter additional username/password pairs, repeat Step 2. To finish, press **Enter** at the **Login** prompt.

6.3.1.3 Setting Up a Path Mapping File

If your Job needs to be able to read from or write to a file, then the file must be accessible by the Workstation that the Job runs on.

Jobs that run on Windows Workstations that need to be able to access files must do so through a drive letter. Such files are typically located on:

- A Windows share (for example,
\\\ger.corp.intel.com\ec\users\daves)
- An NFS file server, available through Samba (for example,
\\\isamba\nfs\iil\iec\sws\work\daves)

Users can use a **path mapping file** to specify how paths are mapped to drive letters.

Users can do one of the following:

- Use your own path mapping file, as described below.
- Use the global path mapping file, provided by the Netbatch Administrator (if such a file exists). To do this, simply leave the **NBNTPATHTABLE** environment variable unset.

To use your own path mapping file:

1. Create a file containing the following:

NB6

```
<UNC_path> {<letter>|*} <property>=<value>[ ...]  
[ ...]
```

where:

- **UNC_path** is the UNC path of the Windows/Samba share.
- **letter** is the drive letter (or * for the next available drive letter).
- **property** is one of the following:
 - ◆ **MUST_MOUNT**—mount is forced, even if there are no references in **env/cmdline/job**.



- **IMPORT**—the UNIX path
 - **LOGIN**—use the specified login to mount this UNC.
 - **value** is the value assigned to the corresponding **property**.
2. Save the file in a location that can be accessed by any Windows Workstation (i.e., on a Windows or Samba share).
 3. Set the value of the **NBNTPATHTABLE** environment variable to the location of the file.

This will vary depending on where you are submitting your Windows Jobs from. In Windows, it can be a UNC path to a Windows or Samba share. .

When a Windows Job is dispatched for execution, the specified path(s) is mapped to the specified drive letter(s). When the Job ends, the path is unmapped.

For Example

You could create a mapping file that looks like this:

NB6

\\\isamba\\nfs\\home\\daves T MUST_MOUNT=true

This specifies that the specified Samba share is to be mapped to the drive letter T:. Jobs can then access the files in this location via T:.

6.3.1.4 Disabling Drive Mapping

When Netbatch logs on to a Windows Workstation to run a Job there, by default it maps all of your mapped drives. This can take a long time.

To avoid this problem, disable mapping for all drives other than those required by the Job. To do this, set the value of the **NB_NOMOUNT** environment variable.

For Example

*To disable mapping for drives X: and Y:, set the value of **NB_NOMOUNT** to:*

X,Y

For Example

*To disable mapping for all drives, set the value of **NB_NOMOUNT** to:*



6.3.1.5 Checking Your Scripts

Using hard-coded drive letters or UNC paths in your scripts can cause problems. (For example, if Netbatch dispatches two such Jobs to the same Workstation, and they both try to use the same network share, one Job will fail.)

The solution is to replace drive letters and UNC paths with environment variables that hold the drive letters or paths, and then set these environment variables at the beginning of the script.

For Example

A script that contains the following lines will not run correctly through Netbatch:

```
\FS1\abc\xyz\mytool.exe  
\FS1\abc\xyz\myfile.dat  
P:\iA64\simulators\sphinx.exe
```

(where P: = \FS2\bla.)

These paths must be replaced with environment variables, as follows:

```
%FS1%\xyz\mytool.exe  
%FS1%\xyz\myfile.dat  
%FS2%\iA64\simulators\sphinx.exe
```

Then, at the beginning of the script, set the environment variables:

```
Set FS1=/share1  
Set FS2=P:          (if running the script interactively)  
Set FS2=/share2    (if running the script as a Netbatch Job)
```

You must also add the appropriate lines to your path mapping table file (see [Section 6.3.1.3](#), above, for more details):

```
FS1  abc    /share1    *    NTFS  
FS2  bla    /share2    *    NTFS
```



6.3.2 Submitting Jobs

If the Netbatch command binaries have been installed on your Windows machine, you can submit, track, and manipulate Jobs from it.

Simply open a command shell (click **Start | Run**, type `cmd`, and press **Enter**), then type a Netbatch command.

There are three possible scenarios, which are covered in the following sections:

- Submitting a Windows Job from a Windows Workstation—see [Section 6.3.2.1](#)
- Submitting a UNIX Job from a Windows Workstation—see [Section 6.3.2.2](#)
- Submitting a Windows Job from a UNIX Workstation—see [Section 6.3.2.3](#)

(Submitting a UNIX Job from a UNIX machine is the standard case, and so no special actions are required.)

Note

A Pool that includes both Windows and UNIX machines will usually use a different class for each OS. Use these classes to ensure that your Job runs on the OS that you want it to.

6.3.2.1 Submitting a Windows Job from a Windows Workstation

When you submit a Windows Job from a Windows Workstation, you must:

1. Set your **NBWD** environment variable in Windows or use the `--work-dir` switch when submitting Jobs.
2. Explicitly specify the log file path with the `--log-file-dir` switch.

If you do not do this, Netbatch will not be able to write the Job log file and so the Job will fail.

Note

If you only ever submit Windows Jobs from Windows, you only need to set NBWD once.

**Note**

*On Windows, **NBWD** must contain a single Windows path, and nothing else. Multiple values are not supported.*

If the Job needs to read from or write to files, you must also set up a mapping file so that the Job will be able to access the files from the Windows Workstation that it will run on. See [Section 6.3.1.3](#).

6.3.2.2 Submitting a UNIX Job from a Windows Workstation

When you submit a UNIX Job from a Windows Workstation, you must:

1. Set your **NBWD** environment variable in Windows or use the **--work-dir** switch when submitting Jobs.
2. Explicitly specify the log file path with the **--log-file-dir** or **--log-file** switch.

If you do not do this, Netbatch will not be able to write the Job log file and so the Job will fail.

Note

If the Job needs to read from or write to files, you must submit it from UNIX, not from Windows.

Note

*If you only ever submit UNIX Jobs from Windows, you only need to set **NBWD** once.*

For Example

Set **NBWD** to: **/nfs/iil/iec/sws/work/martin/**.

Submit the Job by typing:

```
nbjob run --log-file
/nfs/iil/iec/sws/work/martin/log.txt
--class unix sleep 10
```



6.3.2.3 Submitting a Windows Job from a UNIX Workstation

When you submit a Windows Job from a UNIX Workstation, you must:

1. Set your **NBWD** environment variable in UNIX or use the **--work-dir** switch when submitting Jobs.
2. Explicitly specify the log file path with the **--log-file-dir** switch.

If you do not do this, Netbatch will not be able to write the Job log file and so the Job will fail.

Note

If you only ever submit Windows Jobs from UNIX, you only need to set NBWD once.

Note

NBWD may not include both UNIX and Windows paths.

If the Job needs to read from or write to files, you must set up a mapping file so that the Job will be able to access the files from the Windows Workstation that it will run on. See [Section 6.3.1.3](#).

Note

The directory that you set as the value of NBWD must be one that the Windows Workstation can cd to. So a Samba directory cannot be used.

6.4 Submitting a Job to Netbatch with nbjob run

Submitting an application to run as a Netbatch Job is exactly the same as running the application from the command line, except that the application's command is preceded by **nbjob run** and other optional Netbatch command-line options.

The general syntax of **nbjob run** is:

```
nbjob run [options] command [parameters]
```

where:

- **options** are nbjob run switches.
- **command** is the command to be run through Netbatch.



- **parameters** are the command's parameters.

By default, Netbatch dispatches a Job along with your local environment variables and sends the output file to your current working directory.

Note

If Jobs that you submit repeatedly fail with certain exit codes, Netbatch stops dispatching your Jobs from the Qslot to which you submitted them for a predetermined period, and sends you mail informing you of this (if the Netbatch Administrator has enabled this feature). Netbatch may also suspend other Jobs that you submitted to the same Qslot. (You will still be able to submit Jobs to the Qslot, but they will not be dispatched for execution.)

If this happens, it is usually a sign that something is wrong with the Jobs you are submitting. Once you have fixed the problem, you can use the following commands view the actions that Netbatch has taken as a result of such Job failures and to unlock the Qslot so that it will start dispatching your Jobs for execution again.

To view the actions that Netbatch has taken, type:

```
nbstatus monitor-actions --target <pool_name>  
"User=='<username>'"
```

where username is your username.

To reverse these actions, type:

```
nbadmin monitor-actions reset  
--target <pool_name> "Qslot=='<Qslot>'"
```

where Qslot is the name of the Qslot to which you submitted the problem Jobs.

The following sections describe **nbjob run** command-line options:

- [Specifying a Pool, Section 6.4.1](#)
- [Specifying a Queue, Section 6.4.2](#)
- [Specifying a Qslot, Section 6.4.3](#)
- [Specifying the Workstation Class, Section 6.4.5](#)
- [Email Options, Section 6.4.6](#)
- [Job Output \(Log File\) Location and Options, Section 6.4.7](#)
- [Suspended Job Options, Section 6.4.8](#)



- [**Specifying a License, Section 6.4.9**](#)
- [**Job Environment Variables, Section 6.4.11**](#)
- [**Job Execution Limits, Section 6.4.12**](#)
- [**Job Hung Limits, Section 6.4.13**](#)
- [**Specifying a Job Pre-Execution Utility, Section 6.4.14**](#)
- [**Specifying a Job Post-Execution Utility, Section 6.4.15**](#)
- [**Specifying a Job Starter, Section 6.4.16**](#)
- [**Specifying the Job Priority, Section 6.4.17**](#)
- [**Specifying a Task Name, Section 6.4.18**](#)
- [**Submitting Groups of Jobs, Section 6.4.19**](#)
- [**Specifying Resource Limits, Section 6.4.20**](#)
- [**Protecting Jobs from Blackholes, Section 6.4.21**](#)
- [**Working with Job Dependencies, Section 6.5**](#)
- [**Jobs with Special Resource Requirements, Section 6.6**](#)
- [**Submitting a Job with Resource Reservation, Section 6.9**](#)
- [**Submitting a Parallel Job, Section 6.10**](#)
- [**Interactive Batch Jobs, Section 6.14**](#)
- [**Terminal Jobs, Section 6.16**](#)
- [**Submitting a Job when Workstations Might not Have Access to Required Files, Section 6.18**](#)
- [**Using Job Constraints, Section 6.19**](#)
- [**Logging Job Resource Usage Data, Section 6.20**](#)
- [**Specifying a Retry Timeout, Section 6.21**](#)

**Note**

An `nbjob run` command-line option usually overrides an environment variable that has been set to specify the same parameter.

In most cases (but not all), it also overrides any default values that the Netbatch Administrator has set for the parameter.

See [Appendix B: Precedence of Configuration Directives, Environment Variables, Job Profiles, and Command-Line Switches](#).

6.4.1 Specifying a Pool

To find out which Pools exist in your site, and which Pools are available for your use you can either:

- Ask your Netbatch Administrator.
- Locate the file named `pools` in the **NBCONF** directory. The `pools` file contains a list of all of the Pools and Queues that are configured at your site, along with the hostnames of the Netbatch Pool Masters that host them.

Note

If you submit your Job to a Virtual Pool, you can explicitly specify the Physical Pool(s) in the Virtual Pool that you want to use. See [Section 6.4.4, Specifying a Virtual Pool](#), below.

All of the Netbatch commands require information that will indicate which Pool Master should be contacted. The best way to provide this information is by specifying the name of the Pool preceded by the `-P` switch.

For Example

```
nbjob run --target myPool echo hello
```

This command will submit the Job `echo hello` for execution by the Netbatch pool named `myPool`.

You can avoid typing the name of the Pool for each Netbatch command you invoke by setting the **NBPOOL** environment variable.

For Example

```
setenv NBPOOL myPool
```



Once the **NBPOOL** environment variable is set, you may omit the name of the Pool from all of your Netbatch commands.

For Example

```
nbjob run echo hello
```

This command will also submit the echo hello Job for execution by the Netbatch pool named myPool.

An alternate method of specifying the location of the Netbatch Pool Master is by providing the name of the host on which it is running, preceded by the **--target** switch.

For Example

```
nbjob run --target myPoolMaster.intel.com echo hello
```

This command will submit the Job echo hello for execution by the Pool Master on the machine identified by the name myPoolMaster.intel.com.

Note

*Specifying a Pool name or Pool Master hostname explicitly in a Netbatch command will override the Pool specification in the **NBPOOL** environment variable.*

If the **--target** switch is not used on the command line of a Netbatch command, the following search sequence will be performed:

1. Netbatch will check for a Pool name in the **NBPOOL** environment variable.
2. The configuration files in the **NBCONF** directory will be scanned for the most appropriate Pool.
3. Netbatch will issue an error message.

Note

The Netbatch team recommends that you do not specify the Pool by its Master machine hostname. The Master machine for a Pool can change.



6.4.2 Specifying a Queue

Use `--queue` to specify a Queue as follows:

```
nbjob run --target <poolName> --queue <queueName>
<myJob>
```

This tells Netbatch to send the Job `<myJob>` to the default Qslot in the Queue `<queueName>` of the Pool `<poolName>`.

To avoid having to type `--queue <queueName>` for many successive Netbatch commands, you can set the **NBQUEUE** environment variable as follows:

```
setenv NBQUEUE <queueName>
```

You can now send the Job `<myJob>` to the default Qslot in the Queue `<queueName>` of the Pool `<poolName>` with a shorter version of the command:

```
nbjob run --target <poolName> <myJob>
```

If you do not specify a Queue explicitly, Netbatch will submit your Job to the Pool's Default Queue.

Note

If you specify a class requirement that cannot be satisfied by any of the Workstations to which the Queue can dispatch Jobs, the Job is not Queued.

6.4.3 Specifying a Qslot

Use `--qslot` to specify a Qslot as follows:

```
nbjob run --target <poolName> --queue <queueName> --
qslot <qslot> <myJob>
```

If you do not specify a Qslot, Netbatch:

1. Looks for the Qslot with the highest numerical value (that is, its alias, or its name if the name is a number) that you have permissions to submit Jobs to.
2. If you do not have permissions to submit Jobs to any Qslot with an alias or numerical name, then Netbatch checks whether you have permissions to submit Jobs to the specified Queue's default Qslot.
3. If you do not, the Job cannot be queued.



If the Qslot that you specify does not exist, or if it does exist but you do not have permissions to submit Jobs to it, Netbatch:

1. Checks whether you have permissions to submit Jobs to the specified Qslot's parent Qslot. If not, it continues up the hierarchy to the parent's parent, then to its parent, and so on, until it reaches the top-level Qslot (that is, the Qslot whose parent is the Queue).
2. If you do not have permissions to submit Jobs to any of these Qslots, Netbatch looks for the Qslot with the highest numerical value (that is, its alias, or its name if the name is a number) that you have permissions to submit Jobs to.
3. If you do not have permissions to submit Jobs to any Qslot with an alias or numerical name, then Netbatch checks whether you have permissions to submit Jobs to the specified Queue's default Qslot.
4. If you do not, the Job cannot be queued.

Note

If the Netbatch Administrator has disabled the Qslot search feature, a Job with no specified Qslot (or a specified Qslot that does not exist or for which you do not have permissions) will not be queued.

Note

If you specify a class requirement that cannot be satisfied by any of the Workstations to which the Qslot can dispatch Jobs, the Job remains in the wait state indefinitely (and is never dispatched for execution).

For Example

The following command tells Netbatch to submit the Job sleep 100 to Qslot number 76, of the default Queue in the Pool linux_idc:

```
nbjob run --target linux_idc --qslot 76 sleep 100
```

The Qslot name can be either a numeric value or a path string.

For Example

```
nbjob run --target linux_idc --queue tapeout
--qslot /TMN/NORTH sleep 100
```

This instructs Netbatch to submit the Job sleep 100 to Qslot /TMN/NORTH in the Queue tapeout of the Pool linux_idc.

**Note**

On rare occasions, NIS problems prevent Netbatch from correctly ascertaining the UNIX groups that you belong to. If this happens, your Job may be queued in a Qslot other than the one you specified. In such cases, your Job will be queued in the Qslot with the highest numerical alias to which anyone may submit Jobs.

6.4.4 Specifying a Virtual Pool

When submitting a Job to a Virtual Pool, you can explicitly specify the Physical Pool(s) in the Virtual Pool that you want to use. To do this, use the **--remote-pools** switch (in addition to specifying the Virtual Pool with the **--target** switch):

```
nbjob run --target <vpool> --remote-pools
{<phys_pool>|<alias>}[,...]
```

where:

- **vpool** is the name of the Virtual Pool that you are submitting the Job to.
- **phys_pool** is the name of a Physical Pool in the Virtual Pool.
- **alias** is the alias assigned to one or more Physical Pools in the Virtual Pool.

The order determines the order in which the Pools will be tried. Netbatch will try the first Pool, and will only try the next one if the previous one has reached its **WaitJobsLimit** or **MaxJobsLimit**.

If an alias is specified, Netbatch uses its existing algorithm to decide which Pool to send the Job to. (Multiple Pools can have the same alias. For example, all the Physical Pools at the same site as the Virtual Pool Master might have the alias **local**.)

Alternatively, you can specify these Pools in the **NB_POOLS** environment variable. The **--remote-pools** switch overrides **NB_POOLS**.

Note

When submitting a Job to a Virtual Pool where there is a chance that the Job will be sent to a Pool at a remote site, you may have to enable file copying so that the Job will have access to the files that it needs. See [Section 6.18, Submitting a Job when Workstations Might not Have Access to Required Files](#).



6.4.5 Specifying the Workstation Class

Use the `--class` switch to specify the Workstation class as follows:

```
nbjob run --target <poolName> --class  
<booleanExpression> <command>
```

For Example

```
nbjob run --target <poolName> --class BIGMEM echo hello
```

This specifies that the Job echo hello can only run on Workstations of the BIGMEM class.

You can specify multiple classes with a simple boolean expression enclosed in quotation marks as follows:

```
nbjob run --target <poolName> --class "BIGMEM && RISC"  
command
```

This string specifies that the Job can run only on a Workstation that is both a **BIGMEM** and **RISC** class Workstation.

The `--class` switch can accept an arbitrarily complex boolean expression combining any number of static or dynamic resource requirements. For more information on specifying complex resource requirements, see [Section 6.6.4, Submitting a Job with Resource Requirements](#).

You can use the `nbstatus classes` command (see [Appendix A.3.1](#)) to find out which classes are supported by a Pool, Resource Set, or Queue, and to see how many Workstations support each class (and which Workstations they are):

- To see the classes supported by the Workstations in the Pool, type:

```
nbstatus classes
```

- To see the classes supported by the Workstations in a Resource Set, type:

```
nbstatus classes "resouceset='<resource_set>'"
```

where `resource_set` is the name of a Resource Set.

- To see the classes supported by the Workstations available to a Queue, type:

```
nbstatus classes "queue='<queue_name>'"
```

where `queue_name` is the name of a Queue.

**Note**

If you submit a Job to a Virtual Pool with a class requirement, the Job will be accepted and the Virtual Pool Master will check whether the class requirements can be satisfied by any of the eligible physical Pools in the background.

If they cannot be satisfied, the Job will not be dispatched and any Jobs that are subsequently submitted with the same class requirements will not be accepted. See [Section 3.2.1, How Netbatch Checks Class Requirements](#).

6.4.6 Email Options

Netbatch can be instructed to notify you via email when your Job begins executing, when it ends, or when it is suspended on a Workstation for a predetermined amount of time.

Table 3 lists the command line switches used for setting the email options.

Table 3: Email Options

Switch	Description
--mail S	Sends an email as the Job starts
--mail E	Sends an email as the Job ends
--mail no	Does not send any emails.
--mail <min>	Sends an email if the Job is suspended on the Workstation for more than the specified number of minutes

For Example

```
nbjob run --target <poolName> --mail "S 10" command
```

This command instructs Netbatch to send an email when the Job command starts running and if it is suspended on the Workstation for more than 10 minutes. For information on why Netbatch suspends Jobs, see: [Section 3.3.5, Suspended Jobs](#).

Note

The email address used for status reporting is derived from your login user name. Make sure that you have a .forward file in your UNIX home directory (~) to redirect mail if you are using Microsoft® Outlook as your primary email client.



6.4.7 Job Output (Log File) Location and Options

By default, Netbatch sends the standard output and error of a Job to your current working directory and names the file as follows:

```
##Date-Time#.<Workstation>
```

If more than one Job starts at the same time on the same Workstation, one file will be named in the above format, and for each of the other Jobs, the output file name will have a unique serial number appended to it.

To create a Job output file at a different location and/or with a different name, use the **--log-file** switch:

```
nbjob run --log-file <jobname> <command>
```

This Job Name appears in the output of the **nbstatus jobs** command (but only if you specify that the **Name** field is to be displayed).

To specify that the Job output file should be written to a different location, use the **--log-file-dir** switch:

```
nbjob run --log-file-dir <path> <command>
```

Note

If both --log-file-dir and --log-file are used and --log-file's argument includes a path, --log-file-dir is ignored.

Note

If you want to know whether a Job has been resubmitted (and how many times), use the --incremental-log switch.

With this switch, each time the Job is resubmitted, a new log file is created with an incremental suffix. (For example, if the original log file is called myjob.log, additional log files are called myjob.log.1, myjob.log.2, and so on.)

Note

If the WSM is down for some reason when the Job ends, writing the log file is not completed, as Netbatch does not have the required information. When the WSM comes back up, writing of the log file is completed.

**For Example**

```
nbjob run --log-file myJob command
```

*This string instructs Netbatch to send the output of the Job command to the file **myJob** in the current working directory.*

For Example

```
nbjob run --log-file /home/username/testdir/Joboutput  
command
```

*This string instructs Netbatch to send the output of the Job command to the directory **/home/username/testdir** and to name the file **Joboutput**.*

For Example

```
nbjob run --log-file-dir /home/username/testdir  
command
```

*This instructs Netbatch to send the output of the Job command to the directory **/home/username/testdir**. The file will have the standard name.*

For Example

```
nbjob run --log-file /tmp/Joboutput  
--log-file-dir /home/username/testdir command
```

*This instructs Netbatch to send the output of the Job command to the **/tmp** directory and to name the file **Joboutput**. The **--log-file-dir** switch is ignored..*

Caution

If you give the name of a file that already exists, the file will be overwritten.

Note

If the path contains any spaces, you must enclose it in quotation marks.

You may set the UNIX permissions of the Job's output file by setting the **NBUMASK** environment variable before submitting the Job. For example, you can set it so that other members of your group can overwrite your output files. See [Section 7.2.4, Job File Permissions](#).



6.4.8 Suspended Job Options

Netbatch will sometimes suspend a Job that is executing on a Workstation. See [Section 3.3.5, Suspended Jobs](#).

You may specify in advance what Netbatch should do if your Job is suspended on a Workstation.

Table 4 lists the available options in case your Job is suspended on a Workstation, along with the command line switches that activate the different options.

Table 4: Suspended Job Option Options

Switch	Description
--mail <min>	Sends an email if the Job is suspended for more than the specified number of minutes.
--kill <min>	Kills the Job if it is suspended more than the specified number of minutes, and sends an email message informing that the Job has been killed.
--resubmit <min>	Resubmits the Job to the same Pool if it is suspended more than the specified number of minutes. If the Resubmitted Job is suspended again, Netbatch kills it, and sends an email message informing that the Job has been killed. Note that this does not apply to Jobs that are resubmitted by the Netbatch Administrator or by a user with Queue/Qslot administrative rights (using nb.job suspend).
--trials <retry> --resubmit <min>	The combination of these switches tells Netbatch to resubmit the Job if it is suspended more than the specified number of minutes, up to a specified number of retries. If it is suspended once more, Netbatch kills the Job, and sends an email message. You can verify how many times a submitted Job has been restarted with the following command <code>nbstatus jobs --fields "*",TimesRestarted"</code> (see also: Section 4.3.2, Running Job Takes Too Long)
--resubmit-to-source	If you submit your Job to a Virtual Pool, you can use this switch to specify that if your Job is resubmitted, it is to be resubmitted to the Virtual Pool and not to the physical Pool.

For Example

```
nbjob run --target <poolName> --kill 10 <command>
```

This tells Netbatch to kill the Job `command` if it is suspended for more than 10 minutes and to send you an email to tell you that Netbatch killed the Job.

Another example:

```
nbjob run --target <poolName> --resubmit 10  
--trials 5 <command>
```



This tells Netbatch to resubmit the Job command to the same pool if it is suspended for more than 10 minutes.

The Job will be restarted as necessary up to a total of 5 times. If the Job is suspended again for more than 10 minutes, it will be killed, and you will receive an email message informing you that it has been killed.

6.4.9 Specifying a License

Some Jobs use other software tools while they are running. A license is a software vendor's permission to use a pre-determined number of instances of a software tool.

For more detailed information on license support see [Section 6.7, Jobs That Require Licenses](#).

The `--license` switch tells Netbatch that a submitted Job requires one or more licenses. You may specify complex license requirements with a boolean expression.

For Example

To specify that a Job requires either both `license1` and `license2` or more than one instance of `license3`, type:

```
nbjob run --target <poolName> --queue <queueName> --  
license "(license1 && license2) || license3 > 1" sleep  
100
```

The `--license` option may be used to specify either a feature name or a license alias. If you specify a feature name, the `NB_LICENSE_FILE` environment variable must be correctly defined. See: [Section 6.7.3, Submitting Jobs that Require Licenses](#).

6.4.10 Kerberos Authentication

Depending on your administrator's configuration, Netbatch provides the option of submitting Jobs with Kerberos authentication (`--kauth`). Prior to submitting a Job, acquire a Kerberos ticket-granting ticket (TGT), see [Section 6.8.2](#). This TGT is attached to a submitted Job and grants access to services that require authentication upon execution. A TGT has a predefined validity period. To renew your Kerberos credentials use `nbkrb renew`. This command updates Netbatch with the new Kerberos credentials, see [Section 6.8.3](#).

For Example

To specify that a Job uses Kerberos authentication, type:

```
nbjob run --target <poolname> --kauth on
```



6.4.11 Job Environment Variables

By default, Netbatch uses your local environment variables when it runs a Job. Use the `--no-env` switch to instruct Netbatch to run your Job without your local environment variables.

For Example

```
nbjob run --target <poolname> --no-env command
```

The `-E` switch does not override the Netbatch environment variables described in [Chapter 7](#).

6.4.12 Job Execution Limits

Note

The `--exec-limits` switch described here is deprecated. Use `--job-constraints` instead. (For example, if you add `--job-constraints "wtime>3600:mail:kill"`, Netbatch will kill the Job and mail you if it runs for more than an hour.)

Use **Job Execution Limits** to specify limits on the total execution time of a Job. Job Execution Limits include:

- **Soft Execution Limit**
- **Hard Execution Limit**

A Soft Execution Limit will cause Netbatch to send an email to the Job submitter when the Job runtime execution limit is exceeded.

A Hard Execution Limit is the maximum run time a Job can have before Netbatch kills the process, and cancels the Job. If this happens, the Job submitter is also notified by email.

Time when Jobs are stopped or suspended by Netbatch **does not** count towards the execution time.

Jobs that consume more CPU time than their assigned execution limits and perform no useful operations are defined as **Runaway Jobs**.

You can specify execution limits with the following methods:

- With `nbjob run` and the `--exec-limits` switch
- With the **NB_EXEC_LIMITS** environment variable



Use the following syntax for soft and hard Job Execution Limits:

```
<softexecetime>{s|m|h} : <hardexecetime>{s|m|h}
```

where **<softexecetime>** and **<hardexecetime>** are positive floating point numbers and {s|m|h} is one of three time units (seconds, minutes, or hours) that define the time limit. No spaces are allowed in the time limit specification.

Note

Execution limits are rounded up to the nearest minute.

For Example

```
nbjob run --target linux_idc --class memhog -Q /merced  
--exec-limits 45m:1.5h echo hello
```

This specifies a soft execution limit of 45 minutes, and a hard execution limit of 1.5 hours.

*Alternatively, the same execution limits can be defined in the **NB_EXEC_LIMITS** environment variable as follows:*

```
setenv NB_EXEC_LIMITS 45m:1.5h
```

6.4.13 Job Hung Limits

Note

*The **--hung-limits** switch described here is deprecated. Use **--job-constraints** instead. (For example, if you add **--job-constraints "deltaovertime('15m','utime+stime')<60:mail:kill"** Netbatch will kill the Job and mail you if it accumulates less than 60 seconds of execution time (**utime** plus **stime**) in any 15 minute period.)*

Netbatch users and administrators can specify time limits for detecting hung Jobs. A hung Job is a Job that does not accumulate any CPU time within a specified time window. As with execution limits, Job Hung Limits also include:

- **Soft Hung Limit**
- **Hard Hung Limit**

A Soft Hung Limit will cause Netbatch to send an email to the Job submitter when the Job does not accumulate any CPU time within the specified time.



A Hard Hung Limit is the maximum time a Job can run without accumulating any CPU time, before Netbatch kills the process. If this happens, the Job submitter is also notified by email.

Time when Jobs are stopped or suspended by Netbatch **does not** count towards the hung time.

You can specify Hung Limits as follows:

- With `nbjob run` and the `--hung-limits` switch
- With the **NB_HUNG_LIMITS** environment variable

Use the following syntax for soft and hard Hung Limits:

`<number1>{s|m|h} : <number2>{s|m|h}`

where `<number1>` and `<number2>` are positive numbers that can have a decimal point and `{s|m|h}` is one of three time units (seconds, minutes, ar hours) that define the time limit. No spaces are allowed in the time limit specification.

Note

Hung limits are rounded up to the nearest minute.

For Example

```
nbjob run --target linux_idc --class Linux
--qslot /Banias
--hung-limits 45m:1.5h echo hello
```

This specifies a soft hung limit of 45 minutes, and a hard hung limit of 1.5 hours.

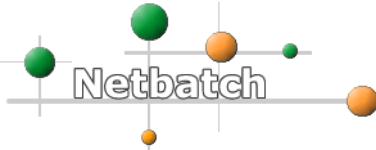
Alternatively the same hung limits can be defined with the NB_HUNG_LIMITS environment variable as follows:

```
setenv NB_HUNG_LIMITS 45m:1.5h
```

6.4.14 Specifying a Job Pre-Execution Utility

Before a Job is run, Netbatch can perform a **Pre-Execution** phase.

During Pre-Execution, a utility program is run on the Workstation where the Job is intended to run.



The purpose of this utility is to verify the existence of certain preliminary conditions that are required for the Job's execution.

Note

You can construct your own Pre-Execution utility programs using any programming or scripting language. Make sure that your Pre-Execution utility program exits with a negative exit status if it fails.

Note

The following environment variables contain information that the pre-exec script can use:

- **__NB_POOL**
- **__NB_QUEUE**
- **__NB_QSLOT**
- **__NB_LOG_FILE** (*but only if the --log-file option is used*)
- **__NB_CLASS**

If the Pre-Execution utility fails with a negative exit status, the Job is sent back to the Wait Queue. This prevents the Job from starting to execute on a Workstation where it would fail.

If the Job's Pre-Execution utility fails 10 times, the Job is cancelled.

To specify a Pre-Execution utility use the **--pre-exec** switch as follows:

```
nbjob run --target <poolName> --pre-exec  
<myPreExecUtility>
```

This instructs Netbatch to submit a Job to the default Queue and Qslot of the specified Pool and to run the specified pre-execution utility before the Job is executed.

You can also specify arguments to the Pre-Execution utility using quotes to enclose the entire Pre-Execution command.

For Example

```
nbjob run --target <poolName> --pre-exec "touch  
/tmp/lockFile" <command>
```

**Note**

You can also use the **NB_PRE_EXEC** environment variable to specify a pre-execution program.

--pre-exec overrides **NB_PRE_EXEC**.

6.4.15 Specifying a Job Post-Execution Utility

After a Job runs, Netbatch can perform a **Post-Execution** phase. During Post-Execution a utility program is run. Typically the Post-Execution utility will perform cleanup tasks.

Note

In the Post-Execution phase, the following environment variables are available:

- **NB_JOB_EXIT_STATUS**—the Job's exit status
- **__NB_CLASS**—the Job's class requirement expression
- **__NB_CMD_LINE**—the Job's full command line
- **__NB_FINISH_TIME**—the time at which the Job finished executing
- **__NB_JOBID**—the Job's ID
- **__NB_LOG_FILE**—the location of the Job log file
- **__NB_POOL**—the name of the Pool that the Job was submitted to
- **__NB_QSLOT**—the Qslot that the Job was submitted to
- **__NB_QUEUE**—the Queue that the Job was submitted to
- **__NB_RUSAGE**—information about the Job's resource usage
- **__NB_START_TIME**—the time at which the Job started executing
- **__NB_SUBMIT_PWD**—the user's working directory when the Job was submitted
- **__NB_TIMES_RESTARTED**—the number of times that the Job was restarted

A Post-Execution phase will take place even if Pre-Execution fails, and the Job is not run.

**Note**

You can construct your own Post-Execution utility programs using any programming or scripting language.

To specify a Post-Execution utility use the **--post-exec** switch as follows:

```
nbjob run --target <poolName> --post-exec  
<myPostExecUtility> <command>
```

This instructs Netbatch to submit a Job to the default Queue and Qslot of the specified Pool and to run the specified Post-Execution utility after the Job has completed.

You can also specify arguments to the Post-Execution utility using quotes to enclose the entire Post-Execution command.

For Example

```
nbjob run --target <poolName> --post-exec "rm -f  
/tmp/netbatch/*" <command>
```

Note

*You can also use the **NB_POST_EXEC** environment variable to specify a post-execution program.*

*--post-exec overrides **NB_POST_EXEC**.*

6.4.16 Specifying a Job Starter

A Job Starter is a wrapper utility that launches a Job.

Job Starters allows easy integration of third-party tools with Jobs that are run through Netbatch. Typically a Job Starter will set environment variables that are required for the correct execution of the Job.

When a Job is submitted to Netbatch with a Job Starter specification, Netbatch passes the Job's command as an argument(s) to the Job Starter. The Job itself is run by the Job Starter, and not directly by Netbatch.

Note

You can construct your own Job Starters using any language or script.



To execute a Job running a Job Starter use the **--job-starter** switch as follows:

```
nbjob run --target <poolName> --job-starter  
<myJobStarter> <command>
```

This instructs Netbatch to submit a Job to the default Queue and Qslot of the specified Pool and to run the Job through the specified Job Starter.

Note

*You can also use the **NB_JOB_STARTER** environment variable to specify a Job Starter.*

*--job-starter overrides **NB_JOB_STARTER**.*

6.4.17 Specifying the Job Priority

Two identical Jobs that are queued in the same Queue and Qslot, will be scheduled for execution according to the following rules, which are considered in this order:

- Jobs with higher assigned priority will be scheduled before Jobs with lower assigned priority.
- Jobs will be scheduled for execution according to the order in which they were submitted to the Pool (FIFO).

Use the priority switch if your Job requires priority over other Jobs that have been submitted to the specified Qslot. The value can be any integer. A higher number means a higher priority.

Note

For Jobs submitted to User Slots, the order is determined primarily by the attributes of the Slot, so priority will only determine the order of Jobs submitted by the same user.

For Example

```
nbjob run --target <poolName> --queue <queueName>  
--priority 12 <command>
```

This specifies that the Job <command> has a higher priority in the Qslot than your other Jobs with a priority of 11 or less.



6.4.18 Specifying a Task Name

A **Task** is typically a collection of Jobs that the user needs to complete in order to achieve a certain goal. Regression is a common example of a task.

Jobs can be associated with a task name as follows:

```
nbjob run --target <poolName> --task <taskname>  
<command>
```

When you want to track, modify, or remove all of the Jobs that belong to a task, the **nbstatus jobs**, **nbjobmodify**, and **nbjob remove** commands can filter according to task name.

For Example

```
nbstatus jobs --target <poolName> "Task=='<taskName>'"  
displays only those Jobs associated with the specified task.  
nbstatus jobs --target <poolName> "Task!=''"  
displays all Jobs, with task names (if any).  
nbjob remove --target <poolName> "Task=='<taskName>'"  
removes all the Jobs in the Wait and Run State Queues that are  
associated with the specified task.
```

To see the task names associated with all the Jobs in a Qslot, type:

```
nbstatus jobs --target <poolName> --fields "*",Task"  
"Queue=='<queueName>'&&Qslot=='<qslot>'"
```

Note

When submitting Jobs with Netbatch Feeder, a task name is automatically assigned to each Job. The task name that is assigned by the Netbatch Feeder is identical to the Netbatch Feeder taskname.

Task Dependency

You can make a Job dependent on a task (that is, so that the Job will not be dispatched until all the Jobs that make up the task have ended).

See [Section 6.5.2, Using nbjob run --triggers to Set Job Dependencies](#) and [Section 6.5.1, Using nbjob modify --triggers to Set Job Dependencies](#).



6.4.19 Submitting Groups of Jobs

To submit Jobs as a group (a "task"), use Netbatch Feeder/Flow Manager. See [Section 10, Submitting and Tracking Jobs with Netbatch Feeder](#).

6.4.20 Specifying Resource Limits

When a process is run, resources are allocated to it by the operating system. The operating system applies certain limits to the resources that may be allocated to every process it runs.

You may view the resource limits that apply to the shell process on a UNIX machine by typing **limit** at the command line:

```
% limit
cputime      unlimited
filesize     unlimited
datasize     unlimited
stacksize    8192 kbytes
coredumpsize 0 kbytes
memoryuse   unlimited
descriptors 1024
memorylocked unlimited
maxproc      2048
openfiles   1024
```

Note

*The **limit** command shows the limits on the Workstation that you are logged into, not the Workstation on which your Job will run.*

*If you need to know the limits that will apply to your Job, run the **limit** command on a Workstation whose capabilities match those required by your Job.*

Note

*The output of the **limit** command is different for each platform.*

**Note**

In bash and ksh, use ulimit -a instead of limit.

When a Job is run on a remote Workstation, default resource limits are applied to all resource types by the local operating system.

You can use the **--rlimits** switch to change the values of these limits:

```
nbjob run --rlimits "<res_limit> [soft|hard] =  
value[,...]"
```

The limits that can be set are listed in the **Table 5**, below. Some are platform-specific. If a limit is set for a platform that does not support it, it is ignored.

If a limit is not set, the limit is inherited from the WSM process that spawned the process. The WSM process, in turn, inherits the limits from the shell from which it was invoked.

Table 5: Resource Limits

Limit	Units	Description	Platform
addressspace	KB	The maximum size of a process's available memory	All
coredumpsize	KB	The maximum size of a core file that a process may create	All
cputime	Seconds	The maximum amount of CPU time that a process may use	All
datasize	KB	The maximum size of a process's heap	All
descriptors	# of files	Maximum number of files that a process may open (may be called openfiles on some platforms/in some shells)	All
filesize	KB	The maximum size of a file that a process may create	All
memorylocked	KB	Maximum locked-in memory address space	Linux
memoryuse	KB	The maximal resident set size of a process	Linux, HP-UX
openfiles	# of files	Maximum number of files that a process may open (may be called descriptors on some platforms/in some shells)	All
maxproc	# of processes	The maximum permitted number of processes ??	Linux

**Table 5: Resource Limits**

Limit	Units	Description	Platform
stacksize	KB	The maximum size of a process's stack (can also be set to <code>unlimited</code>)	All
vmemoryuse	KB	The maximum size of a process's mapped address space	Solaris

For Example

A good example of why you *might* want to change a resource limit is the **stack size limit**. Some Jobs are sensitive to the stack size limit, and will fail if this limit is set too high or too low (for example, the `pthreads` library in Linux).

The stack size limit for a Job that is run through Netbatch will be identical to the stack size limit that applies to the WSM process that spawned it.

If, on a machine with similar capabilities to those required by your Job, the `limit` command displays:

```
stacksize      8192 kbytes
```

then the Job you submit will probably run with a stack size limit of 8,192KB.

You can increase the stack size limit of the Job using the `--rlimits` switch.

For Example

```
nbjob run --rlimits "stacksize = 10000" myJob
```

This ensures that when the Job is run, it has a stack size limit of 10,000KB.

If necessary, you can specify an unlimited stack size:

```
nbjob run --rlimits "stacksize = unlimited" myJob
```

For Example

To increase the size of the core dump file that your Job can create, add the appropriate `--rlimits` switch to the `nbjob run` command:

```
nbjob run --rlimits "coredumpsize = 5000" myJob
```



6.4.21 Protecting Jobs from Blackholes

A Blackhole is a Workstation that accepts Jobs but fails to execute them. If a Job is dispatched to a Blackhole, it will be lost unless you do one of the following:

- Use the `--on-job-finish` switch when you submit the job with `nbjob run`.
- Set the `NB_ON_JOB_FINISH` environment variable.

The `NB_ON_JOB_FINISH` environment variable is explained in [Section 7.4.6, Default Re-queue Policies](#).

The syntax of the `--on-job-finish` switch is as follows:

```
nbjob run --on-job-finish
  "{<exit_code>|<expression>}:<operation>[:....]"
  [,...]
```

This specifies that if the Job exit code is `exit_code` or if it matches `expression`, the specified `operation`(s) are performed.

Here:

- `exit_code` is a numerical Job exit code (see [Exit Status](#)) or one of the following macros:
 - `NBErr` - all Netbatch errors except -8, -9, -10, and -11
 - `Nexit` - all negative Job exit values
 - `Pexit` - all positive Job exit values
 - `NZexit` - all non-zero Job exit values
- `expression` is an arbitrarily complex boolean expression consisting of `<attribute>`s, `<operator>`s, and `<value>`s, where:
 - `attribute` is any `nbstatus jobs` field (though typically, you will use `ExitStatus`, `PreExecExitStatus`, and `PostExecExitStatus`).
 - `operator` defines the relationship between the attribute and the value. `operator` can be one of the following:
 - An arithmetic operator: `+`, `-`, `*`, `/`, `%`, unary `+` (e.g., `"+10"`), or unary `-` (e.g., `"-10"`)
 - A logical operator: `&&`, `||`, or `!`
 - A comparison operator: `<`, `<=`, `>`, `>=`, `!=`, or `==`
 - A non-strict operator: `is` or `isnt`



- ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
- ◆ A conditional operator: ? or :
- ◆ **value** is a number, a string, or another boolean expression.

Expressions can be grouped together using parentheses.

For example,

```
ExitStatus==303&&PreExecExitStatus==2
```

- **operation** is one of the following:

- ◆ **Requeue(<max_attempts>)** - the Job is requeued up to **max_attempts** times.
- ◆ **Exclude(<period>{s|m|h})** - if the Job was submitted to a Virtual Pool, exclude the last resource (Pool) that the Job was dispatched to from those that it can be sent to, for the specified time. (If the Job was *not* submitted to a Virtual Pool, exclude the last Workstation that the Job was dispatched to from the list of Workstations that the Job can run on, for the specified time.)
- ◆ **Modify('<parameter>=<expression>'[,...])** - modify the Job before resubmitting it. You must also specify a **Requeue** action, otherwise the Job will not be resubmitted.

Here, **parameter** is a valid **nbjob modify** parameter (not including Queue) and **expression** specifies the value that is assigned to the parameter. **expression** can include any **nbstatus jobs** field.

If **expression** is a string, it must be enclosed in quotes(" ").

If you specify both **Requeue** and **Exclude** operations, **Exclude** must come first.

If you specify multiple **exit_code|expression:operations**, only the operation for the first match will be performed.

This switch overrides the **NB_ON_JOB_FINISH** environment variable.

Note

*If **Exclude** is used and the Job is submitted to a Virtual Pool or a Netbatch Feeder, the exclude flag is not passed to the physical Pool Master. (If it was passed to the physical Pool Master, the Job could be requeued by both the physical Pool Master and the Virtual Pool Master/Feeder.)*

**Note**

--on-job-finish takes precedence over --autoreq.

For Example

```
--on-job-finish "ExitStatus== -303 &&
PreExecExitStatus== 2 : Exclude(10h) : Requeue(3 )"
```

If the Job exit status is -303 and the exit status of the pre-execution stage is 2, then the Job is to be requeued up to three times. The Workstation that the Job was dispatched to should be excluded from the list of Workstations that it can run on for ten hours.

6.4.22 Specifying Values for Required Properties

The Qslot that you are submitting your Job to may have rules configured that require you to specify values for certain properties when submitting a Job to it.

If you omit required properties when submitting a Job, you will receive an error message that lists the properties that you must supply values for.

To specify values for properties, add the **--properties** switch to the **nbjob run** command:

```
nbjob run --properties "<prop_name>=<value>[ , . . . ]"
<command>"
```

where:

- **prop_name** is the name of a property.
- **value** is the property's value.

6.5 Working with Job Dependencies

In many instances, it is necessary to maintain a definite order of execution of Jobs. Examples of such Jobs are design flows. Jobs that update a design database should precede Jobs that validate the database against the designated design parameters.

Jobs that depend on other Jobs are called **Dependent Jobs**.
Jobs that other Jobs depend on are called **Trigger Jobs**.

Execution dependencies can be specified at Job submission time, so that Jobs are dispatched in the specified order. Each subsequent Job in the execution order is dispatched after the preceding Job has exited the system after completing successfully.

Note

If a trigger Job does not complete successfully, all its directly or indirectly dependent Jobs are suspended.

You will receive email notification if this happens.

The dependency hierarchies in Netbatch are directed acyclic graphs where any number of Jobs can depend on any number of other Jobs. There is no limit to the length of the Job dependency chain.

Job dependencies can also be specified using boolean expressions. For example, you can specify that Job C may not be dispatched for execution until **either** Job A successfully finishes executing **or** Job B ends with a specific exit code.

Dependencies may also be set between Jobs across Queue and Qslot boundaries.

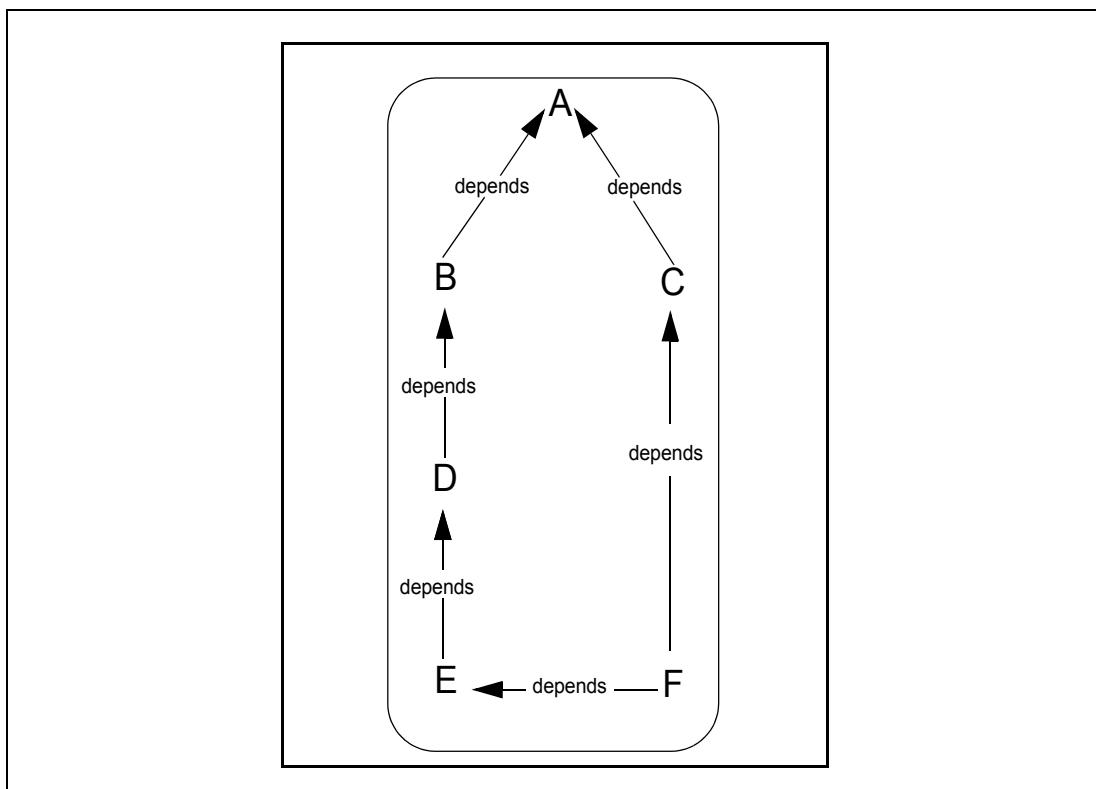


Figure 22: Chained Job Dependencies



Figure 22 illustrates the following scenario:

1. Processes **B** and **C** depend on **A**.
2. Process **E** depends on **D** which depends on **B**.
3. Process **F** depends on **E** and **C**.

In the execution scenario:

- **A** will have to complete before **B** will be considered for dispatch.
- **C** will be considered for dispatch only after **A** and **B** are complete.

If **B** fails (as indicated by a negative exit status) or is canceled (with `nbjob remove`), Jobs **D**, **E**, and **F** are suspended and will not be dispatched, unless they are subsequently resumed (with `nbjob resume`).

There are two ways to create a dependency between two Jobs:

- After the Jobs are submitted, using `nbjob modify` with the `--triggers switch`. See [Section 6.5.1](#).
- During submission of the Jobs, using `nbjob run` with the `--triggers` switch (see [Section 6.5.2](#)).

Note

Using interactive Netbatch commands in conjunction with the `gmake` utility is a more convenient and powerful tool for submitting complex flows of dependent Jobs. See [Section 6.14, Interactive Batch Jobs](#).

6.5.1 Using `nbjob modify --triggers` to Set Job Dependencies

Use `nbjob modify` for post-submission dependency ordering. The dependent Jobs must be in the wait queue. The trigger Jobs may be either running or waiting.

Use `nbjob modify` as follows:

```
nbjob modify <options> --triggers
  "<boolean_expression>"
```

where `boolean_expression` specifies which Job(s) the Job depends on. It replaces any existing dependencies.

`boolean_expression` consists of any number of terms of the form `[!]<JobID>[[<exit_status_expression>]]`, separated by the logical operators `&&` (AND) and `||` (OR).

**Note**

JobID can be either a Job ID or a task name. You are advised to explicitly specify whether it is a Job ID or a task name, because in ambiguous cases, Netbatch can assume that a task name is actually a Job ID, or vice versa.

Use one of the following instead of a Job ID/task name:

```
jobid:<Job_ID>  
task:<task_name>
```

If a task name is used, the exit_status_expression does not evaluate to true until it is true for each of the Jobs that make up the task.

See [Section 6.4.18, Specifying a Task Name](#).

`exit_status_expression` can be one of the following:

- `exit`
- `exit <relational_operator> <exit_code>`
`relational_operator` must be `<`, `<=`, `==`, `>=`, or `>`.

To specify multiple acceptable exit code conditions for a specific Job, use multiple `<JobID>[<exit_status_expression>]` terms separated by `||` (OR) (for example, `1 exit <30 || 1 exit>50`).

If `exit` is used on its own, it means that the Job ended (no matter what its exit status).

If no `exit_status_expression` is specified, `done` is assumed.

Note

Logical expressions are evaluated according to the following order of precedence - () (parentheses), ! (NOT), && (AND), || (OR).

So

```
"1 [exit<30] || !2 && (3 || 4)"
```

is the same as

```
"1 [exit<30] || ((!2) &&(3 || 4))"
```

**For Example**

If a Jobs with JobID 1 and JobID 2 have been submitted and Job 2 is still in the wait queue, make Job 2 dependent on Job 1 as follows:

```
nbjob modify --target <poolname> --triggers 1  
"Jobid==2"
```

Note

If a trigger Job is resubmitted, its dependent Jobs will wait.

6.5.2 Using nbjob run --triggers to Set Job Dependencies

To set a complex Job dependency at submission time, use the **--triggers** switch as follows:

```
nbjob run --target <poolName> --queue <queueName> --  
qslot <qslot> --triggers "<boolean_expression>"  
<command>
```

boolean_expression consists of any number of terms of the form **[!]<JobID>[[<exit_status_expression>]]**, separated by the logical operators **&&** (AND) and **||** (OR).

The nested square brackets denote that **<exit_status_expression>** is optional and that it must be enclosed in square brackets.

When using **!** (NOT), it must be applied to a **<JobID>[[<exit_status_expression>]]** expression. It cannot be used within such an expression.

Note

JobID can be a task name. If a task name is used, the **exit_status_expression** does not evaluate to true until it is true for each of the Jobs that make up the task.

See [Section 6.4.18, Specifying a Task Name](#).

exit_status_expression takes one of the following forms:

- **exit**
- **exit <relational_operator> <exit_code>**



`relational_operator` must be `<`, `<=`, `==`, `>=`, or `>`.

To specify multiple acceptable exit code conditions for a specific Job, use multiple `[!]<JobID>[[<exit_status_expression>]]` terms separated by `||` (OR) (for example, `1 [exit <30] || 1 [exit>50]`).

If `exit` is used on its own, it means that the Job ended (no matter what its exit status).

If no `exit_status_expression` is specified, it means that the Job ended with a non-negative exit status.

Caution***Jobs must not return negative exit codes.***

Netbatch uses negative exit codes to indicate that a Job did not successfully complete its execution for Netbatch reasons (e.g., Job was killed by Netbatch).

Note

Logical expressions are evaluated according to the following order of precedence—() (parentheses), ! (NOT), && (AND), || (OR).

So

`"1 [exit<30] || !2 && (3 || 4)"`

is the same as

`"1 [exit<30] || ((!2) &&(3 || 4))"`

For Example

```
nbjob run --triggers "1 [exit<30] || !2 &&
(3 [exit] || 4 [exit])" myJob
```

Here, myJob is dispatched only if:

- *Job 1 (i.e., the Job with an ID of 1) ended with an an exit code of less than 30 or*
- *Job 2 did not finish executing successfully, and either Job 3 or Job 4 finished executing (no matter what its exit status).*

**For Example**

```
nbjob run --triggers "(1 [exit==30] || 1 [exit==25] ||  
1 [exit>=10]) || 2 && (3 [exit] || 4 [exit])" myJob
```

Here, `myJob` is dispatched only if:

- Job 1 (i.e., the Job with an ID of 1) ended with an exit code of 30, 25, or greater than or equal to 10, or
- Job 2 successfully finished executing and either Job 3 or Job 4 finished executing (no matter what its exit status).

6.6 Jobs with Special Resource Requirements

6.6.1 Overview

A Job may have certain resource requirements. If you specify these when you submit the Job, Netbatch ensures that it is executed on a machine that supports the specified requirements.

When you submit a Job to Netbatch, you can specify its resource requirements in one of the following ways:

- As one or more required **Classes**
- As a **Smart Class** expression

These are described below.

Note

The resources used by a Job when it starts running (for example, memory or disk space) are often less than those that it needs at later stages of execution. This can lead to a situation where two (or more) Jobs are dispatched to a Workstation and successfully begin execution, but hang because their increasing resource requirements exceed the capabilities of the machine.

You can avoid such problems by using Netbatch's resource reservation feature. See [Section 6.9, "Submitting a Job with Resource Reservation."](#)



Classes

A Pool may contain different types of Workstation machines. Each machine may be designated as belonging to one or more **Classes**.

A Class is a name that represents Workstation characteristics such as platform, operating system type, memory size, and CPU power. A Class may apply to more than one Workstation and a Workstation may belong to more than one Class.

Class "Linux_4G", for example, can represent a Linux machine that has 4GB of RAM.

Smart Class

Similar to a Class, a **Smart Class** relates to Workstation characteristics but provides a broader and more dynamic representation of its attributes. A Smart Class is a boolean expression which defines the desired relation between the **attributes** of a Workstation—such as load, number of CPUs or supported classes—and the values of the attributes that are required by the Job. This mechanism is used to match a Job to one or more Workstations that best fit the Job's requirements.

See [Section 6.6.2, Viewing Supported Workstation Attributes](#), below, for a more detailed explanation of Workstation attributes.

As well as attributes, you can also specify **external probe** parameter names. An external probe is something that the Netbatch Administrator sets up to measure parameters that are not measured and available as Workstation (or other) attributes.

See [Section 6.6.3, Viewing Available Probe Parameters](#), below, for details of how to find out what probes and parameters are available.

Boolean smart class expressions are composed of conditions and operators. In fact, Netbatch automatically converts simple Class specifications (-C <class>) into a simple Smart Class expressions of the following form:

```
<class> is true
```

For Example

The expression:

```
"OSName is 'Linux' && (load < 1.5 || CPUCount > 1)"
```

will restrict the Job assignment to Linux workstations that have either a load factor that is lower than 1.5 or more than one CPU.

Another example. The expression:



"Linux_4G && (FVM > 2000)"

requires that the Job execute on a workstation that supports the class "Linux_4G" and has at least 2GB of free virtual memory.

6.6.2 Viewing Supported Workstation Attributes

Workstation attributes are of two types:

- Built-in—These pre-configured attributes exist for every Netbatch Workstation.
- Configured—These are attributes that your Netbatch administrator has configured for specific Workstations in the Pool.

Workstation attributes can further be divided into two categories:

- Static—These are attributes that are constant for a Workstation; for example, the OS type of the Workstation.
- Dynamic—These are attributes that change over time; for example, the load on the Workstation.

Every Workstation attribute takes a value which is of a specific type. These types are listed in **Table 6** below.

Table 6: Attribute Value Types

Type	Examples
Boolean	true, false
Integer	7, -33, 0666, 0x1a
Real	3.14, -2.134, 1e+5
String	"aaa", "\t"
Set	("ws1", "ws2")

To view available attributes use the **nbstatus** command:

```
nbstatus workstations --fields all --format block
```

The **Appearance** column in **Table 7**, below, shows how the fields are displayed in the output of **nbstatus**.

Figure 23, below, shows the output of this command for a Pool with one Workstation configured.



```
PPM on itstl109
Version 7.1.0_0166_03
On since 02/13/2006 10:37:12
Time now 02/15/2006 08:59:47
{
    Server = itstl109
    Status = Accepting
    Load = 0.03/4.00/4.80
    InteractiveUsers = 1/2
    Idle = 15/5
    Sel = 43
    Last Select = 02/13/2006 14:41:16
    Jobs = 0
    On Since = 02/13/2006 10:54:21
    IP = 143.185.3.41
    Version = 7.1.0_0166_03
    NLoad = 0.03/4.00/4.80
    Classes = @, m109, BIGMEM
    RunningJobs =
    SuspendedJobs =
    NicedJobs =
    HungJobs =
    RunAwayJobs =
    ReservedJobs =
    ResourceGroups =
    Resourceset = m109
    HW = i686
    OSName = Linux
    OSRelease = 2.6.5-7.1451xset1-smp
    OSVersion = #1 SMP Thu Jan 27 09:19:29 UTC 2005
    Arch = x86
    CPUCount = 4
    CPUMhz = 2380
    Hyperthreaded = true
    fDS = [/tmp=3825]
    tDS = [/tmp=4102]
    tfDS = 0
    fRM = 1050
    fVM = 4102
    tRM = 2026
    tVM = 4102
    AdminAttributes = []
    Concurrency = 2
```



```
WSCapabilities = [      interactive = 2;      slot = 2; ]
Reservation =
SupportedResourceSets = m109
RunningJobsCount = 0
SuspendedJobsCount = 0
NicedJobsCount = 0
HungJobsCount = 0
RunAwayJobsCount = 0
ReservedJobsCount = 0
WSMLite = false
VMTreshold = 200
CanRun = 2
NotAcceptingReason =
AutoRestart = false
NFactor = 1.0
ExternalProbeData =
CPUConsumption = 0
Host = itstl109
LockReason =
}
```

Figure 23: Viewing Supported Class Attributes

Note

*Note that the **Classes** field in Figure 23 displays a list of the classes that have been assigned to the Workstation.*

To view the attributes of a specific Workstation, type:

```
nbstatus workstations --target <poolName>
"Server=='<WorkstationName>'"
```

The attributes that you can use in a class requirements expression are listed in **Table 7**, below.

6.6.2.1 Built-in Attributes

Table 7, below, lists all the built-in Netbatch Class attributes.

When adding an attribute to a class requirement expression, use its name, as shown in the **Attribute Name** column.

**Table 7: Attributes that Can Be Specified in a Class Requirements Expression**

Attribute Name	Description	Value	Type	Appearance (nbstatus)
Arch	The machine architecture (e.g. "x86")	String	Static	Arch
<class>	The classes supported by the machine (specify as <class> or "<class>==true")	Set	Static	Classes
CPUCount	Number of CPUs	Integer	Static	CPUCount
CPUMhz	The CPU speed	Integer	Static	CPUMhz
fDS('<path>') ¹⁾	The free disk space on the partition where the specified directory resides (in MB)	Integer	Dynamic	fDS
fRM	The size in MB of free real memory on the machine	Integer	Dynamic	fRM
fVM	The size in MB of free virtual memory on the machine	Integer	Dynamic	fVM
Host	The host name	String	Static	Server
HW	The machine hardware (e.g., i686)	String	Static	HW
Hyperthreaded or HT	Whether the Workstation's CPU uses hyperthreading	Boolean	Static	Hyperthreaded
InteractiveUsers	The number of interactive users on the machines	Integer	Dynamic	InteractiveUsers
Load	The load on the machines	Real	Dynamic	Load
NLoad	The normalized load on the Workstation (if Workstation load normalization is enabled)	Real	Dynamic	NLoad
OSName	The OS name (e.g., Linux)	String	Static	OSName
OSRelease	The OS release	String	Static	OSRelease
OSVersion	The OS version	String	Static	OSVersion
tDS('<path>') ¹	The total disk space on the partition where the specified directory resides (in MB)	Integer	Dynamic	tDS
tFDS	Total free disk space (in MB)	Integer	Dynamic	tFDS
tRM	Total real memory (in MB)	Integer	Static	tRM
tVM	Total virtual memory (in MB)	Integer	Static	tVM

¹⁾ You can only specify a directory that has been specified by the Netbatch Administrator. Check the output of nbstatus workstations --fields all to see which directories are available.

**Note**

For each class defined for a Workstation, an additional boolean attribute is defined. This takes the form <class>==true.

These attributes can be included in a Job's class requirements. For example:

```
nbjob run --class "512M"
```

or:

```
nbjob run --class "512M==true"
```

This means that the Job needs to run on a Workstation for which 512M is true (that is, that supports the class 512M).

Another example:

```
nbjob run --class "\!512M"
```

This means that the Job can run on any Workstation except those that support the class 512M.

6.6.2.2 Configured Attributes

The Netbatch Administrator can configure other Workstation attributes, which also appear in **nbstatus workstations --fields all** output.

These include total and free disk space for specified directories (that is, the partitions on which the specified directories reside).

This information is displayed in the following format:

```
fDS = [<path>=<free_space>[ ,... ]]  
tDS = [<path>=<free_space>[ ,... ]]
```

Total free disk space is also displayed:

```
tFDS = <free_space>
```

free_space is in MB.



6.6.3 Viewing Available Probe Parameters

To see the available Workstation probes and the parameters that they measure, run `nbstatus workstations --fields all` and look at the value of the `ExternalProbeData` field.

To see the available Scheduler probes and the parameters that they measure, run `nbstatus probes`.

In both cases, the probe data is in the following format:

```
<probe_name>_<parameter_name>=<value>
reserve={true|false} [static=<static_param_name>]:
[...]
```

where:

- `probe_name` is the name of the probe.
- `parameter_name` is a parameter name.
- `value` is the parameter's value.
- `reserve` specifies whether the parameter can be used in a class reservation expression or not.
- `static_param_name` is the corresponding static parameter (if any).

6.6.4 Submitting a Job with Resource Requirements

When you submit a job with a class requirement, Netbatch checks whether the requirement can ever be satisfied by any of the Workstations to which it can send the Job. (This might include remote Workstations.)

For example, if your Job requires a Workstation with 1GB of free real memory, Netbatch checks whether any of the Workstations that it can send the Job to have **total** real memory of 1GB or more.

If none of the local Workstations meet the requirement, Netbatch accepts the Job while it queries the remote Workstations (if any). If one (or more) of the remote Workstations meets the requirement, the Job will eventually be dispatched for execution.

If no remote Workstation meets the requirement, the Job will remain in the wait state queue until it is removed—it can never be dispatched for execution.



You are recommended to use the `--validate` switch to make sure that your Job's class requirements can be satisfied before you submit it. See [Section 6.6.4.1](#), below.

Once Netbatch has performed the query, it caches the result. Subsequent Jobs with the same requirement will either be accepted (and later dispatched) or rejected immediately.

6.6.4.1 Checking Whether the Required Resources Are Available

Before submitting your Job, you are recommended to first check whether the resources that you are requesting are available, without submitting the Job for execution:

```
nbjob run --class <class_expression> --validate
<command>
```

With the `--validate` switch, `nbjob run` displays one of the following:

- Job not queued (validate mode). Class '`<class_name>`' is served by this pool.
- Job not queued. Class '`<class_name>`' is not served by this pool.

6.6.4.2 Submitting the Job

After making sure that the resources that you are requesting are available, submit your Job to Netbatch as follows:

```
nbjob run --target <pool_name> --class <expression>
<command>
```

An elementary resource requirement expression is of the following form:

```
<attribute> <operator> <value or expression>
```

where:

- `<attribute>` is a Workstation attributes or external probe parameter. See [Section 6.6.2, Viewing Supported Workstation Attributes](#) and [Section 6.6.3, Viewing Available Probe Parameters](#).
- `<operator>` defines the relationship between the attribute and the value. For a list of available Smart Class operators see below.
- `<value or expression>` is a number, a string, or another boolean expression.

**Note**

Netbatch validates smart class requirements to ensure that there is a Workstation in the Resource Set(s) that belongs to the Queue to which you are submitting the Job that meets your Job's requirements (if validation has been enabled by the Netbatch Administrator).

The first time a particular smart class expression is validated, the validation process can take some time. Netbatch caches the validation results, so subsequent validations of the same expression take an insignificant amount of time.

Values are interpreted by Netbatch in the following manner (in this order):

- If <value> can be interpreted as a double it will be handled as such.
- If <value> can be interpreted as an integer it will be handled as such.
- If <value> can be interpreted as a boolean it will be handled as such.
- If <value> is enclosed in apostrophes ('') it will be handled as a string value.
- If <value> can be interpreted as a legal expression it will be evaluated as such.
- Otherwise, <value> is considered illegal.

The available operators are:

- Arithmetic: +, -, /, %, unary + (e.g., "+10"), unary - (e.g., "-10")
- Logical: &&, ||, !
- Comparison: <, <=, >, >=, !=, ==

You can also use the non-strict operators:

- **is**
- **isnt**

A number of expressions can be grouped together using parentheses.

The following examples demonstrate possible uses of boolean expressions in resource requirements:

For Example

```
nbjob run --target <poolName> --class "OS is 'Linux' &&
fDs('/tmp')>1000" <command>
```



Meaning—The Job should run on a Linux Workstation with more than 1000MB of free disk space in the /tmp directory.

For Example

```
nbjob run --target <poolName> --class  
"Load <= 1 || (InteractiveUsers == 0 &&  
fVM > 4000)" <command>
```

*Meaning—The Job should run on a Workstation that either has a load that is less than 1, or that has no interactive users **and** more than 4000MB of free virtual memory.*

The expression that you specify is displayed in the output of the **nbjob run** command, as shown in **Figure 24**, below.

```
%nbjob run --class "OS is Linux && CPUCount >1" sleep 10000  
Your job has been queued (JobID 52, class OS is Linux && CPUCount >1, queue  
myQueue, slot 35)
```

Figure 24: Output of nbjob run Command with Smart Class Specification

Note

If the specified class requirement cannot be satisfied by any of the Workstations to which the specified Queue can dispatch Jobs, the Job is not Queued.

Note

If the specified class requirement cannot be satisfied by any of the Workstations to which the specified Qslot can dispatch Jobs, the Job remains in the wait state indefinitely (and is never dispatched for execution).



6.7 Jobs That Require Licenses

6.7.1 Overview

Software vendors distribute their software packages with license files. These files contain a list of features that are enabled for the specific software suite purchased. To use the software, you (or rather, your Job) must check a license out of a license server and check it back in when it is no longer required. The license server ensures that no more than the permitted number of instances of each license are checked out at the same time.

Note

*In this document, the terms **license** and **feature** are used interchangeably.*

Netbatch can ensure that Jobs that require licensing are dispatched for execution only when the available licenses are available. This prevents the situation where a Job is dispatched for execution on a Workstation but fails repeatedly because the license it required is not available.

License Aliases

You can specify the location of a license that you want to use by specifying:

- The full path of the license keyfile AND the name of the license
- The license keyfile location (<port>@<host>) AND the name of the license
- The license's **Alias**

Using License Aliases is the easier, and recommended, way.

Note

To allow you to use a license's alias, the Netbatch Administrator must have set up an alias for the license that you want to use.

To find out whether an alias is available for the license that you need, use the **nbstatus licenses** command. See [Section 6.7.2, Viewing Available Licenses](#), below, for more details.

**Note**

If you use aliases, you do not need to define the **NB_LICENSE_FILE** environment variable.

6.7.2 Viewing Available Licenses

You can check whether a Netbatch Pool can coordinate the use of the licenses you need by using one of the following commands:

- **nbstatus licenses**—recommended
- **nbstatus logical-licenses**—recommended
- **nblicense**
- **nblicstat**—older, Netbatch 5.x compatible

You can also use these commands to check if a required feature is available, and whether it has an alias configured, prior to submitting a Job.

Note

nbstatus licenses displays details of **physical** licenses; that is, specific licenses in specific keyfiles. A license that appears in two keyfiles is listed **twice**.

nbstatus logical-licenses displays details of **logical** licenses; that is, specific licenses that may appear in more than one keyfile. A license that appears in two keyfiles is listed **once**.

For example, if license *f1* appears in two different keyfiles, with a capacity of 20 in one keyfile and 50 in the other:

- **nbstatus licenses** displays two lines—one for each keyfile in which the license appears.
- **nbstatus logical-licenses** displays a single line for the license with a capacity of 70.

The **nbstatus licenses** command displays the following fields:

- ◆ **LicenseName*** - the license name
- ◆ **Status** - whether the license server is **UP** or **DOWN**
- ◆ **Vendor** - the name of the license vendor



- ◆ **ActualCapacity** - the total number of instances of this license available on the license server(s)
- ◆ **ConfiguredCapacity** - the total number of instances of this license available to Netbatch Jobs
- ◆ **CheckedOut*** - the total number of instances of this license that have been checked out from the license server
- ◆ **ServiceCheckedOut*** - the total number of instances of this license that have been checked out by Netbatch Jobs
- ◆ **Reserved*** - the number of Netbatch Jobs for which instances of the license have been reserved but not yet checked out
- ◆ **Available*** - the number of instances of the license currently available to Netbatch Jobs
- ◆ **Factor** - the number of Jobs that can reserve the license for each available instance of the license.

For example, if it is estimated that on average, Jobs only use the license for half of the time that they are running, **Factor** is set to 2.

- ◆ **Margin** - the number of instances of a license that are reserved for non-Netbatch use
- ◆ **Alias*** - the license alias (if any)
- ◆ **KeyFile** - the license keyfile
- ◆ **SyncStrategy** - how Netbatch calculates the feature's availability. It can be:
 - ◆ **static**—a simple counter is used to track the number of available instances of the feature.
 - ◆ **dynamic**—the license server is queried to check if a feature is available.
- ◆ **ExpirationDate** - the license expiration date
- ◆ **Sessions** - the IDs of the Netbatch license sessions currently using this license
- ◆ **Cost** - configured cost for this feature

The following figure shows sample output for the following command:

```
nbstatus licenses --fields
"LicenseName,Capacity,ServiceCapacity,
Available,Alias" "available>0"
```



PPM on itstl2022					
Version 7.1.0_0163_00					
On since 12/27/2005 09:40:28					
Time now 12/29/2005 15:25:46					
<hr/>					
LicenseName	Capacity	ServiceCapacity	Available	Alias	
<hr/>					
945	80	87	80	null	
994	82	89	82	null	
Fire_Parallel	150	150	150	null	
Ice_Parallel	100	100	100	null	
LEAPFROG-CV	85	90	85	null	
UET	113	119	113	null	
msimcdebug	600	100	100	null	
msimcompare	600	100	100	null	
msimcoverage	600	100	100	null	
msimdataflow	600	100	100	null	
msimhdlcom	600	100	100	null	
msimhdlsim	600	100	100	null	
msimprofile	600	100	100	null	
msimviewer	600	100	100	null	
plotversa	332	344	332	null	
<hr/>					

Figure 25: nbstatus licenses Output

Jobs do not usually need a license for their entire duration of their run. The usage factor takes this into account, as illustrated in the following examples:

- If the usage factor is 2, then for each feature not currently in use, Netbatch attempts to run two Jobs that require the feature. This ensures good utilization of a license if a Job does not require the license for its entire runtime.

However, if more Jobs require the same feature than the number of currently available licenses, those Jobs may hang or fail, depending on the tool.

- If the usage factor is 0.5, then for every two features not currently in use, Netbatch attempts to run one Job. This reserves features for users who use the license outside of Netbatch, but limits the maximum utilization of the license when these users are not actively using it.



6.7.3 Submitting Jobs that Require Licenses

The `nbjob run` command's `--license` switch tells Netbatch that the submitted Job requires one or more licenses. You may specify complex license requirements with a boolean expression.

With the `--license` switch, you can specify either a license or a **License Alias** (or a boolean expression comprised of either of these). See [License Aliases](#), above, for more details.

These two methods of submitting a Job are explained in the following sections:

- [Section 6.7.3.1, Submitting a Job Using a License's Alias](#)
- [Section 6.7.3.2, Submitting a Job by Specifying Keyfile and Feature Name](#)

Note

*You can set the **NBLICENSES** environment variable to specify that ALL Jobs that you submit require the specified licenses. (These can be expressed as either feature names or aliases.)*

*Setting **NBLICENSES** is the same as adding `--license <licenses>` to every `nbjob run` command (that is, the contents of **NBLICENSES** can be a single feature name or alias or a boolean expression comprising feature names and/or aliases).*

Using Job Profiles is a much more flexible way of specifying license requirements, as it allows you to specify different requirements for each Job type. This is described in detail in [Section 6.13, Using Job Profiles](#), below.

6.7.3.1 Submitting a Job Using a License's Alias

If the feature(s) that your Job needs has an alias (use `nbstatus licenses` to check if it does), you can submit the Job by typing:

```
nbjob run --target <pool name> --licenses  
<license_alias> [other options] <command>
```

where `license_alias` can be a single alias or a boolean expression.

If you have permissions for a feature that exists in multiple license keyfiles, each of which has a different alias, type:

```
nbjob run --target <pool name> --license "<expression>"  
[other options] <command>
```



where **expression** consists of any number of **<req_expr>**s and **<operator>**s:

- **req_expr** is a license requirement expression of the following format:

```
"<license>=<value>[ :duration=<duration_time> ]"
```

where:

- **license** is the alias of the required license.
- **value** is the number of instances of **license** that are required.
- **duration_time** is the amount of time (in minutes) for which the license is required (starting from when the Job is dispatched for execution).

If not specified, then **duration** is set to the Job execution time (that is, the license is reserved for the entire duration of the Job's execution).

- **operator** is **&&** or **||**.

These can be grouped together using parentheses.

For Example

```
nbjob run --target linux_idc --license X0 myJob
```

This submits the Job myJob to the linux_idc Pool, and specifies that it requires the feature whose alias is X0.

For Example

```
nbjob run --target linux_idc --queue myQueue  
--license "(X0&&X1)||X2=2" myJob
```

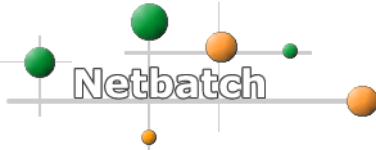
This submits the Job myJob to Queue myQueue of the linux_idc Pool, and specifies that it requires either both X0 and X1 or two instances of X2 (where X0, X1, and X2 are all aliases).

For Example

```
nbjob run --target linux_idc --queue myQueue  
--license "(X0:duration=60&&X1)||  
X2=2:duration=60" myJob
```

This submits the Job myJob to Queue myQueue of the linux_idc Pool, and specifies that it requires either both X0 (for the first 60 minutes of its run time) and X1 (for the entire duration of its run) or two instances of X2 (for the first 60 minutes of its run time).

X0, X1, and X2 are all aliases.



6.7.3.2 Submitting a Job by Specifying Keyfile and Feature Name

To submit a Job by specifying the license keyfile and the feature name:

1. Make sure that the **NB_LICENSE_FILE** environment variable contains the full path(s) of the license keyfile(s) that contains the feature(s) that your Job requires.

`nbstatus keyfiles` output includes this information.

Note

*You can use `nbjob run`'s `--license-keyfile` switch instead of setting **NB_LICENSE_FILE**. (See Step 2, below.)*

Note

*You will usually set the **NB_LICENSE_FILE** environment variable to the value that you set for the **LM_LICENSE_PATH** variable, in order to work with the licensed program directly.*

Note

*You can set **NB_LICENSE_FILE** to define multiple keyfiles (which can be on multiple license servers). Netbatch searches for licenses in the locations specified in **NB_LICENSE_FILE** in the order in which they are specified there.*

Note

*The **NB_LICENSE_FILE** variable may have already been set for you. You can check whether the variable is defined by typing:*

```
echo $NB_LICENSE_FILE
```

If you do not know where the keyfiles you require are, ask your Design Automation group for the path.

If you have permissions for a feature that exists in multiple license keyfiles, include the full path of each keyfile in **NB_LICENSE_FILE**. The paths must be delimited by colons.

**For Example**

*To set **NB_LICENSE_FILE** to `somekeyfile` from the vendor `cadence`, type (on a single line):*

```
setenv NB_LICENSE_FILE  
/nfs/iil/license/license/cadence/somekeyfile
```

You can specify multiple keyfiles with a colon-delimited list as follows:

```
setenv NB_LICENSE_FILE  
"someKeyFile:someOtherKeyFile:..."
```

Note

*Setting the **NB_LICENSE_FILE** environment variable correctly ensures that Netbatch runs Jobs that require licenses correctly. Specifying **NB_LICENSE_FILE** incorrectly or failing to specify it will not cause Netbatch to reject Jobs.*

However, the use of licenses will no longer be coordinated by Netbatch, and as a result, Jobs that require licenses may fail.

Caution

It is important to specify the full path of the keyfiles exactly as they were configured by your Netbatch administrator.

Using soft links that point to the keyfiles will not be interpreted correctly by Netbatch.

For example,

```
/nfs/iil/license/license/synopsys/  
ncg.synopsys.key
```

and

```
/nfs/iil.intel.com/license/license/  
synopsys/ncg.synopsys.key
```

may point to the same file in the file system, but Netbatch will see them as referring to different files.



2. Submit your Job using `nbjob run` with the `--license` switch, specifying the name(s) of the license(s) that the Job requires.

Note

If you did not set `NB_LICENSE_FILE`, you can use the `--license-keyfile` switch when submitting the Job.

Just add one of the following to the `nbjob run` command below:

- `--license-keyfile <full keyfile path>`
- `--license-keyfile "<full keyfile path>[:....]"`
- `--license-keyfile <port>@<host>`
- `--license-keyfile "<port>@<host>[:....]"`

To find out the location (in `<port>@<host>` format) of the keyfile that contains the feature that you need, use `nbstatus keyfiles`.

Type:

```
nbjob run --target <pool name> --license "<expression>"  
[other options] <command>
```

where `expression` consists of any number of `<req_expr>`s and `<operator>`s:

- `req_expr` is a license requirement expression of the following format:
`"<license>=<value>[:duration=
<duration_time>]"`

where:

- `license` is the name of the required license.
- `value` is the number of instances of `license` that are required.
- `duration_time` is the amount of time (in minutes) for which the license is required (starting from when the Job is dispatched for execution).

If not specified, then `duration` is set to the Job execution time (that is, the license is reserved for the entire duration of the Job's execution).

- `operator` is `&&` or `||`.



These can be grouped together using parentheses.

For Example

```
nbjob run --target linux_idc --license license1  
--license-keyfile 83817@ilics03.iil.intel.com myJob
```

This submits the Job myJob to the linux_idc Pool, and specifies that it requires license1, which appears in the specified keyfile (here in <port>@<host> format)

For Example

```
nbjob run --target linux_idc --queue myQueue  
--license "(license1 && license2) || license3 = 2" --  
license-keyfile 83817@ilics03.iil.intel.com myJob
```

This submits the Job myJob to Queue myQueue of the linux_idc Pool, and specifies that it requires either both license1 and license2 or two instances of license3 (which appear in the specified keyfile).

For Example

```
nbjob run --target linux_idc --queue myQueue  
--license "(license1:duration=60 && license2) ||  
license3 = 2" --license-keyfile  
83817@ilics03.iil.intel.com myJob
```

This submits the Job myJob to Queue myQueue of the linux_idc Pool, and specifies that it requires either both license1 (for the first 60 minutes of its run time) and license2 (for the entire duration of its run) or two instances of license3 (for the entire duration of its run). These licenses all appear in the specified keyfile.

6.8 Kerberos Authentication

6.8.1 Overview

Depending on your administrator's configuration, Netbatch provides the option of submitting Jobs with Kerberos authentication. Prior to submitting a Job, you will need to acquire a Kerberos ticket-granting ticket (TGT) for your submission host ([Section 6.8.2](#)). The acquired TGT is attached to a submitted Job and grants access to services that require authentication. A TGT has a predefined validity period. A Job is required to have a valid TGT when accessing services that require authentication upon execution. Netbatch ensures that a Job submitted with `--kauth` on and a valid TGT reaches its destination host with a valid TGT.



6.8.2 Acquiring Kerberos TGT

Kerberos ticket-granting tickets are acquired through an external tool (`kinit`).

Prior to submitting a Job, run `kinit` on your submission host. This command will prompt you to enter your network password, or use a keytab file.

The TGT file is stored either in the default predefined path: `/tmp/krb5cc_<uid>`, or according to the path set in the environment variable **KRB5CCNAME**.

Note

The value of KRB5CCNAME might not persist after the session is closed.

6.8.3 Renewing Kerberos TGT

Due to the time that elapses between submitting a Job and running it, the waiting Job may need to have its credentials renewed.

To ensure the submitted Job has a valid TGT, follow this renew procedure:

1. Acquire a TGT (see [Section 6.8.2](#)).
2. Run `nbkrb renew` (see [Appendix A.7](#)). This command attaches a new TGT to Jobs, even if these Jobs were already submitted and are waiting.

Note

Also, run this command when you are notified that your credentials are about to expire.

6.9 Submitting a Job with Resource Reservation

The resources used by a Job when it starts running (for example, memory or disk space) are often less than those that it needs at later stages of execution. This can lead to a situation where two (or more) Jobs are dispatched to a Workstation and successfully begin execution, but hang because their increasing resource requirements exceed the capabilities of the machine.

When you submit a Job, you can specify:

- The **type** of resource(s) that your Job requires



- The **amount** of the specified resource(s) that your Job requires
- The **amount of time** that your Job needs the specified resource(s)

Netbatch uses this information to ensure that the Job is assigned the necessary resources for the required duration.

Note

Resource requirements and Resource Reservation are not the same.

A Job's **resource requirements** specify conditions that must be satisfied on a Workstation before the Job can be dispatched for execution on that Workstation (for example, free virtual memory).

A Job's **Resource Reservation** specifies that certain resources must be reserved for the Job's exclusive use for a specified time period (or for the duration of its run).

A Job may be submitted with either resource requirements, Resource Reservation, both, or neither.

Note

For tracking Jobs that have been submitted with Resource Reservation, use:

- The **nbstatus jobs** command with the **--fields** switch to display the following fields:
 - ◆ **WaitReason**
 - ◆ **ReservedClasses**
 - ◆ **RUsage**
- The **nbstatus workstations** switch with the **--fields** switch to display the following fields:
 - ◆ **ReservedJobs**
 - ◆ **Reservation**
 - ◆ **ReservedJobsCount**

See [Section 8.3.3, Viewing a Job's Resource Usage](#) for a detailed explanation.



6.9.1 Submitting a Job with Class Reservation

Note

If, after you submit your Job, it seems to be waiting a long time in the Wait State Queue, you can use `nbstatus jobs` with the `--fields` switch to see what is happening—the `ClassReservation` and `ClassImplicitReservation` fields show the requested resource reservation, while the `ReservedClasses` field shows the resources that Netbatch has already accumulated for the Job.

To submit a Job that requires certain physical machine characteristics or that requires that certain measured parameters match required values, use `nbjob run` with the `--class-reservation` switch. Type:

```
nbjob run --target <pool name> [other options]
--class <class specification>
--class-reservation <reservation_item>[,...]
```

Note

The Job's actual class reservation (as specified here and/or applied by a Job profile) is available in the `__NB_CLASSRESERVATION` environment variable (in the Job's environment) at runtime.

Each `reservation_item` string must be in the following format:

```
"<key>=<key_value>
[:duration=<duration_time>[\{h|m|s\}]]
[:decrease=<decrease_time>]
[:delta=<delta_value>]
```

where:

- `key` is the type of resource to be reserved. It can take the following values:
 - `fVM`
 - `fRM`
 - `dedicated`
 - The name of a custom capability or attribute that has been defined for one or more Workstations in the Pool

You can use the `nbstatus workstations` command to see a list of these attributes:

```
nbstatus workstations --fields "* , WSCapabilities"
```



These capabilities can either relate to total real or virtual memory or be arbitrary.

Note

Workstation capability requirements can also be defined at the Queue or Qslot level. Such requirements apply to all Jobs submitted to the Queue/Qslot.

Note

*If you specify a requirement for a Workstation capability that does not exist in the Pool, it will stay in the wait state indefinitely. Make sure that the capability exists before submitting the Job. (Use **nbstatus workstations** to see if the required capability appears in the **WSCapabilities** field for any of the Pool's Workstations.)*

- The name of an external probe parameter that can be used for resource reservation

To see the available Workstation probes and the parameters that they measure, run **nbstatus workstations --fields all** and look at the value of the **ExternalProbeData** field.

To see the available Scheduler probes and the parameters that they measure, run **nbstatus probes**.

In both cases, the probe data is in the following format:

```
<probe_name>_<parameter_name>=<value>
  reserve={true|false}
  [static=<static_param_name>]: [...]
```

where:

- **probe_name** is the name of the probe.
- **parameter_name** is a parameter name.
- **value** is the parameter's value.
- **reserve** specifies whether the parameter can be used in a class reservation expression or not.
- **static_param_name** is the corresponding static parameter (if any).

**Note**

If **dedicated=true**, then no other Jobs may run on the host while the Job is running (except for Parallel Slave Jobs—multiple Slave Jobs that are part of the same Parallel Job can run on the same host).

If the Job is a Parallel Slave Job and **slots_per_host** was specified when the Master Job was submitted, then Slave Jobs will run on the specified number of execution slots, and no other Jobs will run on the Workstation.

All Resource Reservation specifications are specified when submitting the Master Job (though the actual reservation is owned by either the Master Job or a Slave Job, depending on what is requested)—individual Slave Jobs cannot have different Resource Reservation specifications.

- **key_value** is the value for **key**. It must be one of the following:
 - ◆ For **fVM**, **fRM** or a custom field, it can be any type of number—integer or non-integer value
 - ◆ For **dedicated**, **true** or **false**
- **duration_time** is the amount of time (in hours, minutes, or seconds) for which the resources (**fVM** and **fRM** only) are required (starting from when the Job is dispatched for execution). If you do not specify **h**, **m**, or **s**, the time is in minutes.

If not specified, then **duration** is set to the Job execution time (that is, the resource is reserved for the entire duration of the Job's execution).

- **decrease_time**—During the time specified by **duration**, the committed resource (**fVM** and **fRM** only) is reduced every **decrease_time** minutes by the amount specified in **delta_value**.

If a **delta_value** is specified, but no **decrease_time**, then **decrease = 1** minute.

If neither a **delta_value** nor a **decrease_time** is specified, then the resource is available at the same level for the whole time specified by **duration**.

- **delta_value**—During the time specified by **duration**, the committed resource (**fVM** and **fRM** only) is reduced every **decrease_time** minutes by the amount specified in **delta_value**.

If a **decrease_time** is specified, but no **delta_value**, then **delta = 1** unit.



If neither a `delta_value` nor a `decrease_time` is specified, then the resource is available at the same level for the whole time specified by `duration`.

Note

`--class` and `--class-reservation` are *not the same*:

- `--class` specifies the resources that must be available when the Job starts (without taking into account what may happen later in the Job's execution).
- `--class-reservation` specifies the actual resources that will be required by the Job, and for how long they will be required.

For Example

```
nbjob run --target myPool --class "fVM>10" --class-
reservation "fVM=10" myJob
```

This specifies that Job `myJob` requires 10MB of free virtual memory for the entire duration of its execution.

For Example

```
nbjob run --target myPool --class "fVM>10" --class-
reservation "fVM=10:duration=2" myJob
```

This specifies that Job `myJob` requires 10MB of free virtual memory at some point during its first two minutes of execution.

6.10 Submitting a Parallel Job

Overview

A Parallel Job is a Job that requires more than one **Execution Slot** (that is, a processor or a host), or that consists of several Jobs, each of which requires an execution slot.

A Parallel Job consists of a Master Job and any number of Slave Jobs. The Master Job is responsible for dispatching the Slave Jobs to the hosts assigned to it by Netbatch (using the `nbjob prun` command).

Netbatch supports the **Message Passing Interface (MPI)** and the **Linda** coordination language. Detailed instructions for working with both of these are given in the relevant sections below.

**Note**

If you cancel or resubmit the Master Job, all the Slave Jobs are also cancelled or resubmitted.

Note

If a Parallel Job is preempted, the appropriate action is applied to the whole Parallel Job; that is, the Master Job and all its Slave Jobs.

Caution

*It is possible to **cancel a Slave Job**, but if another Slave Job is dependent on the cancelled Job, the Master Job will never complete, as it must wait for all its Slave Jobs to end.*

Only cancel a Slave Job if you understand the Parallel Job's internal logic and the consequences of cancelling it.

Caution

*It is possible to **resubmit a Slave Job**, but because Slave Jobs are usually interdependent, such an action may have adverse effects.*

Only resubmit a Slave Job if you understand the Parallel Job's internal logic and the consequences of resubmitting it.

Submitting a Parallel Job

To submit a Parallel Job, you must first create the Master Job that specifies the individual Slave Jobs that are to be run. Creating a Master Job is explained in [Section 6.10.1](#), below.

You then submit the Master Job by using `nbjob run` with the `--parallel` switch. Using a network topology mapping file (.ntm), you can optimize the running of parallel Jobs by defining the maximum latency allowed. This is explained in [Section 6.10.2](#), below.



A Parallel Job has a number of separate Job log files (one for the Master Job and one for each Slave Job). These are described in [Section 6.10.4](#), below.

Note

For tracking Parallel Jobs, use the nbstatus jobs and nbstatus workstations commands with the --fields switch to display the following fields:

- **nbstatus jobs:**
 - ◆ **ParallelRequest**—displays parallel reservation requirements (if any), as specified with **--parallel**
 - ◆ **ParallelJobID**—displays the ID of the Master Job
 - ◆ **ParallelHosts**—displays the hosts that Netbatch has assigned to the Parallel Job. If there is more than one execution slot available on the same host, that host's name appears the corresponding number of times.
 - ◆ **ReservedClasses**—displays the resources that have been reserved for the Job, including execution slots.
 - ◆ **ParallelPhase**—shows whether the Job is in the **Inclusive phase** (where the scheduler is looking for and marking execution slots for the Job, and where other Jobs can still run in these slots) or the **Exclusive phase** (where the execution slots are reserved for the Job).
- **nbstatus workstations:**
 - ◆ **ReservedJobs**—IDs of Jobs for which resources have been reserved
 - ◆ **ReservedJobsCount**—the number of Jobs for which resources have been reserved
 - ◆ **Reservation**—the resources that have been reserved, and the Job(s) for which they have been reserved

*For execution slots that have been reserved by Parallel Jobs, this field shows whether the status of the execution slot(s) is **Inclusive** (marked by one or more Parallel Jobs, but can still be used by other Jobs) or **Exclusive** (reserved for a particular Parallel Job). It also shows the Job ID(s) of the Parallel Jobs (only one if its status is Exclusive, one or more if Inclusive).*

*For example, the following **Reservation** field shows that execution slots on the machine have been marked as*



inclusive locked by Jobs 303 and 304 in Pool m2022:

slot: inclusive=[m2022.302, m2022.303] ()

See [Section 8.3.2, Parallel Job Details](#) for a detailed explanation.

6.10.1 Creating a Master Job

A Master Job can be written as a Perl script, a shell script, or any kind of program. When a Master Job begins executing, Netbatch has already set the **NB_PARALLEL_JOB_HOSTS** environment variable, which contains a space-delimited list of hosts that the Slave Jobs can be sent to.

Note

If you use MPI or Linda within Netbatch to run Parallel Jobs, you must make certain changes to allow Netbatch to correctly monitor and control all the Slave Jobs.

After these changes are made, MPI and Linda will use a supplied Netbatch script instead of rsh, which is their default way of communicating with the Workstations on which the Slave Jobs run. (MPI uses rsh to start a daemon process on each Workstation. Linda uses rsh for all communication with the Workstation.)

The script runs nbexec with a combination of switches that cause it to behave in a similar way to rsh.

- For MPI-specific instructions, see [Section 6.10.1.1](#), below.
- For Linda-specific instructions, see [Section 6.10.1.2](#), below.

Note

Netbatch implements safeguards to prevent abuse of the Parallel Job mechanism. For example, if you have been assigned two execution slots on a host listed in NB_PARALLEL_JOB_HOSTS, and you try to send three Jobs to that host (using nbjob prun), then the third Job will fail.

Note

NB_PARALLEL_JOB_HOSTS does not include a slot for the Master Job.

If there is more than one execution slot available on the same host, that host's name appears the corresponding number of times.



The Master Job should use the `nbjob prun` command to send each Slave Job to one of the hosts.

Note

The nbjob prun command does not have a --target switch to specify which host or Pool the Slave Jobs are sent to. Instead, it reads the __NB_PARALLEL_POOL_HOST environment variable (which is set by Netbatch when the Master Job is dispatched for execution).

nbjob prun then sends each Slave Job to the Pool Master specified in __NB_PARALLEL_POOL_HOST for validation before it is executed on the specified host.

Note

The nbjob prun command can fail in the following circumstances:

- *The specified Workstation (host) is down.*
- *The (master) Job is being preempted.*
- *There are no more reservations (nbjob prun or nbexec has been called too many times).*
- *There is no Master Job attached to the Job. (That is, the Job was not sent by a valid Master Job.)*
- *No Workstation was specified.*

Caution

*Do not set __NB_PARALLEL_POOL_HOST yourself.
Netbatch sets it as part of the scheduling process to ensure system consistency.*

The interface of the `nbjob prun` command is very similar to that of `nbjob run`—it supports a subset of the `nbjob run` options—except for the `--target` switch, which specifies the host that the Slave Job is to be sent to.

Note

Though nbjob prun and nbjob run have similar interfaces, they function very differently.



nbjob run queues a Job.

nbjob prun dispatches a Job for execution on a host that has already been reserved for it.

To use **nbjob prun**, add lines like this to your Master Job script/program:

```
nbjob prun --host <host_name> [other_switches]
<command> <arguments>
```

where:

- **host_name** is one of the hosts listed in **NB_PARALLEL_JOB_HOSTS**.
- **other_switches** are any other **nbjob prun** switches.

Note

Slave Jobs do not inherit the Job attributes of their Master Job, unless nbjob prun's --inherit switch is used. For example, if the Master Job is submitted with the -s switch (which causes Netbatch to email the user when the Job starts), all its Slave Jobs that are submitted with nbjob prun --inherit will also be dispatched with this attribute.

- **command** is the name of a Slave Job.
- **arguments** are the Slave Job's arguments, if any.

For a full list of supported **nbjob prun** switches, see [Appendix A.2.9](#).

Note

Although each Slave Job is sent to a reserved execution slot on a host, the Jobs do pass via the Pool Master for validation. Because of this, in the rare situation of the Pool Master being down, the Slave Jobs will not be successfully dispatched.

If you know that later iterations of your Job require fewer execution slots than earlier ones, use **nbjob pmodify** to release execution slots that are no longer required. You can also use it to increase the number of execution slots for the Job. See [Section 6.10.3, Modifying a Parallel Job](#).

If a Slave Job needs to record data for the Master Job (i.e., so that it will appear in the **CustomAttributes** field for the Master rather than for the Slave), it can do so using the **nblog** command with the **--master** switch. See [Appendix A.10, nblog](#).



6.10.1.1 MPI

Netbatch uses the **LAM/MPI** implementation of the Message Passing Interface (MPI). For further information about LAM/MPI, see <http://www.lam-mpi.org>.

If your Master Job uses MPI, it must perform the following five operations, in this order (or use the MPI Job starter script—see below):

1. Set the **LAMRSH** environment variable to force LAM/MPI to use the supplied Netbatch script instead of **rsh**:

```
setenv LAMRSH $NS_HOME/sbin/rsh.nb
```

where **NS_HOME** is the Netbatch installation directory.

2. Read the **NB_PARALLEL_JOB_HOSTS** environment variable and write its contents to a plain text file. Each host name must be on a separate line.

The hostname of the Workstation on which the Master Job is running (i.e., localhost) must also be added to this file.

3. Run **lamboot**:

```
lamboot <hostfile>
```

where **hostfile** is the file that was created in Step 2, above.

LAM/MPI connects to each Workstation (using **rsh.nb** instead of **rsh**) and starts the **lamd** daemon on each one. This way, **lamd** runs as a Netbatch Job, allowing it and all its child processes (the Slave Jobs) to be monitored and controlled by Netbatch.

4. Use **mpirun** to send each Slave Job to be executed on a Workstation.

```
mpirun <arguments>
```

See the LAM/MPI documentation for details of how to use the **mpirun** command to send Slave Jobs to the Workstations. (Go to <http://www.lam-mpi.org/using/docs/> and click the **LAM/MPI User's Guide** link.)

5. Run **lamhalt**.

```
lamhalt
```

When all the Slave Jobs have ended, use **lamhalt** to kill the **lamd** daemon on each Workstation.

**Note**

There is a Job starter script that you can use instead that does all this for you. It is located in the `etc/MPI/` directory of the Netbatch installation (for example, `/nfs/site/gen/adm/netbatch/install/netbatch/etc/MPI/`).

Simply submit the master Job with the `--job-starter` switch, with the path to the MPI starter script as its argument.

6.10.1.2 Linda

If your Master Job uses Linda, you must edit the `$LINDA_HOME/bin/linda_rsh` file (where `LINDA_HOME` is the Linda installation directory) to force Linda to use the supplied Netbatch script instead of `rsh` (or Linda's own version of `rsh`).

To do this:

1. Open `$LINDA_HOME/bin/linda_rsh` in your favorite text editor.
2. Look for this line (close to the end of the file):

```
* ) exec /usr/bin/rsh $host $user -n "$@"
```
3. Replace it with these lines:

```
* ) if [ $__NB_JOBID ]; then
    exec $NS_HOME/sbin/rsh.nb $host $user -n "$@"
else
    exec /usr/bin/rsh $host $user -n "$@"
fi
```
4. Save and close the file.

The Master Job itself should simply read the list of Workstations from the `NB_PARALLEL_JOB_HOSTS` environment variable and insert them, **plus the hostname of the Workstation on which the Master Job is running (i.e., `localhost`)**, as arguments into the `ntsnet` command, which takes the following form:

```
ntsnet -nodelist "<node>[,...]"
```

where `node` is a Workstation hostname.

See <http://www.lindaspaces.com/downloads/lindamanual.pdf> for details of how to use the `ntsnet` command to send Slave Jobs to the Workstations



6.10.2 Submitting a Master Job

Note

A Parallel Master Job usually spends more time in the Wait State Queue than other Jobs, while it waits for Netbatch to accumulate the execution slots that it needs.

To submit a Master Job, type:

```
nbjob run --target <poolName> --queue <queueName> --
qslot <qslot> --parallel <parallel_reservation_string>
--parallel-topology-latency "max<=<int>"/ "max==<int>"
[--class <class_expression>]
[--parallel-slave-class <class_expression>]
[--parallel-job-constraints
"<expression>:<action>[:...][,...]" ]
<command>
```

where:

- **poolName** is the name of the Pool.
- **queueName** is the name of the Queue.
- **qslot** is the name of the Qslot.
- **parallel_reservation_string** is of the form:

```
"slots=[<min_slots>-]<max_slots>[, 
slots_per_host=<min_sph>[-<max_sph>]][, 
reservation_per_host={true|false}][, 
preserve_slot={true|false}][, 
exit_on_master_finish={true|false}][, 
slave_validator=<validate_script>]"
```

where:

- **min_slots** (optional) is the minimum number of execution slots required
- **max_slots** is the maximum number of execution slots required
- **min_sph** is the minimum number of execution slots required on a single host
- **max_sph** (optional) is the maximum number of execution slots required on a single host
- **reservation_per_host** indicates that the requested resource reservation will be made once for each scheduled executing host, regardless of the number of parallel slots scheduled for it.



The default value is **false**.

For example, if two Slave Jobs are scheduled to execute on a Host, each of which has a resource reservation of 10GB of memory:

- If **reservation_per_host** is **false**, 10GB will be reserved for each Job (20GB in total).
- If **reservation_per_host** is **true**, a total of 10GB will be reserved.

Note

*If **dedicated=true** in the Resource Reservation specification (specified with the **--class-reservation** switch), then no other Jobs may run on a host while a Job is running (except for Parallel Slave Jobs—multiple Slave Jobs that are part of the same Parallel Job can run on the same host).*

See [Section 6.9, Submitting a Job with Resource Reservation](#) for details of Resource Reservation.

- **preserve_slot** specifies whether an execution slot will be released (**false**) or not (**true**) when the Slave Job running in it ends. If **preserve_slot=true**, the execution slot can be reused.

The default is **false**.

This behavior can also be specified at the Slave Job level (but only if **preserve_slot=false** for the Master Job). To do this, add **--preserve-slot** to the **nbexec** command for the relevant Slave Job(s).

- **exit_on_master_finish** specifies what happens to Slave Jobs that are still running when the Master Job ends. If your Slave Jobs should no longer be running when the Master Job ends, set this to **true**. Otherwise, set it to **false** (the default).
- **validate_script** is the name of a script that is run to determine whether a particular Workstation can or cannot run one of the Job's Slave Jobs. The user is responsible for creating this script.

If the script determines that the Workstation is suitable, it should return zero. If not, it should return a non-zero value.

If the script returns a non-zero value, Netbatch excludes the Workstation and looks for a replacement.

How it works: Netbatch runs the script for each Workstation only after it has successfully reserved all the requested slots. If the script indicates that one or more Workstations are not suitable, the Job is resubmitted to



the wait queue. Netbatch then starts looking for available execution slots again. Ones for which validation failed are not considered.

- The **--class** and **--parallel-slave-class** switches specify the classes required by the Master and Slave Jobs, as follows:
 - If **--class** is specified without **--parallel-slave-class**, the specified class requirement applies to the Master Job and the Slave Jobs.
If you want to specify multiple acceptable classes or class expressions, but want all the slave Jobs to run on Workstations with the same class or that match the same class expression, use a colon (:) as the OR operator (and not ||). See **--class** in [nbjob run](#).
 - If both **--class** and **--parallel-slave-class** are specified:
 - ◆ The class requirement specified by **--class** applies to the Master Job.
 - ◆ The class requirement specified by **--parallel-slave-class** applies to the Slave Jobs.
- You can add a **--parallel-job-constraints** switch to ensure that your Job:
 - Does not use too many resources, or
 - Does not wait too long in the wait queueFor details, see [Appendix A.2.1, nbjob run](#) or [Appendix A.13, nbq](#).
- **command** is the name of the Master Job.

Note

The only mandatory part of parallel reservation string is slots=<min_slots>. All the rest is optional.

For Example

```
nbjob run --target linux_idc --queue proj1 --qslot
team1 --parallel "slots=2" myMasterJob
```

This specifies that the Parallel Job whose Master Job is myMasterJob needs at least two execution slots.

For Example

```
nbjob run --target linux_idc --queue proj1 --qslot
team1 --parallel "slots=2" myMasterJob
```



*This specifies that the Parallel Job whose Master Job is myMasterJob needs at least **two** execution slots.*

For Example

```
nbjob run --target linux_idc --queue proj1 --qslot
team1 --parallel "slots=2" --class "host!='itstl107'" 
myMasterJob
```

*This specifies that the Parallel Job whose Master Job is myMasterJob needs at least **two** execution slots, and that neither the Master Job nor any Slave Job may run on Workstation itstl107.*

For Example

```
nbjob run --target linux_idc --queue proj1 --qslot
team1 --parallel "slots=2" --class "host='itstl107'" -
--parallel-slave-class "@" myMasterJob
```

*This specifies that the Parallel Job whose Master Job is myMasterJob needs at least **two** execution slots, and that the Master Job must run on Workstation itstl107. (The --parallel-slave-class is required to specify that the -c requirement does **not** apply to the Slave Jobs.)*

For Example

```
nbjob run --target linux_idc --queue proj1 --qslot
team1 --parallel "slots=2" --class "CPUCount>=2"
--parallel-slave-class "BIGMEM" myMasterJob
```

*This specifies that the Parallel Job whose Master Job is myMasterJob needs at least **two** execution slots, and that:*

- *The Master Job must run a Workstation with at least two CPUs.*
- *Each Slave Job must run on a Workstation that supports the BIGMEM class.*

For Example

```
nbjob run --target linux_idc --queue proj1 --qslot
team1 --parallel "slots=4-6,slots_per_host=2,
reservation_per_host=true" myMasterJob
```

*This specifies that the Parallel Job whose Master Job is myMasterJob needs at least **four** execution slots, but no more than **six**, that **two** execution slots must be allocated on each host (so that the possible acceptable combinations are two or three hosts, each with two executions slots), and that the reservation will be made for every scheduled executing host.*



- **parallel-topology-latency** (optional) makes use of your .ntm file latency rankings. "**max<=<int>**" (or "**max==<int>**") defines the maximum latency rating allowed for running the job. (0 = no latency.)

Note

The Master Job does not end until all its Slave Jobs have successfully completed.

6.10.3 Modifying a Parallel Job

If a Parallel Job consists of a number of iterations, where later iterations require fewer execution slots than earlier ones, use the **nbjob pmodify** command to release the execution slots that are no longer required.

The alternative would be to split the Parallel Job into multiple Jobs, each of which would have to wait for the required number of execution slots to become available before it could be dispatched. With this command, after the Job has been dispatched for execution, no further waiting is required.

You can also use **nbjob pmodify** to increase the number of execution slots for the Job.

Reducing the Number of Execution Slots

To reduce the number of execution slots required by a Parallel Job, the Job itself should run the following command:

```
nbjob pmodify [--target <pool_name>|<host_name>]  
--shrink <number_of_slots>
```

where **number_of_slots** is the number of slots that are no longer required.

The output of the command is a list of hostnames that indicate the execution slots still available to the Job. If more than one execution slot on the same Workstation is assigned to the Job, its hostname is repeated the appropriate number of times.

Increasing the Number of Execution Slots

To increase the number of execution slots required by a Parallel Job, the Job itself should run the following command:

```
nbjob pmodify [--target <pool_name>|<host_name>]  
--grow <number_of_slots>
```

where **number_of_slots** is the number of additional slots that are required.



The output of the command is a list of hostnames that indicate the new execution slots. If more than one execution slot on the same Workstation is assigned to the Job, its hostname is repeated the appropriate number of times.

Note

With this switch, the command operates in blocking mode—it does not return all the required execution slots are available.

6.10.4 Parallel Job Log Files

For each executed Job (including the Master Job), Netbatch creates a separate output log file, in the standard format (`##Date-Time#.<machine>`). This can make it difficult to work out which log files belong to your Parallel Job.

You are therefore recommended to do one of the following:

- Use the `--log-file` switch (that is, use `--log-file <filename>`) when you submit a Job. If you do this:
 - ◆ The Master Job log file is called `filename`.
 - ◆ The Slave Job log files are called `filename:index`.
- Use the `--log-file` switch (that is, `--log-file <filename>`) with `nbjob prun` in the Master Job (where each Slave Job is executed by running a separate `nbjob prun` command). In this case, the Slave Job log file is called `filename`.

Note

Netbatch does not create a log file for the Master Job until all its Slave Jobs have successfully completed.

6.11 Submitting a Job Using VBatch

If you have a Job that requires an OS version that is not running on any of the Workstations in the Pool (for example, Windows in a Linux Pool, or a non-current version of SLES), you can use the VBatch feature. With this feature, Netbatch can instantiate a virtual machine running a specified OS image and run your Job on it.

To submit a Job like this, you use `nbjob run` with the `--image` switch. You also need to specify a class reservation expression so that Netbatch will know how many cores and how much memory your Job requires.



Submit your Job as follows:

1. Check if the OS version your Job needs is available as an image. Type:

```
nbstatus vm-image-cache
```

2. If the OS version is available, submit your Job using `nbjob run`:

```
nbjob run --class <class> --qslot <qslot> --image  
<image> --class-reservation <reservation_expression>
```

where:

- `image` is the OS image name (as shown in the output of `nbstatus vm-image-cache`).
- `reservation_expression` is the class reservation expression, which tells Netbatch how much memory and how many cores the Job requires.

6.12 Using Vlon to Run a Virtual Machine through Netbatch

You can run a virtual machine through Netbatch (as a Netbatch Job) for interactive use. You can also create clusters of VMs that work like mini-Pools. (You can submit a Job to a cluster, rather than having to send it to a specific machine.)

To start a virtual machine:

1. Find out the image name for the OS and version that you need. Type:

```
nbstatus vm-image-cache
```

2. **Optional:** create a cluster that the VM will belong to. (If you already have a cluster that you want to add the VM to, you can skip this step.) Type:

```
nbvm cluster-create [--target <pool_name>|<host_name>]  
[--cluster-name <cluster_name>]
```

where `cluster_name` is the name of the cluster.

The output includes the cluster ID, which you will need in a moment.

Note

You do not have to create a cluster—you can start a VM without specifying a cluster—see Step 3, below.

3. Start the VM. Type:

```
nbvm boot [--target <pool_name>|<host_name>]  
--image <image> --cores <cores>  
--memory <memory> [--cluster-id <cluster_id>]  
[--class <class_expression>] [--qslot <qslot>]
```



where:

- **image** is the OS image to be used.
- **cores** is the number of cores required.
- **memory** is the amount of memory required (e.g., **4** for 4GB).
- **cluster_id** is the ID of the cluster that the VM belongs to. If you do not specify a cluster, your default cluster will be used (and will be created if it does not yet exist). Your default cluster is called **<username>_DEFAULT_CLUSTER**.
- **class_expression** specifies the class of machine that the VM will run on.
- **qslot** is the Qslot to which the "Job" will be submitted. If you omit this switch, it will be submitted to the default Qslot.

You can stop the virtual machine using the **nbvm shutdown** command. You can remove a cluster using the **nbvm cluster-remove** command.

6.12.1 Submitting a Job to a VM Cluster

To submit a Job to a cluster of virtual machines, type:

```
nbjob run --target {<pool_name>|<hostname>}
--cluster-id <cluster_id> <options> <command>
```

where **cluster_id** is the ID of the cluster (which appears in the output of the **nbvm cluster-create** command and in **nbstatus vms**).

6.13 Using Job Profiles

6.13.1 Overview

If you find that you use the same options every time for the same type of Job, you can save time and effort by configuring Netbatch to use **Job Profiles**. For example, all Jobs that run a tool by Cadence require a license to run, and may have to be submitted to a special Queue and Qslot.

Job Profiles automatically match the Jobs you submit to the **nbjob run** arguments that they require. This way, all the Jobs that you send will always be submitted with the correct set of Netbatch options.

6.13.2 Enabling Job Profiling

In order to enable Job Profiling, you must:



1. Create a **Job Argument Mapping File** (or simply a **mapping file**). See [Section 6.12.3, Job Argument Mapping File](#).
2. Set the **NB_PROFILE_DIR** environment variable to point to the directory where you have created the mapping file.

For Example

*If you have created the mapping file in the directory ~/netbatch, then set the **NB_PROFILE_DIR** environment variable as follows:*

```
setenv NB_PROFILE_DIR ~/netbatch
```

Note

*If you do not set the **NB_PROFILE_DIR** environment variable, Netbatch will search for the mapping file in your home directory. If the mapping file is not found in your home directory then Netbatch will not use Job profiling.*

Note

*The **NB_PROFILE_FILE** environment variable is deprecated and its use is not encouraged. It is still supported in the current release, but at some point it will cease to be supported.*

6.14 Interactive Batch Jobs

6.14.1 Overview

An interactive batch Job is a Job that is dispatched for execution by Netbatch in a way that makes the Job appear as though it were executing on the submitting host machine.

Interactive Jobs are especially suited for batch execution of complex sequences of interdependent Jobs defined in a Makefile.

Note

Interactive Batch Jobs should not be confused with [Interactive Logon Sessions](#). In an interactive logon session, you logon to a Workstation through Netbatch. See [Section 6.15, Interactive Logon Sessions](#) for more information.

**Note**

As of version 6.3.1, Netbatch automatically senses the context in which an interactive or terminal Job is being executed, and runs it as either an interactive or terminal Job, whichever is appropriate.

Thus, there is no need to use --mode terminal. Whenever you want the Job to run as if it is running locally, use --mode interactive.

6.14.2 Submitting an Interactive Batch Job with nbjob run --mode interactive

Interactive Jobs are submitted using the `nbjob run` command with the `--mode interactive` switch.

The `--mode interactive` switch may be used in conjunction with any other valid switch. See [Section 6.4](#).

Note

To instruct nbjob run to handle signals locally, use --mode interactive-ls switch instead of --mode interactive. (With --mode interactive, Netbatch passes these signals to the remote process.)

When you submit a Job for interactive execution, it executes like a regular Job, with the following modifications:

- On the submitting host, the `nbjob run` command does not return until the completion of the Job.
- Upon termination of the Job, the `nbjob run` command assumes the exit status of the Job. If the Job fails to run, the `nbjob run` command assumes the Netbatch exit status that corresponds to the error that has occurred.
- The standard input stream of the submitting process (the one that runs the `nbjob run` command) is redirected to the standard input stream of the remote process that is executing the Job. As a result, any input provided at the submitting host is forwarded to the Job.
- The standard output and error streams of the remote process that executes the Job are redirected to the standard output and error streams of the submitting process. As a result, any output from the Job is displayed on the submitting host.
- Additional Job progress messages are written to the submitting process's standard error stream.



For Example

To Submit an interactive batch Job that runs tcsh, type:

```
nbjob run --target <poolName> --queue <queue> --qslot
<qslot> --mode interactive tcsh
```

Figure 26, below, shows an example of the interactive `nbjob run` command as it runs tcsh. Bold characters indicate input typed at the command line.

```
%nbjob run --target myPool --queue myQueue --qslot /ProjectB/team1 --mode
interactive tcsh
Your job has been queued (JobID 183, Class @, Queue queue, Slot qslot)
Mon Feb 13 12:36:08 2006: Your job has been queued (JobID 183, Class @,
Queue queue, Slot qslot)
Mon Feb 13 12:36:13 2006: Job 183 has started on itstl109
ls
bin@
build2.34_0067/
build2.34_0070/
conf/
doc/
etc@
gfile
jar@
java.policy*
lib@
log/
release@
exit
Mon Feb 13 12:36:28 2006: Job 183 has finished with exit status 0
```

Figure 26: Running tcsh as an Interactive Batch Job

Figure 27 below shows an example of piping input and output from an interactive batch Job that performs a CPU intensive task (in this case, a sort). Note that the fact that the sort command was run on a remote server is completely transparent (all of the Netbatch scheduling messages were written to the standard error).



```
cat someVeryLongFile | nbjob run --target <poolName> --queue<queue>
--qslot <qslot> --mode interactive
sort | head
Thu Nov 21 11:33:01 2002: Your Job has been queued (JobID 364, class
@, Queue queue, Slot qslot)
Thu Nov 21 11:33:03 2002: Job 364 has started on itst106
Thu Nov 21 11:33:03 2002: Job 364 has finished with exit status 0
someVeryLongContents1
someVeryLongContents2
someVeryLongContents3
someVeryLongContents4
someVeryLongContents5
someVeryLongContents6
someVeryLongContents7
someVeryLongContents8
someVeryLongContents9
someVeryLongContents10
```

Figure 27: Piping Input and Output from an Interactive Batch Job

6.14.3 Scheduling Interactive Batch Jobs

All the scheduling policies that apply to regular batch Jobs also apply to interactive batch Jobs. This means that like any other batch Job submitted to Netbatch, an interactive batch Job may not be execute immediately.

If response time is important for an interactive batch Job, make sure that you submit it to a Pool and to a Queue and Qslot that have been especially configured for that purpose by your Netbatch administrator.

6.14.4 Accelerating gmake with Interactive Batch Jobs

6.14.4.1 The Concept

Interactive batch Jobs and the **gmake** utility together provide a very powerful tool for executing complex sequences of interdependent CPU-intensive Jobs through Netbatch.

Interactive Netbatch commands can be transparently embedded in the command lines of a **Makefile**. The **gmake** utility can then be run on the Makefile in parallel execution mode (with the **-j** switch). At any given time, **gmake** will dispatch in parallel all of the Netbatch commands whose prerequisites are fulfilled.



Should any of the Jobs in a sequence fail, `gmake` will also stop. If the Makefile is correctly constructed, you can fix the problem that caused the failure and re-run `gmake`. `gmake` will resume the sequence where it left off.

For a the `gmake` manual, see:

http://www.gnu.org/manual/make-3.80/html_mono/make.html

6.14.4.2 A Simple Example

In this example we want to run three Jobs: **a**, **b**, and **c**.

Each Job has a product which it displays to the standard output.

Job **a** and Job **b** do not have any prerequisites.

Job **c** requires the concatenated output from **a** and **b** for its operation.

Figure 28 shows what the Makefile should look like:

```
%cat Makefile
c.out: a.out b.out c
        cat b.out >> a.out
        cat a.out | nbjob run --mode interactive ./c > c.out
a.out: a
        nbjob run --mode interactive ./a > a.out
b.out: b
        nbjob run --mode interactive ./b > b.out
clean:
        rm -f a.out b.out c.out
```

Figure 28: Makefile with Interactive Batch Jobs

In the Makefile, we have chosen to append `.out` to the name of each Job to denote the file that will contain the product of that Job.

In our example, we will use the scripts shown in **Figure 29** as the Jobs to be performed:



```
%cat a
#!/bin/sh
echo Result of a

%cat b
#!/bin/sh
echo Result of b

%cat c
#!/bin/sh
read -r var1
read -r var2
echo $var1 $var2 Result of c
```

Figure 29: Example Interactive Batch Jobs

Now run **gmake**:

```
% gmake -j 3
bin/nbjob run --mode interactive ./a > a.out
bin/nbjob run --mode interactive ./b > b.out
Thu Nov 21 17:28:34 2002: Your Job has been queued (JobID 53, Class @,
Queue Slot 1)
Thu Nov 21 17:28:34 2002: Your Job has been queued (JobID 54, Class @,
Queue Slot 1)
Thu Nov 21 17:28:37 2002: Job p6.53 has started on itst106
Thu Nov 21 17:28:37 2002: Job p6.53 has finished with exit status 0
Thu Nov 21 17:28:38 2002: Job p6.54 has started on itst107
Thu Nov 21 17:28:37 2002: Job p6.54 has finished with exit status 0
cat b.out >> a.out
cat a.out | bin/nbjob run --mode interactive ./c > c.out
Thu Nov 21 17:28:38 2002: Your Job has been queued (JobID 55, Class @,
Queue Slot 1)
Thu Nov 21 17:28:42 2002: Job p6.55 has started on itst106
Thu Nov 21 17:28:42 2002: Job p6.55 has finished with exit status 0
% cat c.out
Result of a Result of b Result of c
```

Figure 30: Output of gmake Running Interactive Batch Jobs

The output of the **gmake** command is shown in in [Figure 30](#), above. Note that Jobs **a** and **b** were dispatched and executed in parallel on two different Netbatch servers.



6.14.5 Interactive Batch Job Messages and Exit Status

The following messages are written to standard error during submission and execution of an interactive batch Job:

- When a Job is submitted:

```
<Day Date Time Year>: Your Job has been queued (JobID  
<JobId>, Class <class>, Queue <queue> Slot <qslot>)
```

- When a Job starts executing:

```
<Day Date Time Year>: Job <JobId> has started on  
<server>
```

- When the Job terminates:

```
<Day Date Time Year>: Job <JobId> has finished with exit  
status <exitStatus>
```

As a rule, the `nbjob run --mode interactive` command exits with the exit status of the Job. There are two exceptions to this rule:

- If the Job cannot run. For example, if the executable to be run is not found (-4), or if the Job was canceled before it could execute (-8).
- If the Job cannot be submitted. For example, if there is a network failure, or if the Netbatch Pool Master server is down (-12).

6.14.6 Special Aspects of Interactive Jobs

6.14.6.1 Cancelling an Interactive Batch Job

Interactive batch Jobs can be cancelled in a number of ways:

- Using the `nbjob remove` command
- By killing the `nbjob run` process
- By pressing **Ctrl-C** during operation of the `nbjob run` command (unless `--mode interactive-ls` is used)

Cancelling the Job will cause `nbjob run` to exit with status -8.

Note

*If **Ctrl-C** is pressed during the operation of `gmake`, all of the running interactive Jobs will be cancelled, and the `gmake` utility tool will terminate.*



6.14.6.2 Resubmitting an Interactive Batch Job

The input to and output from an interactive Job is not persisted by Netbatch. Recalling or re-submitting a running interactive Job is therefore not allowed, as the input and output would be lost. If an interactive Job is still in the waiting queue, it may be re-submitted in the usual way.

6.14.6.3 Suspending an Interactive Batch Job

An interactive batch Job can be suspended using the `nbjob suspend` command. Input streamed to the suspended Job is buffered by Netbatch.

To resume the suspended interactive Job, use `nbjob resume`.

Note

*Netbatch now passes **Ctrl-Z** to the remote Job (unless `--mode interactive-ls` is used). (In earlier versions, pressing **Ctrl-Z** during the operation of the `nbjob run` command suspended the `nbjob run` command's process, but did not affect the remote Job.)*

6.14.7 Netbatch Version

Interactive Job submission is available on Netbatch Pools and `nbjob run` clients of version 6.1 and higher. Interactive Netbatch Jobs should not be submitted or moved to earlier version Netbatch pools.

Virtual Netbatch pools of version 6.1 and higher correctly handle interactive Netbatch Jobs.

To find out which version of Netbatch Pool Master is running your Pool, run any `nbstatus` command. For example:

```
nbstatus jobs --target <poolName>
```

Figure 31, below shows the output of this command. The version number is shown in bold.



PPM on itstl109						
Version 7.1.0_0166_03						
On since 02/13/2006 10:37:12						
Time now 02/13/2006 12:54:33						

Status	Jobid	Class	Qslot	User	Cmdline	workstation
Run	184	@	/1	martinpx	tcsh	itstl109

Figure 31: Version of Netbatch Pool Master

To find out which version of the Netbatch `nbjob` client you are using, type:

```
nbjob -v
```

```
Client version: netbatch release 7.1.0_0166_03 - Feb 6 2006
Server version: NetBatch 7.1.0 (Jan 16 2005)
```

Figure 32: Version of Netbatch nbjob Client

6.15 Interactive Logon Sessions

In an interactive logon session, you logon to a Workstation through Netbatch.

Note

Interactive Batch Jobs should not be confused with *interactive logon sessions*. An *interactive batch Job* is a batch Job that has its input and output redirected to the submitting host, whereas an *interactive logon session* allows you to logon to a Workstation through Netbatch.

To begin an interactive logon session, use `nbjob run`'s `--mode interactive` switch, and submit the Job to a specially designated logon Queue. (Ask your Netbatch Administrator what this Queue is called.)

For Example

```
nbjob run --target <pool> --mode interactive --queue
<logon_queue> tcsh
```



6.16 Terminal Jobs

Terminal Jobs allow you to run interactive applications that manipulate user input and screen display via a terminal device through Netbatch. Examples include vi, less, emacs, and tcsh.

Note

As of version 6.3.1, Netbatch automatically senses the context in which an interactive or terminal Job is being executed, and runs it as either an interactive or terminal Job, whichever is appropriate.

Thus, there is no need to use --terminal. Whenever you want the Job to run as if it is running locally, use --mode interactive. (See [Section 6.14, Interactive Batch Jobs](#).)

To start a terminal Job, type:

```
nbjob run [other options] --mode terminal <command>
```

For Example

```
nbjob run --target linux2 --mode terminal tcsh
```

This opens a tcsh session through Netbatch. tcsh will run on a Workstation in the linux2 Pool.

For Example

```
nbjob run --target linux2 --mode terminal emacs
```

This runs emacs through Netbatch. emacs will run on a Workstation in the linux2 Pool.

6.16.1 Special Aspects of Terminal Jobs

6.16.1.1 Delays and Interruptions

When a Terminal Job is submitted, it is scheduled for execution like any other Job, and therefore may not be executed immediately.

Once it is running on a Workstation, the application behaves as if it was running locally, but with the following differences:

- Because the application is running on a remote machine, interaction may be subject to delays caused by network latency.



- It can still be niced, suspended, or killed if thresholds are exceeded on the Workstation. Therefore it is recommended to use Resource Reservation to minimize the chances of this happening. (See [Section 6.9, Submitting a Job with Resource Reservation](#).)

Note

Terminal Jobs cannot be resubmitted once they have started running.

6.16.1.2 Running a Shell as a Terminal Job

When you run a shell, such as tcsh, as a terminal Job, it looks very similar to logging in to a Workstation using telnet. However, it is not the same—there are a number of differences, among them the fact that the startup scripts in your home directory are not executed.

6.16.1.3 Job Completion

When a terminal session (or other application that spawns child processes) running as a terminal Job ends, its child processes (that is, commands that were issued within it) become regular batch Jobs, and are treated as such by Netbatch. However, they cannot be resubmitted or moved, thought they can be cancelled.

This is explained in greater detail in [Section 3.3.2, Job Completion Detection](#).

6.17 ClearCase Jobs

ClearCase Jobs (that is, Jobs that use the IBM ClearCase environment) must be submitted to a Queue or Qslot that the Netbatch Administrator has designated and configured especially for ClearCase Jobs.

Also, an additional step must be taken before submitting ClearCase Jobs.

To submit a ClearCase Job:

1. Set the Netbatch working directory environment variable to /tmp:

```
setenv NBWD /tmp
```

(Netbatch cannot "see" or write to your present working directory, so it needs **NBWD** to be set so that it can initialize correctly, even though it will not actually use /tmp as the working directory.)

2. Submit your Job to the Queue or Qslot designated by the Netbatch administrator for ClearCase Jobs, and use the **--log-file** and **--log-file-dir** switches to specify a location for the log file:



```
nbjob run --queue <queueName>
[--qslot <QslotName>] --log-file-dir <path>
--log-file <filename>
```

(Netbatch cannot write to your present working directory, so you must specify a log file path and name that it *can* write to.)

Note

When the Job is dispatched for execution on a Workstation, a special starter script sets the view on the Workstation to your current working view.

To submit additional ClearCase Jobs, simply repeat Step 2.

6.18 Submitting a Job when Workstations Might not Have Access to Required Files

In some cases, when you submit a Job, the Workstation to which it is dispatched for execution does not have access to all the files required by the Job, for example:

- When submitting a Job to a cross-site Virtual Pool
- When submitting a Job directly to a physical Pool at a different site
- When the required files or directories are on a local filesystem

In such cases, use **START** or rsync to copy the required files or directories to a location that is accessible by the Workstations in the Pool to which you are submitting the Job.

6.19 Using Job Constraints

You can add a **--job-constraints** switch to ensure that your Job:

- Does not use too many resources, or
- Does not wait too long in the wait queue

For details, see [Appendix A.2.1, nbjob run](#) or [Appendix A.13, nbq](#).



6.20 Logging Job Resource Usage Data

You can add a `--dumpRusage` switch to log a Job's resource usage information to a file. This is useful for problem-solving.

```
nbjob run --target <pool_name> --dumpRusage  
<interval>:<path> <command>
```

where:

- `interval` specifies how often data is written to the file.
- `path` is the path to the file.

If the file already exists, it is appended to.

Note

You can also use the `nbprofile` command to do the same thing (without having to run a command through Netbatch). See [Appendix A.12, nbprofile](#).

6.21 Specifying a Retry Timeout

In certain situations, Netbatch cannot handle a Job submission request, for example, if:

- The Pool Master is down.
- A request limit has been reached. (If this happens, you see a `Service denied` message.)
- The Pool Master is under heavy load.

You can add the `--retry-timeout` switch to commands to specify that the command client should keep trying to submit the command for the specified period.

For Example

```
nbjob run --retry-timeout 30m myJob
```

If Netbatch cannot handle the command when you submit it, the command client keeps trying for 30 minutes.



6.22 Conditionally Resubmitting a Job on Finish

You can specify that if your Job finishes with a certain exit code, it should be modified and resubmitted. To do this, use the `--on-job-finish` switch:

```
nbjob run --on-job-finish "{<exit_code>|<expression>}:  
Modify('<parameter>=<expression>'[;...]):  
Requeue(<max_attempts>)" <other_switches> <command>
```

where:

- **exit_code** is a numerical Job exit code (see [Exit Status](#)) or one of the following macros:
 - **NBErr** - all Netbatch errors except -8, -9, -10, and -11
 - **Nexit** - all negative Job exit values
 - **Pexit** - all positive Job exit values
 - **NZexit** - all non-zero Job exit values
- **expression** is an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:
 - **attribute** is any [nbstatus jobs](#) field (though typically, you will use **ExitStatus**, **PreExecExitStatus**, and **PostExecExitStatus**).
 - **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")
 - ◆ A logical operator: &&, ||, or !
 - ◆ A comparison operator: <, <=, >, >=, !=, or ==
 - ◆ A non-strict operator: **is** or **isnt**
 - ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
 - ◆ A conditional operator: ? or :
 - **value** is a number, a string, or another boolean expression.

Expressions can be grouped together using parentheses.

- In the **Modify** section:
 - **parameter** is a valid **nbjob modify** parameter (not including Queue).
 - **expression** specifies the value that is assigned to the parameter. **expression** can include any [nbstatus jobs](#) field.

Examples:



```
Modify('class="c2"')
Modify('priority=priority+1')
```

6.23 Enabling Checkpointing

To enable checkpointing for a Job (using a tool such as DMTCP), submit it with the following switches:

```
nbjob run --checkpoint-tool <tool_name>
[--checkpoint-tool-data <string>]
[--checkpoint-directory <path>]
[--checkpoint-interval <time>[<s|m>]]
[--num-of-checkpoints <number>]
<other_switches> <command>
```

where:

- **tool_name** is the name of the checkpointing tool (for example, **dmtcp**).
- **string** is the data to be passed to the checkpointing tool. Each tool defines its own specific syntax for this string.
- **path** is where the checkpoint files are to be stored. If not specified, the files are stored in the Job's working directory.
- **time** is how often a checkpoint should be created, in seconds (**s**) or minutes (**m**). If no units are specified, the time is in minutes. If the switch is omitted, a checkpoint is created every 15 minutes.
- **number** is the number of checkpoints to save. Older checkpoints will be deleted. If number is 0, all checkpoints are saved. If the switch is omitted, the last five checkpoints are saved.

Once a Job has been submitted with checkpointing enabled, you can:

- Use the **nbjob checkpoint**: command to create a checkpoint manually for one or more Jobs.
- Use **nbjob recall** with the **--restore-from-checkpoint** or **--restore-from-last-checkpoint** switches to rerun the Job from a particular checkpoint.
- View information about the Job's checkpoint in the **Checkpoint*** fields in the output of **nbstatus jobs**.

You can submit the job with the **--on-job-finish** switch as well to resubmit it when it ends to run it again from the last checkpoint.



CONFIDENTIAL

Netbatch
v8.2.2

Revision 4

Submitting Jobs to Netbatch



7 Using Netbatch Environment Variables

7.1 Overview

Environment variables can be used to control various aspects of Netbatch behavior.

You do not have to set all the environment variables. Netbatch will use just the ones that you have set. Often you set environment variables (sometimes without being aware of it) by running scripts supplied to you by members of your group.

Environment variables can be used for:

- [Setting the Netbatch Job Execution Environment](#), see [Section 7.2](#)
- [Setting Job Licensing Parameters](#), see [Section 7.3](#)
- [Setting Default Command Parameters](#), see [Section 7.4](#)

As well as the Netbatch environment variables that you can set, there are a number of environment variables that Netbatch sets when a Job is dispatched for execution. These are described in [Section 7.6, Environment Variables Set by Netbatch when a Job Is Run](#).

Note

An nbjob run command-line option usually overrides an environment variable that has been set to specify the same parameter.

In most cases (but not all), the environment variable overrides any default values that the Netbatch Administrator has set for the parameter.

See [Appendix B: Precedence of Configuration Directives, Environment Variables, Job Profiles, and Command-Line Switches](#).

7.2 Setting the Netbatch Job Execution Environment

7.2.1 Netbatch Configuration Directory

For correct operation of all Netbatch commands, set the **NBCONF** environment variable to the location of the Netbatch configuration directory.

To specify the Netbatch configuration directory:

1. Ask your Netbatch Administrator where the Netbatch configuration directory is located.



2. Set the value of the **NBCONF** environment variable to this location.

If you do not set **NBCONF**, Netbatch assumes that the Netbatch configuration directory is at its default location of `/usr/local/lib/netbatch/conf`.

7.2.2 Default Job Output Directory

The **NBWD** environment variable specifies one or more backup directories where Netbatch sends Job output if it cannot access your working directory when your Job ends.

Note

*You are recommended to specify multiple directories in **NBWD**, so that if your Job is dispatched to a Workstation at a different site (which can happen with Virtual Pools and Global Pools), Netbatch has more chance of finding a directory to which it can write the Job log file.*

When a Job is submitted to Netbatch, `nbjob run` records the user's current working directory, along with the value of the **NBWD** environment variable. The Workstation that receives the Job for execution first verifies that it can write to the current working directory. If this verification fails, the Workstation defaults to the first directory specified in the **NBWD** environment variable. If it cannot write to the first one, it tries the second, and so on, until it finds a directory that it can write to. If the Workstation cannot write to any of the directories specified in **NBWD**, the Job will not run, and an exit status of -31 is recorded.

Note

*If you do not set **NBWD**, Netbatch automatically sets it (when a Job is run, in the Job's own environment) to the directory where the `nbjob run` command was run.*

Note

*On Windows, **NBWD** must contain a single Windows path, and nothing else. Multiple values are not supported.*



7.2.3 Project Name

The Netbatch Pool Master maintains a special log file that records all the Jobs that are run in the Pool (usually called `matbatch.log`). The Jobs that you submit to the Pool can be tagged in the log file with a Project Name. This helps the Netbatch administrator track resource usage in the Pool.

The **NBPROJ** environment variable specifies the project name that is assigned to Jobs that you submit to the Pool.

You do not need to set the **NBPROJ** variable yourself. If project names are used at your site, then the Netbatch administrator or Design Automation group should set this variable automatically for you.

7.2.4 Job File Permissions

A running Job can create two types of files:

- A Job log file—This file is always created and maintained by Netbatch. The Job log file contains the standard output and error generated by the Job, along with a header and footer generated by Netbatch. See [Section 6.5.7, Job Output \(Log File\) Location and Options](#).
- Files created by the Job itself—Not all Jobs create their own files. This depends on the application that is run as a Job.

By default, Netbatch creates the Job log file with UNIX permissions of 777 (-rwxrwxrwx), and files created by the Job itself have UNIX permissions of 666 (-rw-rw-rw-).

The **NBUMASK** environment variable allows you to change the permissions for these files. For example, you can set it so that other members of your group can overwrite your Job's log file.

The **NBUMASK** is an octal mask that is subtracted from the default permissions 777 or 666.

If you do not set the **NBUMASK** environment variable, a default mask of 022 is used, so the Job log file is created with permissions of 755 (-rwxr-x-r-x), and files generated by the Job have permissions of 644 (-rwx-r--r--).

For Example

You can loosen the access restrictions to the generated files by setting NBUMASK to 000, as follows:

```
setenv NBUMASK 000
```



Job log files will now be created with permissions of 777, and files created by the running Job will have permissions of 666.

Likewise, you can further restrict access to the generated files by using a mask that is greater than 022. For example:

```
setenv NBUMASK 033
```

7.2.5 Mail Notification of Failed Jobs

Use the **NBMAILONERR** environment variable to receive email notification about a Jobs that do not succeed.

Setting **NBMAILONERR** is very useful if you are running thousands of regression Jobs and are only interested in knowing about failed cases.

Set this environment variable before submitting Jobs with **nbjob run**, as follows:

```
setenv NBMAILONERR <error type ...>
```

where **<error type>** can be any integer (error code), or one or more of the macros listed in the following table.

Table 8: NBMAILONERR <error type> Macros

Macro	Description
NBErr	All Netbatch errors except -8, -9, -10, and -11
Nexit	All negative Job exit values
Pexit	All positive Job exit values
NZexit	All non-zero Job exit values

For Example

```
setenv NBMAILONERR "Pexit -2"
```

Instructs Netbatch to send a mail notification on all Jobs that exit with an exit status that is either positive or equal to -2.

Mail messages are sent to the user that submitted the Job that matches the specified criteria.



7.3 Setting Job Licensing Parameters

7.3.1 License Keyfile Location

The **NB_LICENSE_FILE** environment variable is used to specify the location (full path and filename) of the keyfile(s) containing the license feature(s) that are required by the Jobs you want to run on Netbatch.

Setting the **NB_LICENSE_FILE** environment variable ensures that Netbatch correctly runs Jobs that require licenses. See [Section 6.8.3, Submitting Jobs that Require Licenses](#).

If you specify more than one keyfile path, use colons (:) as delimiters.

Note

You can set NB_LICENSE_FILE to define multiple keyfiles (which can be on multiple license servers). Netbatch searches for licenses in the locations specified in NB_LICENSE_FILE in the order in which they are specified there.

Note

If you use license aliases, you do not need to define the NB_LICENSE_FILE environment variable.

See [License Aliases](#) for more details.

You can use nbjob run's --license-keyfile switch instead of setting NB_LICENSE_FILE. See [Section 6.8.3.2, Submitting a Job by Specifying Keyfile and Feature Name](#).

7.3.2 Default License Requirements

The **NBLICENSES** environment variable is used to specify the license requirements for every Job that you submit to Netbatch. It is the same as adding `--license <licenses>` to every `nbjob run` command (that is, the value that you set **NBLICENSES** to uses the same syntax as the expression that follows the `--license` switch). For more details of the syntax of the `--license` switch, see [Section 6.8.3, Submitting Jobs that Require Licenses](#).

If you run a Netbatch command and explicitly specify different license requirements with the `--license` switch, this overrides the **NBLICENSES** environment variable.

**Note**

*Setting **NBLICENSES** causes the same license requirements to be specified for every Job.*

Using Job Profiles is a much more flexible way of specifying license requirements, as it allows you to specify different requirements for each Job type. See [Section 6.14, Using Job Profiles](#).

7.4 Setting Default Command Parameters

7.4.1 Default Pool Name

The **NBPOOL** environment variable is used to specify the Pool(s) that Netbatch uses by default for all Netbatch commands. It is the same as adding **--target <PoolName>** to every Netbatch command.

For Example

*To set **myPool** as the default Pool name, from tcsh, type:*

```
setenv NBPOOL myPool
```

If you run a Netbatch command and explicitly specify a different Pool or host name with the **--target** switch, these specifications override the **NBPOOL** environment variable.

7.4.2 Default Queue

The **NBQUEUE** environment variable is used to specify the Queue that Netbatch uses by default for Netbatch commands. It is the same as adding **--queue <queueName>** to every Netbatch command where it applies.

For Example

*To set **myQueue** as the default Queue, from tcsh, type:*

```
setenv NBQUEUE myQueue
```

If you run a Netbatch command and explicitly specify a different Queue with the **--queue** switch, this specification overrides the **NBQUEUE** environment variable.

7.4.3 Default Qslot

The **NBQSLOT** environment variable is used to specify the Qslot that Netbatch uses by default for Netbatch commands. It is the same as adding **--qslot <qslot>** or to every Netbatch command where it applies.

**For Example**

To set `myQslot` as the default Qslot, from tcsh, type:

```
setenv NBQSLOT myQslot
```

If you run a Netbatch command and explicitly specify a different Qslot with the `--qslot` switch directive, this specification overrides the **NBQSLOT** environment variable.

7.4.4 Default Physical Pools (for Jobs Submitted to a Virtual Pool)

The **NB_POOLS** environment variable is used to specify the Physical Pool(s) in the Virtual Pool that you want to use. It is the same as adding a `--remote-pools` switch to every `nbjob run` command.

Syntax:

```
setenv NB_POOLS { "[ - ]<phys_pool>" |  
" [ - ]<alias>" } [ , ... ]
```

You can specify Pools by:

- Inclusion—only the listed physical Pools will be considered when deciding which Pool to send the Job to.
- Exclusion—precede Pool/alias names with `-`. All relevant physical Pools will be considered except those that are listed.
- A combination—if there are both included and excluded Pools/aliases in the list, the excluded ones are excluded from the list of included Pools.

You can use this to specify that a specific physical Pool should be excluded from the set of Pools represented by an alias.

This situation can also arise when the request uses inclusion and a Job Profile uses exclusion, or vice-versa.

The order determines the order in which the Pools will be tried. Netbatch will try the first Pool, and will only try the next one if the previous one has reached its **WaitJobsLimit** or **MaxJobsLimit**.

If an alias is specified, Netbatch uses its existing algorithm to decide which Pool to send the Job to. (Multiple Pools can have the same alias. For example, all the Physical Pools at the same site as the Virtual Pool Master might have the alias `local`.)

**For Example**

To specify that all your Jobs that are submitted to a Virtual Pool should only be submitted to the Physical Pools with the alias `local`, type:

```
setenv NB_POOLS local
```

The `--remote-pools` switch overrides `NB_POOLS`.

7.4.5 Default Class

The **NBCLASS** environment variable is used to specify the Class that Netbatch uses by default for the `nbjob run` command. It is the same as adding `--class <class>` to every `nbjob run` command.

Any valid Class specification string may be set as a default in the **NBCLASS** environment variable. This includes boolean expressions.

For Example

To set "BIGMEM && Linux" as the default Class, from tcsh, type:

```
setenv NBCLASS "BIGMEM && Linux"
```

If you run `nbjob run` and explicitly specify a different Class with the `--class` switch, this specification overrides the **NBCLASS** environment variable.

7.4.6 Default Re-queue Policies

The **NB_ON_JOB_FINISH** environment variable specifies that if Job ends with a certain exit code or if a specified expression is true for it, the specified action is performed.

You can use it to protect your Jobs from blackhole machines—if you do not set **NB_ON_JOB_FINISH** and one of your Jobs is sent to a Blackhole machine, your Job will be lost.

When sending Jobs to a Virtual Pool, it can also be used to prevent a Job being sent back to a physical Pool on which it failed (for example, because the Job does not have access to the files that it needs in that Pool).

Note

You can use `nbjob run`'s `--on-job-finish` switch to achieve the same results.



For the settings in **NB_ON_JOB_FINISH** to take effect, the variable must have already been set when a Job is submitted with **nbjob run**.

Use **NB_ON_JOB_FINISH** as follows:

```
setenv NB_ON_JOB_FINISH
" {<exit_code> | <expression>} :<operation>[ :... ]"
[ ,... ]
```

where:

- **exit_code** is a numerical Job exit code (see [Exit Status](#)) or one of the following macros:
 - **NBErr** - all Netbatch errors except -8, -9, -10, and -11
 - **Nexit** - all negative Job exit values
 - **Pexit** - all positive Job exit values
 - **NZexit** - all non-zero Job exit values
- **expression** is an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:
 - ◆ **attribute** is any [nbstatus jobs](#) field (though typically, you will use **ExitStatus**, **PreExecExitStatus**, and **PostExecExitStatus**).
 - ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")
 - ◆ A logical operator: &&, ||, or !
 - ◆ A comparison operator: <, <=, >, >=, !=, or ==
 - ◆ A non-strict operator: **is** or **isnt**
 - ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
 - ◆ A conditional operator: ? or :
 - ◆ **value** is a number, a string, or another boolean expression.

Expressions can be grouped together using parentheses.

For example,

```
ExitStatus== -303 && PreExecExitStatus==2
```

- **operation** is one of the following:
 - ◆ **Requeue(<max_attempts>)** - the Job is requeued up to **max_attempts** times.



- ♦ **Exclude(<period>{s|m|h})** - if the Job was submitted to a Virtual Pool, exclude the last resource (Pool) that the Job was dispatched to from those that it can be sent to, for the specified time. (If the Job was *not* submitted to a Virtual Pool, exclude the last Workstation that the Job was dispatched to from the list of Workstations that the Job can run on, for the specified time.)

If you specify both **Requeue** and **Exclude** operations, **Exclude** must come first.

If you specify multiple **exit_code|expression:operations**, only the operation for the first match will be performed.

The **nbjob run --on-job-finish** switch overrides the **NB_ON_JOB_FINISH**.

7.4.7 Default Job Execution Limits

Note

*The **NB_EXEC_LIMITS** environment variable is deprecated. Use Job constraints instead. See **--job-constraints** in [nbjob run](#).*

The **NB_EXEC_LIMITS** environment variable is used to specify the Execution Limits that Netbatch uses by default for the **nbjob run** command. It is the same as adding **--exec_limits <execLimits>** to every **nbjob run** command. See [Section 6.5.12, Job Execution Limits](#).

For Example

To set 45m:1.5h as the default Job Execution Limits, from tcsh, type:

```
setenv NB_EXEC_LIMITS 45m:1.5h
```

If you run a Netbatch command and explicitly specify different Job Execution Limits with the **--exec_limits** switch, this specification overrides the **NB_EXEC_LIMITS** environment variable.

Note

Execution limits are rounded up to the nearest minute.



7.4.8 Default Job Hung Limits

Note

*The **NB_HUNG_LIMITS** environment variable is deprecated. Use Job constraints instead. See --job-constraints in [nbjob run](#).*

The **NB_HUNG_LIMITS** environment variable is used to specify the Hung Limits that Netbatch uses by default for the `nbjob run` command. It is the same as adding `--hung-limits <hungLimits>` to every `nbjob run` command. See [Section 6.5.13, Job Hung Limits](#).

For Example

To set 45m:1.5h as the default Job Hung Limits, from tcsh, type:

```
setenv NB_HUNG_LIMITS 45m:1.5h
```

If you run a Netbatch command and explicitly specify different Job Hung Limits with the `--hung-limits` switch, this specification overrides the **NB_HUNG_LIMITS** environment variable.

Note

Hung limits are rounded up to the nearest minute.

7.4.9 Default Pre-execution Program

The **NB_PRE_EXEC** environment variable specifies a pre-execution program that is executed immediately before every Job.

For Example

To specify that the script ~/myScript be executed immediately before every Job, type:

```
setenv NB_PRE_EXEC ~/myScript
```

If you run a Netbatch command and explicitly specify a different pre-execution program with the `--pre-exec` switch, this specification overrides the **NB_PRE_EXEC** environment variable.

7.4.10 Default Post-execution Program

The **NB_POST_EXEC** environment variable specifies a post-execution program that is executed immediately after every Job.

**Note**

The post-execution program is executed even if the Job is killed or suspended.

For Example

To specify that the script ~/myScript be executed immediately after every Job, type:

```
setenv NB_POST_EXEC ~/myScript
```

If you run a Netbatch command and explicitly specify a different post-execution program with the **--post-exec** switch, this specification overrides the **NB_POST_EXEC** environment variable.

7.4.11 Default Job Starter Program

The **NB_JOB_STARTER** environment variable specifies a Job starter program that performs certain operations and then runs the Job.

For Example

To specify that the script ~/myJobStarter be used as a Job stater for every Job, type:

```
setenv NB_JOB_STARTER ~/myJobStarter
```

If you run a Netbatch command and explicitly specify a different Job starter program with the **--job-starter** switch, this specification overrides the **NB_JOB_STARTER** environment variable.

7.4.12 Setting the SIGKILL (Kill) Timeout

To specify the time that Netbatch waits before sending the SIGKILL signal to your Jobs to kill them (after the SIGTERM signal is sent), set the **NB_SIGKILL_TIMEOUT** environment variable.

You might want to do this to allow your Jobs more time to perform cleanup tasks (such as closing files) than that allowed by the default set by the Netbatch Administrator.

If this variable's value is greater than the default maximum set by the Netbatch Administrator, then the default maximum value is used.

For Example

```
setenv NB_SIGKILL_TIMEOUT 150
```



7.4.13 Setting the SIGSTOP (Suspend) Timeout

To specify the time that Netbatch waits before sending the SIGSTOP signal to your Jobs to suspend them (after the SIGTSTP signal is sent), set the **NB_SIGSTOP_TIMEOUT** environment variable.

If this variable's value is greater than the default maximum set by the Netbatch Administrator, then the default maximum value is used.

For Example

```
setenv NB_SIGSTOP_TIMEOUT 150
```

7.4.14 Default Task Name

If you want the same task name to be assigned to all your Jobs, set the **NBTASK** environment variable to the task name.

For Example

```
setenv NBTASK myTask
```

7.4.15 Enabling Wash Groups

If you belong to more than 16 NIS groups, you can use the following environment variables to specify the groups to be applied to your Jobs:

- **NB_WASH_GROUPS**—specifies the groups to be applied to your Jobs (as a comma-separated list of up to 15 groups).
- **NB_WASH_ENABLED**—specifies that these groups should be applied to every Job that you submit. (This is the same as adding the **--wash** switch to **nbjob run**.)

For Example

```
setenv NB_WASH_GROUPS group1,group2,group3  
setenv NB_WASH_ENABLED
```

Note

A Job's owner must belong to all of its wash groups according to the group map.



7.4.16 Setting Properties for Your Jobs

The Qslot that a Job is submitted to may have configured rules that require you to specify values for certain properties when submitting a Job. Optionally, you can also supply values for any configured property.

If you omit required properties when trying to submit a Job, you will receive an error message that lists the properties that you must supply values for.

To set properties for Jobs, you can do one of the following:

- Add the `--properties` switch when submitting a Job with `nbjob run`
- Set the **NB_PROPERTIES** environment variable

Note

If a property is set in --properties and in NB_PROPERTIES, the one set in --properties has precedence. Other properties are merged.

The syntax for **NB_PROPERTIES** is as follows:

```
setenv NB_PROPERTIES "<prop_name>=<value>[ , . . . ]"
```

where `prop_name` is the name of a property. `value` is the property's value.

7.5 Environment Variables for Netbatch on Windows

The following environment variables are used to enable Netbatch Jobs to run correctly on Windows Workstations:

- **NB_NOMOUNT**

See [Section 6.3.1.4, Disabling Drive Mapping](#).

7.6 Environment Variables Set by Netbatch when a Job Is Run

Caution

Do not attempt to set any of the following environment variables (except NBWD). Doing so may cause unexpected results.



7.6.1 __NB_JOBID

When Netbatch dispatches a Job for execution, it sets **__NB_JOBID** (in the Job's environment) to:

```
<pool name>.<jobID>
```

where:

- **pool name** is the name of the Pool that the Job was submitted to.
- **jobID** is the Job's ID.

You can use the value of **__NB_JOBID** to find out a Job's JobID. This is useful for tracking Jobs that you have submitted. See [Section 8, Tracking Job Status](#) for more details.

7.6.2 NBWD

If you do not set **NBWD**, Netbatch automatically sets it (when a Job is run, in the Job's own environment) to the directory where the **nbjob run** command was run.

7.6.3 NB_PARALLEL_JOB_HOSTS

NB_PARALLEL_JOB_HOSTS lists the hosts on which an execution slot has been allocated for executing a Parallel Job. The host names are used with **nbexec** to dispatch the Slave Jobs.

Note

If two or more execution slots are used on the same Workstation, its name appears the corresponding number of times in the list.

*This list does **not** include the host that runs the Master Job.*

The hosts are delimited by spaces.

7.6.4 __NB_PARALLEL_POOL_HOST

__NB_PARALLEL_POOL_HOST contains the name of the Pool Master to which the hosts listed in **NB_PARALLEL_JOB_HOSTS** belong.

The **nbexec** command sends a Slave Job (part of a Parallel Job) to the Pool Master specified in **__NB_PARALLEL_POOL_HOST** for validation before it is executed on the specified host.



7.6.5 NB_SUBMIT_PWD

_NB_SUBMIT_PWD contains the value of the PWD environment variable when you submitted the Job (that is, your working directory when the Job was submitted).

This is useful for ClearCase Jobs, where the directory does not exist until after the view is set. (See [Section 6.18, ClearCase Jobs](#).)

7.7 Environment Variables Set by the Netbatch Administrator

7.7.1 **NBFEED_CONF**

The Netbatch Administrator sets this for you to specify a common configuration file that will be used for all Netbatch Feeders that you start.



8 Tracking Job Status

8.1 Overview

Every Job submitted to a Pool has a specific Netbatch state associated with it. The Netbatch state indicates whether the Job is waiting for execution, is running, or has already run. The Netbatch state can also indicate other special conditions of a Job.

Conceptually each Queue in the Pool can be seen as being subdivided into six **State Queues**, each of which contains those Jobs that are currently in a particular state.

The six State Queues are:

- **Run State Queue**—contains all the Jobs that are running, including ones that have started running and have been suspended or niced.
- **Wait State Queue**—contains all the Jobs that are waiting to be run.
- **Moved State Queue**—contains all the Jobs that have been moved to a different Pool, and which have not yet completed.
- **Notify State Queue**—contains all the completed Jobs that were originally moved from another Pool or Queue. These Jobs must notify the original Pool (where they were submitted) that they have completed. After the original Pool is notified, these Jobs move to the History State Queue.
- **History State Queue**—contains all the Jobs that have recently exited the Netbatch system.
- **Dirty State Queue**—no longer used.

If you have successfully submitted a Job to Netbatch, it will appear in one of these State Queues.



8.2 Viewing Job Information

You can query Netbatch to find out about Jobs that you have submitted, using the `nbstatus jobs` command. This allows you to see where your Jobs are in their life cycle. See [Section 3.3, Job Life Cycle](#).

Note

When Netbatch dispatches a Job for execution, it sets the `__NB_JOBID` environment variable (in the Job's environment) to:

`<pool name>.<jobID>`

where:

- *pool name* is the name of the Pool that the Job was submitted to.
- *jobID* is the Job's ID.

You can use the value of `__NB_JOBID` to identify your Job(s) in `nbstatus jobs` output, where it is displayed in the `Fullid` field.

The syntax of `nbstatus jobs` is:

```
nbstatus jobs[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {table|block|csv}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--history <time>] [--timeout <time>]  
[--retry-timeout <time>{h|m}]  
[<specification>]
```

(See [Appendix A.3.10, nbstatus jobs](#) for a full explanation of this command.)

Note

In certain situations, Netbatch cannot handle a user request (such as an `nbstatus` command), for example, if:

- *The Pool Master is down.*
- *A request limit has been reached. (If this happens, you see a `Service denied` message.)*
- *The Pool Master is under heavy load.*



You can add the **--retry-timeout** switch to commands to specify that the command client should keep trying to submit the command for the specified period.

For example:

```
nbstatus jobs --retry-timeout 30m
```

If Netbatch cannot handle the command when you submit it, the command client keeps trying for 30 minutes.

Usually you only need to see what is happening with the Jobs that you have submitted to Netbatch. By default, **nbstatus jobs** only displays your running and waiting Jobs.

To view all Jobs, including those submitted by other users and those that have ended, add the specification **all**:

```
nbstatus jobs all
```

To view all your Jobs, including those that have ended, type:

```
nbstatus jobs "user=='<username>'"
```

To view a specific Job, type:

```
nbstatus jobs "jobid=='<jobID>'"
```

The output of **nbstatus jobs** includes information that can help you find out what is happening to your Jobs. For example, it can help you figure out why a Job is taking longer than expected to be dispatched for execution, or once running, why a Job seems to be taking longer than expected to complete.

See also [Section 4.3, Basic Troubleshooting](#).

8.2.1 Finding out Why a Job Is Waiting Too Long

To find out why a Job seems to be taking longer than expected to be dispatched for execution, look at the following fields:

- **WaitReason**—why Netbatch cannot dispatch the Job. It can be one of the following:
 - **no resource is available**—no matching Workstation is available to run the Job, or Netbatch has not yet started searching for a matching Workstation.

This could indicate that the class that you specified (if any) is not served by many Workstations.

- **qslot inactive**—the Qslot to which the Job was submitted is inactive.



If there is another Qslot that you can use, try resubmitting the Job to it.

- **qslot running policies exceeded**—the Qslot to which the Job was submitted has reached its running Jobs limit.

If there is another Qslot that you can use, try resubmitting the Job to it.

- **Dependency**—the Job cannot start executing until one or more Jobs that it depends on finish running. (See also **Triggers**, below.)
- **reservation is not complete**—some or all of the resources that the Job reserved are not yet available.

These can include classes that you specified in a class reservation expression or execution slots (for a Parallel Job).

- **Suspend**—the Job has been suspended.

Netbatch can suspend a Job if load or interactive user thresholds are exceeded for the Workstation on which the Job is running, or if it needs to run a Job from a Qslot that preempts the Qslot to which you submitted the Job. (See [Section 5.3.3, Preemption](#).) The **SuspendReason** field shows why the Job is suspended.

- **Failed to run pre execution (last failure on machine <machine>)**—the specified pre-execution program failed (on machine).
- **License_not_available <...>**—one or more licenses that the Job requires are not available. This includes the situation where one or more of the licenses that the Job requires has not yet been reserved. (See [LicenseReservationRequest](#) and [ReservedLicenses](#), below.)
- **LicenseReservationRequest** and **ReservedLicenses**—you can compare the licenses that you requested for the Job and the licenses that have so far been reserved for the Job.
- **ClassReservation**, **ClassImplicitReservation**, and **ReservedClasses**—you can compare the classes that you requested to be reserved for the Job, the reservation that was actually made for the Job (which can include additional requirements that are automatically applied to Jobs that are submitted to a particular Queue or Qslot), and the classes that have so far been reserved for the Job.
- **TriggerJobs**—Jobs that the Job depends on. The Job will not be dispatched until all trigger Jobs have ended.



8.2.2 Finding Out Why a Job Is Taking a Long Time to Complete

To find out why a Job that has been dispatched for execution seems to be taking longer than expected to finish running, look at the following fields:

- **Status**—if the Job's status is **Suspend**, then it has been suspended. Netbatch can suspend a Job if load or interactive user thresholds are exceeded for the Workstation on which the Job is running, or if it needs to run a Job from a Qslot that preempts the Qslot to which you submitted the Job. (See [Section 5.3.3, Preemption](#).)

The **SuspendReason** field shows why the Job is suspended.

- **SubmitTime**, **Utime**, **Stime**, **TimeOnMachine**, **TimeInRunning**, **ExecTime**, **IdleTime**—these can give you an idea of how much the Job has actually been running during its time on the Workstation.
- **TimesPolicyRestarted**, **TimesRestarted**—these show how many times the Job has been restarted (if at all).
- **IsHung**—indicates whether the Job is hung (i.e., has exceeded its own or the Pool's hung limits)
- **IsRunaway**, **ExecTime**—shows whether the Job is a runaway Job, and if so, how long it has been in this state.

8.2.3 Viewing Jobs in the Different State Queues

8.2.3.1 Viewing Jobs in the Wait State Queue

Jobs submitted to a Pool are maintained in the Wait State Queue until dispatched for execution.

To view only waiting Jobs, type:

```
nbstatus jobs "status=='wait'"
```

For Example

```
nbstatus jobs "status=='wait'&&user='<username>'"
```

This instructs Netbatch to display all your waiting Jobs.

When a Job is dispatched for execution on a Workstation it exits the Wait State Queue and enters the Run State Queue.



8.2.3.2 Viewing Jobs in the Run State Queue

Jobs that have been dispatched to a Workstation are maintained in the Run State Queue, and remain in this State Queue until one of the following events occurs:

- The Job terminates successfully. It is then moved into the History Queue.
- The Job executes and fails. It is then moved into the History Queue.
- The Job fails to start executing on a Workstation. It is then moved into the History Queue.
- The Job is cancelled. It is then moved into the History Queue.
- The Job is resubmitted. It is then re-entered into the Wait State Queue.
- The Job is moved to another Pool or Queue. It is then moved into the Moved Queue.

To view your running Jobs, type:

```
nbstatus jobs "status=='run'"
```

For Example

```
nbstatus jobs "status=='run'&&user='<username>'"
```

This instructs Netbatch to display all your running Jobs.

8.2.3.3 Viewing Jobs in the History State Queue

The History State Queue contains Jobs that have recently exited the Netbatch system.

These Jobs include those that have been successfully executed, as well as those Jobs that have, for some reason, failed to execute, or have failed during execution.

The maximum number of Jobs that are maintained at any given time in this State Queue is determined by your Netbatch administrator.

To view your Jobs in the History State Queue, type:

```
nbstatus jobs "status=='comp'&&user=='<username>'"
```

where **username** is your user name.



8.2.3.4 Viewing Older Jobs

To view your Jobs that are no longer in the History State Queue, use `nbstatus jobs` with the `--history` switch:

```
nbstatus jobs --history <time_spec>
"user='<username>'"
```

where:

- `time_spec` specifies the timeframe for the query.
- `username` is your user name.

For a more detailed explanation, see [--history <number>{M|H|D}](#) in [Appendix A.3.10, nbstatus jobs](#).

For Example

To view all your Jobs that ran in the last five days, type:

```
nbstatus jobs --history 5D "user='<username>'"
```

where `username` is your user name.

8.3 More Job Tracking Options

8.3.1 Job Totals

You can see the total number of Jobs that belong to you in a Queue by typing:

```
nbstatus jobs --target <pool_name> --group-by "user"
--fields "user,jobid:count"
"user=='<username>'&&queue=='<queue_name>'"
```

This can give an indication of the progress of a series of Jobs that you have submitted to this Queue.

You can see the total number of Jobs in a Queue by typing:

```
nbstatus jobs --target <pool_name> --fields
"jobid:count" "queue=='<queue_name>'"
```



8.3.2 Parallel Job Details

Basic Output

The output of `nbstatus jobs` lists each Parallel slave Job individually. It also lists the Master Job. The following parallel-related fields are available:

- **ParallelHosts**—the hosts that Netbatch has assigned to the Parallel Job. If there is more than one execution slot available on the same host, that host's name appears the corresponding number of times.
- **ParallelJobConstraints**—Parallel Job constraints (as specified with the `--parallel-job-constraints` switch)
- **ParallelJobID**—if the Job is a Parallel Job, displays the full ID of the Master Job. If the Job is not a Parallel Job, displays the Job's full ID.
- **ParallelPhase**—if the Job is a Parallel Job, this field shows whether the Job is in the **Inclusive** phase (where the scheduler is looking for and marking execution slots for the Job, and where other Jobs can still run in these slots) or the **Exclusive** phase (where the execution slots are reserved for the Job). See [Section 6.11, Submitting a Parallel Job](#)
- **ParallelRequest**—the Job's parallel reservation requirements (if any), as specified with `--parallel`
- **ParallelReservedSlotsCount**—the number of execution slots that are currently reserved for slave Jobs
- **ParallelRunningSlavesCount**—the number of slave Jobs that are currently running
- **ParallelSlaveClass**—for Parallel Master Jobs, the class of machine that the Job's Slave Jobs require

8.3.3 Viewing a Job's Resource Usage

The **Rusage** field of `nbstatus jobs` allows you to view information about the resources it is currently using and has been using:

- **wtime** - wall clock time
- **utime** - user time
- **stime** - system time
- **RM** - current real memory usage (in MB)
- **VM** - current virtual memory usage (in MB)
- **MaxRSS** - maximum resident set size (i.e., the maximum number of pages the Job's processes have had in real memory so far)



- **MaxVSZ** - maximum swap (virtual memory) size (i.e., the maximum number of pages of virtual memory that the Job's processes have used at any one time so far)

This information is updated every three minutes. Netbatch gets this information from `/proc/<pid>/stat` and `/proc/<pid>/status`.

Note

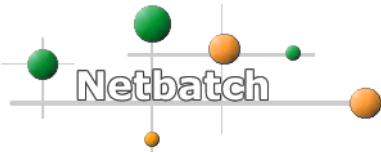
The size of a page of memory may be different for different operating systems.

With the `--history` switch, the available resource usage fields are as follows:

- **Utime** - as above
- **Stime** - as above
- **Wtime** - as above
- **MaxRSS** - as above
- **ixRSS** - integral shared memory size
- **idRSS** - integral unshared data size
- **isRSS** - integral unshared stack size
- **PAGE_FAULT_WITHOUT_IO** - number of page faults (reclaimed)
- **PAGE_FAULT_REQUIRED_IO** - number of page faults (I/O required)
- **NUM_OF_MEM_SWAPS** - number of times swapped out
- **NUM_OF_FS_INPUTS** - number of times file system performed input
- **NUM_OF_FS_OUTPUTS** - number of times file system performed output
- **NUM_OF_IPC_MESSAGE_SENT** - number of messages sent
- **NUM_OF_IPC_MESSAGE RECEIVED** - number of messages received
- **NUM_OF_SIGNAL_DELIVERED** - number of signals delivered
- **NUM_OF_VOLUNTARY_CTX_SWITCH** - number of voluntary context switches
- **NUM_OF_INVOLUNTARY_CTX_SWITCH** - number of involuntary context switches

For Example

To view the resource usage for the Job with a Job ID of 117560620, type:



CONFIDENTIAL

Netbatch
v8.2.2

Revision 4

Tracking Job Status

```
nbstatus jobs --fields "jobid,rusage::120"  
"jobid==117560620"
```



9 Moving, Changing, and Cancelling Jobs

9.1 Overview

Once a Job has been submitted to Netbatch, you can manipulate its state and location in the Pool using the following commands:

- **`nbjob modify`**—modify the parameters of one or more Jobs that have already been submitted.
- **`nbjob remove`**—remove one or more Jobs that have already been submitted.
- **`nbjob recall`**—resubmit one or more Jobs that have already been submitted.

You can also suspend and resume Jobs—see [Appendix A.2.6](#) and [Appendix A.2.7](#).

Note

You can only manipulate Jobs that belong to you, that is, Jobs that you have submitted. Superusers can manipulate any Job in the Pool.

Note

To use these commands to move, cancel, or resubmit a Parallel Job, simply specify the Job ID of the Master Job.

If you cancel or resubmit the Master Job, all the Slave Jobs are also cancelled or resubmitted.

9.2 Modifying One or More Jobs

You can use the **`nbjob modify`** command to change one or more of the following parameters (for one or more Jobs that have already been submitted):

- The Queue and/or Qslot to which the Job was submitted
- Class requirements
- Class (resource) reservation requirements
- License requirements
- License reservation requirements



- License keyfile to be used
- Parallel reservation requirements
- Priority
- File copying parameters
- Jobs on which the Job depends (trigger Jobs)
- Job constraints
- The remote (physical) Pools that the Job can use (Virtual Pool only)

If one of the Jobs that you specify is already running, it is **resubmitted** with the modified parameters (unless you also specify the **--no-restart** switch).

To modify one or more Jobs, type:

```
nbjob modify [--target <pool_name>|<host_name>]
[<options>] [--no-restart] <specification>
```

where:

- The **--target** switch specifies the Pool to which the Job(s) was submitted (by Pool name or Pool Master hostname).
- **options** specify the new Job parameters (see [Section 9.2.1](#), below).
- **--no-restart** specifies that changes should be applied without restarting the Job(s).
- **specification** specifies which Jobs are to be modified (see [Section 9.5](#), below).

Note

In certain situations, Netbatch cannot handle a user request (such as an nbjob modify command), for example, if:

- *The Pool Master is down.*
- *A request limit has been reached. (If this happens, you see a Service denied message.)*
- *The Pool Master is under heavy load.*

You can add the --retry-timeout switch to commands to specify that the command client should keep trying to submit the command for the specified period.

For example:



```
nbjob modify --retry-timeout 30m --class BIGMEM  
1322
```

If Netbatch cannot handle the command when you submit it, the command client keeps trying for 30 minutes.

9.2.1 Job Options

A switch(es) specifies which parameter(s) is to be changed for the specified Job(s). These switches are described in the following subsections.

9.2.1.1 Moving the Job(s) to a Different Queue/Qslot

To move the Job(s) to a different Queue and/or Qslot, use the `--queue` and/or `--qslot` switches:

```
nbjob modify [--target <pool_name>|<host_name>]  
--queue <queue_name> --qslot <qslot_name>  
<specification>
```

For Example

```
nbjob modify --target linux_idc  
--queue dt --qslot lvt/lfe 33349682
```

This moves the Job with ID 33349682 to qslot lvt/lfe in Queue dt.

Note

33349682 is the same as "jobid==33349682".

9.2.1.2 Changing Class Requirements

To change the class requirements of one or more Jobs, use the `--class` switch:

```
nbjob modify [--target <pool_name>|<host_name>]  
--class <class_expression> <specification>
```

where `class_expression` is an arbitrarily complex boolean expression. (See [Appendix .i.nbjob_checkpoint](#); for more details.)

For Example

```
nbjob modify --target linux_idc  
--class !bigmem "jobid==33349682 | jobid==33349684"
```

This changes the class requirements of the Jobs with IDs 33349682 and 33349684 to !bigmem.



9.2.1.3 Changing Class Reservation Requirements

To change the class reservation requirements of one or more Jobs, use the **--class-reservation** switch:

```
nbjob modify [--target <pool_name>|<host_name>]
--class-reservation <reservation_item>[,...]
<specification>
```

where **reservation_item** is in the following format:

```
"<key>=<key_value>[:duration=<duration_time>]
[:decrease=<decrease_time>]
[:delta=<delta_value>]"
```

If the Job(s) is running, it is **resubmitted** with the new class reservation requirements and all reserved resources are released.

If the Job(s) is waiting, all reserved resources are released.

See [Appendix .i.nbjob checkpoint](#); for more details.

For Example

```
nbjob modify --target linux_idc
--class-reservation "fVM=100:duration=2" 33349682
```

This changes the class reservation requirement of Job 33349682 to specify that it requires 10MB of free virtual memory at some point during its first two minutes of execution.

9.2.1.4 Changing License Requirements

To change the license requirements of one or more Jobs, use the **--license** switch:

```
nbjob modify [--target <pool_name>|<host_name>]
--license <feature>[,...] <specification>
```

For Example

```
nbjob modify --target linux_idc
--license specman 33349682
```

This changes the license requirements of Job 33349682 to specify that it requires the feature called specman.



9.2.1.5 Changing the License Keyfile

To change the license keyfile used by one or more Jobs, use the **--license-keyfile** switch:

```
nbjob modify [--target <pool_name>|<host_name>]
--license-keyfile <keyfile>[....] <specification>
```

where **keyfile** is either the full keyfile path or the port and host of the license server (in **<port>@<host>** form).

For Example

```
nbjob modify --target linux_idc
--license-keyfile 83817@ilics01.iil.intel.com 33349682
```

This changes the license keyfile for Job 33349682 to the one that has a <port>@<host> of 83817@ilics01.iil.intel.com 33349682.

9.2.1.6 Changing Job Constraints

To change the Job constraints of one or more Jobs, use the **--job-constraints** switch:

```
nbjob modify [--target <pool_name>|<host_name>]
--job-constraints "<expression>:<action>[....][,...]" 
<specification>
```

where **expression** is an arbitrarily complex boolean expression or **idle(<interval_in_sec>,<cpu_in_percent>)**. See [Appendix , .i.nbjob checkpoint](#); for more details.

For Example

```
nbjob modify --target linux_idc
--job-constraints "RM>500:mail:kill" 33349682
```

This changes the Job constraint of Job 33349682 to specify that if the its real memory usage exceeds 500MB, Netbatch should kill it and mail the user.

Note

*If you use **--job-constraints** with the **--no-restart** switch, the Job constraints are modified without resubmitting the Job.*

*If you omit the **--no-restart** switch, the Job is resubmitted with the new Job constraints.*



9.2.1.7 Changing Parallel Reservation Requirements

To change the parallel reservation requirements of one or more Parallel Master Jobs, use the `--parallel` switch:

```
nbjob modify [--target <pool_name>|<host_name>]
--parallel <parallel_reservation_string>
<specification>
```

where `parallel_reservation_string` takes the following form:

```
"slots=<min_slots>[-<max_slots>],
slots_per_host=<min_sph>[-<max_sph>][,
reservation_per_host={true|false}][,
preserve_slot={true|false}]"
```

See [Appendix .i.nbjob checkpoint](#); for more details.

If you modify a running Job's Parallel Job requirements, the Job is **resubmitted** with the new Parallel Job requirements and all reserved execution slots are released.

If you modify a waiting Job's Parallel Job requirements, all reserved execution slots are released.

For Example

```
nbjob modify --target linux_idc
--parallel "slots=2" 33349682
```

This changes to the parallel requirements of Job 33349682 to specify that it requires at least two execution slots.

9.2.1.8 Changing Priority

To change the priority of one or more Jobs, use the `--priority` switch:

```
nbjob modify [--target <pool_name>|<host_name>]
--priority <priority> <specification>
```

This changes the priority of the Job immediately, even if it is already running.

Note

If the Job is a Parallel master Job, the new priority is applied to the master Job and all the slave Jobs. (If the Job is a slave Job, the command is ignored.)

**For Example**

```
Nbjob modify --target linux_idc  
--priority 5 33349682
```

This changes to the priority of Job 33349682 to 5.

9.2.1.9 Changing File Copying Parameters

To change the file copying parameters of one or more Jobs, add whichever of the following switches that you require to the **nbjob modify** command:

- **--file-copy-in <source_in>:<target_in>[,...]**
- **--file-copy-out <source_out>:<target_out>[,...]**
- **--file-delete <delete_path>[,...]**
- **--file-copy-command-args <arguments>**
- **--copy-arg-file <argfile_path>**

See the following sections of [Appendix .i.nbjob checkpoint](#); for more details:

- [**--file-copy-in <source>:<target>\[,...\]**](#)
- [**--file-copy-out <source>:<target>\[,...\]**](#)
- [**--file-delete <path>\[,...\]**](#)
- [**--file-copy-command-args <arguments>**](#)

9.2.1.10 Adding, Changing, or Removing Job Dependencies

To change the Job(s) on which the Job depends, or to add Job dependencies if it does not already have any, use the **--triggers** switch:

```
nbjob modify [--target <pool_name>|<host_name>]  
--triggers "<boolean_expression>" <specification>
```

where **boolean_expression** consists of any number of terms of the form **[!]<JobID>[[<exit_status_expression>]]**, separated by the logical operators **&&** (AND) and **||** (OR).

This expression completely replaces any existing Job dependency expression.

**Note**

JobID can be either a Job ID or a task name. You are advised to explicitly specify whether it is a Job ID or a task name, because in ambiguous cases, Netbatch can assume that a task name is actually a Job ID, or vice versa.

Use one of the following instead of a Job ID/task name:

```
jobid:<Job_ID>  
task:<task_name>
```

If a task name is used, the `exit_status_expression` does not evaluate to true until it is true for each of the Jobs that make up the task.

See [Section 6.5.18, Specifying a Task Name](#).

`exit_status_expression` can be one of the following:

- `exit`
- `exit <relational_operator> <exit_code>`
`relational_operator` must be `<`, `<=`, `==`, `>=`, or `>`.

To specify multiple acceptable exit code conditions for a specific Job, use multiple `<JobID>[<exit_status_expression>]` terms separated by `||` (OR) (for example, `1 exit <30 || 1 exit>50`).

If `exit` is used on its own, it means that the Job ended (no matter what its exit status).

If no `exit_status_expression` is specified, `done` is assumed.

Note

Logical expressions are evaluated according to the following order of precedence - () (parentheses), ! (NOT), && (AND), || (OR).

So

```
"1 [exit<30] || !2 && (3 || 4)"
```

is the same as

```
"1 [exit<30] || ((!2) &&(3 || 4))"
```

**For Example**

```
nbjob modify --target linux_idc --triggers "27&&28" 30
```

This replaces any existing Job dependencies for Job 30 so that it now depends on Jobs 27 and 28 (that is, it cannot start running until they end).

9.2.1.11 Changing the Remote Pools that Can Be Used

To change the remote (physical) Pools that can be used by one or more Jobs that were submitted to a Virtual Pool, use the **--remote-pools** switch:

```
nbjob modify [--target <pool_name>|<host_name>]  
--remote-pools {"[-]<phys_pool>" | "[-]<alias>" }[,...]  
<specification>
```

where:

- **phys_pool** is the name of a physical Pool.
- **alias** is a physical Pool's alias.
- You can specify the Pools by:
 - Inclusion—only the listed physical Pools will be considered when deciding which Pool to send the Job to.
 - Exclusion—precede Pool/alias names with **-**. All relevant physical Pools will be considered except those that are listed.
 - A combination—if there are both included and excluded Pools/aliases in the list, the excluded ones are excluded from the list of included Pools.

You can use this to specify that a specific physical Pool should be excluded from the set of Pools represented by an alias.

This situation can also arise when the request uses inclusion and a Job Profile uses exclusion, or vice-versa.

Note

The order determines the order in which the Pools will be tried. Netbatch will try the first Pool, and will only try the next one if the previous one has reached its **WaitJobsLimit** or **MaxJobsLimit**.

If an alias is specified, Netbatch uses its existing algorithm to decide which Pool to send the Job to. (Multiple Pools can have the same alias. For example, all the Physical Pools at the same site as the Virtual Pool Master might have the alias **local**.)



This switch overrides the **NB_POOLS** environment variable.

For Example

```
nbjob modify --target vpool_idc  
--remote-pools "-sles_idc" 33349682
```

*This removes **sles_idc** from the list of physical Pools that Job 33349682 can use.*

9.3 Removing One or More Jobs

To remove one or more Jobs, type:

```
nbjob remove [--target <pool_name>|<host_name>]  
[--reason "<reason>" ] <specification>
```

where:

- The **--target** switch specifies the Pool to which the Job(s) was submitted (by Pool name or Pool Master hostname).
- **--reason** specifies why the Job is being removed. (This information is recorded in NB Tracker.)
- **specification** specifies which Jobs are to be modified (see [Section 9.5](#), below).

Note

*In certain situations, Netbatch cannot handle a user request (such as an **nbjob remove** command), for example, if:*

- *The Pool Master is down.*
- *A request limit has been reached. (If this happens, you see a **Service denied** message.)*
- *The Pool Master is under heavy load.*

*You can add the **--retry-timeout** switch to commands to specify that the command client should keep trying to submit the command for the specified period.*

For example:

```
nbjob remove --retry-timeout 30m 1322
```



If Netbatch cannot handle the command when you submit it, the command client keeps trying for 30 minutes.

For Example

```
nbjob remove --target linux_idc 12660
```

This removes the Job with ID 12660.

For Example

```
nbjob remove --target linux_idc  
"jobid==12658 | jobid==45129"
```

This removes the Jobs 12658 and 45129.

For Example

```
nbjob remove --target linux_idc  
"(jobid>=12650)&&(jobid<=12660)"
```

This removes the Jobs with IDs in the range 12650 to 12660 (inclusive).

For Example

```
nbjob remove --target linux_idc all
```

This removes all your Jobs.

Note

To remove a Parallel Job, specify the Master Job's Job ID.

If you remove the Master Job, all the Slave Jobs are also removed.

**Caution**

It is possible to remove a Parallel Job's Slave Job, but the Master Job will never complete, as it must wait for all its Slave Jobs to end.

Only cancel a Slave Job if you understand the Parallel Job's internal logic and the consequences of cancelling it.

9.4 Resubmitting One or More Running Jobs

To resubmit (recall) one or more Jobs that are already running, type:

```
nbjob recall [--target <pool_name>|<host_name>]  
[--reason "<reason>"] <specification>
```

where:

- The **--target** switch specifies the Pool to which the Job(s) was submitted (by Pool name or Pool Master hostname).
- **--reason** specifies why the Job is being resubmitted. (This information is recorded in NB Tracker.)
- **specification** specifies which Jobs are to be modified (see [Section 9.5](#), below).

You might want to resubmit a running Job if:

- There is a problem with the input.
- The Job is idle because it is waiting for a license.
- The Job is running on a slow machine.
- Netbatch suspends the Job because load or user thresholds are exceeded.

The resubmitted Job(s) is placed at the head of the wait Queue; that is, it will be dispatched before the other waiting Jobs (assuming that it can run on the next Workstation that becomes available).

When you resubmit a Job, it retains its Job ID.

Note

To resubmit a Parallel Job, specify the Job ID of the Master Job.



If you resubmit the Master Job, all the Slave Jobs are also resubmitted.

Caution

It is possible to resubmit a Parallel Job's Slave Job, but because Slave Jobs are usually interdependent, such an action may have adverse effects.

Only resubmit a Slave Job if you understand the Parallel Job's internal logic and the consequences of resubmitting it.

Note

In certain situations, Netbatch cannot handle a user request (such as an `nbjob recall` command), for example, if:

- *The Pool Master is down.*
- *A request limit has been reached. (If this happens, you see a `Service denied` message.)*
- *The Pool Master is under heavy load.*

You can add the `--retry-timeout` switch to commands to specify that the command client should keep trying to submit the command for the specified period.

For example:

```
nbjob recall --retry-timeout 30m 1322
```

If Netbatch cannot handle the command when you submit it, the command client keeps trying for 30 minutes.

9.5 Job Specification

The `specification` argument is a boolean expression that specifies which Jobs are to be modified, removed, resubmitted, etc.

`specification` is either `all` or an arbitrarily complex boolean expression consisting of `<attribute>`s, `<operator>`s, and `<value>`s, where:

- `attribute` is a valid Job field (see `nbstatus jobs --fields`, [Appendix A.3.10](#)).



- **operator** defines the relationship between the attribute and the value.
operator can be one of the following:
 - An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")
 - A logical operator: &&, ||, or !
 - A comparison operator: <, <=, >, >=, !=, or ==
 - A non-strict operator: is or isnt
 - A bitwise operator: &, |, ~, ^, <<, >>, or >>>
 - A conditional operator: ? or :
- <value> is a number, a string, or another boolean expression.

Note

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If **value** is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

Expressions can be grouped together using parentheses.

If **specification** is all, all the user's Jobs are modified.

The **startswith()** and **glob()** functions can also be used:

- "startswith(<field_name>,'<string>')"

startswith() matches fields whose value starts with the specified string.

- "glob(<field_name>,'<expression>')"

glob() matches fields that match **expression**. **expression** can contain:

- Any character
- . - matches any character
- \w - matches any letter (upper- or lower-case)
- \d - matches any digit
- \s - matches any space character



- * - zero or more of the previous character
- ? - exactly one of the previous character

For Example

```
nbjob modify --target linux_idc <options> jobid==12658
```

This specifies that the Job with ID 12658 is to be modified.

For Example

```
nbjob modify --target linux_idc <options>  
"jobid==12658 | jobid==45129"
```

This specifies that Jobs 12658 and 45129 are to be modified.

For Example

```
nbjob modify --target linux_idc <options>  
"(jobid>=12650)&&(jobid<=12660)"
```

This specifies that all the Jobs with IDs in the range 12650 to 12660 (inclusive) are to be modified.

For Example

```
nbjob modify --target linux_idc <options>  
"glob(jobid,'12\d??')"
```

This specifies that all your Jobs that have a five-digit Job ID that starts with 12 are to be modified.

For Example

```
nbjob modify --target linux_idc <options> all
```

This specifies that all your Jobs are to be modified.



CONFIDENTIAL

Netbatch
v8.2.2

Revision 4

Moving, Changing, and Cancelling Jobs



10 Submitting and Tracking Jobs with Netbatch Flow Manager

10.1 Overview

Netbatch Feeder allows you to efficiently feed a large number of related Jobs (a **task**) to Netbatch, and then to track and monitor the progress of these Jobs.

When you submit a task to the Feeder, it submits the task's Jobs to the specified Pool in trickle mode (that is, in a regulated way) based on Pool availability.

The Netbatch Flow Manager is a graphical front-end for Netbatch Feeder, and is the recommended way to work with Netbatch Feeders. It is a GUI application that allows you to:

- Start and manage feeders.
- Create and edit tasks and submit them to a feeder.
- Graphically create dependencies between tasks.
- Monitor and manage tasks and Jobs.

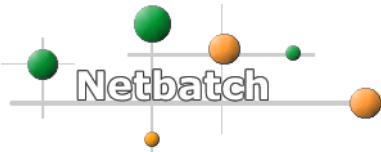
For more information, see the Flow Manager Help. (In Flow Manager, select **Help | Help Contents.**)

You can also perform these operations at the command line. See [Appendix C.1, nbfeeder](#) and [Appendix C.2, nbtask](#).

10.2 Starting the Flow Manager

To start the Flow Manager:

1. cd to /usr/intel/pkgs/netbatch/<latest_version>/bin. (where **latest_version** is the latest version in the **netbatch** directory).
2. Type:
`./nbflow`



CONFIDENTIAL

Netbatch
v8.2.2

Revision 4 Submitting and Tracking Jobs with Netbatch Flow



11 Using Netbatch Tracker

Netbatch has a number of commands that allow you to find out information about Jobs that are currently running or that are waiting to be dispatched for execution. These commands provide limited information about Jobs that have finished running.

Netbatch Tracker records information about Netbatch Jobs. By default, it keeps this information for two months.

You can query Netbatch Tracker to find out information about:

- Any Jobs that were submitted to Netbatch in the last two months, including ones that have just been submitted. (Netbatch Tracker is updated in real time.)

Note

Netbatch Tracker was previously called the Netbatch Accounting System (NAS).

In the recommended usage model, all the Netbatch Pools in a region report to a single Netbatch Tracker database. The database, and therefore the data that can be queried, only includes information from those Pools that report to it.

To query Netbatch Tracker, use `nbstatus` command with the `--history` switch (see [Appendix A.3, nbstatus](#)).

Note

The `nbquery` command has been deprecated.



CONFIDENTIAL

Netbatch
v8.2.2

Revision 4

Using Netbatch Tracker



12 Increasing Throughput with Virtual Pools

12.1 Overview

A **Virtual Pool** is a Pool of virtual resources. Each virtual resource is a Physical Pool. The Virtual Pool accepts Jobs like a regular Pool and schedules them for execution on the best available resource.

A **Virtual Pool Master** process controls the flow of Jobs in the Virtual Pool. The Virtual Pool Master matches Jobs that are waiting in the Queues of the Virtual Pool with Physical Pools that have unused computing resources.

12.2 Why Use a Virtual Pool?

Virtual Pools offer the following advantages:

- **Scalability and performance**—The scalability of a single Physical Pool is limited. A Physical Pool Master can support a relatively limited number of Workstations, Jobs, and simultaneous user requests. As the limits of a single Physical Pool Master are exceeded, there is a noticeable degradation in its performance. A Virtual Pool allows each Physical Pool Master to handle a smaller load, and hence to provide good performance, while offering the combined power of all its member Physical Pools.
- **Geographically dispersed sites**—A Virtual Pool can be configured to send your Jobs for execution on remote campuses or sites. Often, hours of high load at one site coincide with hours of low load at another. Sending your Jobs to a Virtual Pool allows you to tap into unused resources at other sites.
- **Administration**—Virtual Pools also afford certain administrative advantages for Netbatch administrators.

It is advisable to use a Virtual Pool when the load on its member Physical Pools is uneven, or unpredictable. A Virtual Pool can automatically do the load balancing work and send your Jobs to the most available Pool.

For Example

Some Pools may have many queued Jobs, while other Pools may have idle Workstations at the same time. This situation is exacerbated when two projects are at their peak usage in different time zones, where one user's prime time may be another user's nighttime.

The Virtual Pool is designed for those customers who would like to use multiple Pools efficiently. A Virtual Pool is like a one stop shop for several Netbatch Pools.



12.3 Structure

12.3.1 Virtual Queues and Virtual Qslots

Similar to a Physical Pool, a Virtual Pool is divided up into Queues, called Virtual Queues. Each Virtual Queue contains a hierarchical structure of Qslots call Virtual Qslots.

12.3.2 Migration Rules

When Jobs are submitted to a Virtual Pool, the Virtual Pool Master follows a set of pre-configured rules called **Migration Rules**.

The Migration Rules:

- Define the number of Jobs that can be sent to each Physical Pool and to each Queue and Qslot in the Physical Pools.
- Define the mapping between Virtual Queues and Qslots in the Virtual Pool and their equivalent Physical Queues and Qslots in the different Physical Pools.
- Define the mapping of Classes from one Pool to the equivalent Classes in another Pool.
- Specify policies applied to the Jobs sent to each Physical Pool and virtual Qslot (for example, a threshold time after which a Job waiting in a Qslot of a Physical Pool is resubmitted to another Physical Pool).
- Specify the times during which each rule is enforced.

The Virtual Pool migration rules are created and maintained by the Netbatch Administrator.

12.3.3 Resource Allocation in Virtual Pools

Each Virtual Pool has its own configured Queues and Qslots and the overall resource allocation attempts to match the Virtual Pool resource allocation.

The Virtual Pool cannot force resource allocation on the Physical Pools. Once a Job is migrated to a Physical Pool, it may have to wait in the Wait State Queue, according to the availability of resources in the Physical Pool.



12.4 Using Virtual Pools

A Virtual Pool behaves exactly like a Physical Pool and supports the same commands and output. An additional command, `nbpoolstat`, provides information about the virtual resources of the Virtual Pool.

A Virtual Pool also tracks Job status and responds to user status requests as well as user requests, such as a Job kill request.

12.4.1 Submitting a Job to a Virtual Pool

To submit a Job to a Virtual Pool, type:

```
nbjob run --target <virtual_pool_name>
[--remote-pools {<phys_pool>|<alias>}[,...]] <command>
```

where:

- `virtual_pool_name` is the name of the Virtual Pool.
- The `--remote-pools` switch specifies that the Virtual Pool must not send the Job to physical Pools other than the specified ones.

The order determines the order in which the Pools will be tried. Netbatch will try the first Pool, and will only try the next one if the previous one has reached its `WaitJobsLimit` or `MaxJobsLimit`.

If an alias is specified, Netbatch uses its existing algorithm to decide which Pool to send the Job to. (Multiple Pools can have the same alias. For example, all the Physical Pools at the same site as the Virtual Pool Master might have the alias `local`.)

- `command` is the Job command.

When a Job lands in the Virtual Pool, it is forwarded to one of the pre-configured Physical Pools according to Physical Pool availability and Queue and Qslot allocation.

Prerequisites for running Jobs in Virtual Pools fall into two categories, as follows:

- If the Physical Pools are all located at the local site:
 - ◆ Basic understanding of how to run a Job in Netbatch
 - ◆ List of Physical Pools running Netbatch 6.0 or higher
 - ◆ Allocation and permission to run Jobs in at least one Qslot in each of the configured Physical Pools
 - ◆ Pre-configured Virtual Pool Master with migration rules for each Physical Pool and Qslot. This is done by the Netbatch administrator.



- If one or more of the Physical Pools are located at remote sites:
 - ◆ All the items above
 - ◆ A UNIX account at each site that may run the Jobs. The accounts must share the same name and UID.
 - ◆ Quotas and read/write permission on the local network file system at each site
 - ◆ Accessibility to all the tools and software required for running the Jobs, at each site
 - ◆ Availability of all the input and configuration files at each of the sites

12.5 Commands

You can submit and manage Jobs with the same commands that you use for Jobs submitted to a regular (physical) Pool; that is:

- `nbjob run`
- `nbjob modify`
- `nbjob remove`
- `nbjob suspend`
- `nbjob resume`
- `nbstatus jobs`
- `nbstatus remote-availability`

The `nbstatus qslots` output is the same as for a physical pool, with the following exceptions:

- The **Waiting** field shows the number of Jobs waiting in each virtual Qslot.
- The **Dispatched** field shows the number of Jobs that were submitted to each virtual Qslot that are now waiting in a physical Qslot.
- The **Running** field shows the number of running Jobs that were submitted to each virtual Qslot.

The `nbstatus remote-availability` command allows you to check whether there are Job slots available to run your Job in any of a Virtual Pool's Virtual Resources. (It also displays information about the number of running and waiting Jobs in the specified Qslot, Job limits, etc.) See [Appendix A.3.24, nbstatus remote-availability](#).



Appendix A: User Command Reference

This appendix describes:

- Netbatch commands
- Command-line options
- Permitted combinations of options

The Netbatch executables are a collection of scripts and binary files that implement the Netbatch command interface.

For easy access to the Netbatch executables, ask your Netbatch administrator where the Netbatch 8.2.2 executables reside on your network file system, and append that path to your PATH environment variable. You can do it by typing:

```
setenv PATH <nb_exec_path>:{$PATH}
```

Several generations of Netbatch executables may be resident on your site. Although Netbatch 8.2.2 is backward compatible with all previously released Netbatch executables, the Netbatch team recommends that you use only Netbatch 8.2.2 commands in order to make full use of the product's capabilities.

Note

All options are case insensitive, except -h and -H.

Long switches (that is, those that start with -- and consist of more than a single letter) may be shortened by using only the -- and the first few letters.

For example, nbqslot's --show-hierarchy switch may be shortened to --sh.

A.1 Existing Netbatch 4.x and 5.x Commands

The old Netbatch 4.x and 5.x commands are now deprecated:

- **nbdepend**
- **nblicense**
- **nblicstat**
- **nbq**
- **nbqrm**



- **nbqslot**
- **nbqstat**
- **nbstat**

A.2 nbjob

Name

nbjob - Submit Jobs to Netbatch or perform actions on Jobs already submitted.

Description

nbjob is a set of commands to submit Jobs to Netbatch, perform actions on Jobs already submitted, and to start a Netbatch Feeder, as follows:

- ◆ **nbjob run** - Submit a Job to Netbatch. See [Appendix A.2.1](#).
- ◆ **nbjob checkpoint** - manually create a checkpoint for one or more Jobs. See [Appendix A.2.2](#).
- ◆ **nbjob modify** - Modify the parameters of one or more Jobs that have already been submitted. See [Appendix A.2.3](#).
- ◆ **nbjob remove** - Remove one or more Jobs that have already been submitted. See [Appendix A.2.4](#).
- ◆ **nbjob suspend** - Suspend one or more Jobs that have already been submitted. See [Appendix A.2.5](#).
- ◆ **nbjob resume** - Resume one or more suspended Jobs. See [Appendix A.2.6](#).
- ◆ **nbjob recall** - Resubmit one or more Jobs that have already been submitted. See [Appendix A.2.7](#).
- ◆ **nbjob prun** - Submit a Parallel Slave Job. See [Appendix A.2.8](#).
- ◆ **nbjob pmodify** - Modify a Parallel Job. See [Appendix A.2.8](#).
- ◆ **nbjob signal** - Send a specific signal to a Job. See [Appendix A.2.10](#).

Note

*nbjob commands have no effect on interactive sessions. Use ION's **nbsession** command to work with those.*



Usage (General)

```
nbjob <operation>
[ --target <pool_name> | <host_name> ]
[ <options> ] [ <specification> ]
```

Short Switches

All **nbjob** switches are in the following format:

```
--<switch_name>
```

Switches may be shortened according to the following rules:

- ◆ A short (single-letter) switch (e.g., **-h**) may be used if it is unambiguous.
- ◆ If a single-letter switch is ambiguous, you can use a shortened version of the long switch, including as much of the switch as necessary to be unambiguous. (For example, for **nbjob run**'s **-mode** switch, **-m** is ambiguous, so you can use **--mo**.)

Help, Debug, and Version Information

To get a list of available **nbjob** commands, type:

```
nbjob --help
```

To get help information for a specific **nbjob** command, type:

```
nbjob <command> --help
```

For example:

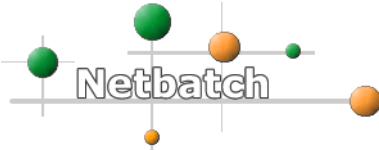
```
nbjob run --help
```

To show debug information, add the **--debug** switch:

```
nbjob <command> --debug
```

To show the Netbatch version, use the **--version** switch:

```
nbjob --version
```



Diagnostics

The command exit codes are listed below.

Note

Not all error codes are relevant for this command.

Note

These are command exit codes, not Job exit codes.

0

Successful execution

1

Specified Pool Master (**-P**) does not exist in **pools** file (older commands only (**nbq**, **nbqrm**, **nbstat**, etc.))

2

Unknown host specified (**--target** for newer commands, **-H** for older commands)

3

Error occurred on master side (for example, PPM error, Job could not be submitted because of missing permissions, non-existent class, etc.)

4

Problem getting user credentials (most likely an NIS issue on the client machine)

5

PPM timeout (The command waits one minute for a response.)

212

Depends on context:

- ◆ CSD error - Job failed
- ◆ Status command - invalid displayed field

**213**

Depends on context:

- ◆ CSD error - bad parameters
- ◆ Status command - invalid filter

214

Depends on context:

- ◆ CSD error - Workstation not available
- ◆ Status command - invalid grouping

215

Depends on context:

- ◆ CSD error - RG not available
- ◆ Status command - invalid sort

216

CSD error - Qslot not available

221

Interrupted

222

Permission denied

223

Timeout

224

Unknown error

225

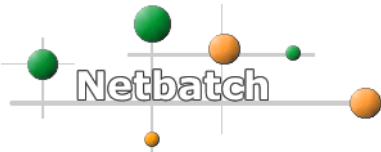
Permissions problem

229

The command was run with the **-v** switch (version)

230

The command was run with the **-h** switch (help)

**231**

Parsing error - mandatory filter not specified

232

Parsing error - invalid value

233

Parsing error - value not specified

234

Parsing error - undefined parameter

235

Parsing error - mandatory parameter not specified

244

No command handler in this context

245

No command handler

248

Communication problem - target service down

249

Communication problem - connection broken during receive stage

250

Server too busy (denial of service response or built-in request limit reached)

251

The server does not support the command.

253

Communication problem - unknown protocol

254

Communication problem - no response

255

Master not responding (probably down)



A.2.1 nbjob run

Name

nbjob run - Submit Jobs to Netbatch.

Description

Submit a Job to Netbatch to execute the given command. The Job can be any UNIX command, shell script, executable binary, almost anything that can be done at the shell prompt.

The Job will be placed in the Wait Queue, and later dispatched for execution based on machine availability and characteristics of the waiting Jobs, such as Qslot and class.

The results of the command execution (standard output and standard error) will be written to a user log file in the directory from which the **nbjob run** command was invoked. The file name format is `##<date>-<time>#.<machine>`, where `<date>` and `<time>` indicate when the Job starts executing, `<machine>` refers to the Workstation on which the Job executes. For example, `##Dec-22-21:08:55#.ilx126`. If a log file name is given with `--log-file` and/or `--log-file-dir`, it will be used instead.

If the directory where **nbjob run** is invoked is unavailable on the remote server, Netbatch will look for the environment variable **NBWD** and try to write the log file there. If **NBWD** is not defined either, a mail message indicating the problem will be sent to the user who submitted the Job. Note that if your site uses automounter you MUST use the **NBWD** variable. For example, if you are working in the directory `/export/cbs_02/pauls`, the `pwd` command will return `/a/export/cbs_02/pauls`. You will have to do `setenv NBWD /export/cbs_02/pauls`. The Job when executed will inherit all the user environment variables at the time of **nbjob run** invocation, unless the `--no-env` switch is used.

Note

If the WSM is down for some reason when the Job ends, writing the log file is not completed, as Netbatch does not have the required information. When the WSM comes back up, writing of the log file is completed.

**Note**

An `nbjob run` command-line option usually overrides an environment variable that has been set to specify the same parameter.

In most cases (but not all), it also overrides any default values that the Netbatch Administrator has set for the parameter.

See [Appendix B: Precedence of Configuration Directives, Environment Variables, Job Profiles, and Command-Line Switches](#).

Usage

```
nbjob run [--target <pool_name>|<host_name>]
[<options>] <command>
```

Target

```
--target <pool_name>|<host_name>
```

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If `--target` is omitted, `nbjob` uses the Pool defined in the `NBPOOL` environment variable, or if `NBPOOL` is undefined, the Pool of which the Workstation where the command was run is a member.)



Options

```
--action-signal
"<action>[:signal=<type>][,timeout=<time>]
[:cmd=<cmd_name>][;...]"
```

Specifies that for the specified action, the Job will be "soft killed", that is, the specified signal will be sent to the specified process (instead of to all the Job's processes, as in the default sequence).

Here:

- ◆ **action** is the action to which the soft kill will be applied. It is one of:
 - ◆ **remove**
 - ◆ **suspend**
 - ◆ **recall**
- ◆ **type** is the signal that will be sent. Any Unix signal may be sent, except SIGKILL and SIGSTOP. (Default: SIGTERM for remove and recall, SIGTSTP for suspend.)
- ◆ **time** is the amount of time (in seconds) to wait between send the first "soft" signal and continuing with the default sequence (for remove and recall, SIGKILL is sent, for suspend SIGSTOP). (Default: five seconds)

This overrides the timeout specified by
NBSIGKILL_TIMEOUT or **NB_SUSPEND_TIMEOUT**
(depending on the action).

- ◆ **cmd_name** is the command name of the process to which the signal will be sent. (Default: the first process under the Job leader)

If multiple processes have the same command name, the signal will be sent to all of them.

Note

If the Job is killed, removed, resubmitted, or suspended by preemption or by a Workstation policy, the signal (and other parameters) specified in --action-signal are used.



```
--autoreq "attempts=<#_attempts>:<expression>" |  
<#_attempts>:<exit_code>[ ,... ]
```

Note

This switch is deprecated. Use --on-job-finish instead.

Specifies that the Job is to be resubmitted (up to #_attempts times) if it ends with particular exit codes.

If this form is used:

```
<#_attempts>:<exit_code>[ ,... ]
```

then if any of the specified exit codes occur, the Job is resubmitted.

If this form is used:

```
attempts=<#_attempts>:<expression>
```

then the Job is resubmitted if expression is true. expression is an arbitrarily complex boolean expression that can consist of any number of terms of the form `exit <relational_operator> <exit_code>`, separated by the logical operators `&&` (AND) and `||` (OR), and together with `!` (NOT). relational_operator can be `>`, `<`, `>=`, `<=`, or `==`.

The following macros may be used instead of exit codes:

- ◆ **NBErr** - All Netbatch errors except -8, -9, -10, and -11
- ◆ **Nexit** - All negative Job exit values
- ◆ **Pexit** - All positive Job exit values
- ◆ **NZexit** - All non-zero Job exit values

For Example

The following are equivalent:

```
--autoreq attempts=4:-8  
--autoreq attempts=4:exit== -8
```

```
--checkpoint-directory <path>
```

Specifies where the checkpoint files are to be stored. If not specified, the files are stored in the Job's working directory. This switch is only valid if --checkpoint-tool is used.

```
--checkpoint-interval <time>[ {s|m}]
```

Specifies how often a checkpoint should be created, in seconds (c) or minutes (m). If no units are specified, the time is in minutes. If the switch is omitted, a checkpoint is created every 15 minutes.



This switch is only valid if **--checkpoint-tool** is used.

--checkpoint-tool <tool_name>

Specifies the checkpointing tool to be used for the Job (for example, dmtcp). See also:

- ◆ **--checkpoint-directory**
- ◆ **--checkpoint-interval**
- ◆ **--checkpoint-tool-data**
- ◆ **--num-of-checkpoints**

--checkpoint-tool-data <string>

The data to be passed to the checkpointing tool. Each tool defines its own specific syntax for this string. This switch is only valid if **--checkpoint-tool** is used.

--class <class_expression>

Specifies the class of machine (that is, the required Workstation attributes) that the Job requires. These can include static attributes (such as OS type and number of CPUs) and dynamic attributes (such as free virtual memory and load).

Note

When you submit a job with a class requirement, Netbatch checks whether the requirement can ever be satisfied by any of the Workstations to which it can send the Job. (This might include remote Workstations.)

*For example, if your Job requires a Workstation with 1GB of free real memory, Netbatch checks whether any of the Workstations that it can send the Job to have **total** real memory of 1GB or more.*

If none of the local Workstations meet the requirement, Netbatch accepts the Job while it queries the remote Workstations (if any). If one (or more) of the remote Workstations meets the requirement, the Job will eventually be dispatched for execution.

If no remote Workstation meets the requirement, the Job will remain in the wait state queue until it is removed—it can never be dispatched for execution.

You are recommended to use the --validate switch to make sure that your Job's class requirements can be satisfied before you submit it.



Once Netbatch has performed the query, it caches the result. Subsequent Jobs with the same requirement will either be accepted (and later dispatched) or rejected immediately.

class_expression is an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ♦ **attribute** is one of the available Workstation attributes or an external probe parameter name.

For a list of Workstation attributes, see [Section 6.7.2, Viewing Supported Workstation Attributes](#).

This includes total and free disk space on partitions on which directories defined by the Netbatch Administrator reside. Run **nbstatus workstations --fields all** and look at the values of the **fDS** and **tDS** fields to see the partitions whose disk space Netbatch monitors.

To see the available Workstation probes and the parameters that they measure, run **nbstatus workstations --fields all** and look at the value of the **ExternalProbeData** field.

To see the available Scheduler probes and the parameters that they measure, run **nbstatus probes**.

See [Appendix A.3.31, nbstatus workstations](#) for more details.

- ♦ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ♦ An arithmetic operator: **+**, **-**, *****, **/**, **%**, unary **+** (e.g., **+10**), or unary **-** (e.g., **-10**)
 - ♦ A logical operator: **&&**, **||**, **:**, or **!**

|| and **:** both mean OR. But for parallel Jobs, **:** means that all the slave Jobs must run on Workstations that match the same class or class expression (ORed item).

If you use **||**, each of the parallel Job's slave Jobs can run on Workstations that match *any* of the specified classes or class expressions (ORed items).

- ♦ A comparison operator: **<**, **<=**, **>**, **>=**, **!=**, or **==**
- ♦ A non-strict operator: **is** or **isnt**
- ♦ A bitwise operator: **&**, **|**, **~**, **^**, **<<**, **>>**, or **>>>**
- ♦ A conditional operator: **?** or **:**

**Note**

If **attribute** is an external probe name and the probe's data is of type collection (that is, it can only take a value from a specified list of values), the only operator that can be used is ==.

- ◆ **value** is a number, a string, or another boolean expression.
Expressions can be grouped together using parentheses.

--class-reservation <reservation_item>[,...]

Specifies the class requirements that the Job will need during execution. Netbatch reserves the required resources, thus preventing the situation where two or more Jobs successfully begin executing, but hang because their increasing resource requirements exceed the capabilities of the machine.

The Job's actual class reservation (as specified here and/or applied by a Job profile) is available in the **__NB_CLASSRESERVATION** environment variable (in the Job's environment) at runtime.

reservation_item must be in the following format:

```
"<key>=<key_value>
[:duration=<duration_time>[{h|m|s}]]
[:decrease=<decrease_time>]
[:delta=<delta_value>]"
```

where:

- ◆ **key** is the type of resource to be reserved. It can take the following values:

- ◆ **fVM**
- ◆ **fRM**
- ◆ **dedicated**

Note

If **dedicated=true**, then no other Jobs may run on the host while the Job is running (except for Parallel Slave Jobs - multiple Slave Jobs that are part of the same Parallel Job can run on the same host).

- ◆ The name of a custom capability or attribute that has been defined for one or more Workstations in the Pool

You can use the **nbstatus workstations** command to see a list of these attributes:



```
nbstatus workstations --fields
"*,WSCapabilities"
```

These capabilities can either relate to total real or virtual memory or be arbitrary.

Note

Workstation capability requirements can also be defined at the Queue or Qslot level. Such requirements apply to all Jobs submitted to the Queue/Qslot.

Note

If you specify a requirement for a Workstation capability that does not exist in the Pool, it will stay in the wait state indefinitely. Make sure that the capability exists before submitting the Job. (Use nbstatus workstations, as described above.)

- ◆ An external probe parameter name

To see the available Workstation probes and the parameters that they measure, run **nbstatus workstations --fields all** and look at the value of the **ExternalProbeData** field.

To see the available Scheduler probes and the parameters that they measure, run **nbstatus probes**.

See [Appendix A.3.31, nbstatus workstations](#) for more details.

- ◆ **key_value** is the value for **key**. It must be one of the following:
 - ◆ For **fVM**, **fRM**, or custom field any type of number–integer or non-integer value
 - ◆ For **dedicated**, **true** or **false**
- ◆ **duration_time** is the amount of time (in hours, minutes, or seconds) for which the resources (**fVM** and **fRM** only) are required (starting from when the Job is dispatched for execution). If you do not specify **h**, **m**, or **s**, the time is in minutes.



If not specified, then **duration** is set to the Job execution time (that is, the resource is reserved for the entire duration of the Job's execution).

- ◆ **decrease_time** - During the time specified by **duration**, the committed resource (**fVM** and **fRM** only) is reduced every **decrease_time** minutes by the amount specified in **delta_value**.

If a **delta_value** is specified, but no **decrease_time**, then **decrease** = 1 minute.

If neither a **delta_value** nor a **decrease_time** is specified, then the resource is available at the same level for the whole time specified by **duration**.

- ◆ **delta_value** - During the time specified by **duration**, the committed resource (**fVM** and **fRM** only) is reduced every **decrease_time** minutes by the amount specified in **delta_value**.

If a **decrease_time** is specified, but no **delta_value**, then **delta** = 1 unit.

If neither a **delta_value** nor a **decrease_time** is specified, then the resource is available at the same level for the whole time specified by **duration**.

--cluster-id <cluster_id>

When submitting a Job to a cluster of virtual machines (see the **nbvm** command), this specifies the cluster by its ID.

--daemon

The Job is a daemon (that is, a process that closes stdin, stdout, and stderr, and continues to run), so Netbatch should use Process Authentication Groups (PAGs) to monitor and control it.

For Parallel Jobs that use MPI, use the **--daemon** switch when submitting the Slave Jobs with **nbjob prun** (see [Appendix A.2.8](#)).

--dumpRusage <interval>:<path>

Specifies that Netbatch should write resource usage (rusage) information to the file at **path** every **interval** seconds. If the specified file already exists, it is appended to.

Rusage information is recorded for the Job itself and for the Job leader process (**nbjobleader.out**). The totals for both of these combined are also displayed.

The information written to the file is as follows:



- ◆ **currentTime** - the time when the information was written to the file
- ◆ **state** - the process state. It can be one of the following:
 - ◆ 1 - sleep
 - ◆ 2 - run
 - ◆ 3 - stopped
 - ◆ 4 - sleep_d
 - ◆ 5 - zombie
 - ◆ 6 - idle
 - ◆ 7 - other
- ◆ **pid** - process ID
- ◆ **ppid** - parent process ID
- ◆ **pgid** - process group ID
- ◆ **commandName** - command name
- ◆ **utime** - user time
- ◆ **stime** - system time
- ◆ **realMem** - amount of real memory used
- ◆ **virtualMem** - amount of virtual memory used
- ◆ **minflt** - number of minor page faults (i.e., those with no disk access)
- ◆ **majflt** - number of major page faults (i.e., those with disk access)
- ◆ **max_rss(max_memory)** - maximum resident set size
- ◆ **ixrss** - integral shared memory size
- ◆ **idrss** - integral unshared data size
- ◆ **isrss** - integral unshared stack size
- ◆ **starttime** - process start time
- ◆ **bread** - number of blocks read (only displayed in totals line)
- ◆ **bwrite** - number of blocks written (only displayed in totals line)
- ◆ **pckti** - number of packets received (only displayed in totals line)
- ◆ **pckto** - number of packets sent (only displayed in totals line)



- ◆ **swpin** - number of virtual memory page reads from swap (only displayed in totals line)
- ◆ **swpout** - number of virtual memory page writes from swap (only displayed in totals line)
- ◆ **nfs getattr** - NFS **getattr** operation count (only displayed in totals line)
- ◆ **nfs setattr** - NFS **setattr** operation count (only displayed in totals line)
- ◆ **nfs lookup** - NFS **lookup** operation count (only displayed in totals line)
- ◆ **nfs access** - NFS **access** operation count (only displayed in totals line)
- ◆ **nfs read** - NFS **read** operation count (only displayed in totals line)
- ◆ **nfs write** - NFS **write** operation count(only displayed in totals line)

--exec-limits <soft_limit>:<hard_limit>

Note

--exec-limits is deprecated. Use **--job-constraints** instead.
(For example, if you add **--job-constraints "wtime>3600:mail:kill"**, Netbatch will kill the Job and mail you if it runs for more than an hour.)

Specifies execution time limits for the Job:

- ◆ **soft_limit** - If exceeded, Netbatch sends email to the user.
- ◆ **hard_limit** - If exceeded, Netbatch kills the process, cancels the Job, and sends email to the user.

Note

Execution limits are rounded up to the nearest minute.

--expected-runtime <time>[{s|m|h|w}]

(HPC Pools only) Specifies the expected run time of the Job. If you do not specify a unit, **time** is in minutes. This is used together with **--renting-condition** to specify that the Job can use resources reserved by a larger Job (but that are not currently being used).



--hung-limits <soft limit>:<hard limit>

Note

--hung-limits is deprecated. Use **--job-constraints** instead.
(For example, if you add **--job-constraints "deltaovertime('15m','utime+stime')<60:mail:kill"**
Netbatch will kill the Job and mail you if it accumulates less than 60
seconds of execution time (utime plus stime) in any 15 minute period.)

Specifies hung limits for the Job.

- ◆ **soft_limit** - If the Job does not accumulate any CPU time within the specified time, Netbatch sends an email to the user.
- ◆ **hard_limit** - If the Job does not accumulate any CPU time within the specified time, Netbatch kills the process, cancels the Job, and sends email to the user.

Note

Hung limits are rounded up to the nearest minute.

--incremental-log

Specifies that each time the Job is resubmitted, a new log file is created with an incremental suffix. (For example, if the original log file is called **myjob.log**, additional log files are called **myjob.log.1**, **myjob.log.2**, and so on.)

--image <os_image>

Specifies that the Job requires a specific OS image. The Job will be run in a virtual machine that is instantiated specifically for this Job. When submitting a Job like this, you must also specify the class of machine required (using **--class**) and a valid class reservation (**--class-reservation**). Netbatch uses the class reservation expression to start the VM with the appropriate parameters.

To see the available OS images, use the **[nbstatus vm-image-cache](#)** command.

Note that it takes around two minutes to boot the virtual machine (and more like 10 minutes for a Windows VM). During this time, the Job remains in the **Send** state.



When the Job ends, the virtual machine stays up for thirty seconds. If another Job is submitted during this time that it can run (because it has the same or lower class reservation requirements, and it was submitted to a Qslot that can submit Jobs to the physical machine that the Vm is running on), the VM is reused. If not, it is shut down.

--job-constraints**"<expression>:<action>[:...][,...]"**

Specifies that the Netbatch should take one or more actions (including stopping, restarting, or resubmitting the Job), depending on various Job-related factors (which it measures once a second).

expression is either an arbitrarily complex boolean expression or an expression of the form:

idle(<interval_in_sec>,<cpu_in_percent>)

This evaluates to true if CPU usage was below the specified percentage during the specified interval.

If **expression** is an arbitrarily complex boolean expression, it consists of **<attribute>**s, **<operator>**s, and **<value>**s, and optionally **<functions>**s, where:

- ◆ **attribute** is one of the following:
 - ◆ **RM** - Jobs real used memory (MB)
 - ◆ **VM** - Jobs virtual memory (MB)
 - ◆ **utime** - Job process user time (seconds)
 - ◆ **nutime** - normalized Job process user time (seconds)
 - ◆ **stime** - Job process system time (seconds)
 - ◆ **nstime** - normalized Job process system time (seconds)
 - ◆ **wtime** - Job process wall clock time (seconds)
 - ◆ **BuffersRM** - the amount of physical RAM used for file buffers (MB)
 - ◆ **CachedRM** - the amount of physical RAM used as cache memory; memory in the pagecache (diskcache) minus SwapCache (the amount of Swap used as cache memory) (MB)
 - ◆ **InactiveRM** - the total amount of buffer or page cache memory that is free and available; this is memory that has not been recently used and can be reclaimed for other purposes by the paging algorithm (MB)
 - ◆ **fRM** - Free real memory; the amount of physical RAM left unused by the system (MB)



- ◆ **fVM** - Free virtual memory, the total amount of swap memory free (MB)
- ◆ **tRM** - Total real memory; the total usable RAM (i.e. physical memory minus a few reserved bytes and the kernel binary code) (MB)
- ◆ **tVM** - Total virtual memory; the total amount of physical swap memory (MB)
- ◆ **fds('<path>')** - Free disk space on the partition on which the specified directory resides (MB). Use **nbstatus workstations --fields "* ,fds"** to see the directories that Netbatch monitors.
- ◆ **users** - Number of active users
- ◆ **load** - Workstation load
- ◆ **Nload** - Normalized Workstation load
- ◆ **nice** - the amount of time that a Job has been niced
- ◆ **susp** - the amount of time the Job has been suspended (in seconds)
- ◆ **waittime** - the amount of time that the Job has been waiting to be dispatched for execution (seconds)
- ◆ **expected_runtime** - the Job's expected run time (in minutes), as specified in the **--expected-runtime** switch.

Note

waittime is only checked every 60 seconds.

Be careful choosing the operator to use with **waittime**. Because Netbatch only checks it every 60 seconds, an expression that uses == is very unlikely ever to be true.

Some actions are irrelevant for **waittime**. The only applicable actions are **kill** and **mail**.

- ◆ **TimeInState('<state>')** - the amount of time that the Job is currently in the specified state (milliseconds)

Valid states are: **send**, **resched**, **approving**, **susp**, **resub**, **del**, **wait remote**, **wait**, **run**, and **disc**.

**Note**

If the specified state is `disc`, the only valid action is `force-kill`.

Note

The `BuffersRM`, `CachedRM`, and `InactiveRM` attributes are available only for SUSE Linux Enterprise Server 9 and later.

Note

Attribute names are case-sensitive.

- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: `+`, `-`, `*`, `/`, `%`, unary `+` (e.g., `+10`), or unary `-` (e.g., `-10`)
 - ◆ A logical operator: `&&`, `||`, or `!`
 - ◆ A comparison operator: `<`, `<=`, `>`, `>=`, `!=`, or `==`
 - ◆ A non-strict operator: `is` or `isnt`
 - ◆ A bitwise operator: `&`, `|`, `~`, `^`, `<<`, `>>`, or `>>>`
 - ◆ A conditional operator: `?` or `:`
- ◆ **value** is a number, a string, or another boolean expression.
- ◆ **function** is one of:
 - ◆ `avgovertime('<time_interval>', '{RM|VM|RM+VM}')`
which returns the average of the sampled values of the specified parameter in `time_interval`.
 - ◆ `deltaovertime('<time_interval>', '{utime|stime|utime+stime}')`
which returns the difference between the maximum and minimum values of the specified parameter during `time_interval`.
 - ◆ `reservation('<capability>')`
which returns the value of the Job's reservation of the specified Workstation capability. (For example, if a Job



reserved two of the capability called **cores**,
reservation("cores") returns 2 for that Job.)

In both functions, **time_interval** is of the form
<number>[{ s | m | h }] (seconds, minutes, or hours). The default is minutes (so 60 means 60 minutes).

These are just the functions specific to Job constraints. For a full list of available functions, see [Appendix H: Functions](#), or run the following command:

```
nbstatus functions --context jobConstraints
```

Note

*When one of these functions is used, the value of the expression that contains the function and its maximum value are displayed in the RUsage field in the output of **nbstatus jobs**.*

For example, if you submit a Job with the following Job constraint:

```
"AvgOverTime('60s','RM')>1:kill"
```

then the nbstatus jobs RUsage field will include something like this:

```
avgovertime(60s;RM)= 0.0 , Max(avgovertime(60s;RM))= (0;64)
```

*For **Max()**, the two numbers are the maximum (of the average or delta), and the time at which the maximum occurred (in seconds from the start of execution).*

Expressions can be grouped together using parentheses.

action is one of the following:

- ◆ **kill**
- ◆ **force-kill**

This kills the Job, even if its state is **disc**. (See **TimeInState()**, above.)

- ◆ **mail**

Mail is sent using the template set up by the Netbatch administrator (or a default template if no such template exists).

- ◆ **mailonce**

This is the same as **mail**, except mail is only sent once (unless the Job is resubmitted).

- ◆ **recall**

- ◆ **nice**



When the condition is no longer true, the Job is reniced.

- ◆ **resubmit**
- ◆ **resubmit(<exit_status>)**

The same as **resubmit**, except that the Job is assigned the specified custom exit status.

- ◆ **suspend**

When the condition is no longer true, the Job is resumed.

- ◆ **resume**
- ◆ **nop**

No operation (for reporting)

- ◆ **kill(<exit_status>)**

The same as **kill**, except that the Job is assigned the specified custom exit status.

- ◆ **modify-reservation(<expression>)**

Modifies the Job's reservation according to **expression**. For example:

```
modify-reservation(memory=memory+2)
```

This Increases the Job's **memory** reservation by 2.

- ◆ **modify-properties(<attribute>=<new_value>[, . . .])**

Modifies the Job's custom attributes.

- ◆ **RunCommand(cmd=<command>)**

The specified command or script is executed on the Workstation where the Job is running.

The command has access to the following environment variables:

- ◆ **__NB_CLASS** - the class requirement with which the Job was dispatched
- ◆ **__NB_CMD** - the Job's command
- ◆ **__NB_DATE** - the date on which the email was sent
- ◆ **__NB_ERROR_MESSAGE** - the error message
- ◆ **__NB_EXIT_STATUS_CODE** - the Job's exit code
- ◆ **__NB_EXIT_STATUS_DESCRIPTION** - the Job's exit status description



- ◆ `__NB_JOB_CONSTRAINTS` - the Job constraints that apply to the the Job (as specified using `nbjob run`'s `--job-constraints` switch or in a Job profile)
- ◆ `__NB_JOB_ID` - the Job's ID
- ◆ `__NB_KILL_IF_SUSPENDED` - the number of minutes specified by the `nbjob run --kill` switch
- ◆ `__NB_MAIL_IF_SUSPENDED` - the number of minutes specified by the `nbjob run --mail __NB_min` switch
- ◆ `__NB_PGRID` - the Job's process group ID
- ◆ `__NB_PID` - the Job's process ID
- ◆ `__NB_POOL_ADMIN` - the ID of the Pool administrator (as specified by the `A` directive—see [Appendix E.2.11](#))
- ◆ `__NB_POOL_NAME` - the Pool name
- ◆ `__NB_QSLOT` - the Qslot from which the Job was dispatched
- ◆ `__NB_RESUBMIT_ATTEMPTS` - the number of resubmission attempts specified by the `nbjob run --trials` switch
- ◆ `__NB_RESUBMIT_IF_SUSPENDED` - the number of minutes specified by the `nbjob run --resubmit` switch
- ◆ `__NB_RUSAGE` - the Job's resource usage
- ◆ `__NB_STIME` - Job process system time
- ◆ `__NB_TIME` - the time at which the email was sent
- ◆ `__NB_USER_LOG_FILE` - the path/filename of the user log file
- ◆ `__NB_USER_NAME` - the Job owner's user
- ◆ `__NB_UTIME` - Job process user time
- ◆ `__NB_WORKING_DIR` - the user's working directory, from which the Job was submitted
- ◆ `__NB_WORKSTATION` - the hostname of the Workstation on which the Job is running
- ◆ `__NB_WTIME` - Job process wall clock time

Multiple actions can be specified for a single expression.

Multiple `<expression>:<action>[:...]` constraints can be specified.



Examples:

```
nbjob run --job-constraints "RM>500:mail:kill"  
myJob
```

If **myJob**'s real memory usage exceeds 500MB, kill it and mail the user.

```
nbjob run --job-constraints "RM>500 | RM>tRM/2:mail,  
RM>700:kill:mail" myJob
```

If **myJob**'s real memory usage exceeds 500MB or half of total real memory, mail the user. If its real memory usage exceeds 700MB, kill it and mail the user.

```
nbjob run --job-constraints  
"RM>500&&fRM<500:suspend:mail,  
fRM>500:resume:mail" myJob
```

If **myJob**'s real memory usage exceeds 500MB and free real memory drops below 500MB, suspend it and mail the user. If free real memory rises above 500MB again, resume **myJob**.

--job-limits "<key>=<value>[, . . .]"

Specifies that Netbatch should limit the amount of the specified resource(s) that the Job uses (only applicable for Workstations running SLES 11 and above and for which cgroups resource usage calculation is enabled). Available **keys** are:

- ◆ **VM**—value in megabytes (default: no limit)
- ◆ **RM**—value in megabytes (default: no limit)
- ◆ **cpushare**—a number that specifies the share of CPU relative to other Jobs (default value: 1024)

--job-starter <exec>

Start the Job with a starter program.

---kauth on|off

This switch specifies whether the Job uses Kerberos authentication or not.

- ◆ **on** - Job requires Kerberos authentication and will not be submitted if the user doesn't have a usable TGT at time of Job submission. This may override a Job Profile, or be overridden by a Job Profile, depending on how the Job Profile is set up.
- ◆ **off** - Job does not use Kerberos authentication. This may override a Job Profile, or be overridden by a Job Profile, depending on how the Job Profile is set up.



If this switch is not specified, and no Job Profile specifies that Netbatch should use Kerberos authentication, the Job is submitted with a TGT—if one is available on the submission host at time of submission. (See [Section 6.9.2](#).)

If this switch is not specified—and the user does NOT have an available TGT on the submission host at time of submission—the submitted Job may fail at execution when trying to access services that require authentication.

Note

A Job submitted with a TGT that expires, or is invalid, may fail at execution. (For renewing TGTs, see [Section 6.9.3](#).)

--kill <minutes>

Specifies that Netbatch should kill the Job if it is suspended for more than the specified number of minutes. This does not apply when the Job is suspended by the owner or the administrator.

--license "<expression>"

Specifies the license(s) that the Job will need during its execution, and (optionally) the amount of time each one is required. This overrides any license requirements defined in the user's Job profile file or **NBLICENSES** environment variable.

expression consists of any number of **<req_expr>**s and **<operator>**s.

req_expr is a license requirement expression of the following format:

"<license>=<value>[:duration=<duration_time>]"

where:

- ◆ **license** is the name or alias of the required license.
- ◆ **value** is the number of instances of **license** that are required.
- ◆ **duration_time** is the amount of time (in minutes) for which the license is required (starting from when the Job is dispatched for execution).

If not specified, then **duration** is set to the Job execution time (that is, the license is reserved for the entire duration of the Job's execution).

operator is **&&** or **||**.



These can be grouped together using parentheses.

```
--license-keyfile "<keyfile path>|<port@host>[ :... ]"
```

Specifies the license keyfiles to be used. (This is instead of setting the **NB_LICENSE_FILE** environment variable.) If multiple arguments are specified, use colons as delimiters.

Note

If you specify a license that the Job requires by its alias, you do not need to specify the license keyfile.

```
--license-reservation <reservation_item>[ ,... ]
```

Note

In earlier versions of Netbatch, this allowed license availability to be calculated much more accurately than just using the --license switch, which meant that license usage was much more efficient.

This functionality is now implemented automatically. All required licenses are automatically reserved.

If you use this switch, Netbatch automatically converts the specified requirement so that it is the same as if you used --license.

Specifies the license(s) that the Job will need during its execution, and the amount of time each one is required.

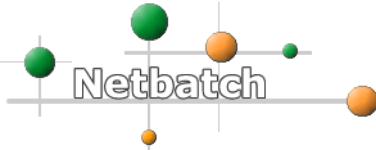
Each **reservation_item** string must be in the following format:

```
"<key>=<key_value>[:duration=<duration_time>]"
```

where:

- ◆ **key** is the feature to be reserved.
- ◆ **key_value** is the number of instances of **key** that are required.
- ◆ **duration_time** is the amount of time (in minutes) for which the feature is required (starting from when the Job is dispatched for execution).

If not specified, then **duration** is set to the Job execution time (that is, the feature is reserved for the entire duration of the Job's execution).

**--log-file <job_name>**

Specifies that the Job's log file will be called **job_name**, and that the Job will have the Job name **job_name** (that is, the Job's name is displayed as **job_name** in the output of the **nbstatus jobs** and **nbqstat** commands).

job_name can be just a filename (e.g., **myJob**), or can include the path (e.g., **/tmp/myJob**).

--log-file-dir <path>

Specifies that the Job's log file will be written in the specified directory.

If **--log-file** is used as well and its argument includes a path, **--log-file-dir** is ignored.

To specify a log file directory for the pre- and/or post-execution script that is different from the Job's log file directory, use the **--log-file-dir-post-exec** or **--log-file-dir-pre-exec** switch, respectively.

--log-file-dir-post-exec <path>

Specifies that the post-execution script's log file will be written in the specified directory. If no post-execution script is specified (with **--post-exec**), this switch is ignored.

--log-file-dir-pre-exec <path>

Specifies that the pre-execution script's log file will be written in the specified directory. If no pre-execution script is specified (with **--pre-exec**), this switch is ignored.

--mail "[S] [E] [no] [<min>]"

Send mail to the user when the Job starts (**s**), ends (**e**), is suspended for more than **min** consecutive minutes, or not at all (**no**). If multiple arguments are used, a space is used as the delimiter (e.g., **mail "S E 60"**)

If **--mail-list** is specified, mail is sent to all specified addresses. (See **--mail-list**.)

--mail-list <address>| "<address>[. . .]"

List of mail recipients for Netbatch messages about the Job. (Mail is sent to these addresses as well as to the user.) If more than one address is specified, a space is used as the delimiter, and the list of addresses must be enclosed in double quotes.

**--master-class <class>**

For a parallel Job, this specifies the master Job's class requirements. The master Job will only run on a Workstation that satisfies both these requirements and those specified in --class.

Note that this switch cannot be used together with --parallel-slave-class (now deprecated).

--mode {interactive|i|interactive-ls|blocking|b|terminal|t}

Run in interactive, interactive (local signals), blocking, or terminal mode:

- ♦ **interactive|i** - the Job appears as though it is executing on the submitting host machine (especially suited for batch execution of complex sequences of interdependent Jobs defined in a Makefile).
- ♦ **interactive-ls** - like **interactive**, except that **nbjob run** handles signals locally, instead of passing them to the remote process.
- ♦ **blocking|b** - no more commands may be executed until the Job ends.
- ♦ **terminal|t** - interactive applications that manipulate user input and screen display via a terminal device can be run as Netbatch Jobs (for example, emacs or tcsh).

Note

As of version 6.3.1, Netbatch automatically senses the context in which an interactive or terminal Job is being executed, and runs it as either an interactive or terminal Job, whichever is appropriate.

Thus, there is no need to use --mode terminal. Whenever you want the Job to run as if it is running locally, use --mode interactive.

See [Section 6.17, Terminal Jobs](#).

Note

*You can shorten **interactive**, **blocking**, and **terminal**. For example, **i**, **in**, and **inte** can all be used instead of **interactive**.*

**--no-env**

Does not pass user environment variables with the Job.

User Job will be executed on remote machine without the user's environment variables.

--num-of-checkpoints <number>

Specifies the number of checkpoints to save. Older checkpoints will be deleted. If number is 0, all checkpoints are saved. If the switch is omitted, the last five checkpoints are saved.

This switch is only valid if **--checkpoint-tool** is used.

--on-job-finish

"{<exit_code>|<expression>}:<operation>[:...]
[,...]

Specifies that if the Job exit code is **exit_code** or if it matches **expression**, the specified **operation**(s) are performed.

exit_code is a numerical Job exit code (see **Exit Status**) or one of the following macros:

- ◆ **NBErr** - all Netbatch errors except -8, -9, -10, and -11
- ◆ **Nexit** - all negative Job exit values
- ◆ **Pexit** - all positive Job exit values
- ◆ **NZexit** - all non-zero Job exit values

expression is an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is any **nbstatus jobs** field (though typically, you will use **ExitStatus**, **PreExecExitStatus**, and **PostExecExitStatus**).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")
 - ◆ A logical operator: &&, ||, or !
 - ◆ A comparison operator: <, <=, >, >=, !=, or ==
 - ◆ A non-strict operator: **is** or **isnt**
 - ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
 - ◆ A conditional operator: ? or :
- ◆ **value** is a number, a string, or another boolean expression.



Expressions can be grouped together using parentheses.

For example,

```
ExitStatus== -303 && PreExecExitStatus == 2
```

operation is one of the following:

- ◆ **Requeue(<max_attempts>)** - the Job is requeued up to **max_attempts** times.
- ◆ **Exclude(<period>{s|m|h})** - if the Job was submitted to a Virtual Pool, exclude the last resource (Pool) that the Job was dispatched to from those that it can be sent to, for the specified time. (If the Job was *not* submitted to a Virtual Pool, exclude the last Workstation that the Job was dispatched to from the list of Workstations that the Job can run on, for the specified time.)
- ◆ **Modify('<parameter>=<expression>' [; . . .])** - modify the Job before resubmitting it. You must also specify a **Requeue** action, otherwise the Job will not be resubmitted.

Here, **parameter** is a valid nbjob **modify** parameter (not including Queue) and **expression** specifies the value that is assigned to the parameter. **expression** can include any **nbstatus** jobs field.

It can also include regex replace functions (see [Appendix G: Regex Replace Functions](#)).

If **expression** is a string, it must be enclosed in quotes. But because the entire on-job-finish expression is already in quotes, you must escape these quotes. In bash, use `\"`. In tcsh, use `"\ "`.

Examples:

```
--on-job-finish
"Nexit:Modify('priority=priority+1';
'class=\"\\"4G\"\\"'):Requeue(1)"
```

If the Job ends with a negative exit status, its priority is incremented and the specified class requirement is added to it before it is requeued.

```
--on-job-finish "Nexit:Modify('job-
constraints=\\"VM>5:mail\"\\"'):Requeue(1)"
```

Here, if the Job ends with a negative exit status, the specified Job constraint is added to it before it is requeued.

- ◆ **requeuefromlastcheckpoint(<num_retries>)** - the Job is requeued up to **max_attempts** times, using the last checkpoint that was created for the Job.



You can only requeue a Job from a checkpoint if it was submitted with the **--checkpoint-tool** switch.

Note

*If you specify both **Requeue** and **Exclude** operations, **Exclude** must come first.*

*If you specify both **Modify** and **Requeue**, **Modify** must come first.*

*If you specify multiple **exit_code|expression:operations**, only the operation for the first match will be performed.*

*This switch overrides the **NB_ON_JOB_FINISH** environment variable.*

*If **Exclude** is used and the Job is submitted to a Virtual Pool or a Netbatch Feeder, the exclude flag is not passed to the physical Pool Master. (If it was passed to the physical Pool Master, the Job could be requeued by both the physical Pool Master and the Virtual Pool Master/Feeder.)*

Note

--on-job-finish takes precedence over **--autoreq**.

For Example

```
--on-job-finish "ExitStatus== -303&&
PreExecExitStatus==2:Exclude(10h):Requeue(3)"
```

If the Job exit status is -303 and the exit status of the pre-execution stage is 2, then the Job is to be requeued up to three times. The Workstation that the Job was dispatched to should be excluded from the list of Workstations that it can run on for ten hours.

--pag on|nosuspend

Netbatch uses Process Authentication Groups (PAGs) to track Jobs. This switch specifies whether a PAG should be added to the Job, and how:

- ◆ **on** - add a PAG to the Job. This may override a Job Profile, or be overridden by a Job Profile, depending on how the Job Profile is set up.
- ◆ **nosuspend** - add a PAG to the Job, but do not suspend child processes that change their process group ID.



If this switch is not specified, and no Job Profile specifies that Netbatch should apply a PAG, Netbatch decides whether to apply a PAG based on how the Workstation is configured.

--parallel <parallel reservation string>

Specifies that the Job is the Master Job of a Parallel Job, to be executed according to the parameters in the string.

The parallel reservation string take the following form:

```
"slots=[<min_slots>-]<max_slots>[,  
slots_per_host=[<min_sph>-]<max_sph>][,  
reservation_per_host={true|false}][,  
preserve_slot={true|false}][,  
exit_on_master_finish={true|false}][,  
slave_validator=<validate_script>]"
```

where:

- ◆ **min_slots** (optional) is the minimum number of execution slots required
- ◆ **max_slots** is the maximum number of execution slots required
- ◆ **min_sph** (optional) is the minimum number of execution slots required on a single host
- ◆ **max_sph** is the maximum number of execution slots required on a single host
- ◆ **reservation_per_host** indicates that the requested resource reservation will be made once for each scheduled executing host, regardless of the number of parallel slots scheduled for it.

The default value is **false**.

For example, if two Slave Jobs are scheduled to execute on a Host, each of which has a resource reservation of 10GB of memory:

- ◆ If **reservation_per_host** is **false**, 10GB will be reserved for each Job (20GB in total).
- ◆ If **reservation_per_host** is **true**, a total of 10GB will be reserved.
- ◆ **preserve_slot** specifies whether an execution slot will be released (**false**) or not (**true**) when the Slave Job running in it ends. If **preserve_slot=true**, the execution slot can be reused.

This behavior can also be specified at the Slave Job level (but only if **preserve_slot=false** for the Master Job). To do



this, add **--preserve-slot** to the **nbexec** command for the relevant Slave Job(s).

- ◆ **exit_on_master_finish** specifies what happens to Slave Jobs that are still running when the Master Job ends. If your Slave Jobs should no longer be running when the Master Job ends, set this to **true**. Otherwise, set it to **false** (the default).
- ◆ **validate_script** is the name of a script that is run to determine whether a particular Workstation can or cannot run one of the Job's Slave Jobs. The user is responsible for creating this script.

If the script determines that the Workstation is suitable, it should return zero. If not, it should return a non-zero value.

If the script returns a non-zero value, Netbatch excludes the Workstation and looks for a replacement.

How it works: Netbatch runs the script for each Workstation only after it has successfully reserved all the requested slots. If the script indicates that one or more Workstations are not suitable, the Job is resubmitted to the wait queue. Netbatch then starts looking for available execution slots again. Ones for which validation failed are not considered.

```
--parallel-job-constraints
"<expression>:<action>[:....][,...]"
```

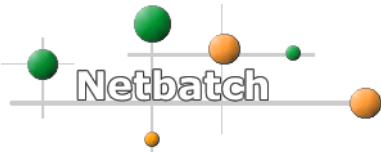
Specifies that the Netbatch should take one or more actions (including stopping, restarting, or resubmitting the Job), depending on various Job-related factors (which it measures once a second). **expression** is either an arbitrarily complex boolean expression or an expression of the form:

```
idle(<interval_in_sec>,<cpu_in_percent>)
```

This evaluates to true if CPU usage was below the specified percentage during the specified interval.

If **expression** is an arbitrarily complex boolean expression, it consists of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is one of the following:
 - ◆ **RM** - the total real memory use by all the Jobs that make up the Parallel Job (MB)
 - ◆ **VM** - the total virtual memory use by all the Jobs that make up the Parallel Job (MB)
 - ◆ **utime** - the user time that has been accumulated by all the Jobs that make up the Parallel Job (seconds)



- ◆ **sftime** - the system time that has been accumulated by all the Jobs that make up the Parallel Job (seconds)
- ◆ **wftime** - the wall clock time that has been accumulated by all the Jobs that make up the Parallel Job (seconds)

Note

These conditions are checked every 5 minutes.

Note

Attribute names are case-sensitive.

- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., +10), or unary - (e.g., -10)
 - ◆ A logical operator: &&, ||, or !
 - ◆ A comparison operator: <, <=, >, >=, !=, or ==
 - ◆ A non-strict operator: **is** or **isnt**
 - ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
 - ◆ A conditional operator: ? or :
- ◆ **value** is a number, a string, or another boolean expression.

Expressions can be grouped together using parentheses.

action is one of the following:

- ◆ **kill**
- ◆ **mail**
- ◆ **recall**
- ◆ **nice**

When the condition is no longer true, the Job is reniced.

- ◆ **resubmit**
- ◆ **suspend**

When the condition is no longer true, the Job is resumed.

Multiple actions can be specified for a single expression.



Multiple `<expression>:<action>[:...]` constraints can be specified.

--parallel-slave-class <class_expression>

Specifies the class of machine that the Parallel Job's Slave Jobs require. (The syntax is the same as for `--class`, above.)

If this switch is not used, any class requirements specified with `--class` apply to the Master Job and all the Slave Jobs.

If you need to specify class requirements for the Master Job only, add the following switch in addition to the `--class` switch:

--parallel-slave-class "@"

--parallel-topology-latency "max<=<int>"

Defines the maximum latency rating allowed for running the job. (0 = no latency.)

--post-exec <exec>

Execute a post-execution program after the Job runs.

To specify a log file directory for the post-execution script that is different from the Job's log file directory, use the `--log-file-dir-post-exec` switch.

Note

In the Post-Execution phase, the following environment variables are available:

- **NB_JOB_EXIT_STATUS**—the Job's exit status
- **NB_CLASS**—the Job's class requirement expression
- **NB_CMD_LINE**—the Job's full command line
- **NB_FINISH_TIME**—the time at which the Job finished executing
- **NB_JOBID**—the Job's ID
- **NB_LOG_FILE**—the location of the Job log file
- **NB_POOL**—the name of the Pool that the Job was submitted to
- **NB_QSLOT**—the Qslot that the Job was submitted to
- **NB_QUEUE**—the Queue that the Job was submitted to
- **NB_RUSAGE**—information about the Job's resource usage
- **NB_START_TIME**—the time at which the Job started executing



- **NB_SUBMIT_PWD**—the user's working directory when the Job was submitted
- **NB_TIMES_RESTARTED**—the number of times that the Job was restarted

Note

The post-execution program is executed even if the Job is killed or suspended.

--pre-exec <exec>

Execute a pre-execution program before the Job runs.

To specify a log file directory for the pre-execution script that is different from the Job's log file directory, use the

--log-file-dir-pre-exec switch.

Note

The following environment variables contain information that the pre-exec script can use:

- **NB_POOL**
- **NB_QUEUE**
- **NB_QSLOT**
- **NB_LOG_FILE** (but only if the **--log-file** option is used)
- **NB_CLASS**

--priority <number>

Specifies the Job priority as a number (any integer, where a higher number means a higher priority).

--properties

"<prop_name>[[<sub_prop>]]=<value>[, . . .]"

Specifies properties for the Job. **prop_name** is the name of a property. **value** is the property's value.

If the property has sub-properties, specify the name of the sub-property in square brackets. For example, for a property called **RunawayFields**, to set the value of the sub-property called **Threshold1**, specify the following:

"RunawayFields[Threshold1]=5"



The Qslot that you are submitting the Job to may have configured rules that require you to specify values for certain properties when submitting a Job.

Optionally, you can also supply values for any configured property. If you omit required properties when trying to submit a Job, you will receive an error message that lists the properties that you must supply values for.

--qslot <qslot>

Specifies the full path or alias of the Qslot the Job should be sent to.

--queue <queue>

Specifies the Queue that the Job is submitted to.

--remote-pools { "[-]<phys_pool>" | "[-]<alias>" } [,...]

Specifies the preferred Physical Pool(s) by name or by alias (only relevant when submitting a Job to a Virtual Pool). You can specify Pools by:

- ◆ Inclusion -- only the listed physical Pools will be considered when deciding which Pool to send the Job to.
- ◆ Exclusion -- precede Pool/alias names with -. All relevant physical Pools will be considered except those that are listed.
- ◆ A combination -- if there are both included and excluded Pools/aliases in the list, the excluded ones are excluded from the list of included Pools.

You can use this to specify that a specific physical Pool should be excluded from the set of Pools represented by an alias.

This situation can also arise when the request uses inclusion and a Job Profile uses exclusion, or vice-versa.

The order determines the order in which the Pools will be tried. Netbatch will try the first Pool, and will only try the next one if the previous one has reached its **WaitJobsLimit** or **MaxJobsLimit**.

If an alias is specified, Netbatch uses its existing algorithm to decide which Pool to send the Job to. (Multiple Pools can have the same alias. For example, all the Physical Pools at the same site as the Virtual Pool Master might have the alias **local**.)

This switch overrides the **NB_POOLS** environment variable.

**--renting-condition <expression>**

(HPC Pools only) Specifies that the Job can borrow resources that have been reserved by other Jobs if **expression** evaluates to true. (**expression** is evaluated in the context of the Job that this switch is applied to.)

If you use this switch, you must also indicate the Job's expected run time using the **--expected-runtime** switch.

--resubmit <minutes>

Specifies that Netbatch should resubmit the Job if it is suspended for more than the specified number of minutes. See also **--trials**.

Note

*This does not apply to Jobs that are resubmitted by the Netbatch Administrator or by a user with Queue/Qslot administrative rights (using **nbjob suspend** or **nbqrm --suspend**).*

--resubmit-to-source

(Virtual Pool only) Specifies that if the Job is resubmitted for whatever reason (for example, if it is preempted) it is to be resubmitted to the Virtual Pool, not the physical Pool.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--rlimits "<res_limit> [soft|hard] = value [, ...]"

Set resource consumption limits:

- ◆ **res_limit** is the name of the limit to be set.
- ◆ **soft|hard** (optional) specifies whether a soft or hard limit is to be set. If not specified, both hard and soft limits are set to the same value.



- ♦ **value** is the value to be assigned to the limit.

The limits that can be set are listed in **Table 1**, below. Some are platform-specific. If a limit is set for a platform that does not support it, it is ignored.

If a limit is not set, the limit is inherited from the WSM process that spawned the process. The WSM process, in turn, inherits the limits from the shell from which it was invoked.

Note

*You can use the built-in **limit** command (in bash and ksh, **ulimit -a**) to view current resource limits. Note, however, that this shows the limits on the Workstation that you are logged into, not the Workstation on which your Job will run.*

*If you need to know the limits that will apply to your Job, run the **limit** command on a Workstation whose capabilities match those required by your Job.*

*The output of the **limit** command is different for each platform.*

Table 1: Resource Limits

Limit	Units	Description	Platform
addressspace	KB	The maximum size of a process's available memory	All
coredumpsize	KB	The maximum size of a core file that a process may create	All
cputime	Seconds	The maximum amount of CPU time that a process may use	All
datasize	KB	The maximum size of a process's heap	All
descriptors	# of files	Maximum number of files that a process may open (may be called openfiles on some platforms/in some shells)	All
filesize	KB	The maximum size of a file that a process may create	All
memorylocked	KB	Maximum locked-in memory address space	Linux
memoryuse	KB	The maximal resident set size of a process	Linux, HP-UX
openfiles	# of files	Maximum number of files that a process may open (may be called descriptors on some platforms/in some shells)	All

**Table 1: Resource Limits**

Limit	Units	Description	Platform
maxproc	# of processes	The maximum permitted number of processes	Linux
stacksize	KB	The maximum size of a process's stack (can also be set to <code>unlimited</code>)	All
vmmemoryuse	KB	The maximum size of a process's mapped address space	Solaris

--silent

Parallel Jobs that use MPI or Linda use the `rsh.nb` script instead of `rsh`. `--silent` suppresses the messages that Netbatch usually sends to stdout when `nbq` or `nbjob run` is run interactively (with `nbq -i` or `nbjob run --mode interactive`). This allows `rsh.nb` to mimic `rsh` as closely as possible.

--task <taskname>

Specifies the Job task name (so that associated Jobs can be identified as such).

You can make a Job dependent on a task (so that it is only dispatched for execution when all the Jobs that make up the task have ended).

--trials <number>

If `--resubmit` (or `-r`) is used, specifies the number of times to resubmit the Job before killing it.

--triggers "<boolean_expression>"

Specifies that the submitted Job depends on other Jobs, as defined in `boolean_expression`.

`boolean_expression` consists of any number of terms of the form `[!]<JobID>[[<exit_status_expression>]]`, separated by the logical operators `&&` (AND) and `||` (OR).

Note

`JobID` can be either a Job ID or a task name. You are advised to explicitly specify whether it is a Job ID or a task name, because in ambiguous cases, Netbatch can assume that a task name is actually a Job ID, or vice versa.

Use one of the following instead of a Job ID/task name:

`jobid:<Job_ID>`

**task:<task_name>**

If a task name is used, the **exit_status_expression** does not evaluate to true until it is true for each of the Jobs that make up the task.

See [Section 6.5.18, Specifying a Task Name](#).

exit_status_expression can be one of the following:

- ◆ **exit**
- ◆ **exit <relational_operator> <exit_code>**

relational_operator must be <, <=, ==, >=, or >.

To specify multiple acceptable exit code conditions for a specific Job, use multiple **<JobID>[<exit_status_expression>]** terms separated by || (OR) (for example, **1 exit <30 || 1 exit>50**).

If **exit** is used on its own, it means that the Job ended (no matter what its exit status).

If no **exit_status_expression** is specified, **done** is assumed.

Note

Logical expressions are evaluated according to the following order of precedence - () (parentheses), ! (NOT), && (AND), || (OR).

So

"1 [exit<30] || !2 && (3 || 4)"

is the same as

"1 [exit<30] || ((!2) &&(3 || 4))"

--validate

Specifies that the Job should not be submitted, but instead that Netbatch should check its class requirements to see whether they can be satisfied by any of the Workstations in the Pool.

With this switch, **nbjob run** displays one of the following:

- ◆ **Job not queued (validate mode). Class `<class_name>' is served by this pool.**
- ◆ **Job not queued. Class `<class_name>' is not served by this pool.**

**--wash**

Specifies that the Job should be run with wash support. That is, only the filesystem groups specified in the **NB_WASH_GROUPS** environment variable will be applied to the Job.

Use this when you belong to more than 16 groups and you want to specify which ones are applied to the Job.

Note

A Job's owner must belong to all of its wash groups according to the group map.

See [Section 6.1.4, What to Do if you Belong to More than 16 Groups](#).

--work-dir <path>

Specifies the directory that the Job is to be run from.

--wsrank-input required_datasets="*<ds_name>[,...]*"

Specifies the CaMa datasets that the Job requires. Netbatch uses this information to match the Job with a Workstation that already has the required datasets (if it is configured to do so).

Output

Netbatch creates a log file containing the Job output named:

##Date-Time#.<machine>

See [Appendix D, Log File Contents and Job Exit Codes](#) for detailed information about Netbatch log file output.

Note

If more than one Job starts at the same time on the same machine, Netbatch appends the Pool name and the Job ID to all the Jobs except the first Job to differentiate between Jobs.

**Note**

For Parallel Jobs, Netbatch names log files as follows:

- For each executed Job, Netbatch creates a separate output log file, in the standard format (see above).
- If a Job name is specified when submitting the Master Job (that is, with `nbjob run --log-file <job_name>` or `nbq -j <job_name>`), then the log files are called `job_name:<index>`.
- If a Job name is specified when submitting the Slave Job (that is, with `nbjob prun --log-file <job_name>` or `nbexec -j <job_name>`), then the Slave Job's log file is called `job_name`.

A.2.2 nbjob checkpoint

Name

nbjob checkpoint - manually create a checkpoint for one or more Jobs.

Description

Manually create a checkpoint for one or more Jobs.

Note

You can only create a checkpoint for a Job if it was submitted with the `--checkpoint-tool` switch.

Usage

```
nbjob checkpoint [--target <pool_name>|<host_name>]  
<specification>
```

Target

```
--target <pool_name>|<host_name>
```

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If `--target` is omitted, `nbjob` uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)



Options

specification

This is a boolean expression that specifies the Jobs for which a checkpoint is to be created.

specification is either **all** or an arbitrarily complex boolean expression.

This expression consists of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid Job field (see **nbstatus jobs --fields**, [Appendix A.3.10](#)).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")
 - ◆ A logical operator: &&, ||, or !
 - ◆ A comparison operator: <, <=, >, >=, !=, or ==
 - ◆ A non-strict operator: **is** or **isnt**
 - ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
 - ◆ A conditional operator: ? or :
- ◆ **<value>** is a number, a string, or another boolean expression.

Note

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If **value** is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

Expressions can be grouped together using parentheses.

If **specification** is **all**, a checkpoint will be created for each of the user's checkpoint-enabled Jobs.

The **startswith()** and **glob()** functions can also be used:

```
"startswith(<field_name>,'<string>')"
```



startswith() matches fields whose value starts with the specified string.

"glob(<field_name>, '<expression>')"

glob() matches fields that match **expression**.
expression can contain:

- ◆ Any character
- ◆ . - matches any character
- ◆ \w - matches any letter (upper- or lower-case)
- ◆ \d - matches any digit
- ◆ \s - matches any space character
- ◆ * - zero or more of the previous character
- ◆ ? - exactly one of the previous character

A.2.3 nbjob modify

Name

nbjob modify - Modify the parameters of a Job that has already been submitted.

Description

Modify one or more of the following Job parameters:

- ◆ The Queue and/or Qslot to which the Job was submitted
- ◆ Class requirements
- ◆ Class (resource) reservation requirements
- ◆ License requirements
- ◆ License keyfile to be used
- ◆ Parallel reservation requirements
- ◆ Priority
- ◆ File copying parameters
- ◆ Jobs on which the Job depends (trigger Jobs)
- ◆ Job properties
- ◆ The remote (physical) Pools that the Job can use (Virtual Pool only)



If the Job is running, it is **resubmitted** with the modified parameters, unless the **--no-restart** switch is used.

When a user runs **nbjob modify**, it only makes changes to a Job belonging to that user.

A user may have privileges to modify the parameters of a Job that belongs to another user in a Qslot. This is defined per Queue or per Qslot.

The Netbatch Administrator can use **nbjob modify** to make changes to a Job that belongs to any user.

Usage

```
nbjob modify [--target <pool_name>|<host_name>]
[<options>] <specification>
```

Target

```
--target <pool_name>|<host_name>
```

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbjob** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

Note

The following switches can be used to modify a Job's parameters, or to add parameters to a Job that was submitted without them.

When used to modify existing parameters, the new parameter completely replaces the old one.

```
--action-signal
"<action>[:signal=<type>][,timeout=<time>]
[:cmd=<cmd_name>][;...]"
```

Specifies that for the specified action, the Job will be "soft killed", that is, the specified signal will be sent to the specified process (instead of to all the Job's processes, as in the default sequence). Here:



- ◆ **action** is the action to which the soft kill will be applied. It is one of:
 - ◆ **remove**
 - ◆ **suspend**
 - ◆ **recall**
- ◆ **type** is the signal that will be sent. Any Unix signal may be sent, except SIGKILL and SIGSTOP. (Default: SIGTERM for remove and recall, SIGTSTP for suspend.)
- ◆ **time** is the amount of time (in seconds) to wait between send the first "soft" signal and continuing with the default sequence (for remove and recall, SIGKILL is sent, for suspend SIGSTOP). (Default: five seconds)

This overrides the timeout specified by **NBSIGKILL_TIMEOUT** or **NB_SUSPEND_TIMEOUT** (depending on the action).
- ◆ **cmd_name** is the command name of the process to which the signal will be sent. (Default: the first process under the Job leader)

If multiple processes have the same command name, the signal will be sent to all of them.

Note

If the Job is killed, removed, resubmitted, or suspended by preemption or by a Workstation policy, the signal (and other parameters) specified in --action-signal are used.

--class <class_expression>

Changes the class of machine (that is, the required Workstation attributes) that the Job requires. These can include static attributes (such as OS type and number of CPUs) and dynamic attributes (such as free virtual memory and load).

class_expression is an arbitrarily complex boolean expression consisting of <attribute>s, <operator>s, and <value>s, where:

- ◆ <attribute> is one of the available Workstation attributes or an external probe parameter name.

For a list of Workstation attributes, see [Section 6.7.2, Viewing Supported Workstation Attributes](#).



This includes total and free disk space on partitions on which directories defined by the Netbatch Administrator reside. Run **nbstatus workstations --fields all** and look at the values of the **fDS** and **tDS** fields to see the partitions whose disk space Netbatch monitors.

To see the available Workstation probes and the parameters that they measure, run **nbstatus workstations --fields all** and look at the value of the **ExternalProbeData** field.

To see the available Scheduler probes and the parameters that they measure, run **nbstatus probes**.

See [Appendix A.3.31, nbstatus workstations](#) for more details.

- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")
 - ◆ A logical operator: &&, ||, :, or !

|| and : both mean OR. But for parallel Jobs, : means that all the slave Jobs must run on Workstations that match the same class or class expression (ORed item).

If you use ||, each of the parallel Job's slave Jobs can run on Workstations that match *any* of the specified classes or class expressions (ORed items).

- ◆ A comparison operator: <, <=, >, >=, !=, or ==
- ◆ A non-strict operator: **is** or **isnt**
- ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
- ◆ A conditional operator: ? or :
- ◆ <**value or expression**> is a number, a string, or another boolean expression.

Expressions can be grouped together using parentheses.

--class-reservation <reservation_item>[,...]

Modifies the Job's class reservation requirements.

If the Job is running, it is **resubmitted** with the new class reservation requirements and all reserved resources are released.

If the Job is waiting, all reserved resources are released.

reservation_item must be in the following format:



```
"<key>=<key_value>[:duration=<duration_time>]
[:decrease=<decrease_time>]
[:delta=<delta_value>]"
```

where:

- ◆ **key** is the type of resource to be reserved. It can take the following values:
 - ◆ **fVM**
 - ◆ **fRM**
 - ◆ **dedicated**
 - ◆ The name of a custom capability or attribute that has been defined for one or more Workstations in the Pool

You can use the **nbstatus workstations** command to see a list of these attributes:

```
nbstatus workstations --fields
"* ,WSCapabilities"
```

These capabilities can either relate to total real or virtual memory or be arbitrary.

Note

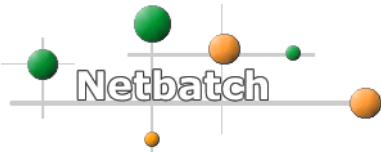
Workstation capability requirements can also be defined at the Queue or Qslot level. Such requirements apply to all Jobs submitted to the Queue/Qslot.

- ◆ An external probe parameter name

To see the available Workstation probes and the parameters that they measure, run **nbstatus workstations --fields all** and look at the value of the **ExternalProbeData** field.

To see the available Scheduler probes and the parameters that they measure, run **nbstatus probes**.

See [Appendix A.3.31, nbstatus workstations](#) for more details.

**Note**

If **dedicated=true**, then no other Jobs may run on the host while the Job is running (except for Parallel Slave Jobs - multiple Slave Jobs that are part of the same Parallel Job can run on the same host).

- ♦ **key_value** is the value for **key**. It must be one of the following:
 - ♦ For **fvm**, **frm**, or a custom field—any type of number, integer or non-integer value
 - ♦ For **dedicated**, **true** or **false**
- ♦ **duration_time** is the amount of time (in minutes) for which the resources (**fvm** and **frm** only) are required (starting from when the Job is dispatched for execution).

If not specified, then **duration** is set to the Job execution time (that is, the resource is reserved for the entire duration of the Job's execution).

- ♦ **decrease_time** - During the time specified by **duration**, the committed resource (**fvm** and **frm** only) is reduced every **decrease_time** minutes by the amount specified in **delta_value**.

If a **delta_value** is specified, but no **decrease_time**, then **decrease = 1 minute**.

If neither a **delta_value** nor a **decrease_time** is specified, then the resource is available at the same level for the whole time specified by **duration**.

- ♦ **delta_value** - During the time specified by **duration**, the committed resource (**fvm** and **frm** only) is reduced every **decrease_time** minutes by the amount specified in **delta_value**.

If a **decrease_time** is specified, but no **delta_value**, then **delta = 1 unit**.

If neither a **delta_value** nor a **decrease_time** is specified, then the resource is available at the same level for the whole time specified by **duration**.

--force

Resubmits the Job with the specified changes, even if it is disconnected>Disc-D or Disc-R). This removes the Job from the Pool Master. Only use this switch if you know that the Workstation that the Job was sent to is actually down.



Only root or a user with administrative permissions for the Qslot that the Job was submitted to can use this switch.

```
--job-constraints
"<expression>:<action>[:...][,...]"
```

Specifies that the Netbatch should take one or more actions (including stopping, restarting, or resubmitting the Job), depending on various Job-related factors (which it measures once a second).

expression is either an arbitrarily complex boolean expression or an expression of the form:

```
idle(<interval_in_sec>,<cpu_in_percent>)
```

This evaluates to true if CPU usage was below the specified percentage during the specified interval.

If **expression** is an arbitrarily complex boolean expression, it consists of <attribute>s, <operator>s, and <value>s, and optionally <functions>s, where:

- ◆ **attribute** is one of the following:
 - ◆ **RM** - Jobs real used memory (MB)
 - ◆ **VM** - Jobs virtual memory (MB)
 - ◆ **utime** - Job process user time (seconds)
 - ◆ **nutime** - normalized Job process user time (seconds)
 - ◆ **stime** - Job process system time (seconds)
 - ◆ **nstime** - normalized Job process system time (seconds)
 - ◆ **wtime** - Job process wall clock time (seconds)
 - ◆ **BuffersRM** - the amount of physical RAM used for file buffers (MB)
 - ◆ **CachedRM** - the amount of physical RAM used as cache memory; memory in the pagecache (diskcache) minus SwapCache (the amount of Swap used as cache memory) (MB)
 - ◆ **InactiveRM** - the total amount of buffer or page cache memory that is free and available; this is memory that has not been recently used and can be reclaimed for other purposes by the paging algorithm (MB)
 - ◆ **fRM** - Free real memory; the amount of physical RAM left unused by the system (MB)
 - ◆ **fVM** - Free virtual memory, the total amount of swap memory free (MB)



- ◆ **tRM** - Total real memory; the total usable RAM (i.e. physical memory minus a few reserved bytes and the kernel binary code) (MB)
- ◆ **tVM** - Total virtual memory; the total amount of physical swap memory (MB)
- ◆ **fDS ('<path>')** - Free disk space on the partition on which the specified directory resides (MB). Use **nbstatus workstations --fields "* ,fDS"** to see the directories that Netbatch monitors.
- ◆ **users** - Number of active users
- ◆ **load** - Workstation load
- ◆ **Nload** - Normalized Workstation load
- ◆ **nice** - the amount of time that a Job has been niced
- ◆ **susp** - the amount of time the Job has been suspended (in seconds)
- ◆ **waittime** - the amount of time that the Job has been waiting to be dispatched for execution (seconds)
- ◆ **expected_runtime** - the Job's expected run time (in minutes), as specified in the **--expected-runtime** switch.

Note

waittime is only checked every 60 seconds.

*Be careful choosing the operator to use with **waittime**. Because Netbatch only checks it every 60 seconds, an expression that uses == is very unlikely ever to be true.*

*Some actions are irrelevant for **waittime**. The only applicable actions are **kill** and **mail**.*

- ◆ **TimeInState('<state>')** - the amount of time that the Job is currently in the specified state (milliseconds)

Valid states are: **send**, **resched**, **approving**, **susp**, **resub**, **del**, **wait remote**, **wait**, **run**, and **disc**.

Note

*If the specified state is **disc**, the only valid action is **force-kill**.*

**Note**

The **BuffersRM**, **CachedRM**, and **InactiveRM** attributes are available only for SUSE Linux Enterprise Server 9 and later.

Note

Attribute names are case-sensitive.

- ♦ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ♦ An arithmetic operator: +, -, *, /, %, unary + (e.g., +10), or unary - (e.g., -10)
 - ♦ A logical operator: &&, ||, or !
 - ♦ A comparison operator: <, <=, >, >=, !=, or ==
 - ♦ A non-strict operator: **is** or **isnt**
 - ♦ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
 - ♦ A conditional operator: ? or :
- ♦ **value** is a number, a string, or another boolean expression.
- ♦ **function** is one of:
 - ♦ **avgovertime('<time_interval>', '{RM|VM|RM+VM}')**
which returns the average of the sampled values of the specified parameter in **time_interval**.
 - ♦ **deltaovertime('<time_interval>', '{utime|stime|utime+stime}')**
which returns the difference between the maximum and minimum values of the specified parameter during **time_interval**.
 - ♦ **reservation('<capability>')**
which returns the value of the Job's reservation of the specified Workstation capability. (For example, if a Job reserved two of the capability called **cores**, **reservation("cores")** returns 2 for that Job.)

In both functions, **time_interval** is of the form **<number>[{s|m|h}]** (seconds, minutes, or hours). The default is minutes (so 60 means 60 minutes).



These are just the functions specific to Job constraints. For a full list of available functions, see [Appendix H: Functions](#), or run the following command:

```
nbstatus functions --context jobConstraints
```

Note

When one of these functions is used, the value of the expression that contains the function and its maximum value are displayed in the RUsage field in the output of [nbstatus jobs](#).

For example, if you submit a Job with the following Job constraint:

```
"AvgOverTime('60s','RM')>1:kill"
```

then the nbstatus jobs RUsage field will include something like this:

```
avgovertime(60s;RM)= 0.0 , Max(avgovertime(60s;RM))= (0;64)
```

*For **Max()**, the two numbers are the maximum (of the average or delta), and the time at which the maximum occurred (in seconds from the start of execution).*

Expressions can be grouped together using parentheses.

action is one of the following:

- ◆ **kill**
- ◆ **force-kill**

This kills the Job, even if its state is **disc**. (See **TimeInState()**, above.)

- ◆ **mail**

Mail is sent using the template set up by the Netbatch administrator (or a default template if no such template exists).

- ◆ **mailonce**

This is the same as **mail**, except mail is only sent once (unless the Job is resubmitted).

- ◆ **recall**
- ◆ **nice**

When the condition is no longer true, the Job is reniced.

- ◆ **resubmit**
- ◆ **resubmit(<exit_status>)**



The same as **resubmit**, except that the Job is assigned the specified custom exit status.

- ◆ **suspend**

When the condition is no longer true, the Job is resumed.

- ◆ **resume**

- ◆ **nop**

No operation (for reporting)

- ◆ **kill(<exit_status>)**

The same as **kill**, except that the Job is assigned the specified custom exit status.

- ◆ **modify-reservation(<expression>)**

Modifies the Job's reservation according to **expression**. For example:

```
modify-reservation(memory=memory+2)
```

This Increases the Job's **memory** reservation by 2.

- ◆ **modify-properties(<attribute>=<new_value>[,...])**

Modifies the Job's custom attributes.

- ◆ **RunCommand(cmd=<command>)**

The specified command or script is executed on the Workstation where the Job is running.

The command has access to the following environment variables:

- ◆ **__NB_CLASS** - the class requirement with which the Job was dispatched
- ◆ **__NB_CMD** - the Job's command
- ◆ **__NB_DATE** - the date on which the email was sent
- ◆ **__NB_ERROR_MESSAGE** - the error message
- ◆ **__NB_EXIT_STATUS_CODE** - the Job's exit code
- ◆ **__NB_EXIT_STATUS_DESCRIPTION** - the Job's exit status description
- ◆ **__NB_JOB_CONSTRAINTS** - the Job constraints that apply to the the Job (as specified using **nbjob run**'s **--job-constraints** switch or in a Job profile)
- ◆ **__NB_JOB_ID** - the Job's ID



- ◆ **__NB_KILL_IF_SUSPENDED** - the number of minutes specified by the `nbjob run --kill` switch
- ◆ **__NB_MAIL_IF_SUSPENDED** - the number of minutes specified by the `nbjob run --mail __NB_min` switch
- ◆ **__NB_PGID** - the Job's process group ID
- ◆ **__NB_PID** - the Job's process ID
- ◆ **__NB_POOL_ADMIN** - the ID of the Pool administrator (as specified by the `A` directive—see [Appendix E.2.11](#))
- ◆ **__NB_POOL_NAME** - the Pool name
- ◆ **__NB_QSLOT** - the Qslot from which the Job was dispatched
- ◆ **__NB_RESUBMIT_ATTEMPTS** - the number of resubmission attempts specified by the `nbjob run --trials` switch
- ◆ **__NB_RESUBMIT_IF_SUSPENDED** - the number of minutes specified by the `nbjob run --resubmit` switch
- ◆ **__NB_RUSAGE** - the Job's resource usage
- ◆ **__NB_STIME** - Job process system time
- ◆ **__NB_TIME** - the time at which the email was sent
- ◆ **__NB_USER_LOG_FILE** - the path/filename of the user log file
- ◆ **__NB_USER_NAME** - the Job owner's user
- ◆ **__NB_UTIME** - Job process user time
- ◆ **__NB_WORKING_DIR** - the user's working directory, from which the Job was submitted
- ◆ **__NB_WORKSTATION** - the hostname of the Workstation on which the Job is running
- ◆ **__NB_WTIME** - Job process wall clock time

Multiple actions can be specified for a single expression.

Multiple `<expression>:<action>[:...]` constraints can be specified.

**Note**

If you use **--job-constraints** with the **--no-restart** switch, the Job constraints are modified without resubmitting the Job.

If you omit the **--no-restart** switch, the Job is resubmitted with the new Job constraints.

--job-limits " <key>=<value>[, . . .] "

Modifies the limit that Netbatch applies to the amount of the specified resource(s) that the Job uses (only applicable for Workstations running SLES 11 and above and for which cgroups resource usage calculation is enabled). Available **keys** are:

- ◆ **VM**—value in megabytes (default: no limit)
- ◆ **RM**—value in megabytes (default: no limit)
- ◆ **cpushare**—a number that specifies the share of CPU relative to other Jobs (default value: 1024)

--license "<expression>"

Change the Job's license requirements to **expression**. **expression** specifies the license(s) that the Job will need during its execution, and (optionally) the amount of time each one is required. This overrides any license requirements defined in the user's Job profile file or **NBLICENSES** environment variable.

expression consists of any number of **<req_expr>**s and **<operator>**s.

req_expr is a license requirement expression of the following format:

```
"<license>=<value>[:duration=
<duration_time>]"
```

where:

- ◆ **license** is the required license.
- ◆ **value** is the number of instances of **license** that are required.
- ◆ **duration_time** is the amount of time (in minutes) for which the license is required (starting from when the Job is dispatched for execution).

If not specified, then **duration** is set to the Job execution time (that is, the license is reserved for the entire duration of the Job's execution).

operator is **&&** or **||**.



These can be grouped together using parentheses.

```
--license-keyfile "<keyfile path>|<port@host>[ :... ]"
```

Specifies the license keyfiles to be used. (This is instead of setting the **NB_LICENSE_FILE** environment variable.) If multiple arguments are specified, use colons as delimiters.

Note

If you specify a license that the Job requires by its alias, you do not need to specify the license keyfile.

```
--no-restart
```

Changes are applied without resubmitting the Job.

```
--parallel <parallel_reservation_string>
```

Modifies the Job's Parallel Job requirements. The original parallel reservation string is replaced with the one specified.

If the Job is running, it is **resubmitted** with the new Parallel Job requirements and all reserved licenses are released.

If the Job is waiting, all reserved execution slots are released.

The parallel reservation string take the following form:

```
"slots=[<min_slots>[-<max_slots>]][,slots_per_host=<min_sph>[-<max_sph>]][,reservation_per_host={true|false}][,preserve_slot={true|false}][,exit_on_master_finish={true|false}][,slave_validator=<validate_script>]"
```

where:

- ◆ **min_slots** is the minimum number of execution slots required
- ◆ **max_slots** (optional) is the maximum number of execution slots required
- ◆ **min_sph** is the minimum number of execution slots required on a single host
- ◆ **max_sph** (optional) is the maximum number of execution slots required on a single host
- ◆ **reservation_per_host** indicates that the requested resource reservation will be made once for each scheduled executing host, regardless of the number of parallel slots scheduled for it.



The default value is `false`.

For example, if two Slave Jobs are scheduled to execute on a Host, each of which has a resource reservation of 10GB of memory:

- ◆ If `reservation_per_host` is `false`, 10GB will be reserved for each Job (20GB in total).
- ◆ If `reservation_per_host` is `true`, a total of 10GB will be reserved.
- ◆ `preserve_slot` specifies whether an execution slot will be released (`false`) or not (`true`) when the Slave Job running in it ends. If `preserve_slot=true`, the execution slot can be reused.

This behavior can also be specified at the Slave Job level (but only if `preserve_slot=false` for the Master Job). To do this, add `--preserve-slot` to the `nbexec` command for the relevant Slave Job(s).

- ◆ `exit_on_master_finish` specifies what happens to Slave Jobs that are still running when the Master Job ends. If your Slave Jobs should no longer be running when the Master Job ends, set this to `true`. Otherwise, set it to `false` (the default).
- ◆ `validate_script` is the name of a script that is run to determine whether a particular Workstation can or cannot run one of the Job's Slave Jobs. The user is responsible for creating this script.

If the script determines that the Workstation is suitable, it should return zero. If not, it should return a non-zero value.

If the script returns a non-zero value, Netbatch excludes the Workstation and looks for a replacement.

```
--parallel-job-constraints
"<expression>:<action>[:....][,...]"
```

Specifies that the Netbatch should take one or more actions (including stopping, restarting, or resubmitting the Job), depending on various Job-related factors (which it measures once a second).

`expression` is either an arbitrarily complex boolean expression or an expression of the form:

```
idle(<interval_in_sec>,<cpu_in_percent>)
```

This evaluates to true if CPU usage was below the specified percentage during the specified interval.

If `expression` is an arbitrarily complex boolean expression, it consists of `<attribute>`s, `<operator>`s, and `<value>`s, where:



- ◆ **attribute** is one of the following:
 - ◆ **RM** - the total real memory use by all the Jobs that make up the Parallel Job (MB)
 - ◆ **VM** - the total virtual memory use by all the Jobs that make up the Parallel Job (MB)
 - ◆ **utime** - the user time that has been accumulated by all the Jobs that make up the Parallel Job (seconds)
 - ◆ **stime** - the system time that has been accumulated by all the Jobs that make up the Parallel Job (seconds)
 - ◆ **wtime** - the wall clock time that has been accumulated by all the Jobs that make up the Parallel Job (seconds)

Note

These conditions are checked every 5 minutes.

Note

Attribute names are case-sensitive.

- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., +10), or unary - (e.g., -10)
 - ◆ A logical operator: &&, ||, or !
 - ◆ A comparison operator: <, <=, >, >=, !=, or ==
 - ◆ A non-strict operator: **is** or **isnt**
 - ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
 - ◆ A conditional operator: ? or :
- ◆ **value** is a number, a string, or another boolean expression.

Expressions can be grouped together using parentheses.

action is one of the following:

- ◆ **kill**
- ◆ **mail**
- ◆ **recall**
- ◆ **nice**



When the condition is no longer true, the Job is reniced.

- ◆ **resubmit**
- ◆ **suspend**

When the condition is no longer true, the Job is resumed.

Multiple actions can be specified for a single expression.

Multiple **<expression>:<action>[: . . .]** constraints can be specified.

--parallel-slave-class <class_expression>

Modifies the class of machine that the Parallel Job's Slave Jobs require. (The syntax is the same as for **--class**, above.)

--priority <priority>

Changes the Job's priority immediately (even if it is already running).

--properties <prop_name>=<value>[, . . .]

Specifies properties for the Job. **prop_name** is the name of a property. **value** is the property's value.

You can only add properties or modify existing ones. You cannot remove them.

--qslot <qslot_path>

Specifies the Qslot to which the Job is to be moved.

--queue <queue_name>

Specifies the Queue to which the Job is to be moved.

--remote-pools { "[-]<phys_pool>" | "[-]<alias>" }[, . . .]

Specifies the preferred Physical Pool(s) by name or by alias (only relevant when submitting a Job to a Virtual Pool). You can specify Pools by:

- ◆ Inclusion -- only the listed physical Pools will be considered when deciding which Pool to send the Job to.
- ◆ Exclusion -- precede Pool/alias names with -. All relevant physical Pools will be considered except those that are listed.
- ◆ A combination -- if there are both included and excluded Pools/aliases in the list, the excluded ones are excluded from the list of included Pools.

You can use this to specify that a specific physical Pool should be excluded from the set of Pools represented by an alias.



This situation can also arise when the request uses inclusion and a Job Profile uses exclusion, or vice-versa.

The order determines the order in which the Pools will be tried. Netbatch will try the first Pool, and will only try the next one if the previous one has reached its `WaitJobsLimit` or `MaxJobsLimit`.

If an alias is specified, Netbatch uses its existing algorithm to decide which Pool to send the Job to. (Multiple Pools can have the same alias. For example, all the Physical Pools at the same site as the Virtual Pool Master might have the alias `local`.)

This switch overrides the **NB_POOLS** environment variable.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--triggers "<boolean_expression>"

Changes the Job's dependencies so that it depends on other Jobs, as defined in `boolean_expression`.

`boolean_expression` consists of any number of terms of the form `[!]<JobID>[[<exit_status_expression>]]`, separated by the logical operators `&&` (AND) and `||` (OR).

Note

`JobID` can be either a Job ID or a task name. You are advised to explicitly specify whether it is a Job ID or a task name, because in ambiguous cases, Netbatch can assume that a task name is actually a Job ID, or vice versa.

Use one of the following instead of a Job ID/task name:

`jobid:<Job_ID>`



task:<task_name>

If a task name is used, the **exit_status_expression** does not evaluate to true until it is true for each of the Jobs that make up the task.

See [Section 6.5.18, Specifying a Task Name](#).

exit_status_expression can be one of the following:

- ♦ **exit**
- ♦ **exit <relational_operator> <exit_code>**

relational_operator must be <, <=, ==, >=, or >.

To specify multiple acceptable exit code conditions for a specific Job, use multiple **<JobID>[<exit_status_expression>]** terms separated by || (OR) (for example, **1 exit <30 || 1 exit>50**).

If **exit** is used on its own, it means that the Job ended (no matter what its exit status).

If no **exit_status_expression** is specified, **done** is assumed.

Note

Logical expressions are evaluated according to the following order of precedence - () (parentheses), ! (NOT), && (AND), || (OR).

So

"1 [exit<30] || !2 && (3 || 4)"

is the same as

"1 [exit<30] || ((!2) &&(3 || 4))"

specification

This is a boolean expression that specifies which Jobs are to be modified.

specification is either **all** or an arbitrarily complex boolean expression.

This expression consists of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ♦ **attribute** is a valid Job field (see **nbstatus jobs --fields**, [Appendix A.3.10](#)).



- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")
 - ◆ A logical operator: &&, ||, or !
 - ◆ A comparison operator: <, <=, >, >=, !=, or ==
 - ◆ A non-strict operator: is or isnt
 - ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
 - ◆ A conditional operator: ? or :
- ◆ <value> is a number, a string, or another boolean expression.

Note

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If value is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

Expressions can be grouped together using parentheses.

If specification is all, all the user's Jobs are modified.

The startsWith() and glob() functions can also be used:

`"startsWith(<field_name>,'<string>')"`

startsWith() matches fields whose value starts with the specified string.

`"glob(<field_name>,'<expression>')"`

glob() matches fields that match expression.
expression can contain:

- ◆ Any character
- ◆ . - matches any character
- ◆ \w - matches any letter (upper- or lower-case)
- ◆ \d - matches any digit
- ◆ \s - matches any space character
- ◆ * - zero or more of the previous character



- ◆ ? - exactly one of the previous character

A.2.4 nbjob remove

Name

nbjob remove - Remove one or more Jobs that have already been submitted.

Description

Cancel one or more Netbatch Jobs.

When a user runs **nbjob remove**, it only cancels Jobs belonging to that user.

A user may have privileges to cancel Jobs of other users in a Qslot. This is defined per Queue or per Qslot.

The system administrator can use **nbjob remove** to cancel any user's Jobs.

nbjob remove can be used to cancel Parallel Jobs. Simply use **nbjob remove** with the JobID of the Master Job.

Usage

```
nbjob remove [--target <pool_name>|<host_name>]
[--reason "<reason>"] <options> <specification>
```

Switches

--target <pool_name>|<host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbjob** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)



```
--action-signal
"<action>[:signal=<type>][,timeout=<time>]
[:cmd=<cmd_name>][;...]"
```

Specifies that for the specified action, the Job will be "soft killed", that is, the specified signal will be sent to the specified process (instead of to all the Job's processes, as in the default sequence).

Here:

- ◆ **action** is the action to which the soft kill will be applied. It is one of:
 - ◆ **remove**
 - ◆ **suspend**
 - ◆ **recall**
- ◆ **type** is the signal that will be sent. Any Unix signal may be sent, except SIGKILL and SIGSTOP. (Default: SIGTERM for remove and recall, SIGTSTP for suspend.)
- ◆ **time** is the amount of time (in seconds) to wait between send the first "soft" signal and continuing with the default sequence (for remove and recall, SIGKILL is sent, for suspend SIGSTOP). (Default: five seconds)

This overrides the timeout specified by
NBSIGKILL_TIMEOUT or **NB_SUSPEND_TIMEOUT**
(depending on the action).

- ◆ **cmd_name** is the command name of the process to which the signal will be sent. (Default: the first process under the Job leader)

If multiple processes have the same command name, the signal will be sent to all of them.

--force

Remove the specified Jobs, even if they are disconnected (status **Disc-D** or **Disc-R**). This removes the Job from the Pool Master. Only use this switch if you know that the Workstation that the Job was sent to is actually down.

Only root or a user with administrative permissions for the Qslot that the Job was submitted to can use this switch.

--reason "<reason>"

Specifies why the Job is being removed. This information is recorded in NB Tracker.

**--retry-timeout <time>{h|m}**

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

specification

This is a boolean expression that specifies which Jobs are to be removed.

specification is either **all** or an arbitrarily complex boolean expression.

This expression consists of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid Job field (see **nbstatus jobs --fields**, [Appendix A.3.10](#)).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")
 - ◆ A logical operator: &&, ||, or !
 - ◆ A comparison operator: <, <=, >, >=, !=, or ==
 - ◆ A non-strict operator: is or isnt
 - ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
 - ◆ A conditional operator: ? or :
- ◆ **<value>** is a number, a string, or another boolean expression.

Note

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.



If **value** is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

Expressions can be grouped together using parentheses.

For example, "(**jobid**>=10)&&(**jobid**<=20)" specifies that all Jobs with a Job ID between 10 and 20 (inclusive) are to be removed.

If **specification** is **all**, all the user's Jobs are removed.

The **startswith()** and **glob()** functions can also be used:

"startswith(<field_name>,'<string>')"

startswith() matches fields whose value starts with the specified string.

"glob(<field_name>,'<expression>')"

glob() matches fields that match **expression**.

expression can contain:

- ◆ Any character
- ◆ . - matches any character
- ◆ \w - matches any letter (upper- or lower-case)
- ◆ \d - matches any digit
- ◆ \s - matches any space character
- ◆ * - zero or more of the previous character
- ◆ ? - exactly one of the previous character



A.2.5 nbjob suspend

Name

nbjob suspend - Suspend one or more Jobs that have already been submitted.

Description

Suspend one or more Jobs. A suspended Job can be resumed by using **nbjob resume**.

When a user runs **nbjob suspend**, it only suspends Jobs belonging to that user.

A user may have privileges to suspend Jobs of other users in a Qslot. This is defined per Queue or per Qslot.

The system administrator can use **nbjob suspend** to suspend any user's Jobs.

Usage

```
nbjob suspend [--target <pool_name>|<host_name>]
[--reason "<reason>"] [<options>] [<specification>]
```

Switches

--target <pool_name>|<host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbjob** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

--action-signal

```
"<action>[:signal=<type>][,timeout=<time>]
[:cmd=<cmd_name>][;...]"
```

Specifies that for the specified action, the Job will be "soft killed", that is, the specified signal will be sent to the specified process (instead of to all the Job's processes, as in the default sequence). Here:

- ♦ **action** is the action to which the soft kill will be applied. It is one of:
 - ♦ **remove**



- ◆ **suspend**
- ◆ **recall**
- ◆ **type** is the signal that will be sent. Any Unix signal may be sent, except SIGKILL and SIGSTOP. (Default: SIGTERM for remove and recall, SIGTSTP for suspend.)
- ◆ **time** is the amount of time (in seconds) to wait between send the first "soft" signal and continuing with the default sequence (for remove and recall, SIGKILL is sent, for suspend SIGSTOP). (Default: five seconds)

This overrides the timeout specified by
NBSIGKILL_TIMEOUT or **NB_SUSPEND_TIMEOUT**
(depending on the action).

- ◆ **cmd_name** is the command name of the process to which the signal will be sent. (Default: the first process under the Job leader)

If multiple processes have the same command name, the signal will be sent to all of them.

--force

Remove the specified Jobs, even if they are disconnected (status **Disc-D** or **Disc-R**).

--reason "<reason>"

Specifies why the Job is being suspended. This information is recorded in NB Tracker. (It is not displayed in the output of **nbstatus jobs** and has nothing to do with the **SuspendReason** field.)

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.



specification

This is a boolean expression that specifies which Jobs are to be suspended.

specification is either **all** or an arbitrarily complex boolean expression.

This expression consists of <attribute>s, <operator>s, and <value>s, where:

- ◆ **attribute** is a valid Job field (see `nbstatus jobs --fields`, [Appendix A.3.10](#)).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")
 - ◆ A logical operator: &&, ||, or !
 - ◆ A comparison operator: <, <=, >, >=, !=, or ==
 - ◆ A non-strict operator: **is** or **isnt**
 - ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
 - ◆ A conditional operator: ? or :
- ◆ <value> is a number, a string, or another boolean expression.

Note

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If **value** is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

Expressions can be grouped together using parentheses.

For example, "(jobid>=10)&&(jobid<=20)" specifies that all Jobs with a Job ID between 10 and 20 (inclusive) are to be suspended.

If **specification** is **all**, all the user's Jobs are suspended.

The `startswith()` and `glob()` functions can also be used:

```
"startswith(<field_name>,'<string>')"
```



startswith() matches fields whose value starts with the specified string.

"glob(<field_name>, '<expression>')"

glob() matches fields that match **expression**.
expression can contain:

- ◆ Any character
- ◆ . - matches any character
- ◆ \w - matches any letter (upper- or lower-case)
- ◆ \d - matches any digit
- ◆ \s - matches any space character
- ◆ * - zero or more of the previous character
- ◆ ? - exactly one of the previous character

A.2.6 nbjob resume

Name

nbjob resume - Resume one or more suspended Jobs.

Description

Resume one or more suspended Jobs (which were suspended with **nbjob suspend** or **nbqrm --suspend**).

When a user runs **nbjob resume**, it only resumes Jobs belonging to that user.

A user may have privileges to resume Jobs of other users in a Qslot. This is defined per Queue or per Qslot.

The system administrator can use **nbjob resume** to resume any user's Jobs.

Usage

```
nbjob resume [--target <pool_name>|<host_name>]
[--reason "<reason>" ] [<options>] [<specification>]
```



Switches

--target <pool_name> | <host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbjob** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

--reason "<reason>"

Specifies why the Job is being resumed. This information is recorded in NB Tracker.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

specification

This is a boolean expression that specifies which Jobs are to be resumed.

specification is either **all** or an arbitrarily complex boolean expression.

This expression consists of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid Job field (see **nbstatus jobs --fields**, [Appendix A.3.10](#)).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")
 - ◆ A logical operator: &&, ||, or !
 - ◆ A comparison operator: <, <=, >, >=, !=, or ==



- ◆ A non-strict operator: `is` or `isnt`
- ◆ A bitwise operator: `&`, `|`, `~`, `^`, `<<`, `>>`, or `>>>`
- ◆ A conditional operator: `?` or `:`
- ◆ `<value>` is a number, a string, or another boolean expression.

Note

A double equals sign (`==`) is used. `!=` (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If `value` is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

Expressions can be grouped together using parentheses.

For example, "`(jobid>=10)&&(jobid<=20)`" specifies that all suspended Jobs with a Job ID between 10 and 20 (inclusive) are to be resumed.

If `specification` is `all`, all the user's suspended Jobs are resumed.

The `startswith()` and `glob()` functions can also be used:

`"startswith(<field_name>,'<string>')"`

`startswith()` matches fields whose value starts with the specified string.

`"glob(<field_name>,'<expression>')"`

`glob()` matches fields that match `expression`.

`expression` can contain:

- ◆ Any character
- ◆ `.` - matches any character
- ◆ `\w` - matches any letter (upper- or lower-case)
- ◆ `\d` - matches any digit
- ◆ `\s` - matches any space character
- ◆ `*` - zero or more of the previous character
- ◆ `?` - exactly one of the previous character



A.2.7 nbjob recall

Name

nbjob recall - Resubmit one or more Jobs that have already been submitted.

Description

Resubmit one or more Jobs (only meaningful for running Jobs). If a Job is running, it is killed and resubmitted to the wait state Queue.

When a user runs **nbjob recall**, it only resubmits Jobs belonging to that user.

A user may have privileges to resubmit Jobs of other users in a Qslot. This is defined per Queue or per Qslot.

The system administrator can use **nbjob recall** to resubmit any user's Jobs.

Usage

```
nbjob recall [--target <pool_name>|<host_name>]
[--reason "<reason>"] [<options>] [<specification>]
```

Switches

--target <pool_name>|<host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If --target is omitted, **nbjob** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

--action-signal "<action>[:signal=<type>][,timeout=<time>] [:cmd=<cmd_name>][;...]"

Specifies that for the specified action, the Job will be "soft killed", that is, the specified signal will be sent to the specified process (instead of to all the Job's processes, as in the default sequence). Here:

- ♦ **action** is the action to which the soft kill will be applied. It is one of:
 - ♦ **remove**



- ◆ **suspend**
- ◆ **recall**
- ◆ **type** is the signal that will be sent. Any Unix signal may be sent, except SIGKILL and SIGSTOP. (Default: SIGTERM for remove and recall, SIGTSTP for suspend.)
- ◆ **time** is the amount of time (in seconds) to wait between send the first "soft" signal and continuing with the default sequence (for remove and recall, SIGKILL is sent, for suspend SIGSTOP). (Default: five seconds)

This overrides the timeout specified by
NBSIGKILL_TIMEOUT or **NB_SUSPEND_TIMEOUT**
(depending on the action).

- ◆ **cmd_name** is the command name of the process to which the signal will be sent. (Default: the first process under the Job leader)

If multiple processes have the same command name, the signal will be sent to all of them.

--reason <reason>

Specifies why the Job is being recalled. This information is recorded in NB Tracker.

--restore-from-checkpoint <checkpoint_name>

Resubmits the Job(s) from the specified checkpoint.

You can only restore a Job from a checkpoint if it was submitted with the **--checkpoint-tool** switch.

--restore-from-last-checkpoint

Resubmits the Job(s) from the last checkpoint.

You can only restore a Job from a checkpoint if it was submitted with the **--checkpoint-tool** switch.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.



If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

specification

This is a boolean expression that specifies which Jobs are to be resubmitted.

specification is either **all** or an arbitrarily complex boolean expression.

This expression consists of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid Job field (see **nbstatus jobs --fields**, [Appendix A.3.10](#)).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")
 - ◆ A logical operator: &&, ||, or !
 - ◆ A comparison operator: <, <=, >, >=, !=, or ==
 - ◆ A non-strict operator: **is** or **isnt**
 - ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
 - ◆ A conditional operator: ? or :
- ◆ **<value>** is a number, a string, or another boolean expression.

Note

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If **value** is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

Expressions can be grouped together using parentheses.

For example, "(jobid>=10)&&(jobid<=20)" specifies that all Jobs with a Job ID between 10 and 20 (inclusive) are to be resubmitted.

If **specification** is **all**, all the user's Jobs are resubmitted.



The **startswith()** and **glob()** functions can also be used:

```
"startswith(<field_name>,'<string>')"
```

startswith() matches fields whose value starts with the specified string.

```
"glob(<field_name>,'<expression>')"
```

glob() matches fields that match **expression**.

expression can contain:

- ◆ Any character
- ◆ . - matches any character
- ◆ \w - matches any letter (upper- or lower-case)
- ◆ \d - matches any digit
- ◆ \s - matches any space character
- ◆ * - zero or more of the previous character
- ◆ ? - exactly one of the previous character



A.2.8 nbjob prun

Name

nbjob prun - Execute a Parallel Job's Slave Jobs.

Description

nbjob prun executes a Parallel Job's Slave Jobs. (A Parallel Job is one that requires more than one Execution Slot (that is, a processor or a host), or that consists of several Jobs, each of which requires an execution slot.)

When a Parallel Job is dispatched for execution, Netbatch adds all the hosts that the Slave Jobs are to run on to the

NB_PARALLEL_JOB_HOSTS environment variable (in a space-delimited list).

The Parallel Job's Master Job is responsible for reading the list of hosts in **NB_PARALLEL_JOB_HOSTS** and using **nbjob prun** to dispatch the Slave Jobs to these hosts.

nbjob prun sends each Slave Job to the Pool Master specified in the **_NB_PARALLEL_POOL_HOST** environment variable for validation before it is executed on the specified host.

Caution

*Do not set **_NB_PARALLEL_POOL_HOST** yourself.
Netbatch sets it as part of the scheduling process to
ensure system consistency.*

Note

*The **nbjob prun** (or **nbexec**) command can fail in the following circumstances:*

- *The specified Workstation (host) is down.*
- *The (master) Job is being preempted.*
- *There are no more reservations (**nbjob prun** or **nbexec** has been called too many times).*



- *There is no Master Job attached to the Job. (That is, the Job was not sent by a valid Master Job.)*
- *No Workstation was specified.*

Usage

```
nbjob prun --host <host_name> [options]
<slave job> <slave job arguments>
```

Host

--host <host_name>

Specifies the host that the Slave Job will be sent to.

Options

--daemon

Tells Netbatch that this Slave Job will be a daemon (that is, a process that closes stdin, stdout, and stderr, and continues to run), and that it should use Process Authentication Groups (PAGs) to monitor and control it.

--exec-limits <soft_limit>:<hard_limit>

Note

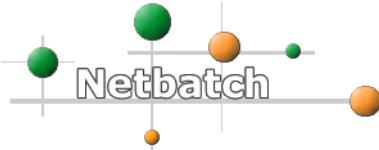
--exec-limits is deprecated. Use **--job-constraints** instead.
(For example, if you add **--job-constraints "wtime>3600:mail:kill"**, Netbatch will kill the Job and mail you if it runs for more than an hour.)

Specifies execution time limits for the Job:

- **soft_limit** - If exceeded, Netbatch sends email to the user.
- **hard_limit** - If exceeded, Netbatch kills the process, cancels the Job, and sends email to the user.

Note

Execution limits are rounded up to the nearest minute.



```
--hung-limits <soft limit>:<hard limit>
```

Note

--hung-limits is deprecated. Use **--job-constraints** instead.
(For example, if you add **--job-constraints "deltaovertime('15m','utime+stime')<60:mail:kill"**
Netbatch will kill the Job and mail you if it accumulates less than 60
seconds of execution time (utime plus stime) in any 15 minute period.)

Specifies hung limits for the Job.

- ◆ **soft_limit** - If the Job does not accumulate any CPU time within the specified time, Netbatch sends an email to the user.
- ◆ **hard_limit** - If the Job does not accumulate any CPU time within the specified time, Netbatch kills the process, cancels the Job, and sends email to the user.

Note

Execution limits are rounded up to the nearest minute.

--inherit

The Slave Job inherits the attributes of the Master Job. (By default, it does not.) The Slave Job does **not** inherit the following attributes:

- ◆ Job starter
- ◆ Command line
- ◆ Environment
- ◆ Current working directory
- ◆ Execution limits (-**y**) (if specified for Slave Job)
- ◆ Hung limits (-**z**) (if specified for Slave Job)
- ◆ **-K, -R, -#, -S, -N** switches (if specified)
- ◆ **rlimits** (if specified)
- ◆ Job name (if specified)
- ◆ Log file
- ◆ PAG option
- ◆ Interactive settings (logon, blocking, etc.)



```
--job-constraints
"<expression>:<action>[:....][,...]"
```

Specifies that the Netbatch should take one or more actions (including stopping, restarting, or resubmitting the Job), depending on various Job-related factors (which it measures once a second).

expression is either an arbitrarily complex boolean expression or an expression of the form:

```
idle(<interval_in_sec>,<cpu_in_percent>)
```

This evaluates to true if CPU usage was below the specified percentage during the specified interval.

If **expression** is an arbitrarily complex boolean expression, it consists of **<attribute>**s, **<operator>**s, and **<value>**s, and optionally **<functions>**s, where:

- ◆ **attribute** is one of the following:
 - ◆ **RM** - Jobs real used memory (MB)
 - ◆ **VM** - Jobs virtual memory (MB)
 - ◆ **utime** - Job process user time (seconds)
 - ◆ **nutime** - normalized Job process user time (seconds)
 - ◆ **stime** - Job process system time (seconds)
 - ◆ **nstime** - normalized Job process system time (seconds)
 - ◆ **wtime** - Job process wall clock time (seconds)
 - ◆ **BuffersRM** - the amount of physical RAM used for file buffers (MB)
 - ◆ **CachedRM** - the amount of physical RAM used as cache memory; memory in the pagecache (diskcache) minus SwapCache (the amount of Swap used as cache memory) (MB)
 - ◆ **InactiveRM** - the total amount of buffer or page cache memory that is free and available; this is memory that has not been recently used and can be reclaimed for other purposes by the paging algorithm (MB)
 - ◆ **fRM** - Free real memory; the amount of physical RAM left unused by the system (MB)
 - ◆ **fVM** - Free virtual memory, the total amount of swap memory free (MB)
 - ◆ **tRM** - Total real memory; the total usable RAM (i.e. physical memory minus a few reserved bytes and the kernel binary code) (MB)



- ◆ **tvm** - Total virtual memory; the total amount of physical swap memory (MB)
- ◆ **fDS('<path>')** - Free disk space on the partition on which the specified directory resides (MB). Use **nbstatus workstations --fields "* ,fds"** to see the directories that Netbatch monitors.
- ◆ **users** - Number of active users
- ◆ **load** - Workstation load
- ◆ **Nload** - Normalized Workstation load
- ◆ **nice** - the amount of time that a Job has been niced
- ◆ **susp** - the amount of time the Job has been suspended (in seconds)
- ◆ **waittime** - the amount of time that the Job has been waiting to be dispatched for execution (seconds)
- ◆ **expected_runtime** - the Job's expected run time (in minutes), as specified in the **--expected-runtime** switch.

Note

waittime is only checked every 60 seconds.

Be careful choosing the operator to use with **waittime**. Because Netbatch only checks it every 60 seconds, an expression that uses == is very unlikely ever to be true.

Some actions are irrelevant for **waittime**. The only applicable actions are **kill** and **mail**.

- ◆ **TimeInState('<state>')** - the amount of time that the Job is currently in the specified state (milliseconds)
- Valid states are: **send**, **resched**, **approving**, **susp**, **resub**, **del**, **wait remote**, **wait**, **run**, and **disc**.

Note

If the specified state is **disc**, the only valid action is **force-kill**.

**Note**

The **BuffersRM**, **CachedRM**, and **InactiveRM** attributes are available only for SUSE Linux Enterprise Server 9 and later.

Note

Attribute names are case-sensitive.

- ♦ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ♦ An arithmetic operator: +, -, *, /, %, unary + (e.g., +10), or unary - (e.g., -10)
 - ♦ A logical operator: &&, ||, or !
 - ♦ A comparison operator: <, <=, >, >=, !=, or ==
 - ♦ A non-strict operator: **is** or **isnt**
 - ♦ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
 - ♦ A conditional operator: ? or :
- ♦ **value** is a number, a string, or another boolean expression.
- ♦ **function** is one of:
 - ♦ **avgovertime('<time_interval>', '{RM|VM|RM+VM}')**
which returns the average of the sampled values of the specified parameter in **time_interval**.
 - ♦ **deltaovertime('<time_interval>', '{utime|stime|utime+stime}')**
which returns the difference between the maximum and minimum values of the specified parameter during **time_interval**.
 - ♦ **reservation('<capability>')**
which returns the value of the Job's reservation of the specified Workstation capability. (For example, if a Job reserved two of the capability called **cores**, **reservation("cores")** returns 2 for that Job.)

In both functions, **time_interval** is of the form **<number>[{s|m|h}]** (seconds, minutes, or hours). The default is minutes (so 60 means 60 minutes).



These are just the functions specific to Job constraints. For a full list of available functions, see [Appendix H: Functions](#), or run the following command:

```
nbstatus functions --context jobConstraints
```

Note

When one of these functions is used, the value of the expression that contains the function and its maximum value are displayed in the RUsage field in the output of [nbstatus jobs](#).

For example, if you submit a Job with the following Job constraint:

```
"AvgOverTime('60s','RM')>1:kill"
```

then the nbstatus jobs RUsage field will include something like this:

```
avgovertime(60s;RM)= 0.0 , Max(avgovertime(60s;RM))= (0;64)
```

*For **Max()**, the two numbers are the maximum (of the average or delta), and the time at which the maximum occurred (in seconds from the start of execution).*

Expressions can be grouped together using parentheses.

action is one of the following:

- ◆ **kill**
- ◆ **force-kill**

This kills the Job, even if its state is **disc**. (See **TimeInState()**, above.)

- ◆ **mail**

Mail is sent using the template set up by the Netbatch administrator (or a default template if no such template exists).

- ◆ **mailonce**

This is the same as **mail**, except mail is only sent once (unless the Job is resubmitted).

- ◆ **recall**
- ◆ **nice**

When the condition is no longer true, the Job is reniced.

- ◆ **resubmit**
- ◆ **resubmit(<exit_status>)**



The same as **resubmit**, except that the Job is assigned the specified custom exit status.

- ◆ **suspend**

When the condition is no longer true, the Job is resumed.

- ◆ **resume**

- ◆ **nop**

No operation (for reporting)

- ◆ **kill(<exit_status>)**

The same as **kill**, except that the Job is assigned the specified custom exit status.

- ◆ **modify-reservation(<expression>)**

Modifies the Job's reservation according to **expression**. For example:

```
modify-reservation(memory=memory+2)
```

This Increases the Job's **memory** reservation by 2.

- ◆ **modify-properties(<attribute>=<new_value>[,...])**

Modifies the Job's custom attributes.

- ◆ **RunCommand(cmd=<command>)**

The specified command or script is executed on the Workstation where the Job is running.

The command has access to the following environment variables:

- ◆ **__NB_CLASS** - the class requirement with which the Job was dispatched
- ◆ **__NB_CMD** - the Job's command
- ◆ **__NB_DATE** - the date on which the email was sent
- ◆ **__NB_ERROR_MESSAGE** - the error message
- ◆ **__NB_EXIT_STATUS_CODE** - the Job's exit code
- ◆ **__NB_EXIT_STATUS_DESCRIPTION** - the Job's exit status description
- ◆ **__NB_JOB_CONSTRAINTS** - the Job constraints that apply to the the Job (as specified using **nbjob run**'s **--job-constraints** switch or in a Job profile)
- ◆ **__NB_JOB_ID** - the Job's ID



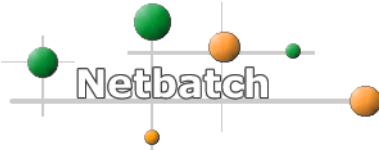
- ◆ **NB_KILL_IF_SUSPENDED** - the number of minutes specified by the **nbjob run --kill** switch
- ◆ **NB_MAIL_IF_SUSPENDED** - the number of minutes specified by the **nbjob run --mail __NB_min** switch
- ◆ **NB_PGRID** - the Job's process group ID
- ◆ **NB_PID** - the Job's process ID
- ◆ **NB_POOL_ADMIN** - the ID of the Pool administrator (as specified by the **A** directive—see [Appendix E.2.11](#))
- ◆ **NB_POOL_NAME** - the Pool name
- ◆ **NB_QSLOT** - the Qslot from which the Job was dispatched
- ◆ **NB_RESUBMIT_ATTEMPTS** - the number of resubmission attempts specified by the **nbjob run --trials** switch
- ◆ **NB_RESUBMIT_IF_SUSPENDED** - the number of minutes specified by the **nbjob run --resubmit** switch
- ◆ **NB_RUSAGE** - the Job's resource usage
- ◆ **NB_STIME** - Job process system time
- ◆ **NB_TIME** - the time at which the email was sent
- ◆ **NB_USER_LOG_FILE** - the path/filename of the user log file
- ◆ **NB_USER_NAME** - the Job owner's user
- ◆ **NB_UTIME** - Job process user time
- ◆ **NB_WORKING_DIR** - the user's working directory, from which the Job was submitted
- ◆ **NB_WORKSTATION** - the hostname of the Workstation on which the Job is running
- ◆ **NB_WTIME** - Job process wall clock time

Multiple actions can be specified for a single expression.

Multiple **<expression>:<action>[:...]** constraints can be specified.

--job-starter

Start the Job with a starter program.

**--kill <minutes>**

Specifies that Netbatch should kill the Job if it is suspended for more than the specified number of minutes.

--log-file <job_name>

Specifies that the Job's log file will be called **job_name**, and that the Job will have the Job name **job_name** (that is, the Job's name is displayed as **job_name** in the output of the **nbstatus jobs** and **nbqstat** commands).

job_name can be just a filename (e.g., **myJob**), or can include the path (e.g., **~/myJob**).

--log-file-dir <path>

Specifies that the Job's log file will be written in the specified directory.

If **--log-file** is used as well and its argument includes a path, **--log-file-dir** is ignored.

--mail [S] [E] [no] [<min>]

Send mail to the user when the Job starts (**s**), ends (**E**), is suspended for more than **min** consecutive minutes, or not at all (**no**). If multiple arguments are used, a space is used as the delimiter (e.g., **mail s E 60**)

If **--mail-list** is specified, mail is sent to all specified addresses. (See **--mail-list**.)

--mail-list <address> | "<address>[...]"

List of mail recipients for Netbatch messages about the Job. (Mail is sent to these addresses as well as to the user.) If more than one address is specified, a space is used as the delimiter, and the list of addresses must be enclosed in double quotes.

--mode interactive|i|blocking|b|terminal|t

Run in interactive (**i**), blocking (**b**), or terminal (**t**) mode:

- ◆ **interactive|i** - the Job appears as though it is executing on the submitting host machine (especially suited for batch execution of complex sequences of interdependent Jobs defined in a Makefile).

See [Section 6.15, Interactive Batch Jobs](#).

- ◆ **blocking|b** - no more commands may be executed until the Job ends.



- ◆ **terminal | t** - interactive applications that manipulate user input and screen display via a terminal device can be run as Netbatch Jobs (for example, emacs or tcsh).

Note

As of version 6.3.1, Netbatch automatically senses the context in which an interactive or terminal Job is being executed, and runs it as either an interactive or terminal Job, whichever is appropriate.

Thus, there is no need to use --mode terminal. Whenever you want the Job to run as if it is running locally, use --mode interactive.

See [Section 6.17, Terminal Jobs](#).

Note

You can abbreviate interactive, blocking, and terminal to i, b, and t respectively.

--no-env

Does not pass user environment variables with the Job.

User Job will be executed on remote machine without the user's environment variables.

--pag on|nosuspend

Netbatch uses Process Authentication Groups (PAGs) to track Jobs. This switch specifies whether a PAG should be added to the Job, and how:

- ◆ **on** - add a PAG to the Job. This may override a Job Profile, or be overridden by a Job Profile, depending on how the Job Profile is set up.
- ◆ **nosuspend** - add a PAG to the Job, but do not suspend child processes that change their process group ID.

If this switch is not specified, and no Job Profile specifies that Netbatch should apply a PAG, Netbatch decides whether to apply a PAG based on how the Workstation is configured.

--post-exec <exec>

Execute a post-execution program after the Job runs.

**Note**

In the Post-Execution phase, the following environment variables are available:

- **NB_JOB_EXIT_STATUS**—the Job's exit status
- **_NB_CLASS**—the Job's class requirement expression
- **_NB_CMD_LINE**—the Job's full command line
- **_NB_FINISH_TIME**—the time at which the Job finished executing
- **_NB_JOBID**—the Job's ID
- **_NB_LOG_FILE**—the location of the Job log file
- **_NB_POOL**—the name of the Pool that the Job was submitted to
- **_NB_QSLOT**—the Qslot that the Job was submitted to
- **_NB_QUEUE**—the Queue that the Job was submitted to
- **_NB_RUSAGE**—information about the Job's resource usage
- **_NB_START_TIME**—the time at which the Job started executing
- **_NB_SUBMIT_PWD**—the user's working directory when the Job was submitted
- **_NB_TIMES_RESTARTED**—the number of times that the Job was restarted

Note

The post-execution program is executed even if the Job is killed or suspended.

--pre-exec <exec>

Execute a pre-execution program before the Job runs.

Note

The following environment variables contain information that the pre-exec script can use:

- **_NB_POOL**
- **_NB_QUEUE**
- **_NB_QSLOT**



- **NB_LOG_FILE** (*but only if the --log-file option is used*)
- **NB_CLASS**

--preserve-slot

Specifies that the execution slot in which the Slave Job is running will not be released when it ends, thus allowing the slot to be reused.

If `preserve_slot=true` for the Master Job (that is, `preserve_slot=true` was specified for the Master Job in `nbq`'s parallel reservation requirements string), then all execution slots are reused.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

**--rlimits "<res_limit> [soft|hard] = value
[,...]"**

Set resource consumption limits:

- ◆ **res_limit** is the name of the limit to be set.
- ◆ **soft|hard** (optional) specifies whether a soft or hard limit is to be set. If not specified, both hard and soft limits are set to the same value.
- ◆ **value** is the value to be assigned to the limit.

The limits that can be set are listed in the table below. Some are platform-specific. If a limit is set for a platform that does not support it, it is ignored.

If a limit is not set, the limit is inherited from the WSM process that spawned the process. The WSM process, in turn, inherits the limits from the shell from which it was invoked.

**Note**

You can use the built-in **limit** command to view current resource limits. Note, however, that this shows the limits on the Workstation that you are logged into, not the Workstation on which your Job will run.

If you need to know the limits that will apply to your Job, run the **limit** command on a Workstation whose capabilities match those required by your Job.

Table 2: Resource Limits

Limit	Units	Description	Platform
addressspace	KB	The maximum size of a process's available memory	All
coredumpsize	KB	The maximum size of a core file that a process may create	All
cputime	Seconds	The maximum amount of CPU time that a process may use	All
datasize	KB	The maximum size of a process's heap	All
filesize	KB	The maximum size of a file that a process may create	All
memorylocked	KB	Maximum locked-in memory address space	Linux
memoryuse	KB	The maximal resident set size of a process	Linux, HP-UX
openfiles	# of files	Maximum number of files that a process may open	All
maxproc	# of processes	The maximum permitted number of processes	Linux
stacksize	KB	The maximum size of a process's stack (can also be set to unlimited)	All
vmemoryuse	KB	The maximum size of a process's mapped address space	Solaris

--silent

Parallel Jobs that use MPI or Linda use the **rsh.nb** script instead of **rsh**. **--silent** suppresses the messages that Netbatch usually sends to stdout when **nbq** is run interactively (with the **-i** switch). This allows **rsh.nb** to mimic **rsh** as closely as possible.

--task <taskname>

Specifies the Job task name (so that associated Jobs can be identified as such).



You can make a Job dependent on a task (so that it is only dispatched for execution when all the Jobs that make up the task have ended).

--triggers " <boolean_expression>"

Specifies that the Slave Job depends other Jobs, as defined in **boolean_expression**.

boolean_expression consists of any number of terms of the form [!]<JobID>[[<exit_status_expression>]], separated by the logical operators **&&** (AND) and **||** (OR).

Note

JobID can be either a Job ID or a task name. You are advised to explicitly specify whether it is a Job ID or a task name, because in ambiguous cases, Netbatch can assume that a task name is actually a Job ID, or vice versa.

Use one of the following instead of a Job ID/task name:

jobid:<Job_ID>
task:<task_name>

If a task name is used, the **exit_status_expression** does not evaluate to true until it is true for each of the Jobs that make up the task.

See [Section 6.5.18, Specifying a Task Name](#).

exit_status_expression can be one of the following:

- ◆ **exit**
- ◆ **exit <relational_operator> <exit_code>**
relational_operator must be <, <=, ==, >=, or >.
To specify multiple acceptable exit code conditions for a specific Job, use multiple **<JobID>[<exit_status_expression>]** terms separated by **||** (OR) (for example, **1 exit <30 || 1 exit>50**).

If **exit** is used on its own, it means that the Job ended (no matter what its exit status).

If no **exit_status_expression** is specified, **done** is assumed.

**Note**

Logical expressions are evaluated according to the following order of precedence - () (parentheses), ! (NOT), && (AND), || (OR).

So

```
"1 [exit<30] || !2 && (3 || 4)"
```

is the same as

```
"1 [exit<30] || ((!2) &&(3 || 4))"
```

Output

Netbatch creates a log file containing the Job output, named as follows:

- ◆ For each Slave Job, Netbatch creates a separate output log file, in the standard format:

```
##Date-Time#.<machine>
```

- ◆ If a Job name was specified when submitting the Master Job (that is, with `nbjob run --log-file <job_name>` or `nbq -j <job_name>`), then the log files are called `job_name:<index>`.
- ◆ If a Job name was specified when submitting the Slave Job (that is, with `nbjob prun --log-file <job_name>`), then the Slave Job's log file is called `job_name`.

See [Appendix D: Log File Contents and Job Exit Codes](#) for detailed information about Netbatch log file output.



A.2.9 nbjob pmodify

Name

nbjob pmodify - modifies a Parallel Job.

Description

Shrinks the number of executions slots required by a Parallel Job. If a Parallel Job consists of a number of iterations, and where later iterations require fewer execution slots than earlier ones, use this command to release the execution slots that are no longer required.

The alternative would be to split the Parallel Job into multiple Jobs, each of which would have to wait for the required number of execution slots to become available before it could be dispatched. With this command, after the Job has been dispatched for execution, no further waiting is required.

This command must be run by the Parallel Job itself (i.e., the Master Job).

The output of the command is a list of hostnames, indicating the execution slots that are still available to the Job.

Usage

```
nbjob pmodify [--target <pool_name>|<host_name>]
--shrink <number_of_slots>
--grow <number_of_slots> [<options>]
```

Switches

--target <pool_name>|<host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbjob** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

--grow <number_of_slots>

Specifies how many additional execution slots are required. When this switch is used, the command operates in **blocking mode**. That is, it does not return all the required execution slots are available.



When it returns, it writes list of hostnames that indicate the new execution slots to stdout. If more than one execution slot on the same Workstation is assigned to the Job, its hostname is repeated the appropriate number of times.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--shrink <number_of_slots>

Specifies how many execution slots are no longer required.

When it returns, it writes list of hostnames that indicate the execution slots still available to the Job to stdout. If more than one execution slot on the same Workstation is assigned to the Job, its hostname is repeated the appropriate number of times.



Output

A list of hostnames that indicate the execution slots still available to the Job. If more than one execution slot on the same Workstation is assigned to the Job, its hostname is repeated the appropriate number of times.

A.2.10 nbjob signal

Name

nbjob signal - sends a specific signal to a Job.

Description

Lets you send a specific signal to a Job.

Usage

```
nbjob signal [--signal <signal>] [--proc-filter <proc_filter>] <specification>
```

Switches

--target <pool_name> | <host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbjob** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

--signal <signal>

Specifies the signal to be sent to the Job(s). Any valid Unix signal may be used.

--proc-filter <proc_filter>

Specifies which of the Job's processes the signal will be sent to. **proc_filter** is a expression of the form **cmd=<command>**, where **command** is the process's command-line invocation.

specification

This is a boolean expression that specifies the Jobs to which the signal is to be sent.

specification is either **all** or an arbitrarily complex boolean expression.



This expression consists of <attribute>s, <operator>s, and <value>s, where:

- ◆ **attribute** is a valid Job field (see `nbstatus jobs --fields`, [Appendix A.3.10](#)).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")
 - ◆ A logical operator: &&, ||, or !
 - ◆ A comparison operator: <, <=, >, >=, !=, or ==
 - ◆ A non-strict operator: **is** or **isnt**
 - ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
 - ◆ A conditional operator: ? or :
- ◆ <value> is a number, a string, or another boolean expression.

Note

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If **value** is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

Expressions can be grouped together using parentheses.

For example, "(jobid>=10)&&(jobid<=20)" specifies that all Jobs with a Job ID between 10 and 20 (inclusive) are to be resubmitted.

If **specification** is **all**, all the user's Jobs are resubmitted.

The **startswith()** and **glob()** functions can also be used:

`"startswith(<field_name>,'<string>')"`

startswith() matches fields whose value starts with the specified string.

`"glob(<field_name>,'<expression>')"`

glob() matches fields that match **expression**.
expression can contain:



- ◆ Any character
- ◆ . - matches any character
- ◆ \w - matches any letter (upper- or lower-case)
- ◆ \d - matches any digit
- ◆ \s - matches any space character
- ◆ * - zero or more of the previous character
- ◆ ? - exactly one of the previous character

A.3 nbstatus

Name

nbstatus - displays status information.

Description

nbstatus is a set of commands for displaying status information, as follows:

- ◆ **nbstatus classes** - displays information about the classes supported by the Workstations in a Pool or Queue. See [Appendix A.3.1](#).
- ◆ **nbstatus constants** - displays Netbatch constants (currently only Job exit statuses). See [Appendix A.3.2](#).
- ◆ **nbstatus custom-fields** - displays list of defined custom fields. See [Appendix A.3.3](#).
- ◆ **nbstatus denial-of-service** - displays the number of requests received from each user and Workstation in the current interval. See [Appendix A.3.4](#).
- ◆ **nbstatus distance-map** - displays the mapping information of the network's topology; the distance between node pairs. See [Appendix A.3.5](#).
- ◆ **nbstatus dynamic-workstations** - displays "dynamic workstations" (that is, virtual machines that have been instantiated to run a particular Job). See [Appendix A.3.6](#).
- ◆ **nbstatus feeders** - displays Feeder information. See [Appendix A.3.7](#).



- ◆ **`nbstatus functions`** - displays list of supported functions. See [Appendix A.3.8](#).
- ◆ **`nbstatus groups`** - displays information about Netbatch and NIS groups. See [Appendix A.3.9](#).
- ◆ **`nbstatus jobs`** - displays Job information. See [Appendix A.3.10](#).
- ◆ **`nbstatus keyfiles`** - displays information about license keyfiles. See [Appendix A.3.11](#).
- ◆ **`nbstatus license-allocation`** - displays allocation information for a specified feature. See [Appendix A.3.12](#).
- ◆ **`nbstatus license-sessions`** - displays information about license sessions. See [Appendix A.3.13](#).
- ◆ **`nbstatus licenses`** - displays license information. See [Appendix A.3.14](#).
- ◆ **`nbstatus locks`** - displays information about Workstation locks (Workstations that are currently locked or scheduled to be locked). See [Appendix A.3.15](#).
- ◆ **`nbstatus logical-licenses`** - displays information about logical licenses (that is, for all available instances of each license, no matter how many different keyfiles they are listed in). See [Appendix A.3.16](#).
- ◆ **`nbstatus monitor-actions`** - displays information about actions taken by the Job failure detection system. See [Appendix A.3.17](#).
- ◆ **`nbstatus permissions`** - displays information about configured permissions. See [Appendix A.3.18](#).
- ◆ **`nbstatus policies`** - displays information about policies. See [Appendix A.3.19](#).
- ◆ **`nbstatus pools`** - displays Pool information. See [Appendix A.3.20](#).
- ◆ **`nbstatus probes`** - displays information about configured scheduler probes. See [Appendix A.3.21](#).
- ◆ **`nbstatus qslots`** - displays Qslot information. See [Appendix A.3.22](#).
- ◆ **`nbstatus remote-availability`** - for a Virtual Pool, displays availability of Workstations for a given Qslot and class. See [Appendix A.3.23](#).
- ◆ **`nbstatus resource-groups`** - displays information about Resource Groups. See [Appendix A.3.24](#).



- ◆ **nbstatus resources** - displays virtual resources. See [Appendix A.3.25](#).
- ◆ **nbstatus services** - displays information about registered services. See [Appendix A.3.26](#).
- ◆ **nbstatus virtualjobs** - displays jobs dispatched to remote resources while in waiting. See [Appendix A.3.27](#).
- ◆ **nbstatus vm-image-cache** - displays available OS images, for running Jobs that need an OS that no physical Workstations are running. Netbatch runs a Job like this in a virtual machine using one of these images. See [Appendix A.3.28](#).
- ◆ **nbstatus vms** - displays information about virtual machines running through Netbatch (that were launched using the **nbvm** command). See [Appendix A.3.29](#).
- ◆ **nbstatus wan** - displays WAN status. See [Appendix A.3.30](#).
- ◆ **nbstatus workstations** - displays Workstation information. See [Appendix A.3.31](#).

Usage (General)

```
nbstatus <object>[,...]
[--target <pool_name>|<host_name>]
[--fields "[*,]<field>[:[:<func>][:<size>]]
[,...]"|all]
[--format {table|block|csv|script}]
[--group-by <field_name>[,...]]
[--sort-by [-]<field_name>[,...]]
[--timeout <time>]
[--retry-timeout <time>{h|m}]
[<specification>]
```

You can combine multiple status requests for different object types into a single request. This allows you, for example, to see all the available information for the Workstation that each Job is running on. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

For example:

```
nbstatus jobs,workstations
"workstations.server==jobs.workstation"
```



For each running Job, this shows the details of the Job (from **nbstatus jobs**) and the Workstation that it is running on (from **nbstatus workstations**).

Note that the name of each field is prefixed with the object type (e.g., **jobs.Status**).

Short Switches

All **nbstatus** switches are in the following format:

--<switch_name>

Switches may be shortened according to the following rules:

- ◆ A short (single-letter) switch (e.g., **-h**) may be used if it is unambiguous.
- ◆ If a single-letter switch is ambiguous, you can use a shortened version of the long switch, including as much of the switch as necessary to be unambiguous. (For example, for **nbjob run**'s **--fields** switch, **-f** is ambiguous, so you can use **--fi**.)

Field Names

The field names that you specify in the arguments to the **--fields**, and **--sort-by** switches, and in the **specification** are **not case-sensitive**.

Help, Debug, and Version Information

To get a list of available **nbstatus** commands, type:

nbstatus --help

To get help information for a specific **nbstatus** command, type:

nbstatus <command> --help

For example:

nbstatus jobs --help

To show debug information, add the **--debug** switch:

nbstatus <command> --debug

To show the Netbatch version, use the **--version** switch:

nbstatus --version



Diagnostics

The command exit codes are listed below.

Note

Not all error codes are relevant for this command.

Note

These are command exit codes, not Job exit codes.

0

Successful execution

1

Specified Pool Master (**-P**) does not exist in **pools** file (older commands only (**nbq**, **nbqrm**, **nbstat**, etc.))

2

Unknown host specified (**--target** for newer commands, **-H** for older commands)

3

Error occurred on master side (for example, PPM error, Job could not be submitted because of missing permissions, non-existent class, etc.)

4

Problem getting user credentials (most likely an NIS issue on the client machine)

5

PPM timeout (The command waits one minute for a response.)

212

Depends on context:

- ◆ CSD error - Job failed
- ◆ Status command - invalid displayed field

**213**

Depends on context:

- ◆ CSD error - bad parameters
- ◆ Status command - invalid filter

214

Depends on context:

- ◆ CSD error - Workstation not available
- ◆ Status command - invalid grouping

215

Depends on context:

- ◆ CSD error - RG not available
- ◆ Status command - invalid sort

216

CSD error - Qslot not available

221

Interrupted

222

Permission denied

223

Timeout

224

Unknown error

225

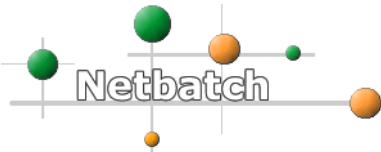
Permissions problem

229

The command was run with the **-v** switch (version)

230

The command was run with the **-h** switch (help)

**231**

Parsing error - mandatory filter not specified

232

Parsing error - invalid value

233

Parsing error - value not specified

234

Parsing error - undefined parameter

235

Parsing error - mandatory parameter not specified

244

No command handler in this context

245

No command handler

248

Communication problem - target service down

249

Communication problem - connection broken during receive stage

250

Server too busy (denial of service response or built-in request limit reached)

251

The server does not support the command.

253

Communication problem - unknown protocol

254

Communication problem - no response

255

Master not responding (probably down)



A.3.1 nbstatus classes

Name

nbstatus classes

Description

Displays information about the classes supported by the Workstations in a Pool or Queue, including the number of Workstations in each category (configured, connected, and available) that support each class and a list of these Workstations.

By default, **nbstatus classes**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.

Usage

```
nbstatus classes[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>[--GMT]]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```



You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

--target <pool_name> | <host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbstatus** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

--date-format <date_format>

Specifies the format to be used for dates and times.

**--fields " [* ,]<field>[:[<func>]][:<size>]]
[, . . .]" | all**

Displays only the selected fields. **--fields all** displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

func specifies a statistical operation that is performed on the field. This affects the output as follows:

- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items



- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, `stodstatus areas --fields "owner:uniq"` displays the number of different owners.) Note that `--group-by` is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which `concat` is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example: `--fields "RequestTimeFull-CreatedTimeFull"` This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as `csv`, except that the header row includes additional information about each field
- ◆ **script** - the same as `csv`, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

**--group-by <field_name>[,...]**

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[,...]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).



- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for glob(), below. It also matches partial strings. And it works with --history. (glob() does not.)

Note that with --history, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with --history, you must use * as a wildcard instead of .* without --history.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: is or isnt
- ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
- ◆ A conditional operator: ? or :
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ **MM/DD/YYYY-HH:mm:ss**



- ◆ **MM/DD/YYYY-HH:mm**
- ◆ **MM/DD/YYYY**

Expressions can be grouped together using parentheses.

Note

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If value is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If specification is all, all records are displayed.

Functions can also be used in specification. To see which functions are available for this nbstatus command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ **all** displays details of all classes supported by Workstations in the Pool.
- ◆ **"queue=='abc'"** displays details of all classes supported by Workstations that can accept Jobs from Queue abc.
- ◆ **"class=='BIGMEM'"** displays information about class BIGMEM (i.e., how many Workstations support the class, and a list of these Workstations).

Fields: classes

The following fields are available (default fields are marked with a *):

- ◆ **Available*** - the number of available Workstations that support the class
- ◆ **AvailableList** - list of available Workstations that support the class
- ◆ **Class*** - the name of the class
- ◆ **Configured*** - the number of configured Workstations that support the class



- ◆ **ConfiguredList** - list of configured Workstations that support the class
- ◆ **Connected*** - the number of connected Workstations that support the class
- ◆ **ConnectedList** - list of connected Workstations that support the class

Output

The **nbstatus classes** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 20, below, shows sample output for the following command:

```
nbstatus classes
```

PPM on itstl008
Version 7.2.0_0178_05
On since 11/06/2006 11:55:35
Time now 11/08/2006 13:45:47

Class Configured Connected Available

@ 1 1 1
burton 1 1 1

Figure 1: nbstatus classes Output

A.3.2 nbstatus constants



Name

nbstatus constants

Description

Displays information about Netbatch constants (currently only Job exit statuses).

By default, **nbstatus constants**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.

Usage

```
nbstatus constants[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>[--GMT]]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```



You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

--target <pool_name> | <host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbstatus** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

--date-format <date_format>

Specifies the format to be used for dates and times.

**--fields " [* ,]<field>[:[<func>]][:<size>]]
[, . . .]" | all**

Displays only the selected fields. **--fields all** displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

func specifies a statistical operation that is performed on the field. This affects the output as follows:

- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items



- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, `stodstatus areas --fields "owner:uniq"` displays the number of different owners.) Note that `--group-by` is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which `concat` is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example: `--fields "RequestTimeFull-CreatedTimeFull"` This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as `csv`, except that the header row includes additional information about each field
- ◆ **script** - the same as `csv`, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

**--group-by <field_name>[,...]**

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[,...]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).



- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for glob(), below. It also matches partial strings. And it works with --history. (glob() does not.)

Note that with --history, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with --history, you must use * as a wildcard instead of .* without --history.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: is or isnt
- ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
- ◆ A conditional operator: ? or :
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ **MM/DD/YYYY-HH:mm:ss**



- ◆ **MM/DD/YYYY-HH:mm**
- ◆ **MM/DD/YYYY**

Expressions can be grouped together using parentheses.

Note

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If value is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If specification is all, all records are displayed.

Functions can also be used in specification. To see which functions are available for this nbstatus command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ all displays details of all constants.

Fields: constants

The following fields are available (default fields are marked with a *):

- ◆ **Description** - the description that corresponds to the numerics value
- ◆ **NumericValue** - the numeric value of the constant
- ◆ **Type** - the constant type:
 - ◆ **CommandExitStatus** - command exit status
 - ◆ **JobExitStatus** - Job exit status
 - ◆ **TaskExitStatus** - task exit status (Feeder only)

Output

The nbstatus constants output depends on:

- ◆ The fields selected for display



- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 20, below, shows sample output for the following command:

```
nbstatus constants --format block "NumericValue==  
8"
```

```
PPM on itstl205  
version 7.5.0_0213_00  
On since 02/18/2009 09:55:33  
Time now 02/18/2009 10:25:16  
{  
    Type = JobExitStatus  
    NumericValue = -8  
    Description = (REMOTE_KILL - Job killed by a specific remote request)  
}
```

Figure 2: nbstatus constants Output

A.3.3 nbstatus custom-fields

Name

```
nbstatus custom-fields
```

Description

Displays list of defined custom fields.

By default, **nbstatus custom-fields**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.



Usage

```
nbstatus custom-fields[,<object>[,...]
[--target <pool_name>|<host_name>]
[--retry-timeout <time>{h|m}]
[--fields "[*,]<field>[:[:<func>][:<size>]]
[,...]"|all]
[--format {rfc4180-csv|typed-csv|
table|block|csv|script|xml}]
[--group-by <field_name>[,...]]
[--sort-by [-]<field_name>[,...]]
[--timeout <time>[--GMT]
[--date-format <date_format>]
[--ignore-invalid-fields] [<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

```
--target <pool_name>|<host_name>
```

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If `--target` is omitted, `nbstatus` uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

```
--date-format <date_format>
```

Specifies the format to be used for dates and times.

```
--fields "[*,]<field>[:[:<func>][:<size>]]
[,...]"|all
```

Displays only the selected fields. `--fields all` displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

`func` specifies a statistical operation that is performed on the field. This affects the output as follows:

- ♦ If `--group-by` is not used, specifying a `func` gives one line of output.



- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that **--group-by** is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example:**--fields "RequestTimeFull-CreatedTimeFull"** This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)



- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

**--sort-by [-]<field_name>[, . . .]**

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

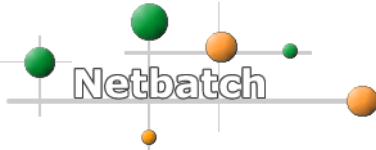
When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for **glob()**, below. It also matches partial strings. And it works with **--history**. (**glob()** does not.)



Note that with **--history**, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with **--history**, you must use * as a wildcard instead of .* without **--history**.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: **is** or **isnt**
- ◆ A bitwise operator: **&**, **|**, **~**, **^**, **<<**, **>>**, or **>>>**
- ◆ A conditional operator: **?** or **:**
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ **MM/DD/YYYY-HH:mm:ss**
 - ◆ **MM/DD/YYYY-HH:mm**
 - ◆ **MM/DD/YYYY**

Expressions can be grouped together using parentheses.

Note

A double equals sign (**==**) is used. **!=** (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If **value** is a string, it must be enclosed in single quotes.

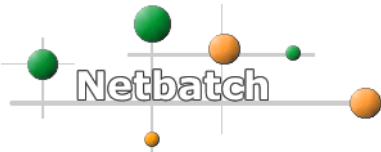
Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If **specification** is **all**, all records are displayed.

Functions can also be used in **specification**. To see which functions are available for this **nbstatus** command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ **all** displays all custom fields.



Fields: custom-fields

The following fields are available (default fields are marked with a *):

- ◆ **AllowedMapKeys** - allowed keys for a custom field type map
- ◆ **Collection** - custom field allowed values collection script
- ◆ **Description*** - the description that corresponds to the custom field
- ◆ **History** - should this field be reported to NBTracker (true/false)
- ◆ **Name*** - the custom field name
- ◆ **Scope*** - custom field scope (Jobs, workstations, and Qslots)
- ◆ **Type*** - custom field data type (e.g. string, boolean, date, etc.)
- ◆ **ValidationExpression** - custom field validation expression of input values
- ◆ **value** - allowed values

Output

The **nbstatus custom-fields** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 20, below, shows sample output for the following command:

```
nbstatus custom-fields --number 10 --fields  
"name,type,scope"
```



```
PPM on inbm001
Version 8.0.0_0400_00
On since 11/07/2011 10:28:01
Time now 11/15/2011 09:02:51
```

Name	Type	Scope
ACE_PROJECT	STRING	Jobs
ACTIVITY	STRING	Jobs
DiskSpace	DOUBLE	Jobs
FUNCTION	STRING	Jobs
LeakingAssertions	DOUBLE	Jobs
MemoryCapabilityPredictionData	STRING	Jobs
MemoryCapabilityPredictionPro>	LONG	Jobs
MemoryCapabilityPredictionRes>	MAP	Jobs
MemoryCapabilityPredictionSta>	STRING	Jobs
MemoryCapabilityPredictionTim>	STRING	Jobs

Figure 3: nbstatus custom-fields Output

A.3.4 nbstatus denial-of-service

Name

nbstatus denial-of-service

Description

Displays the number of requests received from each user and Workstation in the current interval. (The interval and limits for the number of requests for each user and Workstation (and for the total number of requests) are set by the Netbatch Administrator.)

By default, **nbstatus denial-of-service**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report



You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.

Usage

```
nbstatus denial-of-service[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>][--GMT]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

```
--target <pool_name>|<host_name>
```

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If `--target` is omitted, `nbstatus` uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

```
--date-format <date_format>
```

Specifies the format to be used for dates and times.

```
--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all
```

Displays only the selected fields. `--fields all` displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

`func` specifies a statistical operation that is performed on the field. This affects the output as follows:



- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that **--group-by** is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example: **--fields "RequestTimeFull-CreatedTimeFull"** This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:



- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

**--sort-by [-]<field_name>[, . . .]**

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for **glob()**, below. It also matches partial strings. And it works with **--history**. (**glob()** does not.)



Note that with **--history**, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with **--history**, you must use * as a wildcard instead of .* without **--history**.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: **is** or **isnt**
- ◆ A bitwise operator: **&**, **|**, **~**, **^**, **<<**, **>>**, or **>>>**
- ◆ A conditional operator: **?** or **:**
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ **MM/DD/YYYY-HH:mm:ss**
 - ◆ **MM/DD/YYYY-HH:mm**
 - ◆ **MM/DD/YYYY**

Expressions can be grouped together using parentheses.

Note

A double equals sign (**==**) is used. **!=** (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If **value** is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If **specification** is **all**, all records are displayed.

Functions can also be used in **specification**. To see which functions are available for this **nbstatus** command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ **all** displays details of all categories (users and Workstations from which requests have been received in the current interval).



- ◆ **itstl2022** displays the number of requests received from Workstation itstl2022 in the current interval (if any).

Fields: denial-of-service

The following fields are available (default fields are marked with a *):

- ◆ **Category*** - Workstation hostname or username
- ◆ **value*** - the number of requests from the user/Workstation in the current interval (taking into account command weights)

Output

The **nbstatus denial-of-service** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 20, below, shows sample output for the following command:

```
nbstatus denial-of-service
```

```
PPM on itstl109
Version 7.1.0_0166_03
On since 02/16/2006 08:35:38
Time now 02/16/2006 09:58:12
```

```
-----
Category      value
-----
itstl109      4
itstl2022     30
martinpx      34
-----
```

Figure 4: nbstatus denial-of-service Output



A.3.5 nbstatus distance-map

Name

nbstatus distance-map

Description

Displays the mapping information of the network's topology; the distance between node pairs.

By default, **nbstatus distance-map**:

- ◆ Displays all available fields
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field
- ◆ Shows the mapping of all the network's nodes
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.

Usage

```
nbstatus distance-map
[--target <pool_name>|<host_name>]
[--retry-timeout <time>{h|m}]
[--fields "[*,]<field>[:[<func>][:<size>]]
[,...]"|all]
[--format {rfc4180-csv|typed-csv|
table|block|csv|script|xml}]
[--group-by <field_name>[,...]]
[--sort-by [-]<field_name>[,...]]
[--timeout <time>[--GMT]
[--date-format <date_format>]
[--ignore-invalid-fields <filter>] [<specification>]
```



You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

--target <pool_name> | <host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbstatus** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

--date-format <date_format>

Specifies the format to be used for dates and times.

**--fields " [* ,]<field>[:[<func>]][:<size>]]
[, . . .]" | all**

Displays only the selected fields. **--fields all** displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

func specifies a statistical operation that is performed on the field. This affects the output as follows:

- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items



- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, `stodstatus areas --fields "owner:uniq"` displays the number of different owners.) Note that `--group-by` is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which `concat` is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example: `--fields "RequestTimeFull-CreatedTimeFull"` This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as `csv`, except that the header row includes additional information about each field
- ◆ **script** - the same as `csv`, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

**--group-by <field_name>[,...]**

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[,...]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).



- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

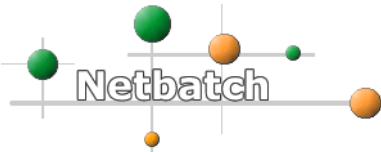
- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for glob(), below. It also matches partial strings. And it works with --history. (glob() does not.)

Note that with --history, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with --history, you must use * as a wildcard instead of .* without --history.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: is or isnt
- ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
- ◆ A conditional operator: ? or :
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ **MM/DD/YYYY-HH:mm:ss**



- ◆ **MM/DD/YYYY-HH:mm**
- ◆ **MM/DD/YYYY**

Expressions can be grouped together using parentheses.

Note

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If value is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If specification is all, all records are displayed.

Functions can also be used in specification. To see which functions are available for this nbstatus command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

Filter by creating an expression the displayed entries will match, for example, "field1=='string1'&&field2==int2"

Fields: distance-map

The following fields are available (default fields are marked with a *):

- ◆ Source* - Source node name
- ◆ Destination* - Destination node name
- ◆ Distance* - Distance (hops) between source to destination

Figure 20, below, shows sample output for the following command:

```
nbstatus distance-map
```



```
PPM on itstl049
Version 8.1.0_0448_01
On since 09/06/2012 12:42:38
Time now 09/09/2012 14:15:35
```

Source	Destination	Distance
<hr/>		
board0.0.0	board0.0.0	1
board_starter0.0.11	board0.0.0	113
board_starter0.0.11	izse.0.0.1	1
board_starter0.0.11	izse.0.0.2	1

Figure 5: nbstatus distance-map Output

A.3.6 nbstatus dynamic-workstations

Name

nbstatus dynamic-workstations

Description

Displays information about "dynamic workstations" (that is, virtual machines that have been instantiated to run a particular Job).

By default, **nbstatus dynamic-workstations**:

- ◆ Displays all available fields
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Shows all dynamic workstations
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.



Usage

```
nbstatus dynamic-workstations[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>[--GMT]]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

```
--target <pool_name>|<host_name>
```

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If `--target` is omitted, `nbstatus` uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

```
--date-format <date_format>
```

Specifies the format to be used for dates and times.

```
--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all
```

Displays only the selected fields. `--fields all` displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

`func` specifies a statistical operation that is performed on the field. This affects the output as follows:

- ♦ If `--group-by` is not used, specifying a `func` gives one line of output.



- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that **--group-by** is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example:**--fields "RequestTimeFull-CreatedTimeFull"** This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)



- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

**--sort-by [-]<field_name>[, . . .]**

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for **glob()**, below. It also matches partial strings. And it works with **--history**. (**glob()** does not.)



Note that with **--history**, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with **--history**, you must use * as a wildcard instead of .* without **--history**.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: **is** or **isnt**
- ◆ A bitwise operator: **&**, **|**, **~**, **^**, **<<**, **>>**, or **>>>**
- ◆ A conditional operator: **?** or **:**
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ **MM/DD/YYYY-HH:mm:ss**
 - ◆ **MM/DD/YYYY-HH:mm**
 - ◆ **MM/DD/YYYY**

Expressions can be grouped together using parentheses.

Note

A double equals sign (**==**) is used. **!=** (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If **value** is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If **specification** is **all**, all records are displayed.

Functions can also be used in **specification**. To see which functions are available for this **nbstatus** command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ **all** displays all dynamic workstations (which is what it does by default if no filter is applied).



- ◆ "PhysicalHost=='icsl2205'" displays only the dynamic workstations that are running on the physical machine called **icsl2205**.
- ◆ "Server=='itstl416'" only displays details of the dynamic workstation whose hostname is **itstl416**.

Fields: dynamic-workstations

The fields that are available are the same as for **nbstatus workstations**, with the following additions/differences (default fields are marked with a *):

- ◆ **OSImage** - the OS image that the virtual machine is running
- ◆ **PhysicalHost** - the hostname of the physical machine that the virtual machine is running on
- ◆ **VMStarterJobID** - the ID of the virtual machine started Job. (This is a special kind of Job that is responsible for starting the VM. It continues running until the VM shuts down. Note that it does not appear in **nbstatus jobs**.)
- ◆ **WSCapabilities** and related fields - in **nbstatus dynamic-workstations**, these show the Workstation capabilities that Netbatch instantiated the dynamic workstation with (which are the same as the requested class reservation requirement).

Output

The **nbstatus dynamic-workstations** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 6, below, shows sample output for the following command:

```
nbstatus dynamic-workstations --format block  
--number 1
```



```
PPM on icsl9579
Version 8.2.1_0597_05
On since 04/28/2014 13:15:39
Time now 05/18/2014 14:58:52
{
    PhysicalHost = icsl2140
    Server = itstl406
    Status = Accepting
    Load = 0.08/99.00/100.00
    InteractiveUsers = 0/0
    Idle = 0/0
    Sel = 0
    Jobs = 0
    Onsince = 05/18/2014 12:50:50
}
```

Figure 6: nbstatus dynamic-workstations Output

A.3.7 nbstatus feeders

Name

nbstatus feeders

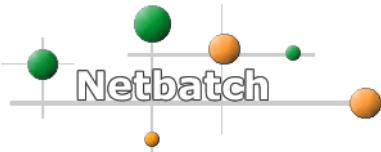
Description

Displays information about running Netbatch Feeders (including ones that have been stopped).

By default, **nbstatus feeders**:

- ◆ Displays all available fields
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Shows all users' Feeders
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.



Usage

```
nbstatus feeders[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>][--GMT]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Note

If a Feeder process is killed, it will still appear in `nbstatus feeders` output until the next routine cleanup (which happens once a day).

Target

```
--target <pool_name>|<host_name>
```

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If `--target` is omitted, `nbstatus` uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

```
--date-format <date_format>
```

Specifies the format to be used for dates and times.

```
--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all
```

Displays only the selected fields. `--fields all` displays all available fields. Fields are displayed in the specified order.



See the Fields section, below, for a list of available fields.

func specifies a statistical operation that is performed on the field.

This affects the output as follows:

- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

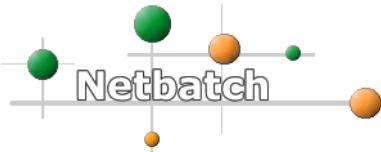
- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that **--group-by** is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example: **--fields "RequestTimeFull-CreatedTimeFull"** This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)



--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--gmt

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.



If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[,...]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.



- ◆ The pattern-matching operator: `=~`. The syntax of the expression that follows this operator is the same as for `glob()`, below. It also matches partial strings. And it works with `--history`. (`glob()` does not.)

Note that with `--history`, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with `--history`, you must use `*` as a wildcard instead of `.*` without `--history`.

Note

`=~` is case-sensitive.

- ◆ A non-strict operator: `is` or `isnt`
- ◆ A bitwise operator: `&`, `|`, `~`, `^`, `<<`, `>>`, or `>>>`
- ◆ A conditional operator: `?` or `:`
- ◆ `value` is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ `MM/DD/YYYY-HH:mm:ss`
 - ◆ `MM/DD/YYYY-HH:mm`
 - ◆ `MM/DD/YYYY`

Expressions can be grouped together using parentheses.

Note

A double equals sign (`==`) is used. `!=` (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If `value` is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If `specification` is `all`, all records are displayed.

Functions can also be used in `specification`. To see which functions are available for this `nbstatus` command, use [nbstatus functions](#). For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:



- ◆ `all` displays all Feeders (not just the user's own Feeders).
- ◆ `"Name=='myFeeder'"` displays only details of the Feeder called `myFeeder`.
- ◆ `"Status=='running'"` only displays details of Jobs submitted to Queue `Q1`.
- ◆ `"User=='andreys'"` displays only Feeders belonging to user `andreys`.
- ◆ `"User=='andreys'&&Status=='running'"` displays only running Feeders that belong to user `andreys`.

Fields: feeders

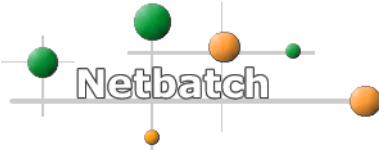
The following fields are available (default fields are marked with a `*`):

- ◆ `Group` - Feeder group
- ◆ `Host*` - The hostname of the Workstation on which the Feeder is running
- ◆ `LastUpdate*` - The last time the Feeder was stopped or restarted, or the last time the Feeder performed a self-update (which happens every six hours)
- ◆ `Name*` - The Feeder name
- ◆ `Patches` - Loaded patches
- ◆ `Port*` - The Feeder's port
- ◆ `StartTime*` - The time the Feeder was started. (Note that if the Feeder is stopped and restarted, the value of this field is unaffected.)
- ◆ `Status*` - The Feeder's current status (either `Running` or `Stopped`)
- ◆ `User*` - The name of the user who owns the Feeder
- ◆ `Version` - Feeder version
- ◆ `WorkArea` - Feeder work area

Output

The `nbstatus feeders` output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order



- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 7, below,The following figure shows sample output for the following command:

```
nbstatus feeders "user='yoavf'"
```

```
DirectoryService on itstl139
Version 6.4.0_0123_05
On since 06/28/2004 10:59:03
Time now 06/28/2004 11:09:39
{
    Host = itstl139
    Port = 35058
    Name =Test
    User = yoavf
    StartTime = 06/28/2004 11:09:24
    Status = Running
    LastUpdate = 06/28/2004 11:09:29
}
```

Figure 7: nbstatus feeders Output

A.3.8 nbstatus functions

Name

nbstatus functions

Description

Displays a list of supported functions.

By default, **nbstatus functions**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report



You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.

Usage

```
nbstatus functions [,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>][--GMT]  
[--date-format <date_format>]  
[--ignore-invalid-fields]  
[--context "<command>"]  
[<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Context

--context "<command>"

Specifies the status command for which valid functions should be listed. For example:

`nbstatus functions --context "nbstatus pools".`

If the status command is not specified, all functions are displayed.

Target

--target <pool_name>|<host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbstatus** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)



Options

--date-format <date_format>

Specifies the format to be used for dates and times.

--fields "[*,]<field>[:[<func>][:<size>]] [, . . .]" | all

Displays only the selected fields. **--fields all** displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

func specifies a statistical operation that is performed on the field. This affects the output as follows:

- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that **--group-by** is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example:**--fields "RequestTimeFull-CreatedTimeFull"** This calculates the difference between these two date fields and displays the result (in seconds).

**Note**

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

**--retry-timeout <time>{h|m}**

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[, . . .]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ♦ **attribute** is a valid field (see **--fields**, above).
- ♦ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ♦ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)



- ◆ A logical operator: `&&`, `||`, or `!`
- ◆ A comparison operator: `<`, `<=`, `>`, `>=`, `!=`, or `==`

Note

For strings, `==` is not case-sensitive. If you need to do a case-sensitive comparison, use the `equals()` function instead.

- ◆ The pattern-matching operator: `=~`. The syntax of the expression that follows this operator is the same as for `glob()`, below. It also matches partial strings. And it works with `--history`. (`glob()` does not.)

Note that with `--history`, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with `--history`, you must use `*` as a wildcard instead of `.*` without `--history`.

Note

`=~` is case-sensitive.

- ◆ A non-strict operator: `is` or `isnt`
- ◆ A bitwise operator: `&`, `|`, `~`, `^`, `<<`, `>>`, or `>>>`
- ◆ A conditional operator: `?` or `:`
- ◆ `value` is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ `MM/DD/YYYY-HH:mm:ss`
 - ◆ `MM/DD/YYYY-HH:mm`
 - ◆ `MM/DD/YYYY`

Expressions can be grouped together using parentheses.

Note

A double equals sign (`==`) is used. `!=` (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If `value` is a string, it must be enclosed in single quotes.



Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If **specification** is **all**, all records are displayed.

Functions can also be used in **specification**. To see which functions are available for this **nbstatus** command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ **all** displays a list of all functions.

Fields: functions

The following fields are available (default fields are marked with a *****):

- ◆ **AllowedValues*** - the allowed values, if applicable
- ◆ **ArgsType*** - function field's argument types (e.g. string, boolean, object)
- ◆ **Description*** - description of the function
- ◆ **Groupable** - whether the function is available as a grouping (true, false)
- ◆ **Name*** - the function name
- ◆ **NumOfArgs*** - the number of arguments
- ◆ **ReturnType*** - function field's return data type (e.g. string, boolean, date, etc.)

Output

The **nbstatus functions** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

[Figure 20](#), below, shows sample output for the following command:

```
nbstatus functions --number 10 --fields  
"name,type,scope"
```



```
PPM on inbm001
Version 8.0.0_0400_00
On since 11/07/2011 10:28:01
Time now 11/15/2011 10:12:05
```

Name	NumOfArgs	ReturnType	ArgsType
<hr/>			
!	1	boolean	boolean
!=	2	boolean	object object
!~	2	boolean	string string
&&	0+	boolean	boolean
*	0+	double	double
+	0+	double	double
-	0+	double	double
/	0+	double	double
<	2	boolean	object object
<=	2	boolean	object object

Figure 8: nbstatus function Output

A.3.9 nbstatus groups

Name

nbstatus groups

Description

Displays information about Netbatch and NIS groups.

By default, **nbstatus groups**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report



You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.

Usage

```
nbstatus groups[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>[--GMT]]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

```
--target <pool_name>|<host_name>
```

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If `--target` is omitted, `nbstatus` uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

```
--date-format <date_format>
```

Specifies the format to be used for dates and times.

```
--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all
```

Displays only the selected fields. `--fields all` displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

`func` specifies a statistical operation that is performed on the field. This affects the output as follows:



- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that **--group-by** is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example: **--fields "RequestTimeFull-CreatedTimeFull"** This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:



- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

**--sort-by [-]<field_name>[, . . .]**

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for **glob()**, below. It also matches partial strings. And it works with **--history**. (**glob()** does not.)



Note that with **--history**, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with **--history**, you must use ***** as a wildcard instead of **.*** without **--history**.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: **is** or **isnt**
- ◆ A bitwise operator: **&**, **|**, **~**, **^**, **<<**, **>>**, or **>>>**
- ◆ A conditional operator: **?** or **:**
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ **MM/DD/YYYY-HH:mm:ss**
 - ◆ **MM/DD/YYYY-HH:mm**
 - ◆ **MM/DD/YYYY**

Expressions can be grouped together using parentheses.

Note

A double equals sign (**==**) is used. **!=** (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If **value** is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If **specification** is **all**, all records are displayed.

Functions can also be used in **specification**. To see which functions are available for this **nbstatus** command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ **all** displays all groups.
- ◆ **"groupname=='zeus'"** display details of the group called **zeus**.



- ◆ "gid=='107'" displays details of the group whose group ID is 107.

Fields: groups

The following fields are available (default fields are marked with a *):

- ◆ **GID*** - The group ID
- ◆ **GroupName*** - The name of the group
- ◆ **Type*** - The type (UNIX (NIS) or Netbatch)
- ◆ **Users*** - The users who belong to the group

Output

The **nbstatus groups** output depends on:

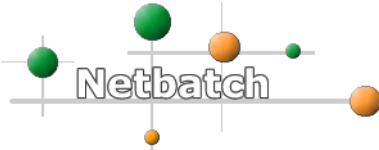
- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 14, below,The following figure shows sample output for the following command:

```
nbstatus groups --format block "groupname=='zeus'"
```

```
AuthorizationService on itstl139
Version 7.0.0_0140_00
On since 06/20/2005 08:41:01
Time now 06/20/2005 12:31:52
{
  GroupName = zeus
  GID = 107
  Type = UNIX
  Users = anatb shohaml cse mkessler pv_help edaniel shroman vkondra
          hadash cben kerem zkorovin sshahin nirnaor zhanna ygreen integ nsasha gilt
          gilka gmeeker mzirin gboris rmark levyg gilamid pauli avardi
}
```

Figure 9: nbstatus groups Output



A.3.10 nbstatus jobs

Name**nbstatus jobs****Description**

Each Netbatch Pool has two Queues - the wait Queue and the run Queue. Any Job submitted to the Pool is either in the wait Queue or the run Queue. This means the Job has either been dispatched to a Workstation for execution or is still waiting for a Workstation to become available. **nbstatus jobs** shows both waiting and running Jobs, though the output can be restricted to show either only running or only waiting Jobs.

By default, **nbstatus jobs**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Only shows the user's own Jobs
- ◆ Only shows Jobs that are running or waiting (but not completed Jobs)
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.



Usage

```
nbstatus jobs[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--history <time>] [--timeout <time>][--GMT]  
[--date-format <date_format>]  
[--ignore-invalid-fields]  
[--tracker <tracker_name>]  
[<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

--target <pool_name>|<host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbstatus** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

--date-format "<date_format>"

Specifies the format to be used for dates and times. **date_format** can include any of the following, in any order, separated by valid separator characters (e.g., space, colon (:), slash (/), etc.):

- ◆ **yyyy** - year (four digits)
- ◆ **yy** - year (two digits)
- ◆ **MM** - month
- ◆ **dd** - date
- ◆ **hh** - hour



- ◆ **mm** - minutes
- ◆ **ss** - seconds
- ◆ **sss** - milliseconds

The default format is "**MM/dd/yyyy hh:mm:ss**".

```
--fields "[*,]<field>[:[<func>][:<size>]]  
[,...]"|all
```

Displays only the selected fields. --fields all displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

func specifies a statistical operation that is performed on the field. This affects the output as follows:

- ◆ If --group-by is not used, specifying a **func** gives one line of output.
- ◆ If --group-by is used, this gives one line of output per group.
- ◆ If --group-by is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that --group-by is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.



You can also perform simple calculations on numeric and date fields. For example:`--fields "RequestTimeFull-CreatedTimeFull"` This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

There are also several function fields available, that are specified in the same way as `field` (above):

- ◆ `countjobs(<expression>)` - displays the number of Jobs on the Workstation that match `expression` (where `expression` is an arbitrarily complex boolean expression that can include any `nbstatus jobs` field).
- ◆ `countsslots(<expression>)` - displays the number of executions slots on the Workstation that match `expression` (where `expression` is an arbitrarily complex boolean expression that can include any `nbstatus jobs` field).
- ◆ `TimeInState(<state>)` - displays the time the Job is in its current state (in milliseconds). Valid states are: `send`, `resched`, `approving`, `susp`, `resub`, `del`, `wait remote`, `wait run`, and `disc`.
- ◆ `CanJobPreempt(<full_job_ID>)` - displays true if the Job belongs to a Qslot that can *currently* preempt the Qslot of the specified Job.
- ◆ `CanRent()` - displays true if

`--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}`

Specifies that output is to be displayed as:

- ◆ `table` - a table
- ◆ `block` - blocks (one block for each item)
- ◆ `csv` - comma-separated values (suitable for importing into Excel)
- ◆ `rfc4180-csv` - RFC 4180-compliant comma-separated values



- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--history <number>{M|H|D}

Requests historical Job information for the last **number** minutes, hours or days,

Note

In Netbatch 8.1 and later, nbstatus jobs --history only shows Jobs from one Pool. Previously, it showed Jobs from all the Pools that report to the same Tracker.

With this switch, the set of available fields is not the same as without the switch.

In addition to the fields listed in **--fields**, there are additional fields available when using the **--history** switch. See Fields section below.

When **--group-by** is used with **--history**, numerical fields display average values. Other fields display -.

Note

By default, Tracker allows no more than four queries from the same user at the same time.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored.

**--number <num>**

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[,...]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

--tracker <tracker_name>

If the Pool uses multiple Trackers, this switch specifies which non-primary Tracker to use for history queries.

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

**Note**

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: `&&`, `||`, or `!`
- ◆ A comparison operator: `<`, `<=`, `>`, `>=`, `!=`, or `==`

Note

For strings, `==` is not case-sensitive. If you need to do a case-sensitive comparison, use the `equals()` function instead.

- ◆ The pattern-matching operator: `=~`. The syntax of the expression that follows this operator is the same as for `glob()`, below. It also matches partial strings. And it works with `--history`. (`glob()` does not.)

Note that with `--history`, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with `--history`, you must use `*` as a wildcard instead of `.*` without `--history`.

Note

`=~` is case-sensitive.

- ◆ A non-strict operator: `is` or `isnt`
- ◆ A bitwise operator: `&`, `|`, `~`, `^`, `<<`, `>>`, or `>>>`
- ◆ A conditional operator: `?` or `:`
- ◆ `value` is a number, a string, or another boolean expression.
If `value` is a date/time, it must be in one of the following formats:
 - ◆ `MM/DD/YYYY-HH:mm:ss`
 - ◆ `MM/DD/YYYY-HH:mm`
 - ◆ `MM/DD/YYYY`

Expressions can be grouped together using parentheses.

**Note**

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If **value** is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If **specification** is **all**, all records are displayed.

Functions can also be used in **specification**. To see which functions are available for **nbstatus jobs**, use **nbstatus functions** with the --context "nbstatus jobs" switch. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ **all** displays all Jobs (not just the user's own Jobs).
- ◆ "**Jobid==2**" displays only details of the Job with ID 2.
- ◆ "**Queue=='Q1'**" only displays details of Jobs submitted to Queue Q1.
- ◆ "**User=='andreys'**" displays only Jobs belonging to user **andreys**.
- ◆ "**User=='andreys' && Queue=='Q1'**" displays only Jobs that were submitted by **andreys** to Queue **Q1**.
- ◆ "**TimeInRunning>60000**" displays only Jobs that have been running for more than 60 seconds.

Fields: jobs

The following fields are available (default **nbstatus jobs** fields are marked with a * and default **nbstatus jobs --history** fields are marked with **):

Note

Additional fields available when using the --history switch are listed after these fields, under [Fields: jobs --history](#).



- ◆ **ActualClassReservation** - the actual class reservation (the higher of **ClassReservation** and **ClassImplicitReservation**)
- ◆ **AllocationCost** - the calculated cost of the Job (depends on Workstation cost and concurrency and Job resource reservation requirements). Only affects scheduling decisions if the Qslot that the Job was submitted to uses cost fair-share scheduling (see [Section 5.1.3.6, Cost Fair-share Resource Allocation](#)).
- ◆ **AlternativeReservationPercentage** - the percentage of surplus reservation that the job will take
- ◆ **AppliedPag** - The PAG that is applied on the Workstation. This depends on the **RequestedPag** and the Workstation configuration. This can be:
 - ◆ **on** - the PAG is added.
 - ◆ **off** - the PAG is not added.
 - ◆ **nosuspend** - the PAG is added, but Netbatch does not suspend child processes that change their process group ID (if **RequestedPag = nosuspend**).
- ◆ **AvgRM** - average real memory usage in MB
- ◆ **AvgVM** - average virtual memory usage in MB
- ◆ **Blocking** - Whether the Job is a blocking Job (i.e., whether it was submitted with **nbq --block** or **nbjob run --mode blocking**) (true/false)
- ◆ **CheckpointTool** - the checkpointing tool that the Job uses, if any.
- ◆ **CheckpointToolData** - the checkpointing tool data that was specified for the Job (if any).
- ◆ **Checkpoints** - all the saved checkpoints for the Job (if any).
- ◆ **CheckpointsDirectory** - the path of the directory where the Job's checkpoints are stored
- ◆ **CheckpointsInterval** - how often checkpoints are created for the Job (if enabled)
- ◆ **CheckpointsToSave** - the number of checkpoints that are saved for the Job (if enabled). Older checkpoints are deleted.
- ◆ **Class*** - Class name or expression, as specified with **nbq -c** or **nbjob run --class** or in **NBCLASS**
- ◆ **ClassImplicitReservation** - Implicit reservation defined at Queue/Qslot level



(The Netbatch Administrator can specify that Jobs submitted to a particular Queue or Qslot are automatically assigned real or virtual memory requirements, or requirements that correspond to defined custom Workstation attributes.)

- ◆ **ClassReservation** - Resource reservation requirements
- ◆ **Cmd** - Executable name
- ◆ **Cmdline*** - Full command line
- ◆ **Cmdname** - The Job command (without arguments)
- ◆ **Credentials** - The user's credentials:
 - ◆ **Username** - specifies who these credentials authenticate
 - ◆ **KerberosRealm** - the TGT's Kerberos realm
 - ◆ **KerberosNextExpiry** - date and time of TGT expiration
 - ◆ **KerberosFinalExpiry** - final TGT expiration date and time—TGT cannot be renewed after the final expiration.
- ◆ **CurrentCost** - the Job's current cost
- ◆ **CurrentReservationPeriodLength** - the current calculated reservation window length (in minutes)
- ◆ **CustomAttributes** - the Job's custom attributes and their values
- ◆ **Cwd** - Job working directory
- ◆ **Daemon** - Whether the Job is running as a daemon (i.e., whether it was submitted with the **--daemon** switch) (true/false)
- ◆ **DependentJobs** - List of dependent Jobs (that is, Jobs for which this Job is a trigger Job)
- ◆ **DispatchedTime** - the time/date when the Job was dispatched to the resource
- ◆ **EnvSubmitted** - Whether user environment variables were passed with the Job (i.e., whether **nbq -E** or **nbjob run --no-env** was used) (true/false)
- ◆ **ExcludedRemoteResources** - (Virtual Pool and Feeder only) list of Pools to which the Job can be sent.
- ◆ **ExecLimit** - The execution limits specified for the Job (if any), as specified with **nbjob run --exec-limits** or **nbq -y** or in **NB_EXEC_LIMITS**
- ◆ **ExecTime** - time Job was in runaway state (in **<hours>h:<mins>m:<secs>s** format)



- ◆ **ExitStatus**** - Job exit status
- ◆ **ExpectedRuntime** - the Job's expected run time (for renting, as specified with the `nbjob run --expected-runtime` switch).
- ◆ **ExpiredReservationCount** - the number of times a resource has been reserved and released for this Job
- ◆ **FinishTime**** - the date and time when the Job finished running
- ◆ **FullId** - The full (qualified) Job ID
- ◆ **HungLimit** - The hung limits specified for the Job (if any), as specified with the `--hung-limits` switch or in **NB_HUNG_LIMITS**
- ◆ **IdleTime** - time Job was in idle state (in `<hours>h:<mins>m:<secs>s` format)
To filter on this field, specify the time in milliseconds.
- ◆ **IncludedRemoteResources** - (Virtual Pool and Feeder only) list of Pools to which the Job cannot be sent
- ◆ **Interactive** - whether the Job is an interactive Job (true/false)
- ◆ **IsHung** - whether the Job is hung or not
- ◆ **IsRunaway** - whether the Job is a runaway Job or not
- ◆ **Iteration** - the number of times the Job has been re-run
- ◆ **JobBasePriority** - the priority that the Job started running with
- ◆ **JobCalculatedShareOverTime** -
- ◆ **JobCalculatedUsageOverTime** -
- ◆ **JobCostShareOverTime** -
- ◆ **JobCostUsageOverTime** -
- ◆ **JobConstraints** - Job constraints, as specified by the `--job-constraints` switch or in a Job Profile
- ◆ **JobCurrentPriority** - the Job's current priority
- ◆ **JobKiller** - the entity that killed or removed the Job
- ◆ **JobLimits** - Job limits that apply to the Job (applied using the `nbjob run/modify --job-limits` switch)
- ◆ **JobLogFile** - the full path of the Job log file



- ◆ **JobStarter** - the specified Job starter program (if any), as specified with the `--job-starter` switch or in **NB_JOB_STARTER**
- ◆ **JobTrace** - the route that the Job took to get where it is (including Virtual Pools, Feeders, and physical Pools)
- ◆ **JobTraceGeneral** - the Job's submission path (the Job's dispatch stations). The path's feeders and pools need to be running Netbatch version 8.1 or later. Values are:
 - ◆ **NBFLM** - Netbatch Flow Manager Feeder
 - ◆ **VP** - Virtual Pool
 - ◆ **PPM** - Physical Pool Master
- ◆ **Jobid*** **- The Job's short ID
- ◆ **KillReason** - the reason why the Job was removed (as specified in `nbjob remove`'s `--reason` switch, or an the appropriate reason if the Job was removed/resubmitted because the Workstation it was running on was locked).
- ◆ **KillThreshold** - The number of minutes that the Job can be suspended before it is killed, as specified with `nbq -k` or `nbjob run --kill`
- ◆ **LMPProject** - The value of the user's **LM_PROJECT** environment variable
- ◆ **LastReservationPeriodLength** - the last calculated reservation window length (in minutes)
- ◆ **LastUpdate** - date/time of last status update
- ◆ **Lenders** - the Job(s) that the Job is renting resources from (if any).
- ◆ **LicenseBundleID** - the Job's license reservation identifier (in the Licensing Service)
- ◆ **LicenseKeyFile** - The license keyfile to be used (if any), as specified with the `--license-keyfile` switch or in **NB_LICENSE_FILE**
- ◆ **LicenseRequest** - The licenses that the Job requires, as specified with `nbq -l` or `nbjob run --license`
- ◆ **LicenseReservationRequest** - The Job's license reservation requirements (if any)
- ◆ **MailDisabled** - Whether sending of mail relating to the Job is disabled (i.e., whether the Job was submitted with `nbjob run --mail no`) (true/false)



- ◆ **MailList** - The list of users (other than the Job's owner) to whom mail about the Job is sent, as specified with `--mail-list`
- ◆ **MailOnEnd** - Whether mail is to be sent to the user when the Job ends, as specified with `nbq -n` or `nbjob run`
`--mail e` (true/false)
- ◆ **MailOnStart** - Whether mail is to be sent to the user when the Job starts running, as specified with `nbq -s` or `nbjob run --mail s` (true/false)
- ◆ **MailThreshold** - The amount of time that the Job can be suspended before mail is sent to the user, as specified with `nbq -m <min>` or `nbjob run --mail <sec>`
- ◆ **MasterClass** - the class requirements of the (Parallel) master Job.
- ◆ **MemoryCapabilityPredictionData** - full memory prediction data
- ◆ **MemoryCapabilityPredictionFields** - list of fields whose values may be changed by memory prediction and their values
- ◆ **MemoryCapabilityPredictionProfileType** - the ID of the specific prediction profile that was matched by the prediction system (used for debugging the prediction system)
- ◆ **MemoryCapabilityPredictionResult** - the Job's predicted memory requirement
- ◆ **MemoryCapabilityPredictionStatus** - whether memory prediction is enabled:
 - ◆ `enable`—memory prediction is enabled.
 - ◆ `disable`—memory prediction is disabled.
 - ◆ `dry`—prediction is enabled without affecting scheduling.
- ◆ **MemoryCapabilityPredictionTimeStamp** - memory prediction timestamp
- ◆ **ModificationHistory** - the Job's modification history specified by map of maps of objects: Queue name -> field name -> modified field value. For example:
`ModificationHistory = tester2={priority=10}`
- ◆ **MultipleWaitReasons** - the Job's wait reasons in the different physical Pools that it is waiting in. (The Job was submitted to a Virtual Pool or to a Feeder and is waiting in multiple physical Pools, and it does not have the same wait reason in all these Pools.) See **MultipleWaitReasons**, above.



Such Jobs have a **WaitReason** of **Multiple wait reasons exist**. See field **MultipleWaitReasons**.

- ◆ **NBPROJ** - the value of the **NBPROJ** environment variable
- ◆ **Name** - Job name, as specified with **nbjob run --log-file** or **nbq -j**
- ◆ **Nbautoreq** - The auto-requeue string (which includes a list of error codes or a boolean expression including error codes that specifies the conditions under which the Job is to be resubmitted), as specified with the **--autoreq** switch or in **NBAUTOREQ**
- ◆ **OnJobFinish** - the value of the **--on-job-finish** request (which can come from the **NB_ON_JOB_FINISH** environment variable, the **--on-job-finish** switch, and/or from a Job Profile)
- ◆ **PROCESS_NAME** - the value of the **PROCESS_NAME** environment variable
- ◆ **PROJECT** - the value of the **PROJECT** environment variable
- ◆ **ParallelHosts** - the hosts that Netbatch has assigned to the Parallel Job. If there is more than one execution slot available on the same host, that host's name appears the corresponding number of times.
- ◆ **ParallelJobConstraints** - Parallel Job constraints (as specified with the **--parallel-job-constraints** switch)
- ◆ **ParallelJobID** - if the Job is a Parallel Job, displays the full ID of the Master Job. If the Job is not a Parallel Job, displays the Job's full ID.
- ◆ **ParallelPhase** - if the Job is a Parallel Job, this field shows whether the Job is in the **Inclusive** phase (where the scheduler is looking for and marking execution slots for the Job, and where other Jobs can still run in these slots) or the **Exclusive** phase (where the execution slots are reserved for the Job). See [Section 6.11, Submitting a Parallel Job](#).
- ◆ **ParallelRequest** - the Job's parallel reservation requirements (if any), as specified with **--parallel**
- ◆ **ParallelReservedSlotsCount** - the number of execution slots that are currently reserved for slave Jobs
- ◆ **ParallelRunningSlavesCount** - the number of slave Jobs that are currently running
- ◆ **ParallelSlaveClass** - for Parallel Master Jobs, the class of machine that the Job's Slave Jobs require
- ◆ **PoolReservedClasses** - reserved pool capabilities



- ◆ **PostExec** - The specified post-execution program (if any), as specified with the **--post-exec** switch or in **NB_POST_EXEC**
- ◆ **PostExecExitStatus** - the exit status of the Job's post-execution stage (if any). If there were multiple post-execution stages, this is the exit status of the last one.
- ◆ **PostExecExitStatusList** - list of exit statuses of the Job's post-execution stage (if any). (A Job can have multiple post-execution stages if the user specified one and/or one or more Job profiles also specified one.)
- ◆ **PreExec** - The specified pre-execution program (if any), as specified with the **--pre-exec** switch or in **NB_PRE_EXEC**
- ◆ **PreExecExitStatus** - the exit status of the Job's pre-execution stage (if any). If there were multiple pre-execution stages, this is the exit status of the last one.
- ◆ **PreExecExitStatusList** - list of exit statuses of the Job's pre-execution stage (if any). (A Job can have multiple pre-execution stages if the user specified one and/or one or more Job profiles also specified one.)
- ◆ **PreExecStartTime** - the date/time when the Job's pre-execution script started running
- ◆ **Priority** - The Job's priority, as specified with **--priority**
- ◆ **ProcessID** - The Job's process ID
- ◆ **Qslot*** - The Qslot in which the Job was queued
- ◆ **Queue** - The Queue in which the Job was queued
- ◆ **RM** - the Job's current real memory use in MB
- ◆ **RUsage** - Resource usage:
 - ◆ **wtime** - wall clock time
 - ◆ **utime** - user time
 - ◆ **stime** - system time
 - ◆ **RM** - current real memory usage (in MB)
 - ◆ **VM** - current virtual memory usage (in MB)
 - ◆ **MaxRSS** - maximum resident set size (i.e., the maximum number of pages the Job's processes have had in real memory so far)



- ◆ **MaxVSZ** - maximum swap (virtual memory) size (i.e., the maximum number of pages of virtual memory that the Job's processes have used at any one time so far)
- ◆ **DiskIOReadKB** - (SLES 11 SP2 and above only) the number of kilobytes the Job's processes have read from disk in the current period
- ◆ **DiskIOWriteKB** - (SLES 11 SP2 and above only) the number of kilobytes the Job's processes have written to disk in the current period
- ◆ **avgovertime()** - (only if a Job constraint was specified that includes the **AvgOverTime()** function) - average value of the specified parameter over the specified time
- ◆ **Max(avgovertime())** - (only if a Job constraint was specified that includes the **AvgOverTime()** function) - maximum value of the average value of the specified parameter, and the time at which the maximum occurred (in seconds from the start of execution)
- ◆ **deltaovertime()** - (only if a Job constraint was specified that includes the **DeltaOverTime()** function) - the difference between the maximum and minimum values of the specified parameter during the specified time
- ◆ **Max(deltaovertime())** - (only if a Job constraint was specified that includes the **DeltaOverTime()** function) - maximum value of the delta value of the specified parameter, and the time at which the maximum occurred (in seconds from the start of execution)

This information is updated every three minutes. Netbatch gets this information from `/proc/<pid>/stat` and `/proc/<pid>/status`.

Resource usage information is also available with `--history`.

Note

The size of a page of memory may be different for different operating systems.

- ◆ **RemoteData** - if the Job was sent to a remote Pool for execution (which can happen if the Job was submitted to a Virtual Pool), this field shows the Job details in the remote Pool; that is, with any Qslot and class migration rules applied.



- ◆ **RemotePoolsCandidates** - (physical Pool only) if the Job was submitted to a Virtual Pool, the Physical Pools that the Job can be sent to (as specified by the user with the `--remote-pools` switch)
- ◆ **RemoteServerHostName** - if the `nbstatus` request was submitted to a Virtual Pool, this is the hostname of the Pool Master of the physical Pool that the Job was sent to. If the request was submitted to a physical Pool, this is the hostname of the Workstation that the Job was sent to.
- ◆ **Renters** - the Job(s) that the Job is lending resources to (if any).
- ◆ **RentingCondition** - the Job's renting condition (as specified with the `nbjob run --renting-condition` switch).
- ◆ **RequestedPag** - The Process Authentication Group request (from a `--pag` switch on the Job submission command line and/or in a Job Profile, depending on which takes precedence). This can be:
 - ◆ **on** - Job submission command or Job Profile contains `--pag on`.
 - ◆ **nosuspend** - Job submission command or Job Profile contains `--pag nosuspend`.
 - ◆ **dontcare** - Neither the Job submission command nor the Job Profile contains a `--pag` switch.
- ◆ **RequestedSlotsNumber** - the number of execution slots requested for the Parallel Master Job
- ◆ **ReservationExpirationFormula** - the reservation window formula that applies to this Job
- ◆ **ReservationExpiration.MaxValue** - the maximum permitted reservation window length that applies to this Job (in minutes)
- ◆ **ReservedClasses** - Resources that have been reserved for the Job. If the Job is a Parallel Job and it is the owner of an execution slot, this field indicates the Workstation where the slot is, and that the Job is the owner.
If a reservation of memory or of a custom Workstation capability has been made for the Job (because the Queue or Qslot to which it was submitted is one where such a requirement is applied automatically), this is also shown.
- ◆ **ReservedLicenses** - Reserved licenses



- ◆ **ResourceSet** - The resource set to which the Workstation on which the Job is running belongs
- ◆ **ResubmitThreshold** - The number of minutes that the Job can be suspended before it is resubmitted, as specified with `nbq -r` or `nbjob run --resubmit`
- ◆ **ResubmitTrials** - If `--resubmit` and `--trials` were specified, this is the number of times Netbatch resubmits the Job (if it is suspended for more than the specified time) before it kills it.
- ◆ **Rlimits** - The Job's resource limits, as specified with `--rlimits`
- ◆ **RunawayPredictionData** - full runaway prediction data
- ◆ **RunawayPredictionFields** - list of fields whose values may be changed by runaway prediction and their values
- ◆ **RunawayPredictionMailingList** - list of email addresses of users who will be alerted if the Job is flagged as a predicted runaway Job
- ◆ **RunawayPredictionProfileType** - the ID of the specific prediction profile that was matched by the prediction system (used for debugging the prediction system)
- ◆ **RunawayPredictionResult** - the Job's predicted runaway probabilities
- ◆ **RunawayPredictionStatus** - whether runaway prediction is enabled:
 - ◆ `enable`—runaway prediction is enabled.
 - ◆ `disable`—runaway prediction is disabled.
 - ◆ `dry`—prediction is enabled without affecting scheduling.
- ◆ **RunawayPredictionTimeStamp** - runaway prediction timestamp
- ◆ **RunningClass** - the class that has been assigned to the Job (from those specified as options in the class expression)
- ◆ **RuntimePredictionData** - full runtime prediction data
- ◆ **RuntimePredictionFields** - list of fields whose values may be changed by runtime prediction and their values
- ◆ **RuntimePredictionProfileType** - the ID of the specific prediction profile that was matched by the prediction system (used for debugging the prediction system)
- ◆ **RuntimePredictionResult** - the Job's predicted runtime probabilities



- ◆ **RuntimePredictionStatus** - whether runtime prediction is enabled:
 - ◆ **enable**—runtime prediction is enabled.
 - ◆ **disable**—runtime prediction is disabled.
 - ◆ **dry**—prediction is enabled without affecting scheduling.
- ◆ **RuntimePredictionTimeStamp** - runtime prediction timestamp
- ◆ **SchedulingOrder** - for a waiting Job, this is the Job's position in the scheduling order.
- ◆ **SequentialIterationID** - the Job's unique, sequential ID (for BI purposes). This counter is reset when the Pool is restarted.
- ◆ **sessionId** - internal Netbatch session ID
- ◆ **startTime**** - the date and time when the Job started running
- ◆ **status*** - the Job's current status
 - ◆ **Comp** - the Job has completed (i.e., it ended or was killed).
 - ◆ **Del** - the Job is being deleted. Waiting for confirmation from Workstation.
 - ◆ **Resub** - the Job is being resubmitted.
 - ◆ **Run** - the Job is running.
 - ◆ **Send** - the Job is being dispatched to a Workstation for execution. The Job is also assigned this state if a pre-execution script is running (as specified with the **nbjob run --pre-exec** switch).
 - ◆ **Susp** - the Job is suspended.
 - ◆ **Wait** - the Job is in the wait Queue.
 - ◆ **Wait susp** - the Job is suspended while in the wait Queue. It will not be dispatched until it is resumed.
 - ◆ **Wait virtual** - the Job is in the wait Queue, and was dispatched to this Pool Master from a Virtual Pool.

All statuses (except **Wait**, **Wait susp**, and **Comp**) can be prepended by **Disc**. This means that the Pool Master cannot communicate with the Workstation on which the Job is/was running. The status is the Job's last reported status.

For Jobs submitted to a Virtual Pool, **Disc** can also mean that the PPM is down.



Statuses for Jobs submitted to a Virtual Pool can also have **Remote** appended to them. This means that the Job is in the specified state in the physical Pool. (So **Wait** means the Job is waiting to be sent to a physical Pool, while **Wait Remote** means it has already been sent to the physical Pool and is waiting there.)

- ◆ **Stime** - Job process system time
- ◆ **SubmissionHost** - The host name of the Workstation from which the Job was submitted
- ◆ **SubmitTime** - The date/time when the Job was submitted
- ◆ **SuspendReason** - The reason the Job is suspended:
 - ◆ **Queue Preemption** - Suspended by Queue/Qslot preemption
 - ◆ **Locked Resource (<reason>)** - Suspended because a resource has been locked. (**reason** is the lock reason. If there are multiple locks, this is the reason for the first lock that was applied.)
 - ◆ **Administration[(<reason>)]** - Suspended by **nbjob suspend**. **<reason>** is whatever is specified for the **--reason** switch
 - ◆ **Threshold** - Suspend because a threshold has been exceeded on the Workstation (number of interactive users, load, or free virtual memory)
 - ◆ **Resource Preemption** - Suspended by Resource Set preemption
 - ◆ **Broken Dependency** - Suspended because of a dependency problem

Note

In all these cases except preemption, the amount of resources allocated to the suspended Job do not change.

In the case of preemption, the amount of resources allocated to the suspended Job are reduced to correct an incorrect allocation of resources. Resources are reallocated to the Job once this imbalance has been corrected.

- ◆ **TOOL_NAME** - the value of the **TOOL_NAME** environment variable
- ◆ **Task** - Task name, as specified with **nbq --task-name** or **nbjob run --task** or in the Flow Manager



- ◆ **TemporaryExcludedResources** - list of resources that have been excluded from those that the Job can be sent to (as specified in --on-job-finish)
- ◆ **Terminal** - Whether the Job is a terminal Job (true/false)
- ◆ **TimeInRunning** - Time in running state (in <hours>h:<mins>m:<secs>s format)
- ◆ **TimeInSuspend** - Time the Job has been in suspended state.
To filter on this field, specify the time in milliseconds.
- ◆ **TimeOnMachine** - Time the Job is on executing Workstation (in <hours>h:<mins>m:<secs>s format)
To filter on this field, specify the time in milliseconds.
- ◆ **TimesExitCodePolicyRestarted** - the number of times the Job has been restarted by because the --autoreq or --on-job-finish criteria have been met (see **njob run**).
To filter on this field, specify the time in milliseconds.
- ◆ **TimesPolicyRestarted** - Number of times Job was restarted by policy
- ◆ **TimesRestarted** - Total number of times Job was restarted for any reason
To filter on this field, specify the time in milliseconds.
- ◆ **TriggerJobs** - List of trigger Jobs, as specified with the --triggers switch
- ◆ **UUID** - the Job's Universally Unique Identifier
- ◆ **User* ***** - Job submitter
- ◆ **Utime** - Job process user time
- ◆ **VM** - the Job's current virtual memory use in MB
- ◆ **VirtualPool** - if the Job was submitted to a Virtual Pool and the nbstatus query is made to the physical Pool Master, the name of the Virtual Pool. If the query is made to the Virtual Pool Master, this field is empty.
- ◆ **VirtualResource** - Job virtual resource name (Virtual Pool only)
- ◆ **WSRank** - Workstation rank (used by the Pool master to decide which Workstation to send the next Job to)
- ◆ **WSReservedClasses** - reserved WSM capabilities (similar to **ReservedClasses**, but WSM-specific)



- ◆ **WaitReason** - reason for which the Job cannot be dispatched for execution:
 - ◆ **no resource is available** - No matching Workstation is available to run the Job, or Netbatch has not yet started searching for a matching Workstation.
 - ◆ **qslot inactive** - The Qslot to which the Job was submitted is inactive.
 - ◆ **qslot running policies exceeded** - The Qslot to which the Job was submitted has reached its running Jobs limit.
 - ◆ **qslot running cost policies exceeded** - The Qslot to which the Job was submitted has reached its **max_cost** or **max_user_cost** limit.
 - ◆ **Dependency** - The Job cannot start executing until one or more dependent Jobs finish running.
 - ◆ **reservation is not complete** - Some or all of the resources that the Job reserved are not yet available.
 - ◆ **Suspend** - The Job has been suspended.
 - ◆ **Failed to run pre execution (last failure on machine <machine>)** - The specified pre-execution program failed (on **machine**).
 - ◆ **License_not_available <...>** - One or more licenses that the Job requires is not available.
 - ◆ **Skipped session because similar jobs failed to dispatch (Reason: <reason>, Job id: <job_id>)** - the Job is waiting because a similar Job is already waiting for the same resource (Qslot, class, reservation requirements, or licence).

The Job ID and wait reason of the Job that this one is similar to are shown in parentheses.

- ◆ **Multiple wait reasons exist. See field MultipleWaitReasons** - the Job was submitted to a Virtual Pool or to a Feeder and is waiting in multiple physical Pools, and it does not have the same wait reason in all these Pools. See **MultipleWaitReasons**, above.

Note

If the Job was submitted to a Virtual Pool or Feeder and is waiting in multiple physical Pools,



- ◆ **WashedGroups** - the filesystem groups applied to the Job
- ◆ **Workstation***** - executing Workstation
- ◆ **WorkstationFullHostname** - the fully-qualified hostname of the Workstation that the Job is running on
- ◆ **Wtime** - Job process wall clock time

Fields: jobs --history

The following additional **nbstatus jobs** fields are available when using the **--history** switch:

- ◆ **Cluster** - record cluster name, as specified by **nblog**'s **--cluster** switch
- ◆ **Context** - record test context (for example, Full Chip/Cluster), as specified by **nblog**'s **--context** switch
- ◆ **Cycles** - the number of validation (RTL) cycles for the Job, as specified by **nblog**'s **--cycles** switch
- ◆ **DBTimestamp** - the date and time that this information was written to the NB Tracker database
- ◆ **Egid** - effective group ID
- ◆ **Frequency** - record test frequency (as specified by **nblog**'s **--frequency** switch)
- ◆ **IDRSS** - integral unshared data size
- ◆ **Int1** - custom integer 1, as specified by **nblog**'s **--custom-int-1** switch
- ◆ **Int2** - customer integer 2, as specified by **nblog**'s **--custom-int-2** switch
- ◆ **ISRSS** - integral unshared stack size
- ◆ **IXRSS** - integral shared memory size
- ◆ **LogExecutionUniqueID** - the ID generated by Tracker for the record
- ◆ **MasterCopy** - whether this Job record is from the physical Pool Master (**true**) or the Virtual Pool Master (**false**).

Jobs submitted to a Virtual Pool are recorded twice in NB Tracker - once by the Virtual Pool Master and once by the physical Pool Master.

The physical Pool record usually has more up-to-date resource usage data than the Virtual Pool record.



- ◆ **MaxRSS** - maximum resident set size (i.e., the maximum number of pages the Job's processes had in real memory at any point while it was running)
- ◆ **MaxVMSize** - maximum virtual memory used by the Job in MB
- ◆ **Model** - the chip model that the Job is working on, as specified by nblog's **--model** switch
- ◆ **Multiplicity** - record multiplicity (for example, number of processors), as specified by nblog's **--multiplicity** switch
- ◆ **NbqCommand** - full path of the command used to submit the Job, plus all of its switches
- ◆ **NBWD** - the value of the **NBWD** environment variable
- ◆ **num_of_fs_inputs** - number of times file system performed input
- ◆ **num_of_fs_outputs** - number of times file system performed output
- ◆ **num_of_involuntary_ctx_switch** - number of involuntary context switches
- ◆ **num_of_ipc_message_received** - number of messages received
- ◆ **num_of_ipc_message_sent** - number of messages sent
- ◆ **num_of_mem_swaps** - number of times swapped out
- ◆ **num_of_signal_delivered** - number of signals delivered
- ◆ **num_of_voluntary_ctx_switch** - number of voluntary context switches
- ◆ **page_fault_required_io** - number of page faults (I/O required)
- ◆ **page_fault_without_io** - number of page faults (reclaimed)
- ◆ **Parallel** - if the Job is a Parallel Slave Job, its index
- ◆ **PersonalPoolSubmitTime** - date and time Job was submitted to the Feeder (if applicable)
- ◆ **Pool** - the Pool name (Note that the query results include Jobs from all the Pools that report to the same NB Tracker.)
- ◆ **PostExecFinishTime** - the date and time when the Job's post-execution program ended
- ◆ **ReportingApplicationType** - the type of the reporting application



- ◆ **ReportingInstanceName** - the name of the reporting instance
- ◆ **SimCPU** - CPU time used by the simulator (as specified by **nblog**'s **--simulator-cpu** switch)
- ◆ **Simulator** - record simulator name, as specified by **nblog**'s **--simulator** switch
- ◆ **SimWTime** - wall clock time used by the simulator (as specified by **nblog**'s **--simulator-wtime** switch)
- ◆ **Str1** - custom string 1, as specified by **nblog**'s **--custom-string-1** switch
- ◆ **Str2** - custom string 2, as specified by **nblog**'s **--custom-string-2** switch
- ◆ **Suspend** - total suspend time
- ◆ **Title** - title, as specified by **nblog**'s **--title** switch
- ◆ **TotalJobWTime** - the Job's total wall clock time, including pre- and post-exec stages
- ◆ **TrackerReportSequentialID** - the sequential ID generated on sending to Tracker
- ◆ **UExit** - user exit status, as specified by **nblog**'s **--exit** switch
- ◆ **URReason** - reason for failure, as specified by **nblog**'s **--reason** switch
- ◆ **VirtualIteration** - Job's iteration in the Virtual Pool
- ◆ **VirtualPoolSubmitTime** - date and time Job was submitted to the Virtual Pool (if applicable)
- ◆ **VirtualSessionID** - Job's session ID in the Virtual Pool
- ◆ **Wait** - the amount of time that the Job had to wait after being submitted before starting to run

Note

*When using the **--history** switch the following **nbstatus** jobs fields are **not** available: **Status**, **WaitReason**, **TimeOnMachine**, **RUsage**, **RM**, **VM**, **TimeInRunning**, **LastUpdate**, and **TemporaryExcludedResources**.*



Output

The **nbstatus jobs** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 10, below, shows sample output for the following command:

```
nbstatus jobs "user='yoavf'"
```

Status	Jobid	Class	Qslot	User	Cmdline	Workstation
Send	114	@	/1	yoavf	sleep 100000	itstl106
Wait	94	@	/999	yoavf	./doit	
Wait	95	@	/999	yoavf	./doit	
Wait	96	@	/999	yoavf	./doit	

Figure 10: nbstatus jobs Output

A.3.11 nbstatus keyfiles

Name

```
nbstatus keyfiles
```

Description

Displays information about keyfiles for licenses configured in Netbatch.

By default, **nbstatus keyfiles**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report



You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.

Usage

```
nbstatus keyfiles[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>][--GMT]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

```
--target <pool_name>|<host_name>  
Specifies the Pool Master, either by Pool name or by Pool Master host name. (If --target is omitted, nbstatus uses the Pool defined in the NBPOOL environment variable, or if NBPOOL is undefined, the Pool of which the Workstation where the command was run is a member.)
```

Options

```
--date-format <date_format>  
Specifies the format to be used for dates and times.
```

```
--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all  
Displays only the selected fields. --fields all displays all available fields. Fields are displayed in the specified order.  
See the Fields section, below, for a list of available fields.
```



func specifies a statistical operation that is performed on the field. This affects the output as follows:

- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

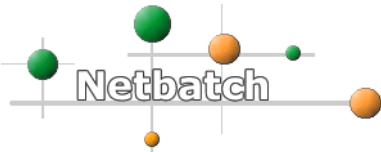
- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that **--group-by** is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example:**--fields "RequestTimeFull-CreatedTimeFull"** This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)



--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--gmt

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.



If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[,...]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.



- ◆ The pattern-matching operator: `=~`. The syntax of the expression that follows this operator is the same as for `glob()`, below. It also matches partial strings. And it works with `--history`. (`glob()` does not.)

Note that with `--history`, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with `--history`, you must use `*` as a wildcard instead of `.*` without `--history`.

Note

`=~` is case-sensitive.

- ◆ A non-strict operator: `is` or `isnt`
- ◆ A bitwise operator: `&`, `|`, `~`, `^`, `<<`, `>>`, or `>>>`
- ◆ A conditional operator: `?` or `:`
- ◆ `value` is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ `MM/DD/YYYY-HH:mm:ss`
 - ◆ `MM/DD/YYYY-HH:mm`
 - ◆ `MM/DD/YYYY`

Expressions can be grouped together using parentheses.

Note

A double equals sign (`==`) is used. `!=` (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If `value` is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If `specification` is `all`, all records are displayed.

Functions can also be used in `specification`. To see which functions are available for this `nbstatus` command, use [nbstatus functions](#). For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:



- ◆ **all** displays all keyfiles.
- ◆ "Status== 'UP'" displays only details of the keyfiles for which the server(s) is up.
- ◆ "Vendor== 'cadence'" only displays details of keyfiles that contain licenses from the vendor called **cadence**.

Fields: keyfiles

The following fields are available (default fields are marked with a *****):

- ◆ **LastLmstatSync** - date and time of last **lmstat** synchronization
- ◆ **Licenses*** - the names of the licenses that are available in this keyfile
- ◆ **Name*** - keyfile path
- ◆ **Port** - the port on which the license server(s) listens
- ◆ **Servers*** - the license server(s) that handle requests for the license listed in the file, and each one's status (**UP** or **DOWN**). There can be either one or three license servers.
- ◆ **Status*** - whether the keyfile is **UP** or **DOWN**. If there are three servers for this keyfile, the keyfile is **DOWN** if two or three of the servers are **DOWN**.
- ◆ **SyncStrategy** - the strategy for this keyfile (**STATIC** or **DYNAMIC**)

Output

The **nbstatus keyfiles** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 14, below,The following figure shows sample output for the following command:

```
nbstatus keyfiles --fields
'Name::15,Status,Servers,Vendors,Licenses'
```



PPM on itstl2022				
Version 7.1.0_0163_00				
On since 12/27/2005 09:40:28				
Time now 12/27/2005 15:09:05				

Name	Status	Servers	Vendors	Licenses
/nfs/iil/disks>	UP	itstl155.iil:UP	SWISS	f3 f2 f1 f4
static_test	UP	static_test_host:UP	SWISS_static	s1 s5 s2

Figure 11: nbstatus keyfiles Output

A.3.12 nbstatus license-allocation

Name

nbstatus license-allocation

Description

Displays allocation information for the specified feature, including the following for each Qslot:

- ◆ Allocation (the maximum number of instances of the feature that Jobs submitted to the Queue/Qslot can reserve at the same time)
- ◆ Number reserved
- ◆ Number checked out

By default, **nbstatus license-allocation**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.



Usage

```
nbstatus license-allocation[,<object>[,...]]  
--context <license>  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>[--GMT]]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

--target <pool_name>|<host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbstatus** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

--date-format "<date_format>"

Specifies the format to be used for dates and times. **date_format** can include any of the following, in any order, separated by valid separator characters (e.g., space, colon (:), slash (/), etc.):

- ◆ **yyyy** - year (four digits)
- ◆ **yy** - year (two digits)
- ◆ **MM** - month
- ◆ **dd** - date
- ◆ **hh** - hour



- ◆ **mm** - minutes
- ◆ **ss** - seconds
- ◆ **sss** - milliseconds

The default format is "**MM/dd/yyyy hh:mm:ss**".

```
--fields "[*,]<field>[:[<func>][:<size>]]  
[,...]"|all
```

Displays only the selected fields. --fields all displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

func specifies a statistical operation that is performed on the field. This affects the output as follows:

- ◆ If --group-by is not used, specifying a **func** gives one line of output.
- ◆ If --group-by is used, this gives one line of output per group.
- ◆ If --group-by is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that --group-by is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.



You can also perform simple calculations on numeric and date fields. For example:`--fields "RequestTimeFull-CreatedTimeFull"` This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--history <time>

Requests historical Job information.

**Note**

In Netbatch 8.1 and later, nbstatus jobs --history only shows Jobs from one Pool. Previously, it showed Jobs from all the Pools that report to the same Tracker.

time specifies the timeframe for the query. It can be one of the following:

- ◆ <int>M - output covers the last int minutes.
- ◆ <int>D - output covers the last int days.
- ◆ <int>H - output covers the last int hours.
- ◆ mm/dd/yyyy[-hh:mm] - output covers the time since the specified date/time.
- ◆ 'mm/dd/yyyy[-hh:mm] mm/dd/yyyy[-hh:mm]' - output covers the specified timeframe.
- ◆ ww<ww>[Y<yyyy>] - output covers the specified workweek. **ww** is the two-digit workweek (e.g., 03). **yyyy** is the four-digit year (e.g., 2005).

With this switch, the set of available fields is not the same as without the switch.

In addition to the fields listed in **--fields**, there are additional fields available when using the **--history** switch. See Fields section below.

When **--group-by** is used with **--history**, numerical fields display average values. Other fields display -.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored.

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.



If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[,...]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

**Note**

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for glob(), below. It also matches partial strings. And it works with --history. (glob() does not.)

Note that with --history, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with --history, you must use * as a wildcard instead of .* without --history.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: is or isnt
- ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
- ◆ A conditional operator: ? or :
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ MM/DD/YYYY-HH:mm:ss
 - ◆ MM/DD/YYYY-HH:mm
 - ◆ MM/DD/YYYY

Expressions can be grouped together using parentheses.

Note

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If value is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.



If **specification** is **all**, all records are displayed.

Functions can also be used in **specification**. To see which functions are available for **nbstatus license-allocation**, use **nbstatus functions** with the **--context "nbstatus license-allocation"** switch. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ **a1** displays information about the specified feature for all Qslots.
- ◆ "**Name=' /a1 '**" displays information about the specified feature for Qslot **a1**.

Fields: license-allocation

The following fields are available (default fields are marked with a *****):

- ◆ **Alias*** - Qslot alias
- ◆ **Dispatched*** - number of instances of the feature that are currently checked out by Jobs that were dispatched from the Queue/Qslot
- ◆ **FullPath** - full Queue/Qslot path
- ◆ **Index** - Qslot index (for Netbatch 5 compatibility)
- ◆ **LastDispatched** - time of last reservation
- ◆ **LastSubmitted** - time last license session submitted to Qslot
- ◆ **Level** - Qslot level (that is, how "deep" in the hierarchy the Qslot is). For example, Level for a Qslot with one "parent" Qslot and one "grandparent" Qslot is 4 (the four levels being the Queue, the grandparent, the parent, and the Qslot itself).
- ◆ **MaxRunning*** - the limit for the number of instances of the feature that can be reserved by Jobs that were dispatched from the Queue/Qslot
- ◆ **Name*** - Queue/Qslot name
- ◆ **Permissions** - users and groups allowed to submit Jobs to the Qslot

This is a space-delimited list of items of the format
**{<user>|<group>}:<type>:{grant | deny} :
{Recursive | nonRecursive}**.



{<user> | <group>} is the user/group to whom the permissions are granted/denied.

- ◆ **Queue** - Queue name
- ◆ **ResourceType** - the name of the feature
- ◆ **Running** - number of instances of the feature that are reserved by Jobs that were dispatched from the Queue/Qslot
- ◆ **status** - feature status (open or closed, active or inactive)
- ◆ **Type** - Qslot type:
 - ◆ **root** - the root of the resource allocation structure
 - ◆ **queue** - Queue
 - ◆ **qslot** - regular Qslot
 - ◆ **user** - User Slot
 - ◆ **/others** - default Qslot
- ◆ **Waiting** - number of waiting license sessions

Output

The **nbstatus license-allocation** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 20, below, shows sample output for the following command:

```
nbstatus license-allocation
```



PPM on itstl008				
Version 7.2.0_0178_03				
On since 10/11/2006 11:46:34				
Time now 10/11/2006 12:06:53				

Name	Alias	MaxReservation	Checkedout	Reserved
/OTHERS			0	0
/a1	10		0	0
/a2	20		0	0
/b			0	0

Figure 12: nbstatus license-allocation Output

A.3.13 nbstatus license-sessions

Name
nbstatus license-sessions

Description

Displays information about license sessions. A license session is created when a Job that requires a license is queued. At this stage, its status is **Defined**. After Netbatch has found a Workstation that can run the Job, it checks whether the required license(s) is available. If it is, it dispatches the Job for execution and changes the status of the session (one per license) to **Reserved**. When the Job checks out a license, the matching session's status changes to **Checkedout**. If the Job checks the license in before the license's requested duration time (which can be the entire run time of the Job) has ended, the session's status returns to **Reserved**. The session's status only changes to **Finished** if the requested duration time has ended.

By default, **nbstatus license-sessions**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report



You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.

Usage

```
nbstatus license-sessions[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>][--GMT]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

```
--target <pool_name>|<host_name>  
Specifies the Pool Master, either by Pool name or by Pool Master host name. (If --target is omitted, nbstatus uses the Pool defined in the NBPOOL environment variable, or if NBPOOL is undefined, the Pool of which the Workstation where the command was run is a member.)
```

Options

```
--date-format <date_format>  
Specifies the format to be used for dates and times.
```

```
--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all  
Displays only the selected fields. --fields all displays all available fields. Fields are displayed in the specified order.
```



See the Fields section, below, for a list of available fields.

func specifies a statistical operation that is performed on the field.

This affects the output as follows:

- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that **--group-by** is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example:**--fields "RequestTimeFull-CreatedTimeFull"** This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)



--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--gmt

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.



If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[,...]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.



- ◆ The pattern-matching operator: `=~`. The syntax of the expression that follows this operator is the same as for `glob()`, below. It also matches partial strings. And it works with `--history`. (`glob()` does not.)

Note that with `--history`, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with `--history`, you must use `*` as a wildcard instead of `.*` without `--history`.

Note

`=~` is case-sensitive.

- ◆ A non-strict operator: `is` or `isnt`
- ◆ A bitwise operator: `&`, `|`, `~`, `^`, `<<`, `>>`, or `>>>`
- ◆ A conditional operator: `?` or `:`
- ◆ `value` is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ `MM/DD/YYYY-HH:mm:ss`
 - ◆ `MM/DD/YYYY-HH:mm`
 - ◆ `MM/DD/YYYY`

Expressions can be grouped together using parentheses.

Note

A double equals sign (`==`) is used. `!=` (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If `value` is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If `specification` is `all`, all records are displayed.

Functions can also be used in `specification`. To see which functions are available for this `nbstatus` command, use [nbstatus functions](#). For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:



- ◆ **all** displays details of all license sessions.

Fields: license-sessions

The following fields are available (default fields are marked with a *):

- ◆ **Ambiguous** - when there is more than one license session for the same license for the same user on the same Workstation, the session is classified as ambiguous.
- ◆ **CheckedInTime** - the date and time that the license was checked in
- ◆ **CheckedOutTime** - the date and time that the license was checked out
- ◆ **Display** - the Job's display (from **lmstat**)
- ◆ **FinishTime** - session finish time
- ◆ **Handle** - **lmstat** handle
- ◆ **Host*** - the hostname of the Workstation to which the Job that requires the license was dispatched
- ◆ **KeyFileName** - path to the keyfile
- ◆ **KeyFileUp** - whether the license server responsible for the license is up or down
- ◆ **LicenseID** - the unique license ID. This has the following format: <vendor>;<keyfile_path>;<feature_name>.
- ◆ **LicenseName** - the name of the license
- ◆ **OwnerID** - full Netbatch Job ID (<pool>.<JobID>), or **External** for an external session
- ◆ **Qslot*** - the Qslot that the session belongs to
- ◆ **Queue** - the Queue that the session belongs to
- ◆ **RequestedLicense*** - the name or alias of the requested license (as specified when the Job was submitted)
- ◆ **ReservationDuration** - the proportion of the Job's run time for which it requires the license. (**infinite** means that it requires the license for the entire duration of its run.)
- ◆ **ReservationTime** - the date and time that the license was reserved
- ◆ **SessionID*** - the session's ID



- ◆ **ShortOwnerID*** - Netbatch Job ID, or **External** for an external session
- ◆ **status*** - session status:
 - ◆ **Defined** - the Job has been submitted with a license requirement
 - ◆ **Reserved** - Netbatch has found available instances for all the Job's license requirements, and has dispatched the Job for execution. A session can return to this status from **CheckedOut** (if the Job checks the license back in before the reservation duration period has ended and continues running).
 - ◆ **CheckedOut** - the Job has checked the license out from the license server.
 - ◆ **Finished** - the reservation duration period has finished for the license that this session applies to, but Job has other license sessions that have not yet ended. (When the Job ends or the session's reservation duration ends, the status of the session is also **Finished**, but because of this, it is no longer displayed.)
- ◆ **statusHistory** - the date and time that the Job was submitted and that the feature was checked out
- ◆ **SubmitTime** - the date and time that the Job that requires the license was submitted
- ◆ **Type** - whether this is an external or internal session:
 - ◆ **Internal** - the license that the Job checked out was specified (with the **--license** switch) when it was submitted.
 - ◆ **External** - the license was not checked out by a Netbatch Job, or the license was checked out by a Netbatch Job, but the license requirement was **not** specified when it was submitted.
- ◆ **User*** - username of the user who submitted the Job that requires the license
- ◆ **Vendor** - the name of the license vendor

Output

The **nbstatus license-sessions** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format



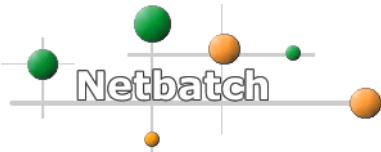
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 20, below, shows sample output for the following command:

```
nbstatus license-sessions
```

PPM on itstl2022					
Version 7.1.0_0163_00					
On since 12/27/2005 09:40:28					
Time now 01/01/2006 14:29:59					
<hr/>					
SessionID	ShortOwnerID	Status	User	Host	RequestedLicense
<hr/>					
3		Defined	martinpx		f1
4		Defined	martinpx		conformal_vhd
589		Defined	martinpx		conformal_vhd
590		Defined	martinpx		conformal_vhd
591		Defined	martinpx		conformal_vhd
598		Defined	martinpx		conformal_vhd
722		Defined	martinpx		conformal_vhd
723		Defined	martinpx		994
743	External	CheckedOut	yhatiel	ics14126	caldesignrev
744	External	CheckedOut	ymto	ito1003	caldesignrev
745	External	CheckedOut	ymto	ito1007	caldesignrev
746	External	CheckedOut	aalmog1	TVYASIC2	msimviewer
749	External	CheckedOut	aalmog1	TVYASIC2	msimhdlsim
759	External	CheckedOut	yshani	ics18219	msimhdlsim
<hr/>					

Figure 13: nbstatus license-sessions Output



A.3.14 nbstatus licenses

Name**nbstatus licenses****Description**

Displays information about physical licenses (that is, specific licenses from specific keyfiles).

By default, **nbstatus licenses**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.

Usage

```
nbstatus licenses[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>[--GMT]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```



You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

--target <pool_name> | <host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbstatus** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

--date-format <date_format>

Specifies the format to be used for dates and times.

**--fields " [* ,]<field>[:[<func>]][:<size>]]
[, . . .]" | all**

Displays only the selected fields. **--fields all** displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

func specifies a statistical operation that is performed on the field. This affects the output as follows:

- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items



- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, `stodstatus areas --fields "owner:uniq"` displays the number of different owners.) Note that `--group-by` is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which `concat` is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example: `--fields "RequestTimeFull-CreatedTimeFull"` This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as `csv`, except that the header row includes additional information about each field
- ◆ **script** - the same as `csv`, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

**--group-by <field_name>[,...]**

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[,...]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).



- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for glob(), below. It also matches partial strings. And it works with --history. (glob() does not.)

Note that with --history, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with --history, you must use * as a wildcard instead of .* without --history.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: is or isnt
- ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
- ◆ A conditional operator: ? or :
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ **MM/DD/YYYY-HH:mm:ss**



- ◆ **MM/DD/YYYY-HH:mm**
- ◆ **MM/DD/YYYY**

Expressions can be grouped together using parentheses.

Note

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If value is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If specification is all, all records are displayed.

Functions can also be used in specification. To see which functions are available for this nbstatus command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

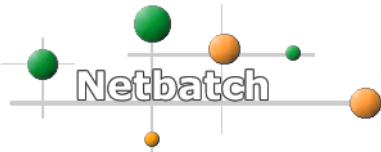
For example:

- ◆ **all** displays all licenses.
- ◆ **"Feature=='Allegro_performance'"** displays only details of the feature called Allegro_performance.
- ◆ **"Alias=='x1'"** only displays details of the feature whose alias is x1.
- ◆ **"Inuse<Capacity"** displays only features for which the number of licenses available for Netbatch Jobs is greater than the number of licenses currently used by Netbatch Jobs.
- ◆ **"Inuse<(Capacity*Riskfactor)"** displays only features for which the number of Netbatch Jobs that require the feature that are permitted to run at the same time is greater than the number of licenses currently used by Netbatch Jobs.

Fields: licenses

The following fields are available (default fields are marked with a *):

- ◆ **ActualCapacity** - the total number of instances of this license available on the license server(s)
- ◆ **Alias*** - the license alias (if any)



- ◆ **Available*** - the number of instances of the license currently available to Netbatch Jobs
- ◆ **CheckedOut*** - the total number of instances of this license that have been checked out from the license server
- ◆ **ConfiguredCapacity** - the total number of instances of this license available to Netbatch Jobs
- ◆ **Cost** - configured cost for this feature
- ◆ **ExpirationDate** - the license expiration date
- ◆ **Factor** - the number of Jobs that can reserve the license for each available instance of the license.

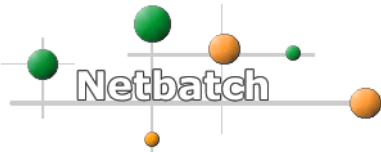
For example, if it is estimated that on average, Jobs only use the license for half of the time that they are running, **Factor** is set to 2.

- ◆ **KeyFile** - the license keyfile
- ◆ **LicenseName*** - the license name
- ◆ **Margin** - the number of instances of a license that are reserved for non-Netbatch use
- ◆ **Reserved*** - the number of Netbatch Jobs for which instances of the license have been reserved but not yet checked out
- ◆ **ServiceCheckedout*** - the total number of instances of this license that have been checked out by Netbatch Jobs
- ◆ **Sessions** - the IDs of the Netbatch license sessions currently using this license
- ◆ **Status** - whether the license server is UP or DOWN
- ◆ **SyncStrategy** - how Netbatch calculates the feature's availability. It can be:
 - ◆ **static**—a simple counter is used to track the number of available instances of the feature.
 - ◆ **dynamic**—the license server is queried to check if a feature is available.
- ◆ **Vendor** - the name of the license vendor

Output

The **nbstatus licenses** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format



- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 14, below, shows sample output for the following command:

```
nbstatus licenses --fields
"LicenseName,Capacity,ConfiguredCapacity,
Available,Alias" "available>0"
```

LicenseName	Capacity	ConfiguredCapacity	Available	Alias
945	80	87	80	null
994	82	89	82	null
Fire_Parallel	150	150	150	null
Ice_Parallel	100	100	100	null
LEAPFROG-CV	85	90	85	null
UET	113	119	113	null
msimcdebug	600	100	100	null
msimcompare	600	100	100	null
msimcoverage	600	100	100	null
msimdataflow	600	100	100	null
msimhdlcom	600	100	100	null
msimhdlsim	600	100	100	null
msimprofile	600	100	100	null
msimviewer	600	100	100	null
plotversa	332	344	332	null

Figure 14: nbstatus licenses Output



A.3.15 nbstatus locks

Name

nbstatus locks

Description

Displays current and scheduled Workstation locks.

By default, **nbstatus locks**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.

Usage

```
nbstatus locks[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>][--GMT]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```



You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

--target <pool_name> | <host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbstatus** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

--date-format <date_format>

Specifies the format to be used for dates and times.

**--fields " [* ,]<field>[:[<func>]][:<size>]]
[, . . .]" | all**

Displays only the selected fields. **--fields all** displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

func specifies a statistical operation that is performed on the field. This affects the output as follows:

- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items



- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, `stodstatus areas --fields "owner:uniq"` displays the number of different owners.) Note that `--group-by` is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which `concat` is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example: `--fields "RequestTimeFull-CreatedTimeFull"` This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as `csv`, except that the header row includes additional information about each field
- ◆ **script** - the same as `csv`, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

**--group-by <field_name>[,...]**

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[,...]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).



- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for glob(), below. It also matches partial strings. And it works with --history. (glob() does not.)

Note that with --history, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with --history, you must use * as a wildcard instead of .* without --history.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: is or isnt
- ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
- ◆ A conditional operator: ? or :
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ **MM/DD/YYYY-HH:mm:ss**



- ◆ **MM/DD/YYYY-HH:mm**
- ◆ **MM/DD/YYYY**

Expressions can be grouped together using parentheses.

Note

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If value is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If specification is all, all records are displayed.

Functions can also be used in specification. To see which functions are available for this nbstatus command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ **all** displays details of all locks.
- ◆ **"Server=='inbm006'"** displays details of the current and scheduled locks for the Workstation whose hostname is **inbm006**.

Fields: locks

The following fields are available (default fields are marked with a *):

- ◆ **ActionOnLock** - What happens to running Jobs when the lock is applied:
 - ◆ **resubmit**
 - ◆ **remove**
 - ◆ **suspend**
- ◆ **Active*** - Whether the lock is currently active (**true** or **false**)
- ◆ **Created*** - When the lock was created
- ◆ **End*** - Lock end time



- ◆ **LockID*** - Lock ID
- ◆ **Name*** - The name of the locked object (Workstation hostname)
- ◆ **Owner*** - Lock owner
- ◆ **Parameters** - Other parameters
- ◆ **Periodic*** - Whether the lock is periodic (**true** or **false**)
- ◆ **PeriodicDays** - The days on which the lock is applied (periodic only)
- ◆ **PeriodicEnd** - Periodic lock end time
- ◆ **PeriodicStart** - Periodic lock start time
- ◆ **Reason** - The reason for the lock
- ◆ **Start*** - Lock start time
- ◆ **Type*** - Locked object type

Output

The **nbstatus locks** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

The following figure shows sample output for the following command:

```
nbstatus locks "server=='itstl126'"
```

Name	Type	Owner	LockID	Periodic	Active	Created	Start	End
<hr/>								
itstl126	machine	abarrapp	1	false	true	10/27/2004	14:35:14	
<hr/>								

Figure 15: nbstatus locks Output



A.3.16 nbstatus logical-licenses

Name**nbstatus logical-licenses****Description**

Displays information about logical licenses (that is, for all available instances of each license, no matter how many different keyfiles they are listed in).

By default, **nbstatus logical-licenses**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.

Usage

```
nbstatus logical-licenses[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>[--GMT]]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```



You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

--target <pool_name> | <host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbstatus** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

--date-format <date_format>

Specifies the format to be used for dates and times.

**--fields " [* ,]<field>[:[<func>]][:<size>]]
[, . . .]" | all**

Displays only the selected fields. **--fields all** displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

func specifies a statistical operation that is performed on the field. This affects the output as follows:

- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items



- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, `stodstatus areas --fields "owner:uniq"` displays the number of different owners.) Note that `--group-by` is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which `concat` is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example: `--fields "RequestTimeFull-CreatedTimeFull"` This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as `csv`, except that the header row includes additional information about each field
- ◆ **script** - the same as `csv`, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

**--group-by <field_name>[,...]**

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[,...]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).



- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for glob(), below. It also matches partial strings. And it works with --history. (glob() does not.)

Note that with --history, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with --history, you must use * as a wildcard instead of .* without --history.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: is or isnt
- ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
- ◆ A conditional operator: ? or :
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ **MM/DD/YYYY-HH:mm:ss**



- ◆ **MM/DD/YYYY-HH:mm**
- ◆ **MM/DD/YYYY**

Expressions can be grouped together using parentheses.

Note

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If value is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If specification is all, all records are displayed.

*Functions can also be used in specification. To see which functions are available for this nbstatus command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).*

For example:

- ◆ **all** displays details of all logical licenses.
- ◆ **"available>0"** displays only those logical licenses that Netbatch Jobs currently available to Netbatch Jobs.

Fields: logical-licenses

The following fields are available (default fields are marked with a *):

- ◆ **ActualCapacity*** - the total number of instances of this license available on the license server(s)
- ◆ **Allocatable** - whether the license can be allocated to Queues and Qslots
- ◆ **Available*** - the number of instances of the license currently available to Netbatch Jobs (for details of how Netbatch calculates availability, see [Section 18.4, How Netbatch Dispatches Jobs that Require Licenses](#))
- ◆ **CheckedOut*** - the total number of instances of this license that have been checked out from the license server(s)
- ◆ **ConfiguredCapacity*** - the total number of instances of this license available to Netbatch Jobs



- ◆ **KeyFiles** - the keyfiles in which this license appears
- ◆ **LicenseName*** - the license name
- ◆ **LicenseSessions** - the IDs of the Netbatch license sessions that are using this logical license
- ◆ **PhysicalLicenses** - the physical licenses (that is, licenses with the same name and vendor from different keyfiles) that make up this logical license
- ◆ **Reserved*** - the number of instances of the license that have been reserved but not yet checked out
- ◆ **ServiceCheckedOut*** - the total number of instances of this license that have been checked out by Netbatch Jobs
- ◆ **Vendor*** - the name of the license vendor

Output

The **nbstatus logical-licenses** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 20, below, shows sample output for the following command:

```
nbstatus logical-licenses --fields
'LicenseName, Vendor, ActualCapacity, Reserved, CheckedOut'
```



PPM on itstl2022				
Version 7.1.0_0163_00				
On since 12/27/2005 09:40:28				
Time now 12/29/2005 09:07:47				
<hr/>				
LicenseName	Vendor	Capacity	Reserved	Checkedout
<hr/>				
f1	SWISS	1	0	0
f2	SWISS	2	0	0
f3	SWISS	3	0	0
f4	SWISS	4	0	0
s1	SWISS_static	1	0	0
s2	SWISS_static	2	0	0
s5	SWISS_static	5	0	0
<hr/>				

Figure 16: nbstatus logical-licenses Output

A.3.17 nbstatus monitor-actions

Name

nbstatus monitor-actions

Description

Displays information about actions taken by the Job failure detection system. You can use **nbadmin monitor-actions** to reverse these actions.

By default, **nbstatus monitor-actions**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.



Usage

```
nbstatus monitor-actions[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>][--GMT]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

```
--target <pool_name>|<host_name>
```

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If `--target` is omitted, `nbstatus` uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

```
--date-format <date_format>
```

Specifies the format to be used for dates and times.

```
--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all
```

Displays only the selected fields. `--fields all` displays all available fields. Fields are displayed in the specified order.
See the Fields section, below, for a list of available fields.
`func` specifies a statistical operation that is performed on the field.
This affects the output as follows:



- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that **--group-by** is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example: **--fields "RequestTimeFull-CreatedTimeFull"** This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:



- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

**--sort-by [-]<field_name>[, . . .]**

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for **glob()**, below. It also matches partial strings. And it works with **--history**. (**glob()** does not.)



Note that with **--history**, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with **--history**, you must use * as a wildcard instead of .* without **--history**.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: **is** or **isnt**
- ◆ A bitwise operator: **&**, **|**, **~**, **^**, **<<**, **>>**, or **>>>**
- ◆ A conditional operator: **?** or **:**
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ **MM/DD/YYYY-HH:mm:ss**
 - ◆ **MM/DD/YYYY-HH:mm**
 - ◆ **MM/DD/YYYY**

Expressions can be grouped together using parentheses.

Note

A double equals sign (**==**) is used. **!=** (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If **value** is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If **specification** is **all**, all records are displayed.

Functions can also be used in **specification**. To see which functions are available for this **nbstatus** command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ **all** displays details of all actions.
- ◆ "**User=='daves'**" displays details of actions that were taken on user **daves**.



- ◆ "Qslot=='avl'" displays details of actions that were taken for Qslot **avl**.

Fields: monitor-actions

The following fields are available (default fields are marked with a *):

- ◆ **ActionId*** - the ID of the action
- ◆ **ActionType*** - the action type:
 - ◆ **LOCK_USER_QSLOT** - Netbatch locked the Qslot to which the defective Jobs were sent for the user who owns the defective Jobs.
 - ◆ **SUSPEND_USER_QSLOT** - Netbatch suspended all the Jobs belonging to the user in the Qslot to which the defective Jobs were sent
- ◆ **ExecutionTime*** - the date and time when the action was taken
- ◆ **Expiration*** - the expiration date and time of the action (after which the lock is automatically removed)
- ◆ **NumberOfFailure** - the number of defective Jobs that were detected in the configured interval
- ◆ **Qslot*** - the Qslot to which the defective Jobs were submitted
- ◆ **Queue*** - the Queue that contains the Qslot to which the defective Jobs were submitted
- ◆ **Threshold** - how often a defective Job must be detected within a particular time interval for a particular user in the same Qslot before Netbatch takes the specified action
- ◆ **User*** - the owner of the defective Jobs

Output

The **nbstatus monitor-actions** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results



Figure 20, below, shows sample output for the following command:

```
nbstatus monitor-actions --fields  
"ActionId, ActionType, User, Expiration"
```

```
PPM on itstl155  
Version 7.1.1_0173_01  
On since 04/16/2006 15:43:22  
Time now 04/16/2006 15:44:32  
-----  
ActionId  ActionType          User   Expiration  
-----  
0         LOCK_USER_QSLOT    adinur 04/16/2006 15:49:09  
1         SUSPEND_USER_QSLOT  adinur 04/16/2006 15:47:09  
-----
```

Figure 17: nbstatus monitor-actions Output

A.3.18 nbstatus permissions

Name

nbstatus permissions

Description

Displays information about configured permissions.

Note

*This command does not display all Workstation permissions. It only displays **explicitly configured** Workstation permissions.*

Note

When a remote Workstation comes up, its permissions are not listed immediately—it takes a few minutes for them to appear.

Similarly, when a remote Workstation goes down, its permissions are still listed for some time.

By default, **nbstatus permissions**:

- ◆ Displays a default set of fields (a subset of all available fields)



- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.

Usage

```
nbstatus permissions[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>[--GMT]]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

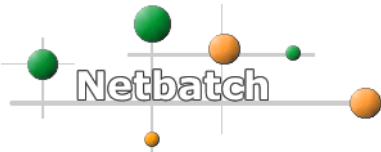
`--target <pool_name>|<host_name>`

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If `--target` is omitted, `nbstatus` uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

`--date-format <date_format>`

Specifies the format to be used for dates and times.



--fields "[*,]<field>[:<func>][:<size>]]
[,...]"|all

Displays only the selected fields. --fields all displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

func specifies a statistical operation that is performed on the field. This affects the output as follows:

- ◆ If --group-by is not used, specifying a func gives one line of output.
- ◆ If --group-by is used, this gives one line of output per group.
- ◆ If --group-by is used without func, sum is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that --group-by is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which concat is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example:--fields "RequestTimeFull-CreatedTimeFull" This calculates the difference between these two date fields and displays the result (in seconds).

**Note**

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

**--retry-timeout <time>{h|m}**

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[, . . .]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ♦ **attribute** is a valid field (see **--fields**, above).
- ♦ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ♦ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)



- ◆ A logical operator: `&&`, `||`, or `!`
- ◆ A comparison operator: `<`, `<=`, `>`, `>=`, `!=`, or `==`

Note

For strings, `==` is not case-sensitive. If you need to do a case-sensitive comparison, use the `equals()` function instead.

- ◆ The pattern-matching operator: `=~`. The syntax of the expression that follows this operator is the same as for `glob()`, below. It also matches partial strings. And it works with `--history`. (`glob()` does not.)

Note that with `--history`, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with `--history`, you must use `*` as a wildcard instead of `.*` without `--history`.

Note

`=~` is case-sensitive.

- ◆ A non-strict operator: `is` or `isnt`
- ◆ A bitwise operator: `&`, `|`, `~`, `^`, `<<`, `>>`, or `>>>`
- ◆ A conditional operator: `?` or `:`
- ◆ `value` is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ `MM/DD/YYYY-HH:mm:ss`
 - ◆ `MM/DD/YYYY-HH:mm`
 - ◆ `MM/DD/YYYY`

Expressions can be grouped together using parentheses.

Note

A double equals sign (`==`) is used. `!=` (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If `value` is a string, it must be enclosed in single quotes.



Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If **specification** is **all**, all records are displayed.

Functions can also be used in **specification**. To see which functions are available for this **nbstatus** command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ **all** displays details of all permissions.
- ◆ "**EntityType== 'Pool'**" displays only permissions configured for the Pool (and not for any other type of entity).
- ◆ "**PermissionType== 'JobSubmit'**" displays only configured Job submission permissions (grant or deny).

Fields: permissions

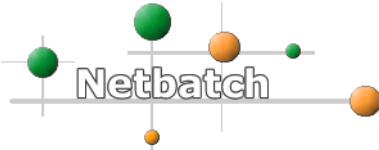
The following fields are available (default fields are marked with a *):

- ◆ **Access*** - whether the permissions are granted (**grant**) or denied (**deny**)
- ◆ **EntityFullName** - The full entity name (e.g., for a Qslot, the full Qslot path including the Queue name)
- ◆ **EntityName*** - The name of the entity
- ◆ **EntityType*** - The entity type (**Pool**, **Queue**, **Qslot**, **Service**, **Workstation**, or **ResourceSet**)
- ◆ **PermissionType*** - The permission type, which is one of the following:
 - ◆ **JobSubmit** - submit Jobs to the Queue/Qslot.
 - ◆ **JobModify** - modify any Job that was submitted to the Queue/Qslot.
 - ◆ **JobRemove** - remove any Job that was submitted to the Queue/Qslot.
 - ◆ **JobAdmin** - perform any Job operation on any Job in the Queue/Qslot.
 - ◆ **QueueOpen** - open the Queue/Qslot.
 - ◆ **QueueClose** - close the Queue/Qslot.
 - ◆ **QueueActivate** - make the Queue/Qslot active.



- ◆ **QueueInactivate** - make the Queue/Qslot inactive.
- ◆ **QueueUserActivate** - make the Queue/Qslot active for a particular user.
- ◆ **QueueUserInactivate** - make the Queue/Qslot inactive for a particular user.
- ◆ **QueueAdmin** - all Queue/Qslot and Job operations
- ◆ **wsstop**—stop the Workstation
- ◆ **WSHup**—hup the Workstation
- ◆ **WSClearBH**—clear a blackhole on the Workstation
- ◆ **wssetLog**—set the logging level for the Workstation
- ◆ **wssyncJobs**—synchronize Jobs on the Workstation
- ◆ **WSSync**—synchronize the Workstation
- ◆ **WSUpgrade**—upgrade the Workstation
- ◆ **WSAdmin**—perform any of the above Workstation actions
- ◆ **RSLock**—lock the Resource Set or any of its Workstations
- ◆ **RSUnlock**—unlock the Resource Set or any of its Workstations
- ◆ **ServiceStart**—start a Pool Master service.
- ◆ **ServiceStop**—stop a Pool Master service.
- ◆ **ServiceHup**—hup a Pool Master service.
- ◆ **ServiceAdmin**—start, stop, or hup a Pool Master service.
- ◆ **PoolSetLog** - set the logging level for the Pool Master.
- ◆ **ReportScheduling** - view scheduler details.
- ◆ **SyncJobs** - synchronize Jobs at the Pool level.
- ◆ **Lock** - lock the Pool.
- ◆ **Unlock** - unlock the Pool.
- ◆ **PoolAdmin** - perform any of the above operations
- ◆ **Recursive*** - (Queues, Qslots, and Pool only) whether the permissions apply to the specified Queue/Qslot only (**false**) or to the specified Queue/Qslot and all the Qslots under it (**true**).

Permissions for the Pool are recursive, as they apply to all the entities in the Pool, including Queues and Qslots.



- ◆ **ServiceName** - one of the following:
 - ◆ For the Pool, Queues, and Qslots - the Pool name
 - ◆ For Pool Master Service - the full hostname of the machine on which the service is running
 - ◆ For a Workstation - the Workstation hostname
- ◆ **UserGroup*** - The user or group to which the permissions are granted or denied (* means all users.)

Output

The **nbstatus permissions** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 20, below, shows sample output for the following command:

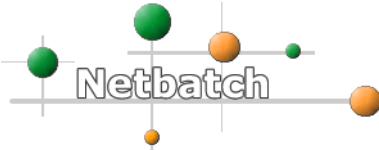
nbstatus permissions



AuthorizationService on itstl2022
Version 7.1.0_0163_00
On since 12/27/2005 09:40:28
Time now 12/27/2005 10:23:21

EntityType	EntityName	UserGroup	PermissionType	Access	Recursive
Pool	m2022	martinpx	PoolSetLog	grant	true
Pool	m2022	nbadmin	PoolAdmin	grant	true
Pool	m2022	nbdevdefault	PoolSetLog	grant	true
Pool	m2022	nbdevdefault	PoolThreadDump	grant	true
Pool	m2022	nbdevdefault	ReportSchedule	grant	true
Pool	m2022	nbdevdefault	SyncJobs	grant	true
Pool	m2022	nbdevdefault	WSSetLog	grant	true
Pool	m2022	nbdevdefault	WSyncJobs	grant	true
Qslot	/1	*	JobSubmit	grant	false
Qslot	/a1	*	JobSubmit	grant	false
Queue	a	martinpx	QueueAdmin	grant	false
ResourceSet	m2022	martinpx	RSLock	grant	false
ResourceSet	m2022	martinpx	RSUnlock	grant	false
ResourceSet	m2022	netbatch	RSLock	grant	false
ResourceSet	m2022	netbatch	RSUnlock	grant	false
Service	Authorization	martinpx	ServiceAdmin	grant	false
Service	GlobalResources	martinpx	ServiceAdmin	grant	false
Service	Groups	martinpx	ServiceAdmin	grant	false
Service	Licensing	martinpx	ServiceAdmin	grant	false
Service	LoadBalancer	martinpx	ServiceAdmin	grant	false
Service	MatbatchLogFile	martinpx	ServiceAdmin	grant	false
Service	NetbatchConfig	martinpx	ServiceAdmin	grant	false
Service	RegistrationSe	martinpx	ServiceAdmin	grant	false
Service	RequestService	martinpx	ServiceAdmin	grant	false
Service	ResourceAllocat	martinpx	ServiceAdmin	grant	false
Service	ResourcesDIREC	martinpx	ServiceAdmin	grant	false
Service	ResourcesUpdate	martinpx	ServiceAdmin	grant	false
Service	Scheduler	martinpx	ServiceAdmin	grant	false
Service	SessionDirecto	martinpx	ServiceAdmin	grant	false
Service	Sessionsupdate	martinpx	ServiceAdmin	grant	false

Figure 18: nbstatus permissions Output



A.3.19 nbstatus policies

Name

nbstatus policies

Description

Displays information about policies. This saves having to look in the configuration file or query the Workstation directly.

By default, **nbstatus policies**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.

Usage

```
nbstatus policies[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>[--GMT]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```



You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

--target <pool_name> | <host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbstatus** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

--date-format <date_format>

Specifies the format to be used for dates and times.

**--fields " [* ,]<field>[:[<func>]][:<size>]]
[, . . .]" | all**

Displays only the selected fields. **--fields all** displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

func specifies a statistical operation that is performed on the field. This affects the output as follows:

- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items



- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, `stodstatus areas --fields "owner:uniq"` displays the number of different owners.) Note that `--group-by` is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which `concat` is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example: `--fields "RequestTimeFull-CreatedTimeFull"` This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as `csv`, except that the header row includes additional information about each field
- ◆ **script** - the same as `csv`, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

**--group-by <field_name>[,...]**

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[,...]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).



- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for glob(), below. It also matches partial strings. And it works with --history. (glob() does not.)

Note that with --history, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with --history, you must use * as a wildcard instead of .* without --history.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: is or isnt
- ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
- ◆ A conditional operator: ? or :
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ **MM/DD/YYYY-HH:mm:ss**



- ◆ **MM/DD/YYYY-HH:mm**
- ◆ **MM/DD/YYYY**

Expressions can be grouped together using parentheses.

Note

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If value is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If specification is all, all records are displayed.

Functions can also be used in specification. To see which functions are available for this nbstatus command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

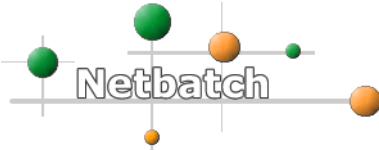
For example:

- ◆ **all** displays details of all policies.
- ◆ **"Name=='P1'"** displays only details of the policy called P1.
- ◆ **"ActOnId=='daves.1271'"** displays only the policies that apply to the specified task.

Fields: policies

The following fields are available (default fields are marked with a *):

- ◆ **ActOn** - the policy target type (Job, task, feeder, WSM, etc.)
- ◆ **ActOnId*** - the policy target ID
- ◆ **Condition*** - the condition that is evaluated to decide which action(s) should be taken
- ◆ **Gatekeeper** - whether the policy is a gatekeeper or not (i.e., whether the WSM auto-tuning service (if enabled) should be allowed to disable the policy or not)
- ◆ **IfFalse** - the action(s) that are taken if the policy's condition is false



- ◆ **IfTrue** - the action(s) that are taken if the policy's condition is true
- ◆ **PolicyName*** - the name of the policy
- ◆ **Schedule** - the policy's start date/time
- ◆ **Time** - days/times at which the policy is active

Output

The **nbstatus policies** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 20, below, shows sample output for the following command:

```
nbstatus policies
```

PPM on inbm002		
Version 7.5.0_0210_00		
on since 01/14/2009 09:37:13		
Time now 02/16/2009 13:57:25		

Name	Target	Condition

PartitionFull	itstl503	fds("/netbatch")<0.>

Figure 19: nbstatus policies Output



A.3.20 nbstatus pools

Name

nbstatus pools

Description

Displays information about the Netbatch Pools.

By default, **nbstatus pools**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.

Usage

```
nbstatus pools[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>][--GMT]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```



You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

--target <pool_name> | <host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbstatus** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

--date-format <date_format>

Specifies the format to be used for dates and times.

**--fields " [* ,]<field>[:[<func>]][:<size>]]
[, . . .]" | all**

Displays only the selected fields. **--fields all** displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

func specifies a statistical operation that is performed on the field. This affects the output as follows:

- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items



- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, `stodstatus areas --fields "owner:uniq"` displays the number of different owners.) Note that `--group-by` is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which `concat` is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example: `--fields "RequestTimeFull-CreatedTimeFull"` This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as `csv`, except that the header row includes additional information about each field
- ◆ **script** - the same as `csv`, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

**--group-by <field_name>[,...]**

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[,...]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).



- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for glob(), below. It also matches partial strings. And it works with --history. (glob() does not.)

Note that with --history, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with --history, you must use * as a wildcard instead of .* without --history.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: is or isnt
- ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
- ◆ A conditional operator: ? or :
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ **MM/DD/YYYY-HH:mm:ss**



- ◆ **MM/DD/YYYY-HH:mm**
- ◆ **MM/DD/YYYY**

Expressions can be grouped together using parentheses.

Note

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If value is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If specification is all, all records are displayed.

*Functions can also be used in specification. To see which functions are available for this nbstatus command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).*

For example:

- ◆ **all** displays details of all Pools.
- ◆ **"Name=='linux_idc'"** displays only details of the Pool called **linux_idc**.
- ◆ **"#Machines>=100"** only displays details of Pools with 100 or more Workstations.
- ◆ **"Host=='inbm006'"** displays only the Pool whose Pool Master machine is **inbm006**.
- ◆ **"Version=='6.4'&&#Machines>=100"** displays Pools that are running version 6.4 of Netbatch with 100 or more Workstations.

Fields: pools

The following fields are available (default fields are marked with a *****):

- ◆ **AdminHosts** - Admin hosts (i.e., hosts that are allowed to perform administrative operations in the Pool)
- ◆ **AdminMail** - The administrator email address
- ◆ **Comment** - Pool description (if defined)



- ◆ **EcdwOptions** - no longer used
- ◆ **GlobalPools** - not used
- ◆ **Host*** - Pool master host name
- ◆ **IP** - The Pool Master's IP address
- ◆ **LastDispatch** - Time last Job dispatched
- ◆ **LastSubmit** - Time last Job submitted
- ◆ **LastUpdate*** - Time the Pool last sent a status update
- ◆ **MachinesOS** - types of machines in the Pool (OSes)
- ◆ **MatbatchLogFile** - The location of the Matbatch log file (if different than the default)
- ◆ **NISDomain** - The NIS domain that the Pool belongs to
- ◆ **Name*** - Pool name
- ◆ **NasServer** - The hostname of the NB Tracker to which the Pool reports
- ◆ **NumOfCores** - number of processor cores
- ◆ **NumOfMachines*** - Number of Workstations in the Pool
- ◆ **NumOfQslots** - Number of Qslots in the Pool
- ◆ **NumOfQueues** - Number of Queues in the Pool
- ◆ **NumOfResourceSets** - Number of Resource Sets in the Pool
- ◆ **NumOfUniqMachines** - number of unique machines
- ◆ **Owner** - Pool owner (if defined)
- ◆ **Patches** - Patches that have been applied
- ◆ **PoolMasterJava** - the version of Java that the Pool master is running
- ◆ **PoolMasterOS** - the OS of the Pool master
- ◆ **PoolRole** - Pool role (if defined)
- ◆ **PoolType** - Pool type (if defined)
- ◆ **Queues** - List of Queues in the Pool
- ◆ **RegID** - Registration ID (When a Pool registers, it is assigned an ID. This allows us to keep track of a Pool even if its name changes.)
- ◆ **ResourceSets** - List of Resource Sets in the Pool
- ◆ **StartTime*** - The date/time that the Pool was started



- ◆ **TrustedHosts** - Trusted hosts (i.e., hosts that are allowed to submit Jobs to the Pool)
- ◆ **Version*** - Version of Netbatch that the Pool is running
- ◆ **virtualPool** - whether the Pool is a virtual Pool or not

Output

The **nbstatus pools** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 20, below, shows sample output for the following command:

```
nbstatus pools --fields name,host,version
```

Name	Host	Version
CSE_HP	pg1nbm03.png.intel.com	6.3.1_0112_13

Figure 20: nbstatus pools Output

A.3.21 nbstatus probes

Name

```
nbstatus probes
```

Description

Displays information about configured scheduler probes. A probe is a generic tool that lets you define a resource that can be monitored using existing tools.

Note

*For information about Workstation probes, use **nbstatus workstations** (see [Appendix A.3.31](#)).*



By default, **nbstatus probes**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.

Usage

```
nbstatus probes[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>[--GMT]]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., **jobs.Status**).

Target

--target <pool_name>|<host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbstatus** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)



Options

--date-format <date_format>

Specifies the format to be used for dates and times.

--fields "[*,]<field>[:[<func>][:<size>]] [, . . .]" | all

Displays only the selected fields. **--fields all** displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

func specifies a statistical operation that is performed on the field. This affects the output as follows:

- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that **--group-by** is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example:**--fields "RequestTimeFull-CreatedTimeFull"** This calculates the difference between these two date fields and displays the result (in seconds).

**Note**

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

**--retry-timeout <time>{h|m}**

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[, . . .]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ♦ **attribute** is a valid field (see **--fields**, above).
- ♦ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ♦ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)



- ◆ A logical operator: `&&`, `||`, or `!`
- ◆ A comparison operator: `<`, `<=`, `>`, `>=`, `!=`, or `==`

Note

For strings, `==` is not case-sensitive. If you need to do a case-sensitive comparison, use the `equals()` function instead.

- ◆ The pattern-matching operator: `=~`. The syntax of the expression that follows this operator is the same as for `glob()`, below. It also matches partial strings. And it works with `--history`. (`glob()` does not.)

Note that with `--history`, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with `--history`, you must use `*` as a wildcard instead of `.*` without `--history`.

Note

`=~` is case-sensitive.

- ◆ A non-strict operator: `is` or `isnt`
- ◆ A bitwise operator: `&`, `|`, `~`, `^`, `<<`, `>>`, or `>>>`
- ◆ A conditional operator: `?` or `:`
- ◆ `value` is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ `MM/DD/YYYY-HH:mm:ss`
 - ◆ `MM/DD/YYYY-HH:mm`
 - ◆ `MM/DD/YYYY`

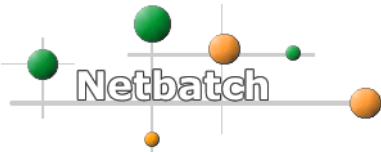
Expressions can be grouped together using parentheses.

Note

A double equals sign (`==`) is used. `!=` (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If `value` is a string, it must be enclosed in single quotes.



Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If **specification** is **all**, all records are displayed.

Functions can also be used in **specification**. To see which functions are available for this **nbstatus** command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ **all** displays details of all probes.
- ◆ "**ProbeName== 'wanlatency'**" displays details of the probe called **wanlatency**.

Fields: probes

The following fields are available (default fields are marked with a *****):

- ◆ **Command***—the script that the probe runs
- ◆ **Data***—the data output by the script

This has the following format:

```
( <probe_name>_<param_name>=<param_val> ,  
limit={<limit_param_name>|<limit_value>} )[ ,  
... ]
```

where:

- ◆ **probe_name** is the name of the probe.
- ◆ **param_name** is a parameter name.
- ◆ **param_val** is the parameter's value.
- ◆ **limit_param_name** is the corresponding limit parameter (if any).
- ◆ **limit_value** is the corresponding limit value (if any)
- ◆ **Interval***—how often the script is run (in minutes)
- ◆ **ProbeName***—the name of the probe
- ◆ **Timeout***—the timeout for the script (in seconds)



Output

The **nbstatus probes** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 22, below, shows sample output for the following command:

```
nbstatus probes --target itstl139
--format block
```

```
PPM on itstl139
Version 7.1.0_0156_00
On since 10/26/2005 09:53:13
Time now 10/26/2005 12:47:20
{
    ProbeName = wan
    Command = /nfs/iil/iec/sws/work/martinpx/wanlatency
    Interval(min) = 1
    Timeout(sec) = 60
    Data = wan_wanlatency=1.7 :
```

Figure 21: nbstatus probes Output

A.3.22 nbstatus qslots

Name

```
nbstatus qslots
```

Description

Displays information about Queues, Qslots, and User Slots.

By default, **nbstatus qslots**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format



- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.

Usage

```
nbstatus qslots[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>][--GMT]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

`--target <pool_name>|<host_name>`

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If `--target` is omitted, `nbstatus` uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

`--date-format <date_format>`

Specifies the format to be used for dates and times.



--fields " [* ,]<field>[:[<func>][:<size>]] [,...]" | all

Displays only the selected fields. **--fields all** displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

func specifies a statistical operation that is performed on the field. This affects the output as follows:

- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that **--group-by** is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example:**--fields "RequestTimeFull-CreatedTimeFull"** This calculates the difference between these two date fields and displays the result (in seconds).

**Note**

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

The following function fields are also available. They are specified in the same way as **field** (above):

- ◆ **CanQslotPreempt(<Qslot_name>)** - displays true if the Qslot can currently preempt the specified Qslot.

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

**--number <num>**

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[,...]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

**Note**

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: `&&`, `||`, or `!`
- ◆ A comparison operator: `<`, `<=`, `>`, `>=`, `!=`, or `==`

Note

For strings, `==` is not case-sensitive. If you need to do a case-sensitive comparison, use the `equals()` function instead.

- ◆ The pattern-matching operator: `=~`. The syntax of the expression that follows this operator is the same as for `glob()`, below. It also matches partial strings. And it works with `--history`. (`glob()` does not.)

Note that with `--history`, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with `--history`, you must use `*` as a wildcard instead of `.*` without `--history`.

Note

`=~` is case-sensitive.

- ◆ A non-strict operator: `is` or `isnt`
- ◆ A bitwise operator: `&`, `|`, `~`, `^`, `<<`, `>>`, or `>>>`
- ◆ A conditional operator: `?` or `:`
- ◆ `value` is a number, a string, or another boolean expression.
If `value` is a date/time, it must be in one of the following formats:
 - ◆ `MM/DD/YYYY-HH:mm:ss`
 - ◆ `MM/DD/YYYY-HH:mm`
 - ◆ `MM/DD/YYYY`

Expressions can be grouped together using parentheses.

**Note**

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If **value** is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If **specification** is all, all records are displayed.

Functions can also be used in **specification**. To see which functions are available for **nbstatus qslots**, use **nbstatus functions** with the --context "nbstatus qslots" switch. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

```
"hasPermissions('<username>')"
```

Matches only Queues, Qslots, and User Slots for which the specified user has Job submission permissions.

Note

If no Queue is specified in the **specification**, then only Qslots in the default Queue are displayed. (If **specification** is all, then all Qslots in all Queues are displayed.)

To apply a filter to a Queue other than the default Queue, use a specification of the following form:

```
"Queue=='<queue_name>' && <condition>"
```

For example:

```
"Queue=='express' && (ShouldGet>GettingNow)"
```

This displays all the Qslots in the express Queue for which ShouldGet is greater than GettingNow.

Note

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.



If **value** is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

For example:

- ◆ `all` displays details of all Qslots.
- ◆ `"Queue=='express' &&Name=='myQslot'"` displays details of the Qslot called `myQslot` in the Queue called `express`.
- ◆ `"Queue=='express' &&State=='open'"` only displays details of open Qslots and User Slots in the `express` Queue.
- ◆ `"Queue=='myQueue'"` displays details of all the Qslots and User Slots in Queue `myQueue`.
- ◆ `"Queue=='myQueue' &&Level>=4"` displays details of all the Qslots and User Slots in Queue `myQueue` that are four or more levels deep in the hierarchy.

Fields: qslots

The following fields are available (default fields are marked with a *). If custom properties are defined for a Qslot, these are also displayed:

- ◆ **ActualGettingNow** - amount of resources the Qslot is getting now, taking into account resources that have been reserved by its Jobs.

The scheduler takes reservation into account when calculating Qslot eligibility, so **ActualGettingNow** gives a truer picture of what is going on than **GettingNow**.

- ◆ **ActualGettingNowCost** - the amount of resources the Qslot/Queue is getting now, taking Job cost into account (Only affects scheduling decisions if the Qslot that the Job was submitted to uses cost fair-share scheduling.)

It also takes into account resources that have been reserved by the Qslot's Jobs. The scheduler takes reservation into account when calculating Qslot eligibility, so **ActualGettingNowCost** gives a truer picture of what is going on than **GettingNowCost**.

- ◆ **ActualGettingNowPerLevel** - the resources that the Qslot is receiving as a percentage of the total resources that it and its "sibling" Qslots are receiving, taking into account resources that have been reserved by the Qslot's Jobs.



The scheduler takes reservation into account when calculating Qslot eligibility, so `ActualGettingNowPerLevel` gives a truer picture of what is going on than `GettingNowPerLevel`.

- ◆ `Alias*` - Qslot alias
- ◆ `Allocation*` - Allocation (for fairshare scheduling)
- ◆ `AverageWaitTime` - average wait time (in minutes) for the Jobs from this Qslot that ran during the configured timeframe
- ◆ `CPUTime` - CPU time (in seconds) accrued by the Jobs from this Qslot that ran during the configured timeframe
- ◆ `CPUTimeShare` - CPU time (in seconds) accrued by the Jobs from this Qslot, expressed as a percentage of the CPU time accrued by Jobs run in the specified context Qslot (if no context Qslot is specified, the Queue that the Qslot belongs to)
- ◆ `CloseReason` - the reason why the Qslot is closed
- ◆ `ConfiguredCapabilities` - configured Pool capabilities
- ◆ `Description` - the configured Qslot description
- ◆ `Dispatched` - Number of Jobs dispatched
- ◆ `DispatchedCost` - same as `RunningCost` (in future this will be used to provide additional information for Virtual Pools)
- ◆ `FairShouldGet` - same as `ShouldGet`, except that Jobs that cannot run are not counted. (This includes Jobs that cannot run because: they are suspended, the Qslot (or a parent Qslot) has reached its `max_running` limit, they have class requirements that cannot be met, or they require a license that is currently unavailable.)

Netbatch uses the value of this field (instead of `ShouldGet`) to decide whether a Job from a preemptive Qslot should preempt one from a preemptable Qslot.

Note

This is only relevant for the FAIRSHARE scheduling method.

- ◆ `FullPath` - Full Qslot path
 - ◆ `GettingNow*` - amount of resources the Qslot is getting now
- This does not take reservation into account. The scheduler **does** take reservation into account when calculating Qslot eligibility. For a truer picture of what is going on, use `ActualGettingNow` instead.



- ◆ **GettingNowCost** - the amount of resources the Qslot/Queue is getting now, taking Job cost into account (Only affects scheduling decisions if the Qslot that the Job was submitted to uses cost fair-share scheduling.)

This does not take reservation into account. The scheduler **does** take reservation into account when calculating Qslot eligibility. For a truer picture of what is going on, use **ActualGettingNowCost** instead.

- ◆ **GettingNowPerLevel** - the resources that the Qslot is receiving as a percentage of the total resources that it and its "sibling" Qslots are receiving

This does not take reservation into account. The scheduler **does** take reservation into account when calculating Qslot eligibility. For a truer picture of what is going on, use **ActualGettingNowPerLevel** instead.

- ◆ **Index** - Qslot index (for Netbatch 5 compatibility)
- ◆ **JobCalculatedShareOverTime** - the accumulated resource cost for the configured time window, weighted to take account of competition, expressed as a percentage of the total for the Qslot and its siblings. (So a Qslot is not penalized for using resources when there is no competition.)
- ◆ **JobCalculatedUsageOverTime** - the accumulated resource cost for the configured time window, weighted to take account of competition. (So a Qslot is not penalized for using resources when there is no competition.)

This is an absolute number, and only has meaning when compared to sibling Qslots.

- ◆ **JobCostShareOverTime** - the accumulated resource cost for the configured time window, expressed as a percentage of the total for the Qslot and its siblings
- ◆ **JobCostUsageOverTime** - the accumulated resource cost for the configured time window

This is an absolute number, and only has meaning when compared to sibling Qslots.

- ◆ **Jobslots** - allocation (number of execution slots) currently used by the Queue/Qslot's running Jobs. A regular Job uses one slot. A Parallel Job can use multiple slots. A Job with dedicated=true uses all the slots on the Workstation that it runs on.
- ◆ **JobslotsShareOverTime** - number of Jobs run, expressed as a percentage of Jobs run in the specified context Qslot (if



no context Qslot is specified, the Queue that the Qslot belongs to)

Only Jobs that **started** running in the current timeframe are counted.

- ◆ **JobsSlotsUsageOverTime** - total number of Jobs from this Qslot that ran during the configured timeframe
 - Only Jobs that **started** running in the current timeframe are counted.
- ◆ **LastDispatched** - The date/time the last Job was dispatched
- ◆ **LastSubmitted** - The date/time the last Job was submitted
- ◆ **Leaf** - whether the Qslot is a "leaf" Qslot or not (If it has no child Qslots, it is a leaf.)
- ◆ **Level** - Qslot level (that is, how "deep" in the hierarchy the Qslot is). For example, **Level** for a Qslot with one "parent" Qslot and one "grandparent" Qslot is 4 (the four levels being the Queue, the grandparent, the parent, and the Qslot itself).
- ◆ **LockedToDirectSubmission** - whether direct submission to the Pool/Queue/Qslot is allowed. (If not, Jobs may only be submitted via a Virtual Pool.)
- ◆ **LockedUsers** - List of users for whom this Qslot has been locked by the Monitoring Service
- ◆ **LockedUsersForResources** - list of users for whom this Qslot has been locked by the **nbadmin qslot lock** command
- ◆ **MaxCost** - maximum permitted Job cost for the Qslot/Queue (Only affects scheduling decisions if the Qslot that the Job was submitted to uses cost fair-share scheduling)
- ◆ **MaxRunning** - Maximum permitted number of running Jobs
- ◆ **MaxWaiting** - Maximum permitted number of waiting Jobs
- ◆ **MemberResources** - The Resource Set(s)/Group(s) that are assigned to the Qslot
- ◆ **Name*** - Qslot name
- ◆ **ParentFullPath** - the full path of the parent Qslot/Queue
- ◆ **ParentName** - the name of the Qslot's parent Qslot/Queue
- ◆ **Permissions** - Users and groups allowed to submit Jobs to the Qslot



This is a space-delimited list of items of the format
`{<user>|<group>}:<type>:{grant|deny}:`
`{Recursive|nonRecursive}.`

`{<user>|<group>}` is the user/group to whom the permissions are granted/denied.

`type` is one of the following:

- ◆ `JobSubmit`
- ◆ `JobModify`
- ◆ `JobRemove`
- ◆ `JobAdmin`
- ◆ `QueueOpen`
- ◆ `QueueClose`
- ◆ `QueueActivate`
- ◆ `QueueInactivate`
- ◆ `QueueAdmin`

`{grant|deny}` specifies whether the permissions are granted or denied.

`{Recursive|nonRecursive}` specifies whether the permissions apply only to this Queue/Qslot (`nonRecursive`), or to this Queue/Qslot and all the Qslots under it (`Recursive`).

- ◆ `Priority*` - Priority (for priority scheduling)
- ◆ `Queue` - The name of the Queue to which the Qslot belongs
- ◆ `ReservedCapabilities` - reserved capabilities (each individual resource can have a double type value in the report output)
- ◆ `Reserving` - the number of Jobs in the Qslot that are currently reserving resources
- ◆ `ReservingCost` - Job cost of the Jobs that are currently reserving resources
- ◆ `ResourceUsageHistory` - the configured timespan for the Qslot in hours (if not configured, the default is 168 hours—seven days)
- ◆ `Running` - Number of running Jobs (or Jobs holding any kind of allocation)
- ◆ `RunningCost` - total cost of running Jobs in the Qslot/Queue (Only affects scheduling decisions if the Qslot that the Job was



submitted to uses cost fair-share scheduling. See [Section 5.1.3.6, Cost Fair-share Resource Allocation](#))

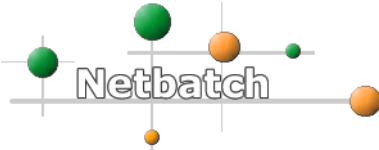
- ◆ **SchedulingMethod** - The scheduling method used:
 - ◆ **Priority**
 - ◆ **Fairshare**
 - ◆ **Complement fairshare**
 - ◆ **Summary fairshare**
 - ◆ **FIFO**
- ◆ **ShouldGet*** - Amount of resources the Qslot should be getting

Note

This is only relevant for the FAIRSHARE scheduling method.

ShouldGet sums to 100 at the Queue level.

- ◆ **ShouldGetPerLevel** - The resources that the Qslot should be receiving as a percentage of the total resources that it and its "sibling" Qslots should be receiving
- ◆ **Status** - Qslot state (**open** or **closed**, **active** or **inactive**)
- ◆ **Type** - Qslot type:
 - ◆ **root** - The root of the resource allocation structure
 - ◆ **queue** - Queue
 - ◆ **qslot** - regular Qslot
 - ◆ **user** - User Slot
 - ◆ **/others** - default Qslot
- ◆ **WTime** - wall clock time (in seconds) accrued by the Jobs from this Qslot that ran during the configured timeframe
- ◆ **WTimeShare** - wall clock time (in seconds) accrued by the Jobs from this Qslot, expressed as a percentage of the wall clock time accrued by Jobs run in the specified context Qslot (if no context Qslot is specified, the Queue that the Qslot belongs to)
- ◆ **Waiting** - Number of waiting Jobs



Output

The **nbstatus qslots** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 22, below, shows sample output for the following command:

```
nbstatus qslots --target itstl107
```

Pool itstl107 on itstl107
Version 6.4_w_0115
On since 02/17/2004 14:54:10
Time now 02/17/2004 15:28:08

Name Alias Priority Allocation ShouldGet GettingNow

/1 1 0 0 100.0 0.0
/OTHERS 0 0 0.0 0.0

Figure 22: nbstatus qslots Output

--format {table|block|csv|script}

Specifies that output is to be displayed as:

- ◆ **table**—a table (the default)
- ◆ **block**—blocks (one block for each Job)
- ◆ **csv**—comma-separated values (suitable for importing into Excel)
- ◆ **script**—the same as **csv**, except that it does not include the header row



it must be in one of the following formats**Target**

--target <pool_name> | <host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbstatus** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

--date-format <date_format>

Specifies the format to be used for dates and times.

--fields "[*,]<field>[:[:<func>][:<size>]] [, . . .]" | all

Displays only the selected fields. **--fields all** displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

func specifies a statistical operation that is performed on the field. This affects the output as follows:

- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields**



"**owner:uniq**" displays the number of different owners.)
Note that **--group-by** is not required in this case.

- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example:**--fields "RequestTimeFull-CreatedTimeFull"** This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

**--ignore-invalid-fields**

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[, . . .]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

**Note**

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: `&&`, `||`, or `!`
- ◆ A comparison operator: `<`, `<=`, `>`, `>=`, `!=`, or `==`

Note

For strings, `==` is not case-sensitive. If you need to do a case-sensitive comparison, use the `equals()` function instead.

- ◆ The pattern-matching operator: `=~`. The syntax of the expression that follows this operator is the same as for `glob()`, below. It also matches partial strings. And it works with `--history`. (`glob()` does not.)

Note that with `--history`, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with `--history`, you must use `*` as a wildcard instead of `.*` without `--history`.

Note

`=~` is case-sensitive.

- ◆ A non-strict operator: `is` or `isnt`
- ◆ A bitwise operator: `&`, `|`, `~`, `^`, `<<`, `>>`, or `>>>`
- ◆ A conditional operator: `?` or `:`
- ◆ `value` is a number, a string, or another boolean expression.
If `value` is a date/time, it must be in one of the following formats:
 - ◆ `MM/DD/YYYY-HH:mm:ss`
 - ◆ `MM/DD/YYYY-HH:mm`
 - ◆ `MM/DD/YYYY`

Expressions can be grouped together using parentheses.

**Note**

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If **value** is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If **specification** is **all**, all records are displayed.

Functions can also be used in **specification**. To see which functions are available for this **nbstatus** command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

A.3.23 nbstatus remote-availability

Name

nbstatus remote-availability - displays details about availability and other information for each of a Virtual Pool's Virtual Resources for a specified class and Qslot.

Description

Displays information about the availability of Job slots, wait times, number of running and waiting Jobs, limits, and so on, for each of the Virtual Pool's Virtual Resources for a specified class and Qslot.

By default, **nbstatus remote-availability**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.



Usage

```
nbstatus remote-availability[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>][--GMT]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

```
--target <pool_name>|<host_name>
```

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If `--target` is omitted, `nbstatus` uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

```
--date-format <date_format>
```

Specifies the format to be used for dates and times.

```
--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all
```

Displays only the selected fields. `--fields all` displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

`func` specifies a statistical operation that is performed on the field. This affects the output as follows:

- ♦ If `--group-by` is not used, specifying a `func` gives one line of output.



- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that **--group-by** is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example:**--fields "RequestTimeFull-CreatedTimeFull"** This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)



- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

**--sort-by [-]<field_name>[, . . .]**

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for **glob()**, below. It also matches partial strings. And it works with **--history**. (**glob()** does not.)



Note that with **--history**, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with **--history**, you must use * as a wildcard instead of .* without **--history**.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: **is** or **isnt**
- ◆ A bitwise operator: **&**, **|**, **~**, **^**, **<<**, **>>**, or **>>>**
- ◆ A conditional operator: **?** or **:**
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ **MM/DD/YYYY-HH:mm:ss**
 - ◆ **MM/DD/YYYY-HH:mm**
 - ◆ **MM/DD/YYYY**

Expressions can be grouped together using parentheses.

Note

A double equals sign (**==**) is used. **!=** (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If **value** is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If **specification** is **all**, all records are displayed.

Functions can also be used in **specification**. To see which functions are available for this **nbstatus** command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ **all** displays details of all Virtual Resources.
- ◆ **"Resource=='vr1'"** only displays details for the Virtual Resource called **vr1**.



- ◆ "Available=='true'" only displays Virtual Resources that have Job slots available for the specified class and Qslot.

Fields: remote-availability

The following fields are available (default fields are marked with a *):

- ◆ **Available*** - whether the Virtual Resource can accept a Job with the specified requirements
- ◆ **AvailableSlots** - number of available Job slots that support the specified requirements
- ◆ **AverageWait** - average wait time in Virtual Resource for Jobs with the specified requirements
- ◆ **EstimatedCapacity** - estimated capacity of the remote Qslot
- ◆ **EstimatedWait** - estimated wait time in Virtual Resource for Jobs with the specified requirements
- ◆ **Host** - the hostname(s) of the physical Pool master(s)
- ◆ **LocalTotalInQslot** - total number of Jobs (waiting and running) from this Virtual Pool in the specified Qslot
- ◆ **LocalTotalInQslotLimit** - limit for the total number of Jobs (waiting and running) from this Virtual Pool in the specified Qslot
- ◆ **LocalTotalInResource** - total number of Jobs (waiting and running) from this Virtual Pool waiting in the remote resource
- ◆ **LocalTotalInResourceLimit** - limit for the total number of Jobs (waiting and running) from this Virtual Pool in the remote resource
- ◆ **LocalWaitingInQslot** - number of Jobs from this Virtual Pool waiting in the specified Qslot
- ◆ **LocalWaitingInQslotLimit** - limit for the number of Jobs from this Virtual Pool waiting in the specified Qslot
- ◆ **LocalWaitingInResource** - number of Jobs from this Virtual Pool waiting in the remote resource
- ◆ **LocalWaitingInResourceLimit** - limit for the number of Jobs from this Virtual Pool waiting in the remote resource
- ◆ **MaxWait** - maximum wait time in Virtual Resource for Jobs with the specified requirements



- ◆ **NotAvailableReason** - reason why Jobs with the specified requirements cannot be accepted
- ◆ **Resource*** - the name of the Virtual Resource
- ◆ **SupportedSlots** - total number of Job slots that support the specified requirements
- ◆ **TargetClass*** - target class in the remote resource
- ◆ **TargetQslot*** - target Qslot in the remote resource

Output

The **nbstatus remote-availability** output depends on:

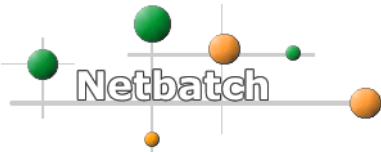
- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

The following figure shows sample output for the following command:

nbstatus remote-availability

```
VP on itstl011
Version 7.3.1_0183_00
On since 09/20/2007 17:59:19
Time now 09/23/2007 09:09:45
-----
Resource      TargetQsl> TargetCla> Available
-----
TestPool1          @        false
TestPool2          @        false
-----
```

Figure 23: nbstatus remote-availability Output



A.3.24 nbstatus resource-groups

Name

nbstatus resource-groups

Description

Displays information about Resource Groups.

By default, **nbstatus resource-groups**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.

Usage

```
nbstatus resource-groups[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>][--GMT]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```



You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

--target <pool_name> | <host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbstatus** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

--date-format <date_format>

Specifies the format to be used for dates and times.

**--fields " [* ,]<field>[:[<func>]][:<size>]]
[, . . .]" | all**

Displays only the selected fields. **--fields all** displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

func specifies a statistical operation that is performed on the field. This affects the output as follows:

- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items



- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, `stodstatus areas --fields "owner:uniq"` displays the number of different owners.) Note that `--group-by` is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which `concat` is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example: `--fields "RequestTimeFull-CreatedTimeFull"` This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as `csv`, except that the header row includes additional information about each field
- ◆ **script** - the same as `csv`, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

**--group-by <field_name>[,...]**

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[,...]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).



- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for glob(), below. It also matches partial strings. And it works with --history. (glob() does not.)

Note that with --history, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with --history, you must use * as a wildcard instead of .* without --history.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: is or isnt
- ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
- ◆ A conditional operator: ? or :
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ **MM/DD/YYYY-HH:mm:ss**



- ◆ **MM/DD/YYYY-HH:mm**
- ◆ **MM/DD/YYYY**

Expressions can be grouped together using parentheses.

Note

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If value is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If specification is all, all records are displayed.

Functions can also be used in specification. To see which functions are available for this nbstatus command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ **all** displays details of all Resource Groups.
- ◆ **"Name=='rs1'"** displays details of Resource Group **rs1**.

Fields: resource-groups

The following fields are available (default fields are marked with a *):

- ◆ **CanRun** - number of Jobs that the Resource Group can currently run
- ◆ **Name*** - the name of the Resource Group
- ◆ **NumOfJobs** - number of running Jobs in the Resource Group
- ◆ **NumOfResources*** - number of Workstations in the Resource Group
- ◆ **ResourcesList** - list of Workstations in the Resource Group

Output

The **nbstatus resource-groups** output depends on:

- ◆ The fields selected for display



- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

The following figure shows sample output for the following command:

```
nbstatus resource-groups "name=='rs1'"
```

Name	NumOfReso>
rs1	3

Figure 24: nbstatus resource-groups Output

A.3.25 nbstatus resources

Name

```
nbstatus resources
```

Description

Displays information about virtual resources.

By default, **nbstatus resources**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.



Usage

```
nbstatus resources[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>][--GMT]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

```
--target <pool_name>|<host_name>
```

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If `--target` is omitted, `nbstatus` uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

```
--date-format <date_format>
```

Specifies the format to be used for dates and times.

```
--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all
```

Displays only the selected fields. `--fields all` displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

`func` specifies a statistical operation that is performed on the field. This affects the output as follows:

- ♦ If `--group-by` is not used, specifying a `func` gives one line of output.



- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that **--group-by** is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example:**--fields "RequestTimeFull-CreatedTimeFull"** This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)



- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

**--sort-by [-]<field_name>[, . . .]**

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for **glob()**, below. It also matches partial strings. And it works with **--history**. (**glob()** does not.)



Note that with **--history**, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with **--history**, you must use * as a wildcard instead of .* without **--history**.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: **is** or **isnt**
- ◆ A bitwise operator: **&**, **|**, **~**, **^**, **<<**, **>>**, or **>>>**
- ◆ A conditional operator: **?** or **:**
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ **MM/DD/YYYY-HH:mm:ss**
 - ◆ **MM/DD/YYYY-HH:mm**
 - ◆ **MM/DD/YYYY**

Expressions can be grouped together using parentheses.

Note

A double equals sign (**==**) is used. **!=** (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If **value** is a string, it must be enclosed in single quotes.

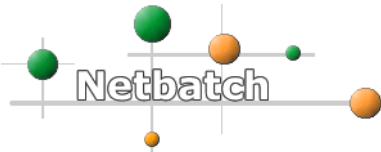
Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If **specification** is **all**, all records are displayed.

Functions can also be used in **specification**. To see which functions are available for this **nbstatus** command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ **all** displays details of all virtual resources in the specified Pool.



- ◆ "Name=='rs1'" displays details of the virtual resources called **rs1**.

Fields: resources

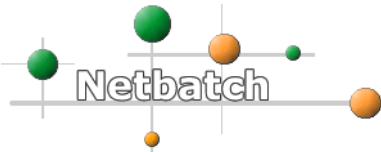
The following fields are available (default fields are marked with a *):

- ◆ **Alias** - alias
- ◆ **Classes** - Classes supported by the virtual resource
- ◆ **LastSelect*** - Date/time that the last Job was sent to the virtual resource
- ◆ **Local** - Whether the virtual resource is local or remote (virtual resource in a Virtual Pool - **false**, Resource Set in a physical Pool - **true**)
- ◆ **Name*** - Name of virtual resource
- ◆ **OnSince*** - Date/time the Pool was started (for a virtual resource, the physical Pool, not the Virtual Pool)
- ◆ **Running*** - Number of running Jobs in the virtual resource
- ◆ **Sel*** - Number of Jobs sent to the virtual resource since the Virtual Pool started running
- ◆ **Server*** - Pool Master hostname
- ◆ **Status*** - Current status:
 - ◆ **Disc** - Disconnected
 - ◆ **Locked** - Locked
 - ◆ **Accepting** - Can accept Jobs
- ◆ **Version** - Netbatch version and build date

Output

The **nbstatus resources** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results



The following figure shows sample output for the following command:

```
nbstatus resources --target linux_pilot
```

Pool linux_pilot on inbm007					
version 6.5_0134					
on since 11/08/2004 09:22:02					
Time now 11/10/2004 15:12:43					

Name	Server	Status	Running On Since	Last Select	Sel
linux_p>	inbm007	Accepting	0	11/08/2004 09:23:54	0
linux_p>	inbm007	Accepting	0	11/08/2004 09:23:54	0
linux_p>	inbm007	Accepting	0	11/08/2004 09:23:54	0
linux_p>	inbm007	Accepting	0	11/08/2004 09:23:54	0

Figure 25: nbstatus resources Output

A.3.26 nbstatus services

Name

```
nbstatus services
```

Description

Displays information about registered services. Currently, registered services include only Netbatch Tracker.

By default, **nbstatus services**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.



Usage

```
nbstatus services[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>[--GMT]]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

```
--target <pool_name>|<host_name>
```

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If `--target` is omitted, `nbstatus` uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

```
--date-format <date_format>
```

Specifies the format to be used for dates and times.

```
--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all
```

Displays only the selected fields. `--fields all` displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

`func` specifies a statistical operation that is performed on the field. This affects the output as follows:

- ♦ If `--group-by` is not used, specifying a `func` gives one line of output.



- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that **--group-by** is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example:**--fields "RequestTimeFull-CreatedTimeFull"** This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)



- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

**--sort-by [-]<field_name>[, . . .]**

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for **glob()**, below. It also matches partial strings. And it works with **--history**. (**glob()** does not.)



Note that with **--history**, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with **--history**, you must use ***** as a wildcard instead of **.*** without **--history**.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: **is** or **isnt**
- ◆ A bitwise operator: **&**, **|**, **~**, **^**, **<<**, **>>**, or **>>>**
- ◆ A conditional operator: **?** or **:**
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ **MM/DD/YYYY-HH:mm:ss**
 - ◆ **MM/DD/YYYY-HH:mm**
 - ◆ **MM/DD/YYYY**

Expressions can be grouped together using parentheses.

Note

A double equals sign (**==**) is used. **!=** (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If **value** is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If **specification** is **all**, all records are displayed.

Functions can also be used in **specification**. To see which functions are available for this **nbstatus** command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ **all** displays details of all services.
- ◆ **"Service=="'Tracker'"** displays details of all Netbatch Tracker services.



Fields: services

The following fields are available (default fields are marked with a *):

- ◆ **ConfigurationPath** - The path to where the service's configuration file is stored
- ◆ **DBHost** - Database host
- ◆ **Host*** - The host on which the service is running
- ◆ **IP** - IP address of the host on which the service is running
- ◆ **LastUpdate*** - Date/time of last status update
- ◆ **LogsPath** - The path where the tracker logs are saved
- ◆ **Patches** - Patches that have been applied
- ◆ **Pools** - Pools that are served by the service
- ◆ **RegID** - The service's registration ID
- ◆ **Service*** - The name of the service
- ◆ **startTime*** - The time/date that the service was started
- ◆ **Version*** - Netbatch version

Output

The **nbstatus services** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

The following figure shows sample output for the following command:

```
nbstatus services --fields "host,service,version"
```



DirectoryService on itstl109		
version 7.1.0_0166_03		
on since 02/16/2006 08:35:38		
Time now 02/16/2006 14:31:34		
<hr/>		
Host	Service	version
<hr/>		
anls0103.an.intel.com	Tracker	6.5.0_0134_27
auslnas1.an.intel.com	Tracker	6.5_0134_19
chlinsn22.ch.intel.com	Tracker	6.5.0_0134_29
filc0152.fm.intel.com	Tracker	6.5.0_0134_30
inbm004.iil.intel.com	Tracker	6.5.0_0134_30
inlc0002.iind.intel.com	Tracker	6.5_0134_19
iws1003.iil.intel.com	SToD	2.2_0016
jernbm1.jer.intel.com	Tracker	6.5.0_0134_30
mmdcs013.hd.intel.com	Tracker	6.5.0_0134_27
mssnbmaster.ims.intel.com	Tracker	6.5.0_0134_29
pglnbm03.png.intel.com	Tracker	6.5.0_0134_28
pta1004.pt.intel.com	Tracker	6.5.0_0134_25
vcs12212.sc.intel.com	Tracker	6.5.0_0134_29
vcs7400.sc.intel.com	Tracker	6.5.0_0134_30
<hr/>		

Figure 26: nbstatus services Output

A.3.27 nbstatus virtualjobs

Name

nbstatus virtualjobs

Description

Displays jobs dispatched to remote resources while in waiting.

By default, **nbstatus virtualjobs**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report



You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.

Usage

```
nbstatus virtualjobs[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>][--GMT]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

```
--target <pool_name>|<host_name>
```

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If `--target` is omitted, `nbstatus` uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

```
--date-format <date_format>
```

Specifies the format to be used for dates and times.

```
--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all
```

Displays only the selected fields. `--fields all` displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

`func` specifies a statistical operation that is performed on the field. This affects the output as follows:



- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that **--group-by** is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example: **--fields "RequestTimeFull-CreatedTimeFull"** This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:



- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

**--sort-by [-]<field_name>[, . . .]**

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for **glob()**, below. It also matches partial strings. And it works with **--history**. (**glob()** does not.)



Note that with **--history**, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with **--history**, you must use * as a wildcard instead of .* without **--history**.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: **is** or **isnt**
- ◆ A bitwise operator: **&**, **|**, **~**, **^**, **<<**, **>>**, or **>>>**
- ◆ A conditional operator: **?** or **:**
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ **MM/DD/YYYY-HH:mm:ss**
 - ◆ **MM/DD/YYYY-HH:mm**
 - ◆ **MM/DD/YYYY**

Expressions can be grouped together using parentheses.

Note

A double equals sign (**==**) is used. **!=** (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If **value** is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If **specification** is **all**, all records are displayed.

Functions can also be used in **specification**. To see which functions are available for this **nbstatus** command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ **all** displays details of all virtual jobs.



Fields: virtualjobs

The following fields are available (default fields are marked with a *):

- ◆ **ActualClassReservation** - the actual class reservation (the higher of **ClassReservation** and **implicitClassReservation**)
- ◆ **AllocationCost** - the Job allocation cost calculated for billing
- ◆ **Class** - the Class name on the remote pool
- ◆ **Fullid*** - the Job's full ID
- ◆ **Qslot** - the Qslot name on the remote pool
- ◆ **RemoteData** - the remote attributes on the remote pool
- ◆ **RemoteServerHostName** - hostname of the remote pool
- ◆ **VirtualResource*** - the name of the remote waiting resource
- ◆ **WaitReason*** - the reason why the Job isn't dispatched for execution

Output

The **nbstatus virtualjobs** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 20, below, shows sample output for the following command:

```
nbstatus virtualjobs --number 10 all
```



```
VP on fmynbm2009
Version 8.0.0_0392_06
on since 10/18/2011 10:29:52
Time now 11/15/2011 02:28:40
```

Fullid	WaitReason	virtualResource
fm_zse_vp.2229	Required resources not y> fm_zse_c1	
fm_zse_vp.2229	no resource is available fm_zse_c0	
fm_zse_vp.2229	no resource is available fm_zse_c2	
fm_zse_vp.2230	no resource is available fm_zse_c0	
fm_zse_vp.2230	no resource is available fm_zse_c2	
fm_zse_vp.2230	reservation is not compl> fm_zse_c1	
fm_zse_vp.2231	no resource is available fm_zse_c0	
fm_zse_vp.2231	no resource is available fm_zse_c2	
fm_zse_vp.2231	reservation is not compl> fm_zse_c1	
fm_zse_vp.2232	skipped due to similar j> fm_zse_c0	

Figure 27: nbstatus virtualjobs Output

A.3.28 nbstatus vm-image-cache

Name

nbstatus vm-image-cache

Description

Displays information about available OS images. You can use one of these images to submit a Job that needs a particular OS version that no actual, physical Workstations are running. Netbatch instantiates a virtual machine using the specified image and runs the Job on it.

By default, **nbstatus vm-image-cache**:

- ◆ Displays all available fields
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)



- ◆ Shows all OS images
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.

Usage

```
nbstatus vm-image-cache[,<object>[,...]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>][--GMT]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

`--target <pool_name>|<host_name>`

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If `--target` is omitted, `nbstatus` uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

`--date-format <date_format>`

Specifies the format to be used for dates and times.

`--fields "[*,]<field>[:[:<func>][:<size>]]
[,...]"|all`

Displays only the selected fields. `--fields all` displays all available fields. Fields are displayed in the specified order.



See the Fields section, below, for a list of available fields.

func specifies a statistical operation that is performed on the field.

This affects the output as follows:

- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that **--group-by** is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example: **--fields "RequestTimeFull-CreatedTimeFull"** This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)



--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--gmt

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.



If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[,...]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.



- ◆ The pattern-matching operator: `=~`. The syntax of the expression that follows this operator is the same as for `glob()`, below. It also matches partial strings. And it works with `--history`. (`glob()` does not.)

Note that with `--history`, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with `--history`, you must use `*` as a wildcard instead of `.*` without `--history`.

Note

`=~` is case-sensitive.

- ◆ A non-strict operator: `is` or `isnt`
- ◆ A bitwise operator: `&`, `|`, `~`, `^`, `<<`, `>>`, or `>>>`
- ◆ A conditional operator: `?` or `:`
- ◆ `value` is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ `MM/DD/YYYY-HH:mm:ss`
 - ◆ `MM/DD/YYYY-HH:mm`
 - ◆ `MM/DD/YYYY`

Expressions can be grouped together using parentheses.

Note

A double equals sign (`==`) is used. `!=` (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If `value` is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If `specification` is `all`, all records are displayed.

Functions can also be used in `specification`. To see which functions are available for this `nbstatus` command, use [nbstatus functions](#). For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:



- ◆ `all` displays all OS images (which is what it does by default if no filter is applied).
- ◆ `"Public=='true'"` displays only public OS images.

Fields: `vm-image-cache`

The following fields are available (default fields are marked with a *):

- ◆ `ImageID` - the image
- ◆ `ImageName` - the name of the OS image
- ◆ `Owner` - the username of the user who own the image
- ◆ `Public` - whether the image is public or not

Output

The `nbstatus vm-image-cache` output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 28, below,The following figure shows sample output for the following command:

`nbstatus vm-image-cache`



```
PPM on icsl9579
Version 8.2.1_0597_05
On since 04/28/2014 13:15:39
Time now 05/20/2014 10:49:51
```

ImageName	ImageID	Public	Owner
<hr/>			
SLES_10_VGROISMA_TEST	10.184.1>	true	rsack
VMT_LinuxSET_EC_SLES10SP3-3_R>	10.184.1>	true	bnirenbe
VMT_LinuxSET_EC_SLES11SP1-1_R>	10.184.1>	true	bnirenbe
VMT_LinuxSET_EC_SLES11SP2-2_R>	10.184.1>	true	rsack
VMT_Windows_7-SP1_x64-CFS	10.184.1>	true	bnirenbe
VMT_Windows_Server_2008R2-SP1>	10.184.1>	true	bnirenbe
<hr/>			

Figure 28: nbstatus vm-image-cache Output

A.3.29 nbstatus vms

Name

nbstatus vms

Description

Displays information about virtual machines that are currently running through Netbatch (in a particular Pool). By default, it only shows the VMs that are owned by the user running the command, but you can use the **all** filter to view all VMs.

By default, **nbstatus vms**:

- ◆ Displays all available fields
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Shows all VMs
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.



Usage

```
nbstatus vms[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>][--GMT]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

```
--target <pool_name>|<host_name>
```

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If `--target` is omitted, `nbstatus` uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

```
--date-format <date_format>
```

Specifies the format to be used for dates and times.

```
--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all
```

Displays only the selected fields. `--fields all` displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

`func` specifies a statistical operation that is performed on the field. This affects the output as follows:

- ♦ If `--group-by` is not used, specifying a `func` gives one line of output.



- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that **--group-by** is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example:**--fields "RequestTimeFull-CreatedTimeFull"** This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)



- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

**--sort-by [-]<field_name>[, . . .]**

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for **glob()**, below. It also matches partial strings. And it works with **--history**. (**glob()** does not.)



Note that with **--history**, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with **--history**, you must use ***** as a wildcard instead of **.*** without **--history**.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: **is** or **isnt**
- ◆ A bitwise operator: **&**, **|**, **~**, **^**, **<<**, **>>**, or **>>>**
- ◆ A conditional operator: **?** or **:**
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ **MM/DD/YYYY-HH:mm:ss**
 - ◆ **MM/DD/YYYY-HH:mm**
 - ◆ **MM/DD/YYYY**

Expressions can be grouped together using parentheses.

Note

A double equals sign (**==**) is used. **!=** (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If **value** is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If **specification** is **all**, all records are displayed.

Functions can also be used in **specification**. To see which functions are available for this **nbstatus** command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ **all** displays all virtual machines. (By default, it only shows those belonging to the user running the command.)



- ◆ "PhysicalHost=='icsl2225'" displays only the VMs that are running on the machine called **icsl2225**.

Fields: vms

The following fields are available (default fields are marked with a *):

- ◆ **ClusterID** - the ID of the cluster that the VM belongs to
- ◆ **Hostname** - the hostname of the VM
- ◆ **IP** - the VM's IP address
- ◆ **ImageID** - the ID of the OS image that was used to instantiate the VM
- ◆ **Memory** - the amount of memory that was requested for the VM
- ◆ **Owner** - the username of the user who own the VM
- ◆ **RUsage** - the VM's resource usage
- ◆ **StartTime** - the time and date when the VM was started
- ◆ **status** - the status of the VM
- ◆ **SubmitTime** - the time and date when the request to create a VM was made
- ◆ **vMid** - the ID of the VM
- ◆ **Workstation** - the hostname of the physical machine that the VM is running on
- ◆ **vCPUs** - the number of cores that were requested for the VM

Output

The **nbstatus vms** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 29, below,The following figure shows sample output for the following command:

```
nbstatus vms --format block --number 1
```



```
PPM on icsl9579
Version 8.2.1_0597_05
On since 04/28/2014 13:15:39
Time now 05/21/2014 12:06:08
{
    VMID = 621554
    Hostname = itstl406
    ClusterID = iil_kvm_621161
    ImageID = VMT_LinuxSET_EC_SLES10SP3-3_Rev_0_ia32e
    PhysicalHost = icsl2140
    Owner = eshmueli
    RequiredCores = 4
    RequiredMemory = 16
    Status = Running
}
```

Figure 29: nbstatus vms Output

A.3.30 nbstatus wan

Name

nbstatus wan

Description

Displays information about the WAN.

By default, **nbstatus wan**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.



Usage

```
nbstatus wan[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>[--GMT]]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

```
--target <pool_name>|<host_name>
```

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If `--target` is omitted, `nbstatus` uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

```
--date-format <date_format>
```

Specifies the format to be used for dates and times.

```
--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all
```

Displays only the selected fields. `--fields all` displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

`func` specifies a statistical operation that is performed on the field. This affects the output as follows:

- ♦ If `--group-by` is not used, specifying a `func` gives one line of output.



- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that **--group-by** is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.

You can also perform simple calculations on numeric and date fields. For example: **--fields "RequestTimeFull-CreatedTimeFull"** This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)



- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored. (The invalid field will be displayed in the output with blank values.)

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

**--sort-by [-]<field_name>[, . . .]**

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

Note

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for **glob()**, below. It also matches partial strings. And it works with **--history**. (**glob()** does not.)



Note that with **--history**, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with **--history**, you must use * as a wildcard instead of .* without **--history**.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: **is** or **isnt**
- ◆ A bitwise operator: **&**, **|**, **~**, **^**, **<<**, **>>**, or **>>>**
- ◆ A conditional operator: **?** or **:**
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ **MM/DD/YYYY-HH:mm:ss**
 - ◆ **MM/DD/YYYY-HH:mm**
 - ◆ **MM/DD/YYYY**

Expressions can be grouped together using parentheses.

Note

A double equals sign (**==**) is used. **!=** (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If **value** is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.

If **specification** is **all**, all records are displayed.

Functions can also be used in **specification**. To see which functions are available for this **nbstatus** command, use **nbstatus functions**. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ **all** displays details of all domains.
- ◆ "**domain=='iil'**" displays details for the **iil** domain.



Fields: wan

The following fields are available (default fields are marked with a *):

- ◆ **CheckTime*** - The last time the WAN status was checked for the domain
- ◆ **Domain*** - The name of the domain
- ◆ **LatencyStatus*** - Latency (in milliseconds) for the domain
- ◆ **LatencyThreshold*** - Latency threshold (in milliseconds) for the domain

Output

The **nbstatus wan** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

A.3.31 nbstatus workstations

Name

nbstatus workstations

Description

Displays information about Workstations.

By default, **nbstatus workstations**:

- ◆ Displays a default set of fields (a subset of all available fields)
- ◆ Displays the output in table format
- ◆ Sorts the output by the first field (then by the second field, then the third, and so on)
- ◆ Does not display a summary report

You can use switches to specify which fields to display, how the output is sorted and grouped, and in which format to display the output.



Usage

```
nbstatus workstations[,<object>[,...]]  
[--target <pool_name>|<host_name>]  
[--retry-timeout <time>{h|m}]  
[--fields "[*,]<field>[:[:<func>][:<size>]]  
[,...]"|all]  
[--format {rfc4180-csv|typed-csv|  
table|block|csv|script|xml}]  
[--group-by <field_name>[,...]]  
[--sort-by [-]<field_name>[,...]]  
[--timeout <time>[--GMT]]  
[--date-format <date_format>]  
[--ignore-invalid-fields] [<specification>]
```

You can combine multiple status requests for different object types into a single request. In this case, you must add a filter expression (specification), otherwise you will get meaningless results.

Note that the name of each field is prefixed with the object type (e.g., `jobs.Status`).

Target

```
--target <pool_name>|<host_name>
```

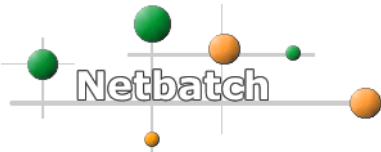
Specifies the Pool Master, either by Pool name or by Pool Master host name. (If `--target` is omitted, `nbstatus` uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

```
--date-format "<date_format>"
```

Specifies the format to be used for dates and times. `date_format` can include any of the following, in any order, separated by valid separator characters (e.g., space, colon (:), slash (/), etc.):

- ◆ `yyyy` - year (four digits)
- ◆ `yy` - year (two digits)
- ◆ `MM` - month
- ◆ `dd` - date
- ◆ `hh` - hour
- ◆ `mm` - minutes



- ◆ **ss** - seconds
- ◆ **sss** - milliseconds

The default format is "**MM/dd/yyyy hh:mm:ss**".

```
--fields "[*,]<field>[:<func>][:<size>]]  
[,...]" | all
```

Displays only the selected fields. **--fields all** displays all available fields. Fields are displayed in the specified order.

See the Fields section, below, for a list of available fields.

func specifies a statistical operation that is performed on the field. This affects the output as follows:

- ◆ If **--group-by** is not used, specifying a **func** gives one line of output.
- ◆ If **--group-by** is used, this gives one line of output per group.
- ◆ If **--group-by** is used without **func**, **sum** is used.

Note

By default, in lines of grouped output, the contents of each field are summed.

Valid operations are as follows:

- ◆ **sum** - sum of all values
- ◆ **count** - the number of matching items
- ◆ **min** - minimum value
- ◆ **max** - maximum value
- ◆ **avg** - average of all values
- ◆ **uniq** - displays the number of unique values for the specified field. (For example, **stodstatus areas --fields "owner:uniq"** displays the number of different owners.) Note that **--group-by** is not required in this case.
- ◆ **concat** - one record displayed. The values of the field for which **concat** is specified are concatenated into a single string.

size specifies the displayed column width for the field.



You can also perform simple calculations on numeric and date fields. For example:`--fields "RequestTimeFull-CreatedTimeFull"` This calculates the difference between these two date fields and displays the result (in seconds).

Note

If you want to perform more complex calculations, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

--format {table|block|csv|rfc4180-csv|typed-csv|script|xml}

Specifies that output is to be displayed as:

- ◆ **table** - a table
- ◆ **block** - blocks (one block for each item)
- ◆ **csv** - comma-separated values (suitable for importing into Excel)
- ◆ **rfc4180-csv** - RFC 4180-compliant comma-separated values
- ◆ **typed-csv** - the same as **csv**, except that the header row includes additional information about each field
- ◆ **script** - the same as **csv**, except that it does not include the header row
- ◆ **xml** - XML

--GMT

Specifies that times should be displayed in GMT.

--group-by <field_name>[,...]

Specifies that the output is to be grouped by the specified field. If more than one field is specified, output is grouped by the first field, then by the second, and so on.

--history <time>

Requests historical Job information.

**Note**

In Netbatch 8.1 and later, nbstatus jobs --history only shows Jobs from one Pool. Previously, it showed Jobs from all the Pools that report to the same Tracker.

time specifies the timeframe for the query. It can be one of the following:

- ◆ <int>M - output covers the last int minutes.
- ◆ <int>D - output covers the last int days.
- ◆ <int>H - output covers the last int hours.
- ◆ mm/dd/yyyy[-hh:mm] - output covers the time since the specified date/time.
- ◆ 'mm/dd/yyyy[-hh:mm] mm/dd/yyyy[-hh:mm]' - output covers the specified timeframe.
- ◆ ww<ww>[Y<yyyy>] - output covers the specified workweek. **ww** is the two-digit workweek (e.g., 03). **yyyy** is the four-digit year (e.g., 2005).

With this switch, the set of available fields is not the same as without the switch.

In addition to the fields listed in **--fields**, there are additional fields available when using the **--history** switch. See Fields section below.

When **--group-by** is used with **--history**, numerical fields display average values. Other fields display -.

--ignore-invalid-fields

Specifies that if invalid fields are specified (in the **--fields** switch), they should be ignored.

--number <num>

Only displays the first **num** results.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.



If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--sort-by [-]<field_name>[,...]

Specifies that the output is to be sorted by the specified field. If more than one field is specified, output is sorted by the first field, then by the second, and so on. The optional - (before the field name) specifies that output is to be sorted in reverse order.

--timeout <time>

Specifies the timeout for the command (in seconds).

specification

The output can be filtered using an expression to specify which information is displayed (which can be a subset or a superset of the default output).

specification is either **all** or an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is a valid field (see **--fields**, above).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")

Note

When using multiple arithmetic operators, use parentheses to make the order of calculation explicit. (The way that Netbatch parses arithmetic expressions means that without parentheses, unexpected results can occur.)

- ◆ A logical operator: &&, ||, or !
- ◆ A comparison operator: <, <=, >, >=, !=, or ==

**Note**

For strings, == is not case-sensitive. If you need to do a case-sensitive comparison, use the equals() function instead.

- ◆ The pattern-matching operator: =~. The syntax of the expression that follows this operator is the same as for glob(), below. It also matches partial strings. And it works with --history. (glob() does not.)

Note that with --history, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with --history, you must use * as a wildcard instead of .* without --history.

Note

=~ is case-sensitive.

- ◆ A non-strict operator: is or isnt
- ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
- ◆ A conditional operator: ? or :
- ◆ **value** is a number, a string, or another boolean expression.
If value is a date/time, it must be in one of the following formats:
 - ◆ MM/DD/YYYY-HH:mm:ss
 - ◆ MM/DD/YYYY-HH:mm
 - ◆ MM/DD/YYYY

Expressions can be grouped together using parentheses.

Note

A double equals sign (==) is used. != (NOT EQUALS) may also be used.

The expression must be enclosed in double quotes.

If value is a string, it must be enclosed in single quotes.

Operator precedence is as follows (highest first) - parentheses, NOT, AND, OR.



If **specification** is **all**, all records are displayed.

Functions can also be used in **specification**. To see which functions are available for **nbstatus workstations**, use **nbstatus functions** with the **--context "nbstatus workstations"** switch. For a full list of all the available functions (with descriptions), see [Appendix H: Functions](#).

For example:

- ◆ **all** displays details of all Workstations.
- ◆ **"Server=='inbm006'"** displays details of the Workstation whose hostname is **inbm006**.
- ◆ **"Status=='Blackhole'"** only displays details of blackhole machines.
- ◆ **"Version=='6.4'"** displays details of all Workstations that are running version 6.4 of Netbatch.
- ◆ **"Version=='6.4'&&Status=='Blackhole'"** displays details of all blackhole machines that are running version 6.4 of Netbatch.

Fields: workstations

The following fields are available (default fields are marked with a *****):

- ◆ **AdminAttributes** - Any attributes defined for the Workstation by the Netbatch Administrator
- ◆ **Arch** - Architecture (e.g., **x86**)
- ◆ **AutoRestart** - whether the Workstation is configured so that the WSM service will automatically be restarted if it fails
- ◆ **BuffersRM** - the amount of physical RAM used for file buffers in MB
- ◆ **CPUConsumption** - the CPU consumption of the WSM process
- ◆ **CPUCount** - number of CPUs
- ◆ **CPUMhz** - CPU speed in MHz
- ◆ **CPUMips** - the MIPS (millions of instructions per second) value for the Workstation (Linux only)
- ◆ **CachedRM** - the amount of physical RAM used as cache memory in MB. Memory in the pagecache (diskcache) minus SwapCache (the amount of Swap used as cache memory)



- ◆ **CanRun** - the number of Jobs that the Workstation can currently run
- ◆ **Classes** - Supported Classes
- ◆ **Concurrency** - the number of Jobs that can run concurrently on the Workstation, according to its configuration.
- ◆ **CurrentConcurrency** - the number of Jobs that can currently run concurrently on the Workstation. If WSM auto-tuning is enabled, this may be different from the value of **Concurrency**.
- ◆ **ExternalProbeData** - External probe data for this Workstation. The value of this field is in the following format:

```
( <probe_name>_<param_name>=<param_val> ,  
limit={<limit_param_name>|<limit_value>} )[ ,  
... ]
```

where:

- ◆ **probe_name** is the name of the probe.
- ◆ **param_name** is a parameter name.
- ◆ **param_val** is the parameter's value.
- ◆ **limit_param_name** is the corresponding limit parameter (if any).
- ◆ **limit_value** is the corresponding limit value (if any)

You can use this information in a class expression (**nbjob run --class**).

- ◆ **fds** - Free disk space (in MB) in partitions on which particular configured directories reside
- ◆ **fRM** - free real memory in MB, the amount of physical RAM left unused by the system
- ◆ **FullHost** - the Workstation's fully-qualified host name
- ◆ **fVM** - free virtual memory in MB, the total amount of swap memory free
- ◆ **HW** - Hardware (e.g., **i686**)
- ◆ **Host** - Workstation host name
- ◆ **HungJobs** - IDs of hung Jobs
- ◆ **HungJobsCount** - number of hung Jobs
- ◆ **Hyperthreaded** - Whether the CPU(s) is a hyperthreading CPU (**true/false**)
- ◆ **IP** - Workstation's IP address



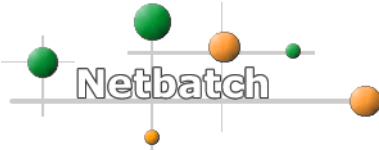
- ◆ **Idle*** - These two numbers define an active user.

The Workstation does not accept new Jobs unless the user has been inactive for the number of minutes defined by the first number.

The Workstation suspends running Jobs if the user is active and they only resume when the user has been inactive for the number of minutes defined by the second number.
- ◆ **InactiveRM** - the total amount of buffer or page cache memory in MB that is free and available; this is memory that has not been recently used and can be reclaimed for other purposes by the paging algorithm
- ◆ **InteractiveUsers*** - Number of users - active users/active user threshold (if exceeded, no new Jobs are accepted and running Jobs are suspended)

Note that the owner of the Job is not counted as an interactive user on the Workstation that the Job was dispatched to.
- ◆ **Jobs*** - the number of the Jobs on the Workstation (in any state--running, suspended, deleting, etc.)

See **RunningJobsCount** for the number of Jobs in the run state on the Workstation.
- ◆ **KernelVersion** - the kernel version (e.g., 2.4.1)
- ◆ **Last Select*** - the date/time that the last Job was sent to the Workstation
- ◆ **LastSelect** - the date/time that the last Job was sent to the Workstation
- ◆ **LastUpdate** - time of last status update (UNIX time)
- ◆ **LastHupTime** - the date/time the workstation was last HUPped
- ◆ **Load*** - Load - current load (average of the number of running processes 1 minute, 5 minutes, and 15 minutes ago)/lower load threshold (if exceeded, no new Jobs are accepted)/upper load threshold (if exceeded, running Jobs are suspended/niced)
- ◆ **LockReason** - the reason why the Workstation is locked (if locked)
- ◆ **MachineCanRun** - sum of **CanRun** for each Resource Set that the Workstation belongs to (but taking into account any Workstation capabilities/reservation that limit this to a lower number)



- ◆ **MachineCost** - the Workstation's assigned cost (Only affects scheduling decisions if the Qslot that the Job was submitted to uses cost fair-share scheduling. See [Section 5.1.3.6, Cost Fair-share Resource Allocation](#))
- ◆ **MachineUtilization** - machine utilization (percent)
If the number of running Jobs is greater than or equal to the number of CPUs, this is 100.

Note

*The **BuffersRM**, **CachedRM**, and **InactiveRM** attributes are available only for SUSE Linux Enterprise Server 9 and later.*

Note

For hyperthreaded CPUs, a multiplier is applied to the number of CPUs. By default the multiplier is 1.

If **MachineCanRun** is zero, this is 100.

Otherwise, it is the number of running Jobs divided by the lower of:

- ◆ The number of CPUs
- ◆ The sum of the number of running Jobs and **MachineCanRun**
- ◆ **NFactor** - Normalization factor for the Workstation
- ◆ **NLoad** - Normalized load (the same as **Load**, except weighted to take account of the Workstation's capabilities)
- ◆ **NicedJobs** - IDs of Jobs currently niced
- ◆ **NicedJobsCount** - Number of niced Jobs currently on the Workstation
- ◆ **NotAcceptingQueues** - the Queue(s) for which the Workstation is not accepting Jobs
- ◆ **NotAcceptingReason** - The reason why the Workstation cannot accept any Jobs:
 - ◆ **HighLoad** - load is above the defined threshold
 - ◆ **FullyInteractive** - the Workstation is not accepting batch Jobs because the number of interactive users is above the defined threshold



- ◆ **LowVirtualMemory** - the Workstation's virtual memory is below the defined threshold
- ◆ **Locked** - the Workstation is locked
- ◆ **Dedicated** - the Workstation is running a Job that was submitted with a class reservation requirement of **dedicated=true**
- ◆ **Blackhole** - the Workstation has been identified as a blackhole machine
- ◆ **blackhole(<configuration_name>)** - the Workstation has been identified as a blackhole machine because it matches the specified blackhole definition.
- ◆ **Max jobs reached** - the Workstation's maximum Jobs limit has been reached
- ◆ **GShareJobs** - the Workstations's **GShareJobs** limit has been reached
- ◆ **Locked by exclusive job** - a Job from an exclusive Queue is running on the Workstation. Only Jobs from the same Queue can run on the machine.
- ◆ **No mapped qslots** - the Resource Group hat the Workstation belongs cannot be used by any Qslot.
- ◆ **Reservation** - all the Queues and Qslots that can send Jobs to this Workstation require some reservation (defined in their configuration) which currently cannot be satisfied by the Workstation.
- ◆ **Preempted resourceset** - the Workstation cannot accept Jobs because of Resource Set preemption (there are Jobs running from the preempting Resource Set). Resource Sets and Resource Set preemption are deprecated, so this reason should not occur.
- ◆ **OSDistributionName** - vendor (e.g., Red Hat, SuSE)
- ◆ **OSDistributionVersion** - distribution version (e.g., 7.2 for Red Hat)
- ◆ **OSImage** - the OS image (only relevant for dynamic workstations)
- ◆ **OSName** - OS name
- ◆ **OSRelease** - OS release (kernel version)
- ◆ **OSVersion** - OS version
- ◆ **On Since*** - Date/time since which the Workstation has been running



- ◆ **OnSince** - Date/time since which the Workstation has been running
- ◆ **Owner** - the machine's configured owner
- ◆ **PatchLevel** - the Workstation's patch level
- ◆ **RANK** - rank value(s) (according to which Workstations are ranked by availability to run the next Job)
- ◆ **Reservation** - Resources that have been reserved on the Workstation for specific Jobs.

For execution slots that have been reserved by Parallel Jobs, this field shows whether the status of the execution slot(s) is **Inclusive** (marked by one or more Parallel Jobs, but can still be used by other Jobs) or **Exclusive** (reserved for a particular Parallel Job).

It also shows the Job ID(s) of the Parallel Jobs (only one if its status is **Exclusive**, one or more if **Inclusive**).

If the slot has an owner, this is also indicated.

If a reservation of memory or of a custom Workstation capability has been made for a Job (because the Queue or Qslot to which it was submitted is one where such a requirement is applied automatically), this is also shown.

- ◆ **ReservedJobs** - Jobs for which resources have been reserved (Job ID, state, reservation string)
- ◆ **ReservedJobsCount** - Number of Jobs for which resources have been reserved
- ◆ **ResourceGroups** - The Resource Group(s) to which the Workstation belongs
- ◆ **Resourceset** - The Resource Set(s) to which the Workstation belongs
- ◆ **RunAwayJobs** - IDs of Runaway Jobs
- ◆ **RunAwayJobsCount** - Number of runaway Jobs currently on the Workstation
- ◆ **RunningJobs** - IDs of Jobs currently running
- ◆ **RunningJobsCount** - Number of Jobs currently running on the Workstation
- ◆ **Sel*** - Number of Jobs sent to the Workstation since the Pool started running
- ◆ **Server*** - Workstation host name
- ◆ **Status*** - Current status:



- ◆ **Disc** - Disconnected (Workstation has been brought down intentionally)
 - ◆ **Down** - Down (Workstation failed for some unknown reason)
 - ◆ **Blackhole** - Blackhole
 - ◆ **Locked** - Locked
 - ◆ **Dedicated** - Workstation is running a Job that prevents other Jobs from running on the same Workstation
 - ◆ **Accepting** - Can accept Jobs
 - ◆ **Lowvmem** - Workstation has low virtual memory
 - ◆ **Running** - Workstation has running Jobs and cannot accept any more
 - ◆ **Reserved** - Workstation has only reserved Jobs (no running Jobs) and cannot accept any more Jobs
 - ◆ **Not accepting** - Workstation cannot accept Jobs for some other reason
 - ◆ **SupportedApplications** - whether the Workstation supports other applications, such as NiSD (**true** or **false**)
 - ◆ **SupportedPools** - list of Pools that the Workstation belongs to
 - ◆ **SupportedResourceSets** - Resource Set(s) of which the Workstation is a member
 - ◆ **SuspendedJobs** - IDs of Jobs currently suspended
 - ◆ **SuspendedJobsCount** - Number of suspended Jobs currently on the Workstation
 - ◆ **SwapRead** - number of swap pages read (from the **vmstat si** field)
 - ◆ **SwapWritten** - number of swap pages written (from the **vmstat so** field)
 - ◆ **tDS** - Total disk space (in MB) in partitions on which particular configured directories reside
 - ◆ **fDS** - Total free disk space (in MB)
 - ◆ **TotalMaxAllowedSessions** - the maximum allowed number of sessions for this Workstation
- If Workstation auto-tuning is enabled, this is set by the auto-tuning mechanism.



If auto-tuning is not enabled, this either takes the value of the limit set by the Netbatch Administrator or if no such limit is set, it is unlimited.

- ◆ **tRM** - total real memory; the total usable RAM in MB (i.e. physical memory minus a few reserved bytes and the kernel binary code) (number)
- ◆ **tVM** - total virtual memory in MB; the total amount of physical swap memory (number)
- ◆ **VMThreshold** - the virtual memory threshold, as specified by **Wvmem** or **Gvmem** (If VM drops below this threshold, the Workstation does not accept new Jobs.)
- ◆ **Version** - Version of Netbatch running on the Workstation
- ◆ **Virtual** - whether the Workstation was started in simulation mode or not.
- ◆ **VmCapable** - whether the Workstation can run virtual machines
- ◆ **WIgnoreThresholdsResources** -
- ◆ **wSCapabilities** - any Workstation capabilities defined for the Workstation. Note that Netbatch assigns memory and cores capabilities automatically, even if they are not explicitly defined in **wscapabilities**.
- ◆ **wSCapabilitiesAvailable** - the amount of defined Workstation capabilities that are available, **including** ones being used by preemptable Jobs
- ◆ **wSCapabilitiesAvailableWithoutPreemption** - the amount of defined Workstation capabilities that are available, **excluding** ones being used by preemptable Jobs
- ◆ **wSCapabilitiesReserved** - Workstation capabilities that are currently reserved by Jobs, **excluding** preemptable Jobs
- ◆ **wSCapabilitiesReservedWithPreemptables** - Workstation capabilities that are currently reserved by Jobs, **including** preemptable Jobs
- ◆ **wsMLite** - Whether the Workstation is running WSM Lite (**true**) or the standard WSM service (**false**)

Output

The **nbstatus workstations** output depends on:

- ◆ The fields selected for display
- ◆ The selected output format



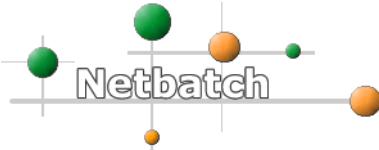
- ◆ The specified sort order
- ◆ The specified grouping
- ◆ The specification used to filter the results

Figure 30, below, shows sample output for the following command:

```
nbstatus workstations
--fields server,status,load,jobs
```

```
Pool itstl107 on itstl107
Version 6.4_w_0115
On since 02/17/2004 14:54:10
Time now 02/17/2004 15:29:16
-----
Server    Status      Load        Jobs
-----
itstl107 Accepting  0.03/4.00/4.80  1
-----
```

Figure 30: nbstatus workstations Output



A.4 nbauth

Name

nbauth - Store user login(s)/password(s) so that Netbatch can securely pass them to a Windows Workstation on which the user's Job is to run.

Description

The user must run **nbauth** (from UNIX only, not Windows) to store logins and passwords so that Netbatch can securely pass them to a Windows Workstation to which the user's Job is dispatched.

nbauth prompts the user for login name (<domain>\<login_name>) and password. Multiple login/password pairs can be entered. It stores these in an encrypted form in the user's home directory.

Usage

```
nbauth [--auth-file <file>]
```

Command-Line Options

--auth-file <file>

Specifies that the file containing the encrypted login/password pairs is to be located in the file specified by **path**, instead of in the user's home directory.



A.5 nbdepend

Name

nbdepend - Specify Job dependencies.

Description

Use to maintain a definite order of execution of Job tasks. To learn more about Job dependencies see [Section 6.6, Working with Job Dependencies](#).

Usage

```
nbdepend [options] <dependentJobId> <triggeringJobId>
```

Command Line Options

-d

Print debug information

-h

Displays help information.

-H <hostName>

Add dependencies between Jobs from master <hostName>

-P <poolName>

Add dependencies between Jobs from Pool <poolName>

-v

Print version number and release date

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.



If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

<dependentJobId>

Job ID of the Job which will become dependent on the Trigger Job. The Dependent Job cannot start execution until the Trigger Job has finished execution.

<triggeringJobId>

Job ID of the Job whose successful execution will trigger dependent Job. If a triggering Job fails due to a Netbatch error then dependent Jobs are suspended.

triggeringJobId can be a task name. If a task name is used, the dependent Job will not be dispatched for execution until all the Jobs that make up the task have ended.

Diagnostics

The command exit codes are listed below.

Note

Not all error codes are relevant for this command.

Note

These are command exit codes, not Job exit codes.

0

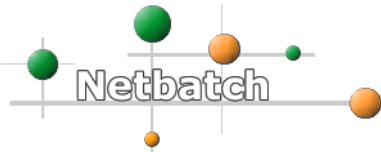
Successful execution

1

Specified Pool Master (**-P**) does not exist in **pools** file (older commands only (**nbq**, **nbqrm**, **nbstat**, etc.))

2

Unknown host specified (**--target** for newer commands, **-H** for older commands)

**3**

Error occurred on master side (for example, PPM error, Job could not be submitted because of missing permissions, non-existent class, etc.)

4

Problem getting user credentials (most likely an NIS issue on the client machine)

5

PPM timeout (The command waits one minute for a response.)

212

Depends on context:

- ◆ CSD error - Job failed
- ◆ Status command - invalid displayed field

213

Depends on context:

- ◆ CSD error - bad parameters
- ◆ Status command - invalid filter

214

Depends on context:

- ◆ CSD error - Workstation not available
- ◆ Status command - invalid grouping

215

Depends on context:

- ◆ CSD error - RG not available
- ◆ Status command - invalid sort

216

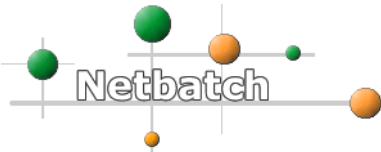
CSD error - Qslot not available

221

Interrupted

222

Permission denied

**223**

Timeout

224

Unknown error

225

Permissions problem

229

The command was run with the **-v** switch (version)

230

The command was run with the **-h** switch (help)

231

Parsing error - mandatory filter not specified

232

Parsing error - invalid value

233

Parsing error - value not specified

234

Parsing error - undefined parameter

235

Parsing error - mandatory parameter not specified

244

No command handler in this context

245

No command handler

248

Communication problem - target service down

249

Communication problem - connection broken during receive stage

**250**

Server too busy (denial of service response or built-in request limit reached)

251

The server does not support the command.

253

Communication problem - unknown protocol

254

Communication problem - no response

255

Master not responding (probably down)

A.6 nbexec

Name

nbexec - Execute a Parallel Job's Slave Jobs.

Note

nbexec is deprecated—use **nbjob prun** instead.



Description

`nbexec` executes a Parallel Job's Slave Jobs. When a Parallel Job is dispatched for execution, Netbatch adds all the hosts that the Slave Jobs are to run on to the **NB_PARALLEL_JOB_HOSTS** environment variable (in a space-delimited list).

The Parallel Job's Master Job is responsible for reading the list of hosts in **NB_PARALLEL_JOB_HOSTS** and using `nbexec` to dispatch the Slave Jobs to these hosts.

`nbexec` sends each Slave Job to the Pool Master specified in the **NB_PARALLEL_POOL_HOST** environment variable for validation before it is executed on the specified host.

Caution

*Do not set **NB_PARALLEL_POOL_HOST** yourself.
Netbatch sets it as part of the scheduling process to
ensure system consistency.*

Note

The `nbexec` command can fail in the following circumstances:

- *The specified Workstation (host) is down.*
- *The (master) Job is being preempted.*
- *There are no more reservations (`nbjob prun` or `nbexec` has been called too many times).*
- *There is no Master Job attached to the Job. (That is, the Job was not sent by a valid Master Job.)*
- *No Workstation was specified.*



Usage

```
nbexec --host <host name> [options]
<slave job> <slave job arguments>
```

Command-line Options

nbexec supports a subset of **nbq**'s command-line options, plus the mandatory **--host** switch:

-d

Displays debugging information.

-E

Does not pass user environment variables with the Job.

The Job will be executed on remote machine without the user's environment variables.

-h

Displays help information.

-i

Submit an interactive Job. To instruct **nbexec** to handle signals locally, add the **--local-signals** switch. (Without **--local-signals**, Netbatch passes these signals to the remote process.)

-J <JobName>

Identifies a Job with the string **JobName**, and determines the name of the Job's log file (see **Output**, below).

-K <minutes>

Specifies that Netbatch should kill the Job if it is suspended for more than the specified number of minutes.

-M <minutes>

Specifies that Netbatch should email the user if the Job is suspended for more than the specified number of minutes.

-N

Instructs Netbatch to email the user when the Job ends.

-S

Instructs Netbatch to email the user when the Job starts.

**-v**

Displays the Netbatch version number and release date.

-y <soft limit:hard limit>

Specifies execution limits for the Job.

Note

Execution limits are rounded up to the nearest minute.

-z <soft limit:hard limit>

Specifies hung limits for the Job.

Note

Hung limits are rounded up to the nearest minute.

--block

Run in blocking mode (no more commands may be executed until the Job ends).

--daemon

For Parallel Jobs that use MPI, **--daemon** tells Netbatch that some of the processes started by the Parallel Master Job will be daemons (that is, processes that close stdin, stdout, and stderr, and continue to run), and that it should use Process Authentication Groups (PAGs) to monitor and control them.

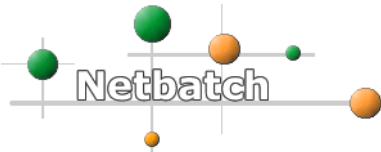
--host <host name>

Specifies the host that the Slave Job will be sent to.

--inherit

The Slave Job inherits the attributes of the Master Job. (By default, it does not.) The Slave Job does **not** inherit the following attributes:

- ◆ Job starter
- ◆ Command line
- ◆ Environment
- ◆ Current working directory
- ◆ Execution limits (-y) (if specified for Slave Job)
- ◆ Hung limits (-z) (if specified for Slave Job)



- ◆ -K, -R, -#, -S, -N switched (if specified)
- ◆ rlimits (if specified)
- ◆ Job name (if specified)
- ◆ Log file
- ◆ PAG option
- ◆ Interactive settings (logon, blocking, etc.)

--job-starter <exec>

Start the Job with a starter program.

--local-signals

(Only with -i (interactive)). Instructs `nbexec` to handle signals locally, instead of passing them to the remote process.

--log-file-dir <path>

Specifies that the Job's log file will be written in the specified directory.

If -j is used as well and its argument includes a path, **--log-file-dir** is ignored.

--pag on|nosuspend

Netbatch uses Process Authentication Groups (PAGs) to track Jobs. This switch specifies whether a PAG should be added to the Job, and how:

- ◆ **on** - add a PAG to the Job. This may override a Job Profile, or be overridden by a Job Profile, depending on how the Job Profile is set up.
- ◆ **nosuspend** - add a PAG to the Job, but do not suspend child processes that change their process group ID.

If this switch is not specified, and no Job Profile specifies that Netbatch should apply a PAG, Netbatch decides whether to apply a PAG based on how the Workstation is configured.

--post-exec <exec>

Execute a post-execution program after the Job runs

Note

The post-execution program is executed even if the Job is killed or suspended.

**--pre-exec <exec>**

Execute a pre-execution program before the Job runs

Note

The following environment variables contain information that the pre-exec script can use:

- **__NB_POOL**
- **__NB_QUEUE**
- **__NB_QSLOT**
- **__NB_LOG_FILE** (*but only if the -j option is used*)
- **__NB_CLASS**

--preserve-slot

Specifies that the execution slot in which the Slave Job is running will not be released when it ends, thus allowing the slot to be reused.

If **preserve_slot=true** for the Master Job (that is, **preserve_slot=true** was specified for the Master Job in **nbc**'s parallel reservation requirements string), then all execution slots are reused.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

**--rlimits "<res_limit> [soft|hard] = value
[,...]"**

Set resource consumption limits:

- ♦ **res_limit** is the name of the limit to be set.



- ◆ **soft | hard** (optional) specifies whether a soft or hard limit is to be set. If not specified, both hard and soft limits are set to the same value.
- ◆ **value** is the value to be assigned to the limit.

The limits that can be set are listed in the table below. Some are platform-specific. If a limit is set for a platform that does not support it, it is ignored.

If a limit is not set, the limit is inherited from the WSM process that spawned the process. The WSM process, in turn, inherits the limits from the shell from which it was invoked.

Note

*You can use the built-in **limit** command to view current resource limits. Note, however, that this shows the limits on the Workstation that you are logged into, not the Workstation on which your Job will run.*

*If you need to know the limits that will apply to your Job, run the **limit** command on a Workstation whose capabilities match those required by your Job.*

Table 3: Resource Limits

Limit	Units	Description	Platform
addressspace	KB	The maximum size of a process's available memory	All
coredumpsize	KB	The maximum size of a core file that a process may create	All
cputime	Seconds	The maximum amount of CPU time that a process may use	All
datasize	KB	The maximum size of a process's heap	All
filesize	KB	The maximum size of a file that a process may create	All
memorylocked	KB	Maximum locked-in memory address space	Linux
memoryuse	KB	The maximal resident set size of a process	Linux, HP-UX
openfiles	# of files	Maximum number of files that a process may open	All
maxproc	# of processes	The maximum permitted number of processes	Linux
stacksize	KB	The maximum size of a process's stack (can also be set to unlimited)	All
vmemoryuse	KB	The maximum size of a process's mapped address space	Solaris

**--silent**

Parallel Jobs that use MPI or Linda use the **rsh.nb** script instead of **rsh**. **--silent** suppresses the messages that Netbatch usually sends to stdout when **nbq** is run interactively (with the **-i** switch). This allows **rsh.nb** to mimic **rsh** as closely as possible.

--stack-size <stack size>**Note**

--stack-size is deprecated. Use **--rlimits** instead (see above).

Specifies the stack size limit for the Job on the Workstation (in KB). See [Section 6.5.20, Specifying Resource Limits](#). If not specified, the stack size limit that applies to the **nbq** command, will apply to the Job as well.

--task-name <taskname>

Specifies the Job task name.

--terminal

Run in terminal emulation mode.

Note

As of version 6.3.1, Netbatch automatically senses the context in which an interactive or terminal Job is being executed, and runs it as either an interactive or terminal Job, whichever is appropriate.

Thus, there is no need to use **--terminal**. Whenever you want the Job to run as if it is running locally, use **-i**.

Output

For each executed Job (including the Master Job), Netbatch creates a separate output log file, in the standard format (#**##Date-Time#**.<machine>). This can make it difficult to work out which log files belong to your Parallel Job.

You are therefore recommended to do one of the following:

- ◆ Use the **-j** switch (that is, use **-j <jobname>**) when you submit a Job. If you do this:
 - ◆ The Master Job log file is called **jobname**.



- ◆ The Slave Job log files are called **jobname:index**.
- ◆ Use the **-j** switch (that is, **-j <jobname>**) with **nbexec** in the Master Job (where each Slave Job is executed by running a separate **nbexec** command). In this case, the Slave Job log file is called **jobname**.

A.7 nbkrb renew

Name

nbkrb renew - Renews Kerberos ticket-granting tickets (TGTs).

Description

nbkrb renew is used to attach a new TGT to Jobs, even if these Jobs were already submitted and are waiting. It is used when you need to ensure that a Job requiring valid authentication credentials has them. Users must acquire a TGT before running this command (see [Section 6.1.6.1, Acquiring Kerberos TGTs](#)).

Usage

```
nbkrb renew [--target <host name> [:<port>]]
```

Output

```
Kerberos credentials successfully renewed
```

A.8 nblicense

Name

nblicense - Displays configured licenses.

Note

nblicense is deprecated—use **nbstatus licenses** instead.



Description

Used to display the configured licenses in the Pool. To learn more about Jobs that require licenses, see [Section 6.8, Jobs That Require Licenses](#).

Usage

```
nblicense [option] [master(s) | pool(s)] ...
```

Command-line Options

-a

Display information for all Pools

-d

Displays debugging information.

-h

Displays help information.

-H <host name>

Specifies the Pool by the master machine name.

-p <PoolName>

Specifies the Pool by Pool name.

-v

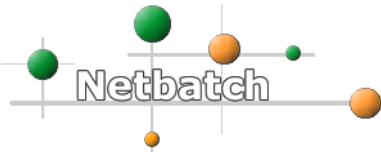
Displays the Netbatch version number and release date.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.



Output

nblicense displays a list with the following information:

- Vendor**—Vendor name
- Feature**—Vendor feature purchased and available
- Alias**—The feature's alias (if any)
- Capacity**—Maximum number of licenses for the feature
- Risk factor**—The factor that the Netbatch Administrator has assigned to the feature to indicate the number of Jobs that can concurrently use the same feature. (For example, if, on average, Jobs use the feature for half of their running time, this is set to 2.)
- In Use**—Number of licenses for the feature currently in use
- Key File**—Name of the keyfile for the specific feature
- Usage Factor**—Enables the user to see how many Jobs will be concurrently scheduled in the Netbatch system at any time (=Usage Factor x Capacity).

Diagnostics

The command exit codes are listed below.

Note

Not all error codes are relevant for this command.

Note

These are command exit codes, not Job exit codes.

0

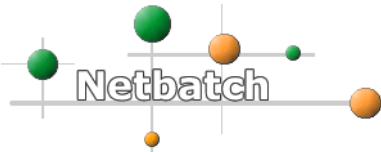
Successful execution

1

Specified Pool Master (**-P**) does not exist in **pools** file (older commands only (**nbq**, **nbqrm**, **nbstat**, etc.))

2

Unknown host specified (**--target** for newer commands, **-H** for older commands)

**3**

Error occurred on master side (for example, PPM error, Job could not be submitted because of missing permissions, non-existent class, etc.)

4

Problem getting user credentials (most likely an NIS issue on the client machine)

5

PPM timeout (The command waits one minute for a response.)

212

Depends on context:

- ◆ CSD error - Job failed
- ◆ Status command - invalid displayed field

213

Depends on context:

- ◆ CSD error - bad parameters
- ◆ Status command - invalid filter

214

Depends on context:

- ◆ CSD error - Workstation not available
- ◆ Status command - invalid grouping

215

Depends on context:

- ◆ CSD error - RG not available
- ◆ Status command - invalid sort

216

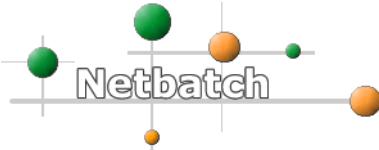
CSD error - Qslot not available

221

Interrupted

222

Permission denied

**223**

Timeout

224

Unknown error

225

Permissions problem

229

The command was run with the **-v** switch (version)

230

The command was run with the **-h** switch (help)

231

Parsing error - mandatory filter not specified

232

Parsing error - invalid value

233

Parsing error - value not specified

234

Parsing error - undefined parameter

235

Parsing error - mandatory parameter not specified

244

No command handler in this context

245

No command handler

248

Communication problem - target service down

249

Communication problem - connection broken during receive stage

**250**

Server too busy (denial of service response or built-in request limit reached)

251

The server does not support the command.

253

Communication problem - unknown protocol

254

Communication problem - no response

255

Master not responding (probably down)

A.9 nblicstat

Name

nblicstat - Displays configured licenses.

Note

nblicstat is deprecated—use ***nbstatus licenses*** instead.

Description

Used to display the configured licenses in the Pool in a Netbatch 5.x compatible format. To learn more about Jobs that require licenses see [Section 6.8 Jobs That Require Licenses](#).

Usage

```
nblicstat [options] [arguments]
```

Command Line Options**-a**

Display information for all Pools.

**-d**

Display debug information

-e

Show Netbatch 6.x extended info.

-h

Display help information

-H <host name>

Specifies the Pool by the Pool master machine.

-P <pool name>**-v**

Print version number and release date

Specifies the Pool by the Pool name.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

Output

nblicstat displays a list with the following information:

LicenseName—Vendor feature purchased and available

cost—The cost defined for the feature.

allocation—Number of license of features available for Netbatch

Reserved—Number license of features reserved out of Intact

Used—Number of license of the feature currently in use

Available—Number of license of the feature currently available

Required—Number of Jobs that require the features

blocked—Number of Jobs that are waiting for the feature



Diagnostics

The command exit codes are listed below.

Note

Not all error codes are relevant for this command.

Note

These are command exit codes, not Job exit codes.

0

Successful execution

1

Specified Pool Master (**-P**) does not exist in **pools** file (older commands only (**nbq**, **nbqrm**, **nbstat**, etc.))

2

Unknown host specified (**--target** for newer commands, **-H** for older commands)

3

Error occurred on master side (for example, PPM error, Job could not be submitted because of missing permissions, non-existent class, etc.)

4

Problem getting user credentials (most likely an NIS issue on the client machine)

5

PPM timeout (The command waits one minute for a response.)

212

Depends on context:

- ◆ CSD error - Job failed
- ◆ Status command - invalid displayed field

**213**

Depends on context:

- ◆ CSD error - bad parameters
- ◆ Status command - invalid filter

214

Depends on context:

- ◆ CSD error - Workstation not available
- ◆ Status command - invalid grouping

215

Depends on context:

- ◆ CSD error - RG not available
- ◆ Status command - invalid sort

216

CSD error - Qslot not available

221

Interrupted

222

Permission denied

223

Timeout

224

Unknown error

225

Permissions problem

229

The command was run with the **-v** switch (version)

230

The command was run with the **-h** switch (help)

**231**

Parsing error - mandatory filter not specified

232

Parsing error - invalid value

233

Parsing error - value not specified

234

Parsing error - undefined parameter

235

Parsing error - mandatory parameter not specified

244

No command handler in this context

245

No command handler

248

Communication problem - target service down

249

Communication problem - connection broken during receive stage

250

Server too busy (denial of service response or built-in request limit reached)

251

The server does not support the command.

253

Communication problem - unknown protocol

254

Communication problem - no response

255

Master not responding (probably down)



A.10 nblog

Name**nblog****Description**

Netbatch Jobs can use the **nblog** command to:

- ◆ Record data in Netbatch Tracker (in addition to the data that Netbatch records automatically)
- ◆ Send data to the Feeder that it was submitted from
 - This data can be queried using **nbstatus jobs**. (The data is displayed in the **CustomAttributes** field.)
- ◆ Send data from a Parallel Slave Job to its Master Job.

To learn more about Netbatch Tracker, see [Section 11, Using Netbatch Tracker](#).

Note that **nblog** is called *from within the Job*.

Note

Data recorded in Netbatch Tracker is automatically collected by iBI. However, if you want it to be available through iBI, the field must be added to the relevant iBI schema.

*You can do this [here](#) (click **iBI Configuration**) if you have sufficient permissions. (If not, contact the iBI development team.)*

Note

Data is sent and is available on the Pool Master almost instantaneously. The data is available on the Virtual Pool Master (where applicable) after its next synchronization with the physical Pool Master.

Note

nblog was previously called **recordJobData**.



Usage

```
nblog [{workstation|pool|feeder}] [options]
```

Note

The default target is workstation.

Target

workstation

The data is sent to NB Tracker (via the Workstation, then via the Pool Master). This is the default.

pool

Only used by Netbatch Interactive (to send display data to the Pool).

feeder

The data is sent to the feeder that submitted the Job.

Command Line Options

-d

Print debug information.

-h

Display help information.

--attributes <key>=<value>[,...]

Specifies key/value pairs to be sent to the Feeder/Master Job.

--cluster <string>

Record the cluster name.

--context {job|task}

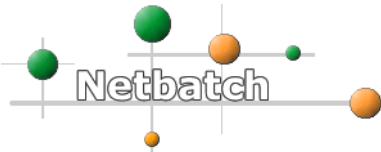
Specifies whether the data is to be logged at the Job level or the task level.

--custom-int-1 <value>

Record a custom integer value.

--custom-int-2 <value>

Record a custom integer value.



--custom-string-1 <value>

Record a custom string value.

--custom-string-2 <value>

Record a custom string value.

--cycles <cycles>

Record the number of validation (RTL) cycles for the Job.

--exit <PASS | FAIL | INTERNAL>

Record the user exit status.

--frequency <int>

Record the test frequency.

--id <id>

Specify a custom ID for the Job.

--master

Specifies that the data is to be recorded for the parallel Master Job instead of for the Slave Job (i.e., so that it will appear in the **CustomAttributes** field for the Master rather than for the Slave). (Parallel Slave Jobs only.)

--model <name>

Record the chip model that the Job is working on.

--multiplicity <int>

Record multiplicity (for example, the number of cores).

--reason <reason>

Record the reason for failure.

--simulator <string>

Record the simulator name.

--simulator-cpu <value>

Record the simulator CPU time.

--simulator-wtime <value>

Record the simulator wtime.

--title <title>

Specify a title for the Job.



A.11 nbpoolstat

Name

nbpoolstat - Displays the status of all the virtual resources in a virtual Pool.

Note

nbpoolstat is deprecated—use **nbstatus resources** instead.

Description

Displays the status of all the virtual resources (Queues) in a virtual Pool.
To learn more about virtual Pools see [Chapter 12 Increasing Throughput with Virtual Pools](#).

Usage

```
nbpoolstat [switches] [virtual pool] or [virtual pool manager]
```

Command Line Options

-c

Specifies that Class information should be displayed for each virtual resource.

-d

Displays debugging information.

-e

Add C/R/W/S info for each server

-h

Displays help information.

-H <host name>

Specifies the name of the manager machine of the Pool to show resource status for.

-j "<virtual_resource>[. . .]|=all

Display Jobs on specified virtual resources or all virtual resources.



-p <pool name>

Specifies the name of the virtual Pool to show resource status for.
(This is the default.)

-s

Specifies that only summary information should be displayed.

-v

Displays the Virtual Pool version number and release date.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

Output

nbpoolstat output consists of an entry for each resource that follows this format:

SERVER STATUS SEL LAST SELECT ON SINCE

Note

The reported resources are the virtual resources (Queues) that are assigned to the virtual Pool.

The output for each resource is described below.

SERVER

The name of the virtual resource.

STATUS

The resource status:

Blank



The resource is not accepting Jobs because the load or number of interactive users exceeds thresholds.

*

The resource can accept Jobs. If there are no other characters, it is not running any Jobs.

<n> R

The resource is running n Jobs.

<n> S

This resource has n suspended Jobs.

This resource has n niced Jobs.

* **<n> R**

The resource is running n Jobs, and can accept more.

L

The resource is locked. This does not affect Jobs that started running before it was locked.

C

The resource is coming up. (This takes about two minutes.)

D

The resource is disconnected (or wrong name specified).

SEL

The number of times the resource has been selected for Job execution since it was connected to the Pool manager.

LAST SELECT

The date and time that the resource was last selected to run a Job.

ON SINCE

The date and time that the resource was connected to the Pool manager.

Diagnostics

The command exit codes are listed below.

Note

Not all error codes are relevant for this command.

**Note**

These are command exit codes, not Job exit codes.

0

Successful execution

1

Specified Pool Master (**-P**) does not exist in **pool\$** file (older commands only (**nbq**, **nbqrm**, **nbstat**, etc.))

2

Unknown host specified (–**target** for newer commands, **-H** for older commands)

3

Error occurred on master side (for example, PPM error, Job could not be submitted because of missing permissions, non-existent class, etc.)

4

Problem getting user credentials (most likely an NIS issue on the client machine)

5

PPM timeout (The command waits one minute for a response.)

212

Depends on context:

- ◆ CSD error - Job failed
- ◆ Status command - invalid displayed field

213

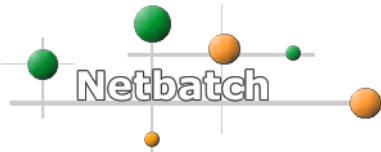
Depends on context:

- ◆ CSD error - bad parameters
- ◆ Status command - invalid filter

214

Depends on context:

- ◆ CSD error - Workstation not available



- ◆ Status command - invalid grouping

215

Depends on context:

- ◆ CSD error - RG not available
- ◆ Status command - invalid sort

216

CSD error - Qslot not available

221

Interrupted

222

Permission denied

223

Timeout

224

Unknown error

225

Permissions problem

229

The command was run with the **-v** switch (version)

230

The command was run with the **-h** switch (help)

231

Parsing error - mandatory filter not specified

232

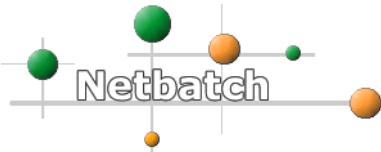
Parsing error - invalid value

233

Parsing error - value not specified

234

Parsing error - undefined parameter

**235**

Parsing error - mandatory parameter not specified

244

No command handler in this context

245

No command handler

248

Communication problem - target service down

249

Communication problem - connection broken during receive stage

250

Server too busy (denial of service response or built-in request limit reached)

251

The server does not support the command.

253

Communication problem - unknown protocol

254

Communication problem - no response

255

Master not responding (probably down)



A.12 nbprofile

Name

nbprofile - writes profiling information about the specified command to a file.

Description

For the specified command, **nbprofile** writes profiling (resource usage) information to a file at specified intervals.

Usage

```
nbprofile --file <filename>
[--interval <interval>] [--post-exec <script>]
<command>
```

Options

--file <filename>

Specifies the file (and path) to which the profiling information is to be written.

--interval <interval>

Specifies how often information is written to the file. The default is five seconds.

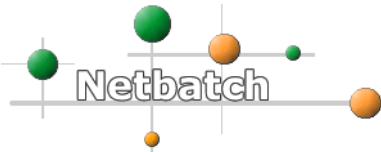
--post-exec <script>

Specifies that the specified script will be executed after **command** finishes running.

Output

The information written to the file is as follows:

- ◆ **currentTime** - the time when the information was written to the file
- ◆ **state** - the process state. It can be one of the following:
 - ◆ 1 - sleep
 - ◆ 2 - run
 - ◆ 3 - stopped
 - ◆ 4 - sleep_d



- ◆ 5 - zombie
- ◆ 6 - idle
- ◆ 7 - other
- ◆ **pid** - process ID
- ◆ **ppid** - parent process ID
- ◆ **pgid** - process group ID
- ◆ **commandName** - command name
- ◆ **utime** - user time
- ◆ **stime** - system time
- ◆ **realMem** - amount of real memory used
- ◆ **virtualMem** - amount of virtual memory used
- ◆ **minflt** - number of minor page faults (i.e., those with no disk access)
- ◆ **majflt** - number of major page faults (i.e., those with disk access)
- ◆ **max_rss(max_memory)** - maximum resident set size
- ◆ **ixrss** - integral shared memory size
- ◆ **idrss** - integral unshared data size
- ◆ **isrss** - integral unshared stack size
- ◆ **starttime** - process start time
- ◆ **bread** - number of blocks read (only displayed in totals line)
- ◆ **bwrite** - number of blocks written (only displayed in totals line)
- ◆ **pckti** - number of packets received (only displayed in totals line)
- ◆ **pckto** - number of packets sent (only displayed in totals line)
- ◆ **swpin** - number of virtual memory page reads from swap (only displayed in totals line)
- ◆ **swpout** - number of virtual memory page writes from swap (only displayed in totals line)
- ◆ **nfs getattr** - NFS **getattr** operation count (only displayed in totals line)
- ◆ **nfs setattr** - NFS **setattr** operation count (only displayed in totals line)



- ◆ **nfs_lookup** - NFS `lookup` operation count (only displayed in totals line)
- ◆ **nfs_access** - NFS `access` operation count (only displayed in totals line)
- ◆ **nfs_read** - NFS `read` operation count (only displayed in totals line)
- ◆ **nfs_write** - NFS `write` operation count(only displayed in totals line)

A.13 nbq

Name

nbq - Submits a Job to Netbatch for execution.

Note

nbq is deprecated—use [**nbjob run**](#) instead.

Description

Submit a Job to Netbatch to execute the given command. The Job can be any UNIX command, shell script, executable binary, almost anything that can be done at the shell prompt.

The Job will be placed in the Wait Queue, and later dispatched for execution based on machine availability and characteristics of the waiting Jobs, such as Qslot and class.

Note

If you submit a Job that requires authentication, make sure to acquire a ticket-granting ticket (TGT) before doing so. (See [**Section 6.1.6.1, Acquiring Kerberos TGTs**](#).) Renew the TGT as needed to ensure the Job has valid credentials (see [**Section 6.1.6.2, Renewing Kerberos TGTs**](#)) for running.



The results of the command execution (standard output and standard error) will be written to a user log file in the directory from which the `nbq` command was invoked. The file name format is `##<date>-<time>#.<machine>`, where `<date>` and `<time>` indicate when the Job starts executing, `<machine>` refers to the Workstation on which the Job executes. For example, `##Dec-22-21:08:55#.i1x126`. If a log file name is given with `-j`, it will be used instead.

If the directory where `nbq` is invoked is unavailable on the remote server, Netbatch will look for the environment variable **NBWD** and try to write the log file there. If **NBWD** is not defined either, a mail message indicating the problem will be sent to the user who submitted the Job. Note that if your site uses automounter you MUST use the **NBWD** variable. For example, if you are working in the directory `/export/cbs_02/pauls`, the `pwd` command will return `/a/export/cbs_02/pauls`. You will have to do `setenv NBWD /export/cbs_02/pauls`. The Job when executed will inherit all the user environment variables at the time of `nbq` invocation, unless the `-E` switch is used.

Note

If the WSM is down for some reason when the Job ends, writing the log file is not completed, as Netbatch does not have the required information. When the WSM comes back up, writing of the log file is completed.

Note

An `nbq` command-line option usually overrides an environment variable that has been set to specify the same parameter.

In most cases (but not all), it also overrides any default values that the Netbatch Administrator has set for the parameter.

See [**Appendix B: Precedence of Configuration Directives, Environment Variables, Job Profiles, and Command-Line Switches.**](#)



Usage

```
nbq [options] command [arguments]
```

Command-line Options

-C <class_expression>

Specifies the class of machine (that is, the required Workstation attributes) that the Job requires. These can include static attributes (such as OS type and number of CPUs) and dynamic attributes (such as free virtual memory and load).

Note

When you submit a job with a class requirement, Netbatch checks whether the requirement can ever be satisfied by any of the Workstations to which it can send the Job. (This might include remote Workstations.)

*For example, if your Job requires a Workstation with 1GB of free real memory, Netbatch checks whether any of the Workstations that it can send the Job to have **total** real memory of 1GB or more.*

If none of the local Workstations meet the requirement, Netbatch accepts the Job while it queries the remote Workstations (if any). If one (or more) of the remote Workstations meets the requirement, the Job will eventually be dispatched for execution.

If no remote Workstation meets the requirement, the Job will remain in the wait state queue until it is removed—it can never be dispatched for execution.

You are recommended to use nbjob run with the --validate switch to make sure that your Job's class requirements can be satisfied before you submit it.

Once Netbatch has performed the query, it caches the result. Subsequent Jobs with the same requirement will either be accepted (and later dispatched) or rejected immediately.

class_expression is an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- **<attribute>** is one of the available Workstation attributes or an external probe parameter name.

For a list of Workstation attributes, see [Section 6.7.2, Viewing Supported Workstation Attributes](#).



This includes total and free disk space on partitions on which directories defined by the Netbatch Administrator reside. Run **nbstatus workstations --fields all** and look at the values of the **fDS** and **tDS** fields to see the partitions whose disk space Netbatch monitors.

To see the available Workstation probes and the parameters that they measure, run **nbstatus workstations --fields all** and look at the value of the **ExternalProbeData** field.

To see the available Scheduler probes and the parameters that they measure, run **nbstatus probes**.

See [Appendix A.3.31, nbstatus workstations](#) for more details.

- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")
 - ◆ A logical operator: &&, ||, or !
 - ◆ A comparison operator: <, <=, >, >=, !=, or ==
 - ◆ A non-strict operator: **is** or **isnt**
 - ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
 - ◆ A conditional operator: ? or :
- ◆ **<value>** is a number, a string, or another boolean expression.

Expressions can be grouped together using parentheses.

-d

Displays debugging information.

-E

Does not pass user environment variables with the Job.

User Job will be executed on remote machine without user's environment variables.

-G <file>

Submits all the Jobs that are specified in the <file>

-h

Displays help information.

**-H <hostname>**

Specifies the name of the master machine for the Pool where the Job is to be run.

-i

Submit an interactive Job. To instruct `nbq` to handle signals locally, add the `--local-signals` switch. (Without `--local-signals`, Netbatch passes these signals to the remote process.)

-J <JobName>

Identifies a Job with the string `JobName`, and determines the name of the Job's log file.

-K <minutes>

Specifies that Netbatch should kill the Job if it is suspended for more than the specified number of minutes. This does not apply when the Job is suspended by the owner or the administrator.

-L "<expression>"

Specifies the license(s) that the Job will need during its execution, and (optionally) the amount of time each one is required. This overrides any license requirements defined in the user's Job profile file or **NBLICENSES** environment variable.

`expression` consists of any number of `<req_expr>`s and `<operator>`s.

`req_expr` is a license requirement expression of the following format:

```
"<license>=<value>[:duration=<duration_time>]"
```

where:

- ◆ `license` is the required license.
- ◆ `value` is the number of instances of `license` that are required.
- ◆ `duration_time` is the amount of time (in minutes) for which the license is required (starting from when the Job is dispatched for execution).

If not specified, then `duration` is set to the Job execution time (that is, the license is reserved for the entire duration of the Job's execution).

`operator` is `&&` or `||`.

These can be grouped together using parentheses.

**-M <minutes>**

Specifies that Netbatch should email the user if the Job is suspended for more than the specified number of minutes.

-N

Instructs Netbatch to email the user when the Job ends.

-P <poolname>

Specifies the name of the Pool where the Job is to be run.

-Q <qslot>

Specifies the full path or alias of the Qslot the Job should be sent to.

-R <minutes>

Specifies that Netbatch should resubmit the Job if it is suspended for more than the specified number of minutes. (If it is suspended again, Netbatch kills it.)

Note

*This does **not** apply to Jobs that are resubmitted by the Netbatch Administrator or by a user with Queue/Qslot administrative rights (using **nbjob suspend** or **nbqrm --suspend**).*

-# <number>

If **-R** is used, specifies the number of times to resubmit the Job before killing it.

-S

Instructs Netbatch to email the user when the Job starts.

-T "<JobID>[[<exit_status_expression>]] [. . .]"

Similar to **--triggers**, but instead of accepting any boolean expression as an argument, this switch accepts a space-delimited list of expressions, each of which consists of a Job ID and (optionally) an exit status expression.

All the specified Jobs must have ended (and their exit status expressions must be true, if specified), before the Job can run.

-V

Displays the Netbatch version number and release date.



-Y <soft limit:hard limit>

Specifies execution limits for the Job.

Note

Execution limits are rounded up to the nearest minute.

-Z <soft limit:hard limit>

Specifies hung limits for the Job.

Note

Hung limits are rounded up to the nearest minute.

```
--autoreq "attempts=<#_attempts>:<expression>" |  
<#_attempts>:<exit_code>[ ,... ]
```

Note

This switch is deprecated. Use --on-job-finish instead.

Specifies that the Job is to be resubmitted (up to #_attempts times) if it ends with particular exit codes.

If this form is used:

```
<#_attempts>:<exit_code>[ ,... ]
```

then if any of the specified exit codes occur, the Job is resubmitted.

If this form is used:

```
attempts=<#_attempts>:<expression>
```

then the Job is resubmitted if **expression** is true. **expression** is an arbitrarily complex boolean expression that can consist of any number of terms of the form **exit <relational_operator> <exit_code>**, separated by the logical operators **&&** (AND) and **||** (OR), and together with **!** (NOT). **relational_operator** can be **>**, **<**, **>=**, **<=**, or **==**.

The following macros may be used instead of exit codes:

- ◆ **NBErr** - All Netbatch errors except -8, -9, -10, and -11
- ◆ **Nexit** - All negative Job exit values
- ◆ **Pexit** - All positive Job exit values
- ◆ **NZexit** - All non-zero Job exit values

**For Example**

The following are equivalent:

```
--autoreq attempts=4:-8
--autoreq attempts=4:exit==8
```

--block

Run in blocking mode (no more commands may be executed until the Job ends).

--class-reservation <reservation_item>[,...]

Specifies the class requirements that the Job will need during execution. Netbatch reserves the required resources, thus preventing the situation where two or more Jobs successfully begin executing, but hang because their increasing resource requirements exceed the capabilities of the machine.

reservation_item must be in the following format:

```
"<key>=<key_value>
[:duration=<duration_time>][{h|m|s}]
[:decrease=<decrease_time>]
[:delta=<delta_value>]"
```

where:

- ◆ **key** is the type of resource to be reserved. It can take the following values:
 - ◆ **fVM**
 - ◆ **fRM**
 - ◆ **dedicated**

Note

If **dedicated=true**, then no other Jobs may run on the host while the Job is running (except for Parallel Slave Jobs—multiple Slave Jobs that are part of the same Parallel Job can run on the same host).

- ◆ The name of a custom capability or attribute that has been defined for one or more Workstations in the Pool

You can use the **nbstatus workstations** command to see a list of these attributes:

```
nbstatus workstations --fields
"*,WSCapabilities"
```



These capabilities can either relate to total real or virtual memory or be arbitrary.

Note

Workstation capability requirements can also be defined at the Queue or Qslot level. Such requirements apply to all Jobs submitted to the Queue/Qslot.

Note

If you specify a requirement for a Workstation capability that does not exist in the Pool, it will stay in the wait state indefinitely. Make sure that the capability exists before submitting the Job. (Use `nbstatus workstations`, as described above.)

- ♦ An external probe parameter name

To see the available Workstation probes and the parameters that they measure, run `nbstatus workstations --fields all` and look at the value of the `ExternalProbeData` field.

To see the available Scheduler probes and the parameters that they measure, run `nbstatus probes`.

See [Appendix A.3.31, nbstatus workstations](#) for more details.

- ♦ `key_value` is the value for `key`. It must be one of the following:
 - ♦ For `fvm` and `frm`, or a custom field, any type of number–integer or non-integer value
 - ♦ For `dedicated`, `true` or `false`
- ♦ `duration_time` is the amount of time (in hours, minutes, or seconds) for which the resources (`fvm` and `frm` only) are required (starting from when the Job is dispatched for execution). If you do not specify `h`, `m`, or `s`, the time is in minutes.

If not specified, then `duration` is set to the Job execution time (that is, the resource is reserved for the entire duration of the Job's execution).



- ◆ **decrease_time**—During the time specified by **duration**, the committed resource (**fVM** and **fRM** only) is reduced every **decrease_time** minutes by the amount specified in **delta_value**.
If a **delta_value** is specified, but no **decrease_time**, then **decrease** = 1 minute.
If neither a **delta_value** nor a **decrease_time** is specified, then the resource is available at the same level for the whole time specified by **duration**.
- ◆ **delta_value**—During the time specified by **duration**, the committed resource (**fVM** and **fRM** only) is reduced every **decrease_time** minutes by the amount specified in **delta_value**.
If a **decrease_time** is specified, but no **delta_value**, then **delta** = 1 unit.
If neither a **delta_value** nor a **decrease_time** is specified, then the resource is available at the same level for the whole time specified by **duration**.

--copy-arg-file <path>

A file containing arguments to be passed to **rsync**.

This switch is only relevant when using the file copying feature to copy files or directories to a location accessible by Workstations to which the Job might be dispatched. Use this feature when Workstations in the Pool to which you are submitting the Job have no access to the current location of files required by the Job. For example:

- ◆ When submitting a Job to a cross-site Virtual Pool
- ◆ When submitting a Job directly to a physical Pool at a different site
- ◆ When the required files or directories are on a local filesystem

The arguments in the specified file override the default arguments that Netbatch uses, which are as follows:

```
-e ssh -azx --rsync-path=/usr/intel/bin/rsync
```

Alternatively, you can specify these arguments on the command line with the **--file-copy-command-args** switch (see below).

See also **--file-copy-in**, **--file-copy-out**, and **--file-delete**.

--cr <reservation_item>[,...]

See **--class-reservation**.

**--daemon**

For Parallel Jobs that use MPI, **--daemon** tells Netbatch that some of the processes started by the Parallel Master Job will be daemons (that is, processes that close stdin, stdout, and stderr, and continue to run), and that it should use Process Authentication Groups (PAGs) to monitor and control them.

--dumpRusage <interval>:<path>

Specifies that Netbatch should write resource usage (rusage) information to the file at **path** every **interval** seconds. If the specified file already exists, it is appended to.

Rusage information is recorded for the Job itself and for the Job leader process (**nbjobleader.out**). The totals for both of these combined are also displayed.

The information written to the file is as follows:

- ◆ **currentTime** - the time when the information was written to the file
- ◆ **state** - the process state. It can be one of the following:
 - ◆ 1 - sleep
 - ◆ 2 - run
 - ◆ 3 - stopped
 - ◆ 4 - sleep_d
 - ◆ 5 - zombie
 - ◆ 6 - idle
 - ◆ 7 - other
- ◆ **pid** - process ID
- ◆ **ppid** - parent process ID
- ◆ **pgid** - process group ID
- ◆ **commandName** - command name
- ◆ **utime** - user time
- ◆ **stime** - system time
- ◆ **realMem** - amount of real memory used
- ◆ **virtualMem** - amount of virtual memory used
- ◆ **minflt** - number of minor page faults (i.e., those with no disk access)
- ◆ **majflt** - number of major page faults (i.e., those with disk access)



- ◆ **max_rss(max_memory)** - maximum resident set size
- ◆ **ixrss** - integral shared memory size
- ◆ **idrss** - integral unshared data size
- ◆ **isrss** - integral unshared stack size
- ◆ **starttime** - process start time
- ◆ **bread** - number of blocks read (only displayed in totals line)
- ◆ **bwrite** - number of blocks written (only displayed in totals line)
- ◆ **pckti** - number of packets received (only displayed in totals line)
- ◆ **pckto** - number of packets sent (only displayed in totals line)
- ◆ **swpin** - number of virtual memory page reads from swap (only displayed in totals line)
- ◆ **swpout** - number of virtual memory page writes from swap (only displayed in totals line)
- ◆ **nfs getattr** - NFS `getattr` operation count (only displayed in totals line)
- ◆ **nfs setattr** - NFS `setattr` operation count (only displayed in totals line)
- ◆ **nfs lookup** - NFS `lookup` operation count (only displayed in totals line)
- ◆ **nfs access** - NFS `access` operation count (only displayed in totals line)
- ◆ **nfs read** - NFS `read` operation count (only displayed in totals line)
- ◆ **nfs write** - NFS `write` operation count(only displayed in totals line)

--file-copy-command-args <arguments>

Specifies arguments to be passed to `rsync`.

This switch is only relevant when using the file copying feature to copy files or directories to a location accessible by Workstations to which the Job might be dispatched. Use this feature when Workstations in the Pool to which you are submitting the Job have no access to the current location of files required by the Job. For example:

- ◆ When submitting a Job to a cross-site Virtual Pool



- ◆ When submitting a Job directly to a physical Pool at a different site
- ◆ When the required files or directories are on a local filesystem

The specified arguments override the default arguments that Netbatch uses, which are as follows:

-e ssh -azx --rsync-path=/usr/intel/bin/rsync

Alternatively, you can specify these arguments in a file which you then specify with the **--copy-arg-file** switch (see above).

See also **--file-copy-in**, **--file-copy-out**, and **--file-delete**.

Note

Copying files between Windows and UNIX machines is not supported.

--file-copy-in <source>:<target>[, . . .]

Specifies the files or directories to be copied before the Job runs and their destination.

This switch is only relevant when using the file copying feature to copy files or directories to a location accessible by Workstations to which the Job might be dispatched. Use this feature when Workstations in the Pool to which you are submitting the Job have no access to the current location of files required by the Job. For example:

- ◆ When submitting a Job to a cross-site Virtual Pool
- ◆ When submitting a Job directly to a physical Pool at a different site
- ◆ When the required files or directories are on a local filesystem

You can use any wildcards that **rsync** accepts.

See also **--file-copy-out**, **--file-delete**, **--file-copy-command-args**, and **--copy-arg-file**.

Note

Copying files between Windows and UNIX machines is not supported.

**Note**

Please bear in mind that if you submit a large number of Jobs at once or a Job (or Jobs) that copy large amounts of data, it can cause serious WAN performance problems.

Note

When you use this feature to copy files from one domain to another, Netbatch checks WAN conditions for the target domain, and only copies the files if conditions are within defined thresholds.

If conditions are outside the defined thresholds, Netbatch does not copy the files (and therefore the Job cannot run) until conditions are within the defined thresholds. (Netbatch checks conditions periodically, with the period being defined by the Netbatch Administrator, unless the number of file copy requests is high, in which case it checks once for every n requests.)

--file-copy-out <source>:<target>[,...]

Specifies the file(s) to be copied after the Job ends and their destination.

This switch is only relevant when using the file copying feature to copy files or directories to a location accessible by Workstations to which the Job might be dispatched. Use this feature when Workstations in the Pool to which you are submitting the Job have no access to the current location of files required by the Job. For example:

- ◆ When submitting a Job to a cross-site Virtual Pool
- ◆ When submitting a Job directly to a physical Pool at a different site
- ◆ When the required files or directories are on a local filesystem

If you add **NBLOG** as one of the **source** arguments, Netbatch copies the Netbatch log file(s) to the specified destination.

You can use any wildcards that **rsync** accepts.

See also **--file-copy-in**, **--file-delete**, **--file-copy-command-args**, and **--copy-arg-file**.

--file-delete <path>[,...]

Specifies files or directories to be deleted after the Job ends.



This switch is only relevant when using the file copying feature to copy files or directories to a location accessible by Workstations to which the Job might be dispatched. Use this feature when Workstations in the Pool to which you are submitting the Job have no access to the current location of files required by the Job. For example:

- ◆ When submitting a Job to a cross-site Virtual Pool
- ◆ When submitting a Job directly to a physical Pool at a different site
- ◆ When the required files or directories are on a local filesystem

You can include wildcards in **path**.

Caution

If you specify a directory, Netbatch deletes the directory and its contents recursively.

See also **--file-copy-in**, **--file-copy-out**, **--file-copy-command-args**, and **--copy-arg-file**.

Note

Copying files between Windows and UNIX machines is not supported.

--incremental-log

Specifies that each time the Job is resubmitted, a new log file is created with an incremental suffix. (For example, if the original log file is called **myjob.log**, additional log files are called **myjob.log.1**, **myjob.log.2**, and so on.)

--job-constraints

"<expression>:<action>[:...][,...]"

Specifies that the Netbatch should take one or more actions (including stopping, restarting, or resubmitting the Job), depending on various Job-related factors (which it measures once a second).

expression is either an arbitrarily complex boolean expression or an expression of the form:

idle(<interval_in_sec>,<cpu_in_percent>)

This evaluates to true if CPU usage was below the specified percentage during the specified interval.



If **expression** is an arbitrarily complex boolean expression, it consists of <attribute>s, <operator>s, and <value>s, and optionally <functions>s, where:

- ◆ **attribute** is one of the following:
 - ◆ **RM** - Jobs real used memory (MB)
 - ◆ **VM** - Jobs virtual memory (MB)
 - ◆ **utime** - Job process user time (seconds)
 - ◆ **nutime** - normalized Job process user time (seconds)
 - ◆ **stime** - Job process system time (seconds)
 - ◆ **nstime** - normalized Job process system time (seconds)
 - ◆ **wtime** - Job process wall clock time (seconds)
 - ◆ **BuffersRM** - the amount of physical RAM used for file buffers (MB)
 - ◆ **CachedRM** - the amount of physical RAM used as cache memory; memory in the pagecache (diskcache) minus SwapCache (the amount of Swap used as cache memory) (MB)
 - ◆ **InactiveRM** - the total amount of buffer or page cache memory that is free and available; this is memory that has not been recently used and can be reclaimed for other purposes by the paging algorithm (MB)
 - ◆ **fRM** - Free real memory; the amount of physical RAM left unused by the system (MB)
 - ◆ **fVM** - Free virtual memory, the total amount of swap memory free (MB)
 - ◆ **tRM** - Total real memory; the total usable RAM (i.e. physical memory minus a few reserved bytes and the kernel binary code) (MB)
 - ◆ **tVM** - Total virtual memory; the total amount of physical swap memory (MB)
 - ◆ **fDS('<path>')** - Free disk space on the partition on which the specified directory resides (MB). Use **nbstatus workstations --fields "*,fds"** to see the directories that Netbatch monitors.
 - ◆ **users** - Number of active users
 - ◆ **load** - Workstation load
 - ◆ **Nload** - Normalized Workstation load
 - ◆ **nice** - the amount of time that a Job has been niced



- ◆ **susp** - the amount of time the Job has been suspended (in seconds)
- ◆ **waittime** - the amount of time that the Job has been waiting to be dispatched for execution (seconds)

Note

waittime is only checked every 60 seconds.

Be careful choosing the operator to use with **waittime**. Because Netbatch only checks it every 60 seconds, an expression that uses == is very unlikely ever to be true.

Some actions are irrelevant for **waittime**. The only applicable actions are **kill** and **mail**.

- ◆ **TimeInState('<state>')** - the amount of time that the Job is currently in the specified state (milliseconds)

Valid states are: **send**, **resched**, **approving**, **susp**, **resub**, **del**, **wait remote**, **wait**, **run**, and **disc**.

Note

If the specified state is **disc**, the only valid action is **force-kill**.

Note

The **BuffersRM**, **CachedRM**, and **InactiveRM** attributes are available only for SUSE Linux Enterprise Server 9 and later.

Note

Attribute names are case-sensitive.

- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., +10), or unary - (e.g., -10)
 - ◆ A logical operator: &&, ||, or !
 - ◆ A comparison operator: <, <=, >, >=, !=, or ==



- ◆ A non-strict operator: `is` or `isnt`
- ◆ A bitwise operator: `&`, `|`, `~`, `^`, `<<`, `>>`, or `>>>`
- ◆ A conditional operator: `?` or `:`
- ◆ `value` is a number, a string, or another boolean expression.
- ◆ `function` is one of:
 - ◆ `avgovertime('<time_interval>', '{RM|VM|RM+VM}')`

which returns the average of the sampled values of the specified parameter in `time_interval`.

- ◆ `deltaovertime('<time_interval>', '{utime|stime|utime+stime}')`

which returns the difference between the maximum and minimum values of the specified parameter during `time_interval`.

- ◆ `reservation('<capability>')`

which returns the value of the Job's reservation of the specified Workstation capability. (For example, if a Job reserved two of the capability called `cores`, `reservation("cores")` returns 2 for that Job.)

In both functions, `time_interval` is of the form `<number>[{s|m|h}]` (seconds, minutes, or hours). The default is minutes (so 60 means 60 minutes).

These are just the functions specific to Job constraints. For a full list of available functions, see [Appendix H: Functions](#), or run the following command:

```
nbstatus functions --context jobConstraints
```

Note

When one of these functions is used, the value of the expression that contains the function and its maximum value are displayed in the RUsage field in the output of `nbstatus jobs`.

For example, if you submit a Job with the following Job constraint:

`"AvgOverTime('60s','RM')>1:kill"`

then the `nbstatus jobs` RUsage field will include something like this:



```
avgovertime(60s;RM)= 0.0 , Max(avgovertime(60s;RM))= (0;64)
```

For **Max()**, the two numbers are the maximum (of the average or delta), and the time at which the maximum occurred (in seconds from the start of execution).

Expressions can be grouped together using parentheses.

action is one of the following:

- ◆ **kill**
- ◆ **force-kill**

This kills the Job, even if its state is **disc**. (See **TimeInState()**, above.)

- ◆ **mail**

Mail is sent using the template set up by the Netbatch administrator (or a default template if no such template exists).

- ◆ **mailonce**

This is the same as **mail**, except mail is only sent once (unless the Job is resubmitted).

- ◆ **recall**

- ◆ **nice**

When the condition is no longer true, the Job is reniced.

- ◆ **resubmit**

- ◆ **resubmit(<exit_status>)**

The same as **resubmit**, except that the Job is assigned the specified custom exit status.

- ◆ **suspend**

When the condition is no longer true, the Job is resumed.

- ◆ **resume**

- ◆ **nop**

No operation (for reporting)

- ◆ **kill(<exit_status>)**

The same as **kill**, except that the Job is assigned the specified custom exit status.

- ◆ **modify-reservation(<expression>)**

Modifies the Job's reservation according to **expression**. For example:

**modify-reservation(memory=memory+2)**

This Increases the Job's `memory` reservation by 2.

- ◆ **modify-properties(<attribute>=<new_value>[, . . .])**
Modifies the Job's custom attributes.
- ◆ **RunCommand(cmd=<command>)**

The specified command or script is executed on the Workstation where the Job is running.

The command has access to the following environment variables:

- ◆ `__NB_CLASS` - the class requirement with which the Job was dispatched
- ◆ `__NB_CMD` - the Job's command
- ◆ `__NB_DATE` - the date on which the email was sent
- ◆ `__NB_ERROR_MESSAGE` - the error message
- ◆ `__NB_EXIT_STATUS_CODE` - the Job's exit code
- ◆ `__NB_EXIT_STATUS_DESCRIPTION` - the Job's exit status description
- ◆ `__NB_JOB_CONSTRAINTS` - the Job constraints that apply to the the Job (as specified using `nbjob run`'s `--job-constraints` switch or in a Job profile)
- ◆ `__NB_JOB_ID` - the Job's ID
- ◆ `__NB_KILL_IF_SUSPENDED` - the number of minutes specified by the `nbjob run --kill` switch
- ◆ `__NB_MAIL_IF_SUSPENDED` - the number of minutes specified by the `nbjob run --mail __NB_min` switch
- ◆ `__NB_PGIN` - the Job's process group ID
- ◆ `__NB_PID` - the Job's process ID
- ◆ `__NB_POOL_ADMIN` - the ID of the Pool administrator (as specified by the `A` directive—see [Appendix E.2.11](#))
- ◆ `__NB_POOL_NAME` - the Pool name
- ◆ `__NB_QSLIST` - the Qslot from which the Job was dispatched
- ◆ `__NB_RESUBMIT_ATTEMPTS` - the number of resubmission attempts specified by the `nbjob run --trials` switch



- ◆ NB_RESUBMIT_IF_SUSPENDED - the number of minutes specified by the nbjob run --resubmit switch
- ◆ NB_RUSAGE - the Job's resource usage
- ◆ NB_STIME - Job process system time
- ◆ NB_TIME - the time at which the email was sent
- ◆ NB_USER_LOG_FILE - the path/filename of the user log file
- ◆ NB_USER_NAME - the Job owner's user
- ◆ NB_UTIME - Job process user time
- ◆ NB_WORKING_DIR - the user's working directory, from which the Job was submitted
- ◆ NB_WORKSTATION - the hostname of the Workstation on which the Job is running
- ◆ NB_WTIME - Job process wall clock time

Multiple actions can be specified for a single expression.

Multiple <expression>:<action>[: . . .] constraints can be specified.

Examples:

```
nbq --job-constraints "RM>500:mail:kill" myJob
If myJob's real memory usage exceeds 500MB, kill it and mail the user.
```

```
nbq --job-constraints "RM>500 | RM>tRM/2:mail,
RM>700:kill:mail" myJob
If myJob's real memory usage exceeds 500MB or half of total real
memory, mail the user. If its real memory usage exceeds 700MB, kill
it and mail the user.
```

```
nbq --job-constraints
"RM>500&&fRM<500:suspend:mail,
fRM>500:resume:mail" myJob
```

```
If myJob's real memory usage exceeds 500MB and free real
memory drops below 500MB, suspend it and mail the user. If free
real memory rises above 500MB again, resume myJob.
```

--job-starter <exec>

Start the Job with a starter program.



```
--license-keyfile "<keyfile path>|<port@host>[ :.... ]"
```

Specifies the license keyfiles to be used. (This is instead of setting the **NB_LICENSE_FILE** environment variable.) If multiple arguments are specified, use colons as delimiters.

Note

If you specify a license that the Job requires by its alias, you do not need to specify the license keyfile.

```
--license-reservation <reservation_item>[ ,.... ]
```

Specifies the license(s) that the Job will need during its execution, and the amount of time each one is required.

In earlier versions of Netbatch, this allowed license availability to be calculated much more accurately than just using the **-1** switch, which meant that license usage was much more efficient.

This functionality is now implemented automatically. All required licenses are automatically reserved.

If you use this switch, Netbatch automatically converts the specified requirement so that it is the same as if you used **-1**.

Each **reservation_item** string must be in the following format:

```
"<key>=<key_value>[:duration=<duration_time>]"
```

where:

- ◆ **key** is the feature to be reserved.
- ◆ **key_value** is the number of instances of **key** that are required.
- ◆ **duration_time** is the amount of time (in minutes) for which the feature is required (starting from when the Job is dispatched for execution).

If not specified, then **duration** is set to the Job execution time (that is, the feature is reserved for the entire duration of the Job's execution).

```
--local-signals
```

(Only with **-i** (interactive)). Instructs **nbq** to handle signals locally, instead of passing them to the remote process.

**--log-file-dir <path>**

Specifies that the Job's log file will be written in the specified directory.

If **-j** is used as well and its argument includes a path,
--log-file-dir is ignored.

--lr <reservation_item>[,...]

See **--license-reservation**.

--mail-list <address>| "<address>[...]"

List of mail recipients for Netbatch messages regarding the Job. If multiple addresses are specified, a space is used as the delimiter.

--on-job-finish

"{<exit_code>|<expression>}:<operation>[:...]"
[,...]

Specifies that if the Job exit code is **exit_code** or if it matches **expression**, the specified **operation**(s) are performed.

exit_code is a numerical Job exit code (see [Exit Status](#)) or one of the following macros:

- ◆ **NBErr** - all Netbatch errors except -8, -9, -10, and -11
- ◆ **Nexit** - all negative Job exit values
- ◆ **Pexit** - all positive Job exit values
- ◆ **NZexit** - all non-zero Job exit values

expression is an arbitrarily complex boolean expression consisting of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is any [nbstatus jobs](#) field (though typically, you will use [ExitStatus](#), [PreExecExitStatus](#), and [PostExecExitStatus](#)).
- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., "+10"), or unary - (e.g., "-10")
 - ◆ A logical operator: &&, ||, or !
 - ◆ A comparison operator: <, <=, >, >=, !=, or ==
 - ◆ A non-strict operator: **is** or **isnt**
 - ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
 - ◆ A conditional operator: ? or :



- ♦ **value** is a number, a string, or another boolean expression.

Expressions can be grouped together using parentheses.

For example,

```
ExitStatus== -303&&PreExecExitStatus==2
```

operation is one of the following:

- ♦ **Requeue(<max_attempts>)** - the Job is requeued up to **max_attempts** times.
- ♦ **Exclude(<period>{s|m|h})** - if the Job was submitted to a Virtual Pool, exclude the last resource (Pool) that the Job was dispatched to from those that it can be sent to, for the specified time. (If the Job was *not* submitted to a Virtual Pool, exclude the last Workstation that the Job was dispatched to from the list of Workstations that the Job can run on, for the specified time.)
- ♦ **Modify('<parameter>=<expression>'[,...])** - modify the Job before resubmitting it. You must also specify a **Requeue** action, otherwise the Job will not be resubmitted.

Here, **parameter** is a valid **nbjob modify** parameter (not including Queue) and **expression** specifies the value that is assigned to the parameter. **expression** can include any **nbstatus jobs** field.

It can also include regex replace functions (see [Appendix G: Regex Replace Functions](#)).

If **expression** is a string, it must be enclosed in quotes. But because the entire on-job-finish expression is already in quotes, you must escape these quotes. In bash, use `\"`. In tcsh, use `"\"`.

Examples:

```
--on-job-finish
"Nexit:Modify('priority=priority+1';
'class="4G"'):Requeue(1)"
```

If the Job ends with a negative exit status, its priority is incremented and the specified class requirement is added to it before it is requeued.

```
--on-job-finish "Nexit:Modify('job-
constraints="VM>5:mail"'):Requeue(1)"
```

Here, if the Job ends with a negative exit status, the specified Job constraint is added to it before it is requeued.

**Note**

If you specify both **Requeue** and **Exclude** operations, **Exclude** must come first.

If you specify both **Modify** and **Requeue**, **Modify** must come first.

If you specify multiple **exit_code|expression:operations**, only the operation for the first match will be performed.

This switch overrides the **NB_ON_JOB_FINISH** environment variable.

If **Exclude** is used and the Job is submitted to a Virtual Pool or a Netbatch Feeder, the exclude flag is not passed to the physical Pool Master. (If it was passed to the physical Pool Master, the Job could be requeued by both the physical Pool Master and the Virtual Pool Master/Feeder.)

Note

--on-job-finish takes precedence over **--autoreq**.

For Example

```
--on-job-finish "ExitStatus== -303&&
PreExecExitStatus==2:Exclude(10h):Requeue(3 )"
```

If the Job exit status is -303 and the exit status of the pre-execution stage is 2, then the Job is to be requeued up to three times. The Workstation that the Job was dispatched to should be excluded from the list of Workstations that it can run on for ten hours.

--pag on|nosuspend

Netbatch uses Process Authentication Groups (PAGs) to track Jobs. This switch specifies whether a PAG should be added to the Job, and how:

- ♦ **on** - add a PAG to the Job. This may override a Job Profile, or be overridden by a Job Profile, depending on how the Job Profile is set up.
- ♦ **nosuspend** - add a PAG to the Job, but do not suspend child processes that change their process group ID.

If this switch is not specified, and no Job Profile specifies that Netbatch should apply a PAG, Netbatch decides whether to apply a PAG based on how the Workstation is configured.

**--parallel <parallel reservation string>**

Specifies that the Job is the Master Job of a Parallel Job, to be executed according to the parameters in the string.

The parallel reservation string take the following form:

```
"slots=[<min_slots>[-<max_slots>]][, slots_per_host=<min_sph>[-<max_sph>]][, reservation_per_host={true|false}][, preserve_slot={true|false}][, exit_on_master_finish={true|false}][, slave_validator=<validate_script>]"
```

where:

- ◆ **min_slots** is the minimum number of execution slots required
- ◆ **max_slots** (optional) is the maximum number of execution slots required
- ◆ **min_sph** is the minimum number of execution slots required on a single host
- ◆ **max_sph** (optional) is the maximum number of execution slots required on a single host
- ◆ **reservation_per_host** indicates that the requested resource reservation will be made once for each scheduled executing host, regardless of the number of parallel slots scheduled for it.

The default value is **false**.

For example, if two Slave Jobs are scheduled to execute on a Host, each of which has a resource reservation of 10GB of memory:

- ◆ If **reservation_per_host** is **false**, 10GB will be reserved for each Job (20GB in total).
- ◆ If **reservation_per_host** is **true**, a total of 10GB will be reserved.
- ◆ **preserve_slot** specifies whether an execution slot will be released (**false**) or not (**true**) when the Slave Job running in it ends. If **preserve_slot=true**, the execution slot can be reused.

This behavior can also be specified at the Slave Job level (but only if **preserve_slot=false** for the Master Job). To do this, add **--preserve-slot** to the **nbexec** command for the relevant Slave Job(s).

- ◆ **exit_on_master_finish** specifies what happens to Slave Jobs that are still running when the Master Job ends. If your



Slave Jobs should no longer be running when the Master Job ends, set this to `true`. Otherwise, set it to `false` (the default).

- ◆ **validate_script** is the name of a script that is run to determine whether a particular Workstation can or cannot run one of the Job's Slave Jobs. The user is responsible for creating this script.

If the script determines that the Workstation is suitable, it should return zero. If not, it should return a non-zero value.

If the script returns a non-zero value, Netbatch excludes the Workstation and looks for a replacement.

--parallel-job-constraints
"<expression>:<action>[:....][,...]"

Specifies that the Netbatch should take one or more actions (including stopping, restarting, or resubmitting the Job), depending on various Job-related factors (which it measures once a second).

expression is either an arbitrarily complex boolean expression or an expression of the form:

idle(<interval_in_sec>, <cpu_in_percent>)

This evaluates to true if CPU usage was below the specified percentage during the specified interval.

If **expression** is an arbitrarily complex boolean expression, it consists of **<attribute>**s, **<operator>**s, and **<value>**s, where:

- ◆ **attribute** is one of the following:
 - ◆ **RM** - the total real memory use by all the Jobs that make up the Parallel Job (MB)
 - ◆ **VM** - the total virtual memory use by all the Jobs that make up the Parallel Job (MB)
 - ◆ **utime** - the user time that has been accumulated by all the Jobs that make up the Parallel Job (seconds)
 - ◆ **stime** - the system time that has been accumulated by all the Jobs that make up the Parallel Job (seconds)
 - ◆ **wtime** - the wall clock time that has been accumulated by all the Jobs that make up the Parallel Job (seconds)

Note

These conditions are checked every 5 minutes.

**Note**

Attribute names are case-sensitive.

- ◆ **operator** defines the relationship between the attribute and the value. **operator** can be one of the following:
 - ◆ An arithmetic operator: +, -, *, /, %, unary + (e.g., +10), or unary - (e.g., -10)
 - ◆ A logical operator: &&, ||, or !
 - ◆ A comparison operator: <, <=, >, >=, !=, or ==
 - ◆ A non-strict operator: is or isnt
 - ◆ A bitwise operator: &, |, ~, ^, <<, >>, or >>>
 - ◆ A conditional operator: ? or :
- ◆ **value** is a number, a string, or another boolean expression.

Expressions can be grouped together using parentheses.

action is one of the following:

- ◆ kill
- ◆ mail
- ◆ recall
- ◆ nice

When the condition is no longer true, the Job is reniced.

- ◆ resubmit
- ◆ suspend

When the condition is no longer true, the Job is resumed.

Multiple actions can be specified for a single expression.

Multiple <expression>:<action>[:...] constraints can be specified.

--parallel-slave-class <class_expression>

Specifies the class of machine that the Parallel Job's Slave Jobs require. (The syntax is the same as for -c, above.)

If this switch is not used, any class requirements specified with -c apply to the Master Job and all the Slave Jobs.

If you need to specify class requirements for the Master Job only, add the following switch in addition to the -c switch:



```
--parallel-slave-job "@"
```

```
--post-exec <exec>
```

Execute a post-execution program after the Job runs

Note

In the Post-Execution phase, the following environment variables are available:

- **NB_JOB_EXIT_STATUS**—the Job's exit status
- **NB_CLASS**—the Job's class requirement expression
- **NB_CMD_LINE**—the Job's full command line
- **NB_FINISH_TIME**—the time at which the Job finished executing
- **NB_JOBID**—the Job's ID
- **NB_LOG_FILE**—the location of the Job log file
- **NB_POOL**—the name of the Pool that the Job was submitted to
- **NB_QSLOT**—the Qslot that the Job was submitted to
- **NB_QUEUE**—the Queue that the Job was submitted to
- **NB_RUSAGE**—information about the Job's resource usage
- **NB_START_TIME**—the time at which the Job started executing
- **NB_SUBMIT_PWD**—the user's working directory when the Job was submitted
- **NB_TIMES_RESTARTED**—the number of times that the Job was restarted

Note

The post-execution program is executed even if the Job is killed or suspended.

```
--pre-exec <exec>
```

Execute a pre-execution program before the Job runs.

Note

The following environment variables contain information that the pre-exec script can use:



- **NB_POOL**
- **NB_QUEUE**
- **NB_QSLOT**
- **NB_LOG_FILE** (*but only if the -j option is used*)
- **NB_CLASS**

--priority <number>

Specifies the Job priority (integer between 5 and 15, where 5 is lowest Job priority).

Note

The --priority default value is 10.

--properties <prop_name>=<value>[,...]

Specifies properties for the Job. **prop_name** is the name of a property. **value** is the property's value.

The Qslot that you are submitting the Job to may have configured rules that require you to specify values for certain properties when submitting a Job.

Optionally, you can also supply values for any configured property.

If you omit required properties when trying to submit a Job, you will receive an error message that lists the properties that you must supply values for.

--queue <queue>

Specifies the Queue that the Job is submitted to.

Note

When specifying -H or -P without --queue, the Job will be submitted to the default Queue of the specified Pool.

--remote-pools { "[-]<phys_pool>" | "[-]<alias>" }[,...]

Specifies the preferred Physical Pool(s) by name or by alias (only relevant when submitting a Job to a Virtual Pool). You can specify Pools by:

- ♦ Inclusion -- only the listed physical Pools will be considered when deciding which Pool to send the Job to.



- ◆ Exclusion -- precede Pool/alias names with -. All relevant physical Pools will be considered except those that are listed.
- ◆ A combination -- if there are both included and excluded Pools/aliases in the list, the excluded ones are excluded from the list of included Pools.

You can use this to specify that a specific physical Pool should be excluded from the set of Pools represented by an alias.

This situation can also arise when the request uses inclusion and a Job Profile uses exclusion, or vice-versa.

The order determines the order in which the Pools will be tried. Netbatch will try the first Pool, and will only try the next one if the previous one has reached its `WaitJobsLimit` or `MaxJobsLimit`. If an alias is specified, Netbatch uses its existing algorithm to decide which Pool to send the Job to. (Multiple Pools can have the same alias. For example, all the Physical Pools at the same site as the Virtual Pool Master might have the alias `local`.)

This switch overrides the **NB_POOLS** environment variable.

--resource-type xx

Specifies the resource type. This switch is deprecated, and is only included for backward-compatibility.

--resubmit-to-source

(Virtual Pool only) Specifies that if the Job is resubmitted for whatever reason (for example, if it is preempted) it is to be resubmitted to the Virtual Pool, not the physical Pool.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.



--rlimits "<res_limit> [soft|hard] = value [, ...]"

Set resource consumption limits:

- ◆ **res_limit** is the name of the limit to be set.
- ◆ **soft | hard** (optional) specifies whether a soft or hard limit is to be set. If not specified, both hard and soft limits are set to the same value.
- ◆ **value** is the value to be assigned to the limit.

The limits that can be set are listed in the table below. Some are platform-specific. If a limit is set for a platform that does not support it, it is ignored.

If a limit is not set, the limit is inherited from the WSM process that spawned the process. The WSM process, in turn, inherits the limits from the shell from which it was invoked.

Note

*You can use the built-in **limit** command to view current resource limits. Note, however, that this shows the limits on the Workstation that you are logged into, not the Workstation on which your Job will run.*

*If you need to know the limits that will apply to your Job, run the **limit** command on a Workstation whose capabilities match those required by your Job.*

Table 4: Resource Limits

Limit	Units	Description	Platform
addressspace	KB	The maximum size of a process's available memory	All
coredumpsize	KB	The maximum size of a core file that a process may create	All
cputime	Seconds	The maximum amount of CPU time that a process may use	All
datasize	KB	The maximum size of a process's heap	All
filesize	KB	The maximum size of a file that a process may create	All
memorylocked	KB	Maximum locked-in memory address space	Linux
memoryuse	KB	The maximal resident set size of a process	Linux, HP-UX
openfiles	# of files	Maximum number of files that a process may open	All
maxproc	# of processes	The maximum permitted number of processes	Linux

**Table 4: Resource Limits**

Limit	Units	Description	Platform
stacksize	KB	The maximum size of a process's stack (can also be set to <code>unlimited</code>)	All
vmemoryuse	KB	The maximum size of a process's mapped address space	Solaris

--service-type xx

Specifies the service type. This switch is deprecated, and is only included for backward-compatibility.

--silent

Parallel Jobs that use MPI or Linda use the `rsh.nb` script instead of `rsh`. **--silent** suppresses the messages that Netbatch usually sends to stdout when `nbq` is run interactively (with the `-i` switch). This allows `rsh.nb` to mimic `rsh` as closely as possible.

--stack-size <Number>{kb}**Note**

--stack-size is deprecated. Use **--rlimits** instead (see above).

Specifies the stack size limit for the Job on the Workstation. See **Section 6.5.20, Specifying Resource Limits**. If not specified, the stack size limit that applies to the `nbq` command, will apply to the Job as well.

--task-name <taskname>

Specifies the Job task name.

--terminal

Run in terminal emulation mode.

Note

As of version 6.3.1, Netbatch automatically senses the context in which an interactive or terminal Job is being executed, and runs it as either an interactive or terminal Job, whichever is appropriate.



Thus, there is no need to use `--terminal`. Whenever you want the Job to run as if it is running locally, use `-i`.

--triggers " <boolean_expression>"

Specifies that the submitted Job depends other Jobs, as defined in `boolean_expression`.

`boolean_expression` consists of any number of terms of the form `[!]<JobID>[[<exit_status_expression>]]`, separated by the logical operators `&&` (AND) and `||` (OR).

Note

`JobID` can be either a Job ID or a task name. You are advised to explicitly specify whether it is a Job ID or a task name, because in ambiguous cases, Netbatch can assume that a task name is actually a Job ID, or vice versa.

Use one of the following instead of a Job ID/task name:

`jobid:<Job_ID>`
`task:<task_name>`

If a task name is used, the `exit_status_expression` does not evaluate to true until it is true for each of the Jobs that make up the task.

See [Section 6.5.18, Specifying a Task Name](#).

`exit_status_expression` can be one of the following:

- ♦ `exit`
- ♦ `exit <relational_operator> <exit_code>`
`relational_operator` must be `<`, `<=`, `==`, `>=`, or `>`.

To specify multiple acceptable exit code conditions for a specific Job, use multiple `<JobID>[<exit_status_expression>]` terms separated by `||` (OR) (for example, `1 exit <30 || 1 exit>50`).

If `exit` is used on its own, it means that the Job ended (no matter what its exit status).

If no `exit_status_expression` is specified, `done` is assumed.

**Note**

Logical expressions are evaluated according to the following order of precedence—() (parentheses), ! (NOT), && (AND), || (OR).

So

```
"1 [exit<30] || !2 && (3 || 4)"
```

is the same as

```
"1 [exit<30] || ((!2) &&(3 || 4))"
```

--wash

Specifies that the Job should be run with wash support. That is, only the filesystem groups specified in the **NB_WASH_GROUPS** environment variable will be applied to the Job.

Use this when you belong to more than 16 groups and you want to specify which ones are applied to the Job.

--wsrank-input required_datasets="*<ds_name>*[,...]"

Specifies the CaMa datasets that the Job requires. Netbatch uses this information to match the Job with a Workstation that already has the required datasets (if it is configured to do so).

Note

A Job's owner must belong to all of its wash groups according to the group map.

See [Section 6.1.4, What to Do if you Belong to More than 16 Groups](#).

Output

Netbatch creates a log file containing the Job output named:

```
##Date-Time#.<machine>
```



See [Appendix D: Log File Contents and Job Exit Codes](#) for detailed information about Netbatch log file output.

Note

If more than one Job starts at the same time on the same machine, Netbatch appends the Pool name and the Job ID to all the Jobs except the first Job to differentiate between Jobs.

Note

For Parallel Jobs, Netbatch names log files as follows:

- For each executed Job, Netbatch creates a separate output log file, in the standard format (see above).*
- If you use the -j switch (that is, you use -j <jobname>) when you submit a Job, then the log files will be called jobname:index.*
- If you use the -j switch (that is, -j <jobname>) with nbexec (where each Slave Job is executed by running a separate nbexec command), the log file of the Slave Job is called jobname.*

Diagnostics

The command exit codes are listed below.

Note

Not all error codes are relevant for this command.

Note

These are command exit codes, not Job exit codes.

0

Successful execution

1

Specified Pool Master (-P) does not exist in pools file (older commands only (**nbq**, **nbqrm**, **nbstat**, etc.))

**2**

Unknown host specified (--target for newer commands, -H for older commands)

3

Error occurred on master side (for example, PPM error, Job could not be submitted because of missing permissions, non-existent class, etc.)

4

Problem getting user credentials (most likely an NIS issue on the client machine)

5

PPM timeout (The command waits one minute for a response.)

212

Depends on context:

- ◆ CSD error - Job failed
- ◆ Status command - invalid displayed field

213

Depends on context:

- ◆ CSD error - bad parameters
- ◆ Status command - invalid filter

214

Depends on context:

- ◆ CSD error - Workstation not available
- ◆ Status command - invalid grouping

215

Depends on context:

- ◆ CSD error - RG not available
- ◆ Status command - invalid sort

216

CSD error - Qslot not available

**221**

Interrupted

222

Permission denied

223

Timeout

224

Unknown error

225

Permissions problem

229

The command was run with the **-v** switch (version)

230

The command was run with the **-h** switch (help)

231

Parsing error - mandatory filter not specified

232

Parsing error - invalid value

233

Parsing error - value not specified

234

Parsing error - undefined parameter

235

Parsing error - mandatory parameter not specified

244

No command handler in this context

245

No command handler

**248**

Communication problem - target service down

249

Communication problem - connection broken during receive stage

250

Server too busy (denial of service response or built-in request limit reached)

251

The server does not support the command.

253

Communication problem - unknown protocol

254

Communication problem - no response

255

Master not responding (probably down)

A.14**A.15 nbqrm****Name**

nbqrm - Removes, moves, and/or resubmits one or more Netbatch Jobs.

Note

nbqrm is deprecated—use [**nbjob remove**](#) instead.



Description

Removes, moves, and/or resubmits one or more Netbatch Jobs from a User Slot, Qslot, Queue, Pool, Class, and/or host.

When a user runs **nbqrm**, it only makes changes to Jobs belonging to that user.

A user may have configured privileges to remove or modify Jobs of other users in the Qslots. This is defined per Queue or per Qslots.

The system administrator can use nbqrm to make changes to any user's Jobs.

nbqrm can be used to cancel, resubmit, and move Parallel Jobs (but not to suspend or resume them). Simply use **nbqrm** with the JobID of the Master Job.

Note

If a Job is moved to a Netbatch 4.x Pool, the Job ID and the Netbatch 6 specific data will not be preserved (including license information and user defined priority).

Usage

```
nbqrm [option] [criteria] [Job specification]
```

Command Line Options

-a

Remove Jobs from any Queue (same as **-b**).

-b

Remove Jobs in any Queue.

-C <class>

Moves (changes) Jobs to the specified Class. You must use this with **-s** or **-M**.

-d

Displays debugging information.

**-f**

Remove the specified Jobs, even if they are disconnected (status **Disc-D** or **Disc-R**). This removes the Job from the Pool Master. Only use this switch if you know that the Workstation that the Job was sent to is actually down.

Only root or a user with administrative permissions for the Qslot that the Job was submitted to can use this switch.

-h

Displays help.

-H <hostname>

Remove Jobs from the specified master.

-L "<expression>"

Change the Job's license requirements to **expression**.

expression specifies the license(s) that the Job will need during its execution, and (optionally) the amount of time each one is required. This overrides any license requirements defined in the user's Job profile file or **NBLICENSES** environment variable.

expression consists of any number of **<req_expr>**s and **<operator>**s.

req_expr is a license requirement expression of the following format:

```
"<license>=<value>[:duration=<duration_time>]"
```

where:

- ◆ **license** is the required license.
- ◆ **value** is the number of instances of **license** that are required.
- ◆ **duration_time** is the amount of time (in minutes) for which the license is required (starting from when the Job is dispatched for execution).

If not specified, then **duration** is set to the Job execution time (that is, the license is reserved for the entire duration of the Job's execution).

operator is one **&&** or **||**.

These can be grouped together using parentheses.

-M <queuename>

Move Jobs to a different Queue.



Use this switch with **-Q** or **-C** to resubmit Jobs to a different Qslot or to a different Class requirement.

If you do not use **-Q** to specify a different Qslot, Netbatch moves the Job to the same Qslot in the new Queue.

Note

If you move Jobs to a different Queue in the same Pool, the Job ID will be preserved. If you move Jobs to a new Pool, a new Job ID will be created.

Note

*Using **nbqrm -M** to move a Job to a different Pool has been deprecated. To move a Job to a different Pool, either:*

- *Kill it and resubmit it to the new Pool, or*
- *Submit it to a Virtual Pool instead.*

-O

Displays **nbqrm** messages in a strict backward-compatible format.

-P <poolname>

Remove Jobs from the named Pool.

-Q <qslot>

Moves Jobs to a new Qslot. **<qslot>** specifies the full path or alias of the required Qslot. You must use this with **-s** or **-M**. It cannot be used alone. The new default Qslot is the same Qslot.

-r

Specifies running Jobs only. Cannot be used with **-a** or **-b**.

-s

Resubmit Jobs to the specified wait Queue while preserving the Job ID number. Use **-s** in combination with the following additional options as required:

- ◆ **-r** to select running Jobs only (default selection)
- ◆ **-w** to select waiting Jobs only
- ◆ **-a** to select all Jobs
- ◆ **-Q <queue>** to resubmit Jobs to a different Qslot



- ◆ -C <class> to resubmit Jobs to a different Class

-v

Displays the Netbatch version number and release date.

-w

Specifies waiting Jobs only. Cannot be used with -s, -a, or -b.

-y

Yank (retract) Jobs from the wait Queue. Use only with the -s option.

This switch can only be used if you have root permissions.

--class-reservation <reservation_item>[,...]

Modifies the Job's class reservation requirements.

If the Job is running, it is **resubmitted** with the new class reservation requirements and all reserved resources are released.

If the Job is waiting, all reserved resources are released.

reservation_item must be in the following format:

```
"<key>=<key_value>[:duration=<duration_time>]  
[:decrease=<decrease_time>]  
[:delta=<delta_value>]
```

where:

- ◆ **key** is the type of resource to be reserved. It can take the following values:
 - ◆ **fVM**
 - ◆ **fRM**
 - ◆ **dedicated**
 - ◆ The name of a custom capability or attribute that has been defined for one or more Workstations in the Pool

You can use the **nbstatus workstations** command to see a list of these attributes:

```
nbstatus workstations --fields  
"*,WSCapabilities"
```

These capabilities can either relate to total real or virtual memory or be arbitrary.

**Note**

Workstation capability requirements can also be defined at the Queue or Qslot level. Such requirements apply to all Jobs submitted to the Queue/Qslot.

- ◆ An external probe parameter name

To see the available Workstation probes and the parameters that they measure, run **nbstatus workstations --fields all** and look at the value of the **ExternalProbeData** field.

To see the available Scheduler probes and the parameters that they measure, run **nbstatus probes**.

See [Appendix A.3.31, nbstatus workstations](#) for more details.

Note

*If **dedicated=true**, then no other Jobs may run on the host while the Job is running (except for Parallel Slave Jobs—multiple Slave Jobs that are part of the same Parallel Job can run on the same host).*

- ◆ **key_value** is the value for **key**. It must be one of the following:
 - ◆ For **fvm** and **frm**, an integer/long value
 - ◆ For **dedicated**, **true** or **false**
- ◆ **duration_time** is the amount of time (in minutes) for which the resources (**fvm** and **frm** only) are required (starting from when the Job is dispatched for execution).

If not specified, then **duration** is set to the Job execution time (that is, the resource is reserved for the entire duration of the Job's execution).

- ◆ **decrease_time**—During the time specified by **duration**, the committed resource (**fvm** and **frm** only) is reduced every **decrease_time** minutes by the amount specified in **delta_value**.

If a **delta_value** is specified, but no **decrease_time**, then **decrease = 1 minute**.



If neither a `delta_value` nor a `decrease_time` is specified, then the resource is available at the same level for the whole time specified by `duration`.

- ◆ `delta_value`—During the time specified by `duration`, the committed resource (`fVM` and `fRM` only) is reduced every `decrease_time` minutes by the amount specified in `delta_value`.

If a `decrease_time` is specified, but no `delta_value`, then `delta = 1` unit.

If neither a `delta_value` nor a `decrease_time` is specified, then the resource is available at the same level for the whole time specified by `duration`.

`--cr <reservation_item>[,...]`

See `--class-reservation`.

`--license-keyfile "<keyfile path>|<port@host>[:....]"`

Changes the license keyfiles to be used. (This is instead of setting the `NB_LICENSE_FILE` environment variable.) If multiple arguments are specified, use colons as delimiters.

Note

If you specify a license that the Job requires by its alias, you do not need to specify the license keyfile.

`--new-priority <priority>`

Change the priority.

`--new-queue <queuename>`

Move Jobs to Queue `<queuename>` in the same Pool

`--parallel <new parallel reservation string>`

Modifies the Job's Parallel Job requirements. The original parallel reservation string is replaced with the one specified.

If the Job is running, it is **resubmitted** with the new Parallel Job requirements and all reserved licenses are released.

If the Job is waiting, all reserved execution slots are released.

`--queue <queuename>`

Remove Jobs from Queue `<queuename>`.

**Note**

When specifying -P or -H without --queue, nbqrm will remove Jobs from the default Queue of the specified Pool.

--reason "<reason>"

Specifies why the Job is being removed, suspended, resumed, etc.
This information is recorded in NB Tracker.

--resume

Resumes Jobs. This option cannot be used for Parallel Jobs.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--suspend

Suspends Jobs. This option cannot be used for Parallel Jobs.

Criteria**class=xx**

Removes Jobs that belong to **class xx**

host=xx

Removes Jobs running on **host xx**

Note

Use host=xx with -s -r.

**number=xx**

Removes Jobs number **xx**.

Use this switch with **user**, **slot**, **class**, or **host** specifications.

slot=xx

Removes Jobs that belong to **slot xx**.

task=xx

Removes Jobs that belong to **task xx**.

user=xx

Removes Jobs belonging to **user xx**.

Job Specifications

Job specifications can be any one of the following:

all

Acts on all Jobs that belong to you.

job_list=xx yy zz

Remove Jobs with Jobid **xx**, **yy**, **zz**, in this order

job_set=xx yy zz

Remove Jobs with Jobid **xx**, **yy**, **zz**.

xx

Acts on the Job with the specified Job ID.

xx-yy

Acts on all Jobs with IDs in the specified range.

Diagnostics

The command exit codes are listed below.

Note

Not all error codes are relevant for this command.

**Note**

These are command exit codes, not Job exit codes.

0

Successful execution

1

Specified Pool Master (**-P**) does not exist in **pools** file (older commands only (**nbq**, **nbqrm**, **nbstat**, etc.))

2

Unknown host specified (–**target** for newer commands, **-H** for older commands)

3

Error occurred on master side (for example, PPM error, Job could not be submitted because of missing permissions, non-existent class, etc.)

4

Problem getting user credentials (most likely an NIS issue on the client machine)

5

PPM timeout (The command waits one minute for a response.)

212

Depends on context:

- ◆ CSD error - Job failed
- ◆ Status command - invalid displayed field

213

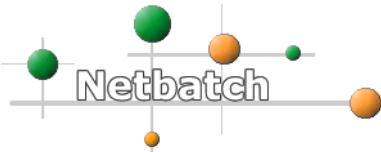
Depends on context:

- ◆ CSD error - bad parameters
- ◆ Status command - invalid filter

214

Depends on context:

- ◆ CSD error - Workstation not available



- ◆ Status command - invalid grouping

215

Depends on context:

- ◆ CSD error - RG not available
- ◆ Status command - invalid sort

216

CSD error - Qslot not available

221

Interrupted

222

Permission denied

223

Timeout

224

Unknown error

225

Permissions problem

229

The command was run with the **-v** switch (version)

230

The command was run with the **-h** switch (help)

231

Parsing error - mandatory filter not specified

232

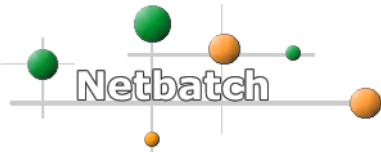
Parsing error - invalid value

233

Parsing error - value not specified

234

Parsing error - undefined parameter

**235**

Parsing error - mandatory parameter not specified

244

No command handler in this context

245

No command handler

248

Communication problem - target service down

249

Communication problem - connection broken during receive stage

250

Server too busy (denial of service response or built-in request limit reached)

251

The server does not support the command.

253

Communication problem - unknown protocol

254

Communication problem - no response

255

Master not responding (probably down)

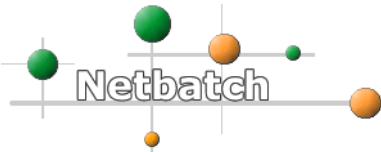
A.16 nbqslot

Name

nbqslot - Displays information about Queues, Qslots, and User Slots.

Note

nbqslot is deprecated—use ***nbstatus qslots*** instead.



Description

Displays information about the Qslots defined for one Queue or all the Queues in one Pool or in all Pools.

It includes the Qslot's weight, ownership, and the number of Jobs from this Qslot that are currently running.

It also displays the Qslot's fairness ratio (i.e. the percentage of resources it should be getting compared to how much it is actually getting).

Usage

```
nbqslot [options] [-p <pool>|-H <host>]
```

Command-line Options

-a

Displays Qslots for all Pools.

-d

Displays debugging information.

-h

Displays help information.

-H <host name>

Specifies the Pool by the Pool manager machine name.

-m

Displays the mapping between Qslots and their aliases in the Queue.

-n

Displays hierarchical names for the Qslots.

-P <pool name>

Specifies the Pool to query.

-r

Print Qslots' statistics.

-v

Displays the Netbatch version number and release date.

**--long-slot**

Changes the output format to display the fully qualified hierarchical Qslot name.

--queue <queue name>|all

Display Qslots of the selected Queue or all Queues.

Note

When specifying -P or -H without --queue, nbqslot will display information of the default Queue of the specified Pool.

--remote-data

Jobs from a Virtual Pool can be either:

- ◆ Waiting (in a Qslot in the Virtual Pool), or
- ◆ Dispatched (sent to the physical Pool, either waiting or running there)

For Qslots in a Virtual Pool, standard **nbqslot** output shows the number of running Jobs.

nbqslot --remote-data (without **-r**) displays the number of dispatched Jobs.

Similarly, in **nbqslot -r** (without **--remote-data**) output, the totals for running Jobs include both dispatched and running Jobs.

With **--remote-data**, **nbqslot -r** output includes an additional row for dispatched Jobs (and the running Jobs totals include only Jobs that are actually running).

--resource-type

Specifies the resource type. This switch is deprecated, and is only included for backward-compatibility.

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.



If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--service-type

Specifies the service type. This switch is deprecated, and is only included for backward-compatibility.

--show-hierarchy

Displays all defined Qslots with their real names, either numeric or hierarchical (aliases are not shown here), and hierarchical indentation.

--summary

Displays summary information for the specified Queue.

Note

nbqslot is used to display the Qslots information whereas *nbqslot --summary* is used to display the cumulative Queue information.

--users-slots

Displays the Qslots with all defined user slots.

Note

Use *nbqslot --show-hierarchy --queue all --summary* to view a summary of all Queues in the system.

Output

nbqslot output displays a list of blocks where each block represents a Queue or a Qslot.

The blocks' format and indentation may vary depending on the command line switches and options that are used and the system scheduling.

To learn more about Queues and Qslots see [Section 2.4 Resource Allocation](#)



Interpreting nbqslot standard output

Figure 31 illustrates the standard `nbqslot` output for each Qslot:

NetBatch SLOT status - pool martin						
Queue master version : 6.2.0						

slot	weight		Running		Should Get	Getting Now
1	0		0		0.00%	[No Running]

NB users						

Unix groups						

Figure 31: nbqslot Output

Interpreting the standard Qslots output

When fairshare scheduling is used to schedule the Qslots the output should be interpreted as follows:

slot

The Qslot's unique number.

Note

The standard format displays only numeric Qslots or Qslots with numeric aliases for compatibility reasons.

weight

The Qslot's weight.

running

Number of running Jobs from this Qslot.

should get

The percentage of resources that the Qslot should be getting from this Queue.

**Getting Now**

The percentage of resources that the Qslot is getting right now from the Queue.

NB users

The Netbatch group or individual user that are permitted to submit Jobs to the Qslots.

Unix Groups

The UNIX (or NIS) group(s) that are permitted to submit Jobs to the Qslots.

When priority scheduling is used to schedule the Qslots the output should be interpreted as follows:

Slot

The Qslot's unique number.

Priority

The Qslot absolute priority.

Running

Number of running Jobs from this Qslot.

Sched Order

The position of the Qslot in the next Queue scheduling iteration.

Getting Now

The percentage of resources that the Qslot is getting right now from the Queue.

NB users

The Netbatch group or individual user that are permitted to submit Jobs to the Qslots.

Unix Groups

The UNIX (or NIS) group(s) that are permitted to submit Jobs to the Qslots.



Interpreting --summary Output

When fairshare scheduling is used to schedule between Queues the output should be interpreted as follows:

Queue

The Queue's unique number.

Weight

The Queue's weight.

Running

The cumulative number of running Jobs from this Queue.

Should Get

The percentage of resources that the Queue should be getting from the Pool.

Getting Now

The percentage of resources that the Queue is getting right now from the Pool.

NB users

The Netbatch group or individual user that are permitted to submit Jobs to the Queue.

Unix Groups

The UNIX (or NIS) group(s) that are permitted to submit Jobs to the Queue.

When priority scheduling is used to schedule the Queues the output should be interpreted as follows:

Queue

The Queue's unique number.

Priority

The Queue's absolute priority.

Running

The cumulative number of running Jobs from this Queue.

Sched Order

The position of the Queue in the next scheduling iteration.

**Getting Now**

The percentage of resources that the Queue is getting right now from the Pool.

NB users

The Netbatch group or individual user that are permitted to submit Jobs to the Qslots.

Unix Groups

The UNIX (or NIS) group(s) that are permitted to submit Jobs to the Qslots.

Interpreting nbqslot -m Output**Qslot name**

The full path to the Qslot within a Queue.

Alias

The alias defined for the Qslot

Index

The position of the Qslot in the Queues configuration order.

Interpreting nbqslot -r Output**Note**

--remote-data changes -r output (see above).

Qslot

The Qslot full hierarchical name.

alias

The alias defined for the Qslot

hierarchical allocation

The relative allocation of the Qslot, comparing to the other Qslots in the same level

node allocation

The relative allocation of the Qslot, comparing to the other sub Qslots in the.

**waiting jobs**

The number of waiting Jobs in the default Qslot (OTHERS).

cumulative waiting jobs

The cumulative number of waiting Jobs in the entire sub tree under the Qslot.

running jobs

The number of running Jobs in the default Qslot (OTHERS).

cumulative running jobs

The cumulative number of running Jobs in the entire sub tree under the Qslot.

pseudo running jobs

The number of pseudo running Jobs in the default Qslot (OTHERS).

cumulative pseudo running jobs

The cumulative number of pseudo running Jobs in the entire sub tree under the Qslot.

Note

Pseudo running Jobs are Jobs that had already finished their execution but still in count for fairshare scheduling

locked jobs

The number of Jobs that are waiting for a license within the default Qslot.

cumulative locked jobs

The cumulative number of Jobs that are waiting for a license within the Qslot sub tree.

Index

The position of the Qslot in the Queue's configuration order.

default class

The default Class defined for the Queue.

node should get

The percentage of resources that the default Qslot should be getting within the entire Queue

**node getting now**

The percentage of resources that the default Qslot is getting within the entire Queue

tree should get

The percentage of resources that the Qslot and its sub-tree should be getting within the entire Queue

tree getting now

The percentage of resources that the Qslot and its sub-tree are getting within the entire Queue

queue state

The Queue state.

Diagnostics

The command exit codes are listed below.

Note

Not all error codes are relevant for this command.

Note

These are command exit codes, not Job exit codes.

0

Successful execution

1

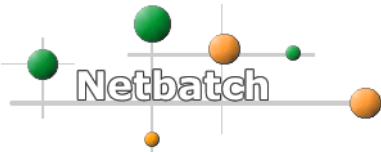
Specified Pool Master (-P) does not exist in **pools** file (older commands only (**nbq**, **nbqrm**, **nbstat**, etc.))

2

Unknown host specified (--target for newer commands, -H for older commands)

3

Error occurred on master side (for example, PPM error, Job could not be submitted because of missing permissions, non-existent class, etc.)

**4**

Problem getting user credentials (most likely an NIS issue on the client machine)

5

PPM timeout (The command waits one minute for a response.)

212

Depends on context:

- ◆ CSD error - Job failed
- ◆ Status command - invalid displayed field

213

Depends on context:

- ◆ CSD error - bad parameters
- ◆ Status command - invalid filter

214

Depends on context:

- ◆ CSD error - Workstation not available
- ◆ Status command - invalid grouping

215

Depends on context:

- ◆ CSD error - RG not available
- ◆ Status command - invalid sort

216

CSD error - Qslot not available

221

Interrupted

222

Permission denied

223

Timeout

**224**

Unknown error

225

Permissions problem

229

The command was run with the **-v** switch (version)

230

The command was run with the **-h** switch (help)

231

Parsing error - mandatory filter not specified

232

Parsing error - invalid value

233

Parsing error - value not specified

234

Parsing error - undefined parameter

235

Parsing error - mandatory parameter not specified

244

No command handler in this context

245

No command handler

248

Communication problem - target service down

249

Communication problem - connection broken during receive stage

250

Server too busy (denial of service response or built-in request limit reached)

**251**

The server does not support the command.

253

Communication problem - unknown protocol

254

Communication problem - no response

255

Master not responding (probably down)

A.17 nbqstat

Name

nbqstat - Displays a list of running or waiting Jobs (or both).

Note

nbqstat is deprecated—use **nbstatus jobs** with the appropriate filters instead.

Description

Each Netbatch Pool has two Queues—the wait Queue and the run Queue. Any Job submitted to the Pool is either in the wait Queue or the run Queue. This means the Job has either been dispatched to a Workstation for execution or is still waiting for a Workstation to become available. **nbqstat** shows both waiting and running Jobs, though the output can be restricted to show either only running or only waiting Jobs.

Note

nbqstat has both **regular options** and **display options** that determine which information is displayed.



Usage

```
nbqstat [regular options]
[display options] [operation options]
[<host>|<pool>][ ... ] [special directive]
```

Note

[<host>|<pool>][...] may be used to specify additional hosts or Pools. If -P is used, these additional terms are taken to be Pools. If -H is used, these additional terms are taken to be hosts.

Regular Options

-a

Displays Jobs in all Pools (all Pools listed in **conf/pools**)

-d

Displays debugging information.

-h

Displays help information.

-H <host name>

Displays Jobs for the specified machine name.

-p <pool name>

Displays Jobs for the specified Pool.

-v

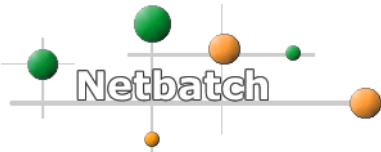
Displays the Netbatch version number and release date.

--queue <queue name>

Displays Jobs for the specified Queue name.

Note

When specifying -P or -H without --queue, nbqstat will display the information that refers to the default Queue of the specified Pool.

**--retry-timeout <time>{h|m}**

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

Display Options

-b

Displays both running and waiting Jobs (this is the default if you do not use **-w** or **-r**).

-c

Displays completed Jobs.

-m

Displays output in Netbatch 5.0-compatible format (that is, with full, unbroken class and Qslot path strings).

-r

Displays running Jobs only.

-s

Displays the total number Jobs without details.

-u

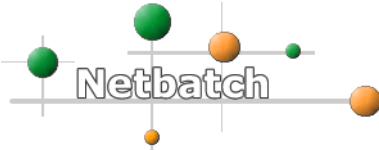
Displays Jobs with unknown (or dirty) Job state information.

-w

Displays waiting Jobs only.

Note

*You can only use one of **-s**, **-f**, **-w**, and **-r**. The exception is that you can use **-r** and **-w** together.*



Operation Options

-f

Displays full format listing of Jobs in the Wait / Run Queue.

-f -e

Displays full format listing of Jobs in all Queues. (Used only with the **-f** option.)

-j

Displays Job name instead of host name for running Jobs.

-l

Print Job dependency information.

-n

Show hierachal names for the Qslots.

--class-reservation

Adds each Job's class reservation requirements to the output

Note

If --class-reservation is used without one of the extended output switches (--verbose, -f, or -f -e), Netbatch automatically appends the --verbose switch.

--cr

See **--class-reservation**.

--fullid

Displays the full Job ID in the format

<original pool name>.<Job ID>

If **--fullid** is not used, displays only the second part (Job ID).

--job-requirements

Display the Job requirements string (smart Class).

Note

If --job-requirements is used without one of the extended output switches (--verbose, -f, or -f -e), Netbatch automatically appends the --verbose switch.

**--license-reservation**

Adds each Job's license reservation requirements to the output.

Note

If **--license-reservation** is used without one of the extended output switches (**--verbose**, **-f**, or **-f -e**), Netbatch automatically appends the **--verbose** switch.

--lr

See **--license-reservation**.

--parallel

Displays additional information for Parallel Jobs.

If **--parallel** is used without **-f** and **-e**, each line with the same JobID in the output represents a Parallel Job's Master and Slave Jobs.

The summary line (**Number of Running Jobs** or **Number of Waiting Jobs**) shows the total number of execution slots. (It does not count a Parallel Job as a single unit.)

If **--parallel** is used with **--rusage**, **nbqstat** shows resource usage for the Parallel Job's Master Job and each of its Slave Jobs separately.

Without **--parallel**, **--rusage** shows the total resource usage for a Parallel Job (that is, totals for the Master Job and the Slave Jobs).

--priority

Displays user-defined priority.

--rc

See **--reserved-class**.

--remote-data

When **nbqstat** is used to view the virtual Pool state, this option is used to display information about remote Jobs. To learn more about virtual Pools see [Chapter 12 Increasing Throughput with Virtual Pools](#).

--reserved-class

Adds details of the resources that have already been committed for each Job to the output.

**Note**

*If **--reserved-class** is used without one of the extended output switches (**--verbose**, **-f**, or **-f -e**), Netbatch automatically appends the **--verbose** switch.*

--reserved-license

Adds details of the licenses that have already been committed for each Job to the output.

--rl

See **--reserved-license**.

Note

*If **--reserved-license** is used without one of the extended output switches (**--verbose**, **-f**, or **-f -e**), Netbatch automatically appends the **--verbose** switch.*

--rusage

Adds details of the resources being used by each running Job to the output.

If **--parallel** is used with **--rusage**, **nbqstat** shows resource usage for the Parallel Job's Master Job and each of its Slave Jobs separately.

Without **--parallel**, **--rusage** shows the total resource usage for a Parallel Job (that is, totals for the Master Job and the Slave Jobs).

Note

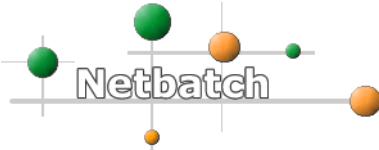
*If **--rusage** is used without one of the extended output switches (**--verbose**, **-f**, or **-f -e**), Netbatch automatically appends the **--verbose** switch.*

--show-hierarchy

Displays hierarchical names for the Qslot.

--task-name

Adds the task name for each Job (if any) to the information displayed.

**--triggers**

For Jobs that were submitted with the **--triggers** switch (that is, that have complex Job dependencies—see [Section 6.6.2, Using nbjob run --triggers to Set Job Dependencies](#)), shows the boolean Job dependency expression that was specified when the Job was submitted.

If **--triggers** is not used, **nbqstat** shows the Job's Trigger Jobs as a simple list.

Note

--triggers only modifies the output when used together with one of the extended output switches (**--verbose**, **-f**, or **-f -e**).

--verbose

Displays full information (like **-f** option) plus priority, submitted from host, time submitted, time on machine, time in running state, times restarted by policy and total times restarted.

Special Directives**Note**

You can use only **one** special directive.

class=<class>

Displays Jobs that require a particular Class of machine.

jobid=<job_ID>

Displays details of the Job with the specified Job ID.

slot=<qslot #>

Displays Jobs for the specified Qslot.

task=<task>

Only displays Jobs that are associated with the specified task.

user=<user>

Displays Jobs owned by the specified user.



Output

Figure 32 illustrates the default `nbqstat` output. **Figure 33** illustrates `nbqstat -f -e` output. **Figure 33** illustrates `nbqstat --verbose` output.

All the available fields are described below (though each display switch causes a different set of fields to be displayed).

```
[itstl107]
      NetBatch QUEUE status - pool martin

      Matbatch v6.2.0_0094 on since: wed Jul 23 11:50:23 2003
      Time now: wed Jul 23 17:01:16 2003

-----
State   JobID    Class     Slot User       Command           Jobname/Server
-----
wait      1 @        1  yoavf   /bin/sleep 1000      null
wait      9 @        1  yoavf   /bin/sleep 1000      null
wait     10 @        1  yoavf   /bin/sleep 1000      null
wait     14 @        1  yoavf   /bin/sleep 1000      null
wait     21 @        1  yoavf   /bin/sleep 1000      null
wait     22 @        1  yoavf   /bin/sleep 1000      null
-----
      Number of jobs in WAITING queue: 6
      No job is RUNNING now.
```

Figure 32: nbqstat Output



[itstl107]

NetBatch QUEUE status - pool martin

Matbatch v6.2.0_0094 on since: Wed Jul 23 11:50:23 2003
Time now: wed Jul 23 17:11:22 2003

Waiting Queue jobs - Extended verbose format

=====

Job id = 1

{

Job Name = /dev/null
Job State = 1 (Wait - waiting job)
Job Descriptor = Job is neither a trigger nor is locked
Trigger Jobs = <Blank>
Dependent Jobs = <Blank>
Licenses = <Blank>
User = yoavf
Command = /bin/sleep 1000
Slot = 1
Class Name = @
Dispatched Class Name = @
Host Name = <Blank>
Source Pool = <Blank>
Source Host = <Blank>
Source Id = <Blank>
Target Pool = <Blank>
Target Host = <Blank>
Target Id = <Blank>
Job Exit Code = <Blank>

}

Number of jobs in the Waiting queue : 13

Figure 33: nbqstat -f -e Output¹

¹⁾The empty Queue messages have been omitted to save space.



```
NetBatch QUEUE status - pool martin

Matbatch v6.5_0134 on since: wed Dec 15 12:32:06 2004
Time now: wed Dec 15 14:27:37 2004

Waiting jobs - Verbose format
=====
User: martinxpx          Job id = 90      Slot: 1 Class Name: @
Priority:      10
Submitted from host: itstl107    Submitted at : 12/15/2004 14:23:22
Job Name:      Host Name:
Wait reason: no resource is available
Times restarted by policy: 0      Total times restarted: 0
Command: sleep 1000

Number of jobs in waiting queue: 1
```

Figure 34: nbqstat --verbose Output¹

Note

-f, --fullid, --verbose, --parallel, and -f -e change the appearance of the output.

Class/Class Name

The Class of machine that the Job needs.

For Jobs that require smart Classes the standard **nbqstat** format will display Class **EXPR**. To view the full smart Class string use **-f** with **--job-requirements** options.

Note

The **EXPR** display for smart Classes is provided only for backwards compatibility, and may not be supported in the future.

To learn more about smart Classes see [Chapter 6.7 Jobs with Special Resource Requirements](#)

¹⁾The empty Queue messages have been omitted to save space.



Class Reservation Request

If the **--class-reservation** switch is used, **nbqstat** adds this field to its output. It shows the Class Reservation that was requested when the Job was submitted.

Class Reservation Implicit Request

If the **--class-reservation** switch is used, **nbqstat** adds this field to its output.

It shows the Implicit Class Reservation that was requested when the Job was submitted—that is, requirements (if any) that are defined for the Queue that the Job was submitted to. These requirements relate to custom attributes defined for each Workstation.

Command

The Job command as submitted with **nbq**.

Dependent Jobs

The list of the Jobs that depend on this Job. If this list is not empty, this Job is a Trigger Job. In other words when this Job is completed successfully, it will trigger (unlock) the dependent Jobs listed here.

If this Job is a Trigger Job for a Job that was submitted with the **--triggers** switch, then it will trigger (unlock) the dependent Job when the condition specified for it (in combination with other specified conditions) is satisfied. (For example, it may have to end with a specific exit code to trigger the dependent Job.)

Dispatched Class Name

The dispatched class name

Host Name

The name of the host where the Job is executed.

For Parallel Jobs, the value is **LIST**.

If the **--parallel** switch is also used, then there is a separate section for each host that the Parallel Job is running on, with indented lines showing:

- ◆ The Job name (that is, the command, not the JobID)
- ◆ The requested class(es)
- ◆ The class(es) of the host
- ◆ A placeholder for the Job exit code

For Parallel Jobs, the first Host subsection is preceded by a **Parallel = Y** line.

**Job Descriptor**

It describes if the Job is locked and/or a trigger Job (that is, another Job is dependent on this one).

Job Exit Code

The Job exit code for the completion status. If the value is greater than or equal to 0, then Netbatch assumes that the Job is completed successfully.

This information is used to unlock the dependent Jobs if it is a successful completion.

JobID

The Job's unique ID number. If **--fullid** is used, the full Job ID, including the Pool name, is displayed.

Jobname/Server

For a waiting Job, the name of the user log file if **-J** was specified.

For a running Job, the name of the workstation that it is running on.

If **-f** or **--verbose** is specified, this information is shown as two separate fields (**Job Name** and **Host Name**).

If the Job is a Parallel Job, the keyword **LIST** appears instead of a Workstation name.

Job Name

Job name defined in **nbq** command with **-j**.

Job State/State - The Job's current state:

Run - The Job is running.

Wait - The Job is in the wait Queue.

Send - The Job is dispatched to a Workstation for execution. The Job is also assigned this state if a pre-execution script is running (as specified with **nbq**'s **--pre-exec** switch).

Fail - Failed to send the Job to the Workstation.

DEL - The Job is being deleted. Waiting for confirmation from Workstation.

Disc-D - Disconnected delete - the delete request has been sent but there is a communication problem between the Pool Master and the Workstation. The Job will be deleted when communication is restored.



Disc-R - Disconnected running - the Job is running on a Workstation, but there is a communication problem between the Pool manager and the workstation. The Job status will change to **Run** when communication is restored.

Move - The Job is being moved to a different Pool.

Licenses

The list of the licenses required by this Job

-Lk slot

Priority

User-defined priority

slot

The name of the Qslot the Job was submitted to, or its numeric alias. If the Qslot does not have a numeric alias, **slot** shows either 0 or nothing. Numeric Qslots with values higher than 999 are cut off after three digits.

Note

nbqstat displays only numeric values of Qslots for backwards compatibility. Use -n or --show-hierarchy to view the full names of string and hierarchical Qslots.

Source Host

For moved Jobs, the name of the source Pool Master host

Source Id

For moved Jobs, the Job ID of the moved Job in the source Pool

Source Pool

For moved Jobs, the name of the source Pool

Submitted at

The date/time when the Job was submitted

Submitted from host

The hostname of the Workstation from which the Job was submitted

**Target Host**

For moved Jobs, the name of the target Pool Master host

Target Id

For moved Jobs, the Job ID of the moved Job in the target Pool

Target Pool

For moved Jobs, the name of the target Pool

Note

Source and Target Host, ID, and Pool are only for moved Jobs.

For example, there is a Job with Job ID 100 in the Pool myPool, whose Netbatch Pool Master is cad911. If we move this Job to the Pool called linuxPool, whose Netbatch Pool Master is cad901, then for the moved Job in linuxPool:

Source Pool will be myPool.

Source host will be cad911.

Source ID will be 100.

The source Pool is notified of this new Job ID. If the Job ID for the moved Job in linuxPool is 5, then in myPool, for the Job with Job ID 100:

Target Pool will be linuxPool.

Target Host will be cad901.

Target ID will be 5.

Times restarted by policy

Number of times Job was restarted by policy

Total times restarted

Total number of times Job was restarted for any reason

Trigger Jobs

The list of the Jobs to which this Job depends, a lock is placed on the Job until the trigger Jobs are completed successfully.

If the **--triggers** switch is used, this displays the boolean Job dependency expression with which the Job was submitted (if the Job was submitted with the **--triggers** switch).

**User**

The Job's owner

Wait reason

Reason for which the Job cannot be dispatched for execution:

- ◆ **no resource is available** - No matching Workstation is available to run the Job, or Netbatch has not yet started searching for a matching Workstation.
- ◆ **qslot inactive** - The Qslot to which the Job was submitted is inactive.
- ◆ **qslot running policies exceeded** - The Qslot to which the Job was submitted has reached its running Jobs limit.
- ◆ **Dependency** - The Job cannot start executing until one or more dependent Jobs finish running.
- ◆ **reservation is not complete** - Some or all of the resources that the Job reserved are not yet available.
- ◆ **Suspend** - The Job has been suspended.
- ◆ **Failed to run pre execution (last failure on machine <machine>)** - The specified pre-execution program failed (on **machine**).
- ◆ **License_not_available <...>** - One or more licenses that the Job requires is not available.

Job Descriptions

If **nbqstat** is used with the **-1** option, the following output results may be expected:

Lk

Job is locked.

Tr

Job triggers open the locked Job.

Bk

Job is both locked and a trigger Job.



Different Job Queues

If `nbqstat` is used with the `-f`, `--fullid`, `--verbose`, or `-f -e` options, a full format listing of Jobs appears in the `nbqstat` output according to the different Job Queues:

Wait

Queue to hold waiting Jobs.

Run

Queue to hold running Jobs.

Move

Queue to hold Jobs that have been moved to other Pools. When the Job is completed at the new Pool, the exit code of the Job is returned to the source Pool.

Notify

Queue to hold completed Jobs, which are originally moved from another Pool. As soon as the exit status of the completed Jobs is notified to the source Pool, these Job are moved into the history Queue. The PoolMaster notifies the source PoolMaster about the exit codes of the completed Jobs as soon as they are completed. If the target Pool PoolMaster is not up or not reachable, the Job will be kept in the notify Queue and PoolMaster will periodically try to notify the source Pool until it successfully completes notification.

History

Queue to hold Jobs that have recently exited the Netbatch system.

Diagnostics

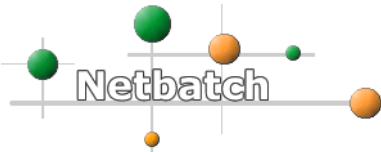
The command exit codes are listed below.

Note

Not all error codes are relevant for this command.

Note

These are command exit codes, not Job exit codes.

**0**

Successful execution

1

Specified Pool Master (-P) does not exist in `pools` file (older commands only (`nbq`, `nbqrm`, `nbstat`, etc.))

2

Unknown host specified (--target for newer commands, -H for older commands)

3

Error occurred on master side (for example, PPM error, Job could not be submitted because of missing permissions, non-existent class, etc.)

4

Problem getting user credentials (most likely an NIS issue on the client machine)

5

PPM timeout (The command waits one minute for a response.)

212

Depends on context:

- ◆ CSD error - Job failed
- ◆ Status command - invalid displayed field

213

Depends on context:

- ◆ CSD error - bad parameters
- ◆ Status command - invalid filter

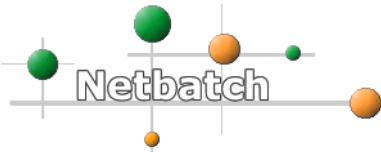
214

Depends on context:

- ◆ CSD error - Workstation not available
- ◆ Status command - invalid grouping

215

Depends on context:



- ◆ CSD error - RG not available
- ◆ Status command - invalid sort

216

CSD error - Qslot not available

221

Interrupted

222

Permission denied

223

Timeout

224

Unknown error

225

Permissions problem

229

The command was run with the **-v** switch (version)

230

The command was run with the **-h** switch (help)

231

Parsing error - mandatory filter not specified

232

Parsing error - invalid value

233

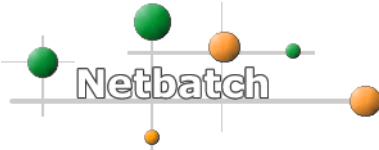
Parsing error - value not specified

234

Parsing error - undefined parameter

235

Parsing error - mandatory parameter not specified

**244**

No command handler in this context

245

No command handler

248

Communication problem - target service down

249

Communication problem - connection broken during receive stage

250

Server too busy (denial of service response or built-in request limit reached)

251

The server does not support the command.

253

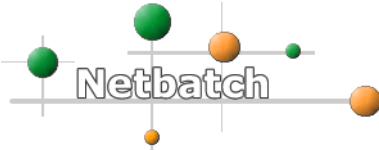
Communication problem - unknown protocol

254

Communication problem - no response

255

Master not responding (probably down)



A.18 nbquery

The **nbquery** command no longer exists—use the relevant **nbstatus** command with the **--history** switch instead.

A.19 nbstat

Name

nbstat - Displays the status of all the Workstations in a Pool.

Note

nbstat is deprecated—use **nbstatus workstations** instead.

Description

Displays the status of all the Workstations in a Pool.

If you need more detailed information than **nbstat** can provide, you can use the relevant **nbstatus** with the **--history** switch to query Netbatch Tracker.

Usage

nbstat [options] [pools] or [Hosts] ...

Command-line Options

-a

Displays status of Workstations in all Pools.

-c

Specifies that Class information should be displayed for each Workstation.

-d

Displays debugging information.

-e

Displays the current “C/R/S/N/H/L” status, where:

- ◆ C is the maximum number of Jobs running concurrently on the server



- ◆ **R** is the number of running Jobs
- ◆ **S** is the number of stopped Jobs
- ◆ **N** is the number of niced Jobs
- ◆ **H** is the number of hung Jobs
- ◆ **L** is the number of runaway Jobs

-h

Displays help information.

-H <host name>

Specifies the name of the manager machine of the Pool to show workstation status for.

-j <hostname> | "<hostname>[. . .]" | =all>

Displays in record format (vs. line format) for each server in the list (or all servers in the case of =all), all the information corresponding to the nbstat options, plus the server name, IP address, version of garcon binary, and the Job IDs for running Jobs, stopped Jobs, niced Jobs, hung Jobs and runaway Jobs. If the corresponding garcon is of version 4.x or earlier, the Job id information will not be available for displaying

-j <hostname> | "<hostname>[. . .]" | =all>**--attributes**

Displays in record format (vs. line format) for each server in the list (or all servers in the case of =all), including the attributes of the servers that can be requested by the Jobs submitted by the user. To learn more about specifying Job requirements see [Chapter 6.7](#)

[**Jobs with Special Resource Requirements**](#).**-j <{a list of servers} |=all> --fullid**

Same as -J, but displays full Job ID format for the Jobs on each server.

-m

Specifies that virtual memory usage information should be displayed for each Workstation.

-p <poolname>

Specifies the name of the Pool to show Workstation status for. (This is the default.)

**-s**

Specifies that only summary information should be displayed.

-v

Displays the Netbatch version number and release date.

--normalize

Display normalized load values (if the administrator has not enabled display of normalized load values as the default).

If the administrator **has** enabled display of normalized load values as the default, using this switch displays the **non-normalized** load values.

--parallel

Modifies the output to take Parallel Jobs into account. It shows machines running Parallel Jobs as:

- ◆ <number> p (for Master Jobs)
- ◆ <number> p (for Slave Jobs)

--parallel can be used together with **-j** to add a list of Parallel Jobs (which is a subset of Running Jobs) to the **-j** output.

--queue <queue_name>

Specifies that output should include only resources that can be used by the specified Queue.

Note

These resources may also be configured so that they can be used by Queues other than the specified one.

--reserved

Specifies that output should include details of resources that have already been committed for use (for Jobs that have been submitted with specified resource reservation requirements).

When used without the **-j** switch, the output includes an extra line below the **JOBs** line, summarizing the reserved resources.

When used with the **-j** switch, the output includes an extra section listing the following for each Job:

- ◆ The JobID
- ◆ The Job's state
- ◆ The resources reserved for the Job, as follows:



- ◆ VM - virtual memory
- ◆ RM - real memory

--resource-group <resourceGroupName>

Specifies that status information should be displayed only for Workstations belonging to the specified Resource Group. Use only when **-H** specifies the physical Pool Master.

--resource-set <resourceSetName>

Specifies that status information should be displayed only for Workstations belonging to the specified Resource Set. Use only when **-H** specifies the physical Pool Master.

Note

*When using the options **-H** or **-P** without specifying **--resource-set**, nbstat displays information for the default Resource Set, i.e., the Resource Set that includes all the Workstations in the Pool.*

--retry-timeout <time>{h|m}

Specifies that if Netbatch cannot handle the request (for example, if the Pool is down, if a request limit has been reached, or if the Pool is under heavy load), the command client will keep trying to submit the request for the specified time. By default, the interval between attempts is 4 minutes, though the Netbatch Administrator can specify a different interval.

If you specify a timeout that is shorter than the retry interval, two attempts are made, once when you run the command, and once at the end of the timeout that you specify.

If the command can never be run successfully (for example, if you send it to a non-existent Pool), no additional attempts to send it are made.

--rusage**Note**

*--rusage must be used together with the **-j** switch.*

Specifies that output should include resource usage information for the specified server.



--rusage adds an extra section to the output, listing the following for each Job:

- ◆ The JobID
- ◆ The Job's state
- ◆ The resources used by the Job, as follows:
 - ◆ VM - virtual memory
 - ◆ RM - real memory
 - ◆ utime - user time
 - ◆ stime - system time
 - ◆ wtime - wall clock time

Output

Figure 35 illustrates nbstat output with the -e option.

```
[itstl107]
[NetBatch pool martin]

Maitrd v6.2.0_0094 on since: Wed Jul 23 11:50:23 2003
Time now: Wed Jul 23 17:15:04 2003

SERVER      STATUS     LOAD     USERS    IDLE     SEL    LAST SELECT    ON SINCE   C/R/S/N/H/L
-----      -----
itstl107      D          (Garcon server not reachable OR Machine is down)

SERVERS: Available:    0  Connected:    0  Configured:    1  Locked:      0
          Running:      0  Stopped:      0  Interactive:   0  High Load:    0
JOBS:    Total:        0  Can_Run:     0  Blackhole:    0  High Memory:  0
CLASSES: martin itstl107
```

Figure 35: nbstat Output with the -e Option

Interpreting the Output

SERVER

The name of the workstation.

STATUS

Workstation status as follows. This can be:

Blank

The Workstation is not accepting Jobs because the load or number of interactive users exceeds thresholds, or because it has been committed (using resource reservation) for a waiting Job.

*



The Workstation can accept Jobs. If there are no other characters, it is not running any Jobs.

<n> R

The Workstation is running n Jobs.

<n> S

This Workstation has n Jobs, at least one is suspended.

<n> N

This Workstation has n Jobs, at least one is niced.

*** <n> R**

The Workstation is running n Jobs, and can accept more.

B

Blackhole

L

The workstation is locked. This does not affect Jobs that started running before it was locked.

V

The workstation virtual memory is low (below configured threshold)

D

The workstation is disconnected (or wrong machine name specified).

Note

If the **--parallel** switch is used, the Workstation Status can also include **P** (the Workstation is running a Parallel Master Job) or **p** (the Workstation is running a Parallel Slave Job).

LOAD

The first number is the **current load** (calculated as an average of the number of running processes 1 minute, 5 minutes, and 15 minutes ago).

The second number is a **load threshold**. If the load exceeds this threshold, no new Jobs are accepted. If the load drops below this threshold, any suspended/niced Jobs are resumed/reniced.

The third number is also a **load threshold**. If the load exceeds this threshold, Jobs that are already running are suspended or niced (depending on the workstation configuration).

**Note**

The load values that are displayed can be either:

- Normalized
- Non-normalized

depending on the way the administrator has configured it.

Use the **--normalize** switch to view the non-default load values (that is, normalized if the default is non-normalized, and non-normalized if the default is normalized).

USERS

The first number is the **number of active users** on the workstation. (This depends on the definition of an active user in the workstation configuration.)

The second number is the **active user threshold**. If the number of active users exceeds this threshold, no new Jobs are accepted and running Jobs are suspended.

IDLE

These two numbers are the parameters that Netbatch uses to determine whether a user is active or not. It uses this information for deciding whether to accept new Jobs and whether to suspend running Jobs.

For the purpose of accepting new Jobs, a user is considered active if there have been no keystrokes for <first number> minutes.

For the purpose of suspending and resuming running Jobs, a user is considered active if there have been no keystrokes for <second number> minutes.

SEL

The number of times the workstation has been selected for Job execution since it was connected to the Pool Master.

LAST SELECT

The date and time that the workstation was last selected to run a Job.

ON SINCE

The date and time that the workstation was connected to the Pool manager.

**C/R/S/N/H/L**

- ◆ **C** is the maximum number of Jobs running concurrently on the Workstation.
- ◆ **R** is the number of running Jobs.
- ◆ **S** is the number of stopped Jobs.
- ◆ **N** is the number of niced Jobs.
- ◆ **H** is the number of hung Jobs.
- ◆ **L** is the number of runaway Jobs.

Note

If the Workstation is running parts of a Parallel Job, then the number of running Jobs includes each individual execution slot used.

*For example, if there are **two** regular Jobs and **two** Parallel Slave Jobs (each of which uses one execution slot) running on a Workstation, then the total number of Running Jobs is **four**.*

Note

C/R/S/N/H/L status is displayed with the -e option.

Interpreting Output with -j Option

Using **nbstat** with the **-j** option displays the following record format for each server.

Each record begins with the keyword **Begin**, ends with the keyword **End**, and is compound of 2 separate sections:

Machine:

- ◆ **Host** - Server name
- ◆ **IP** - Server IP address
- ◆ **Version** - Netbatch version that the server is running
- ◆ **status** - Server state
- ◆ **LOAD** - Load current information and thresholds
- ◆ **USERS** - Interactive users current information and threshold
- ◆ **IDLE** - Idle Jobs threshold



- ◆ **NUM_SELECTED** - The number of time the server was selected to execute Jobs
- ◆ **LAST_SELECTED** - The last time the server was selected to execute a Job.
- ◆ **ON_SINCE** - Server startup time
- ◆ **LOCK_REASON** - The reason why the Workstation is locked (if locked)

Jobs:

- ◆ **Running** - JobID list of Jobs that are executed on the server
- ◆ **Stopped** - JobID list of Jobs that are cancelled on the server
- ◆ **Niced** - JobID list of Jobs that are reniced on the server
- ◆ **Hung** - JobID list of Jobs that are hung on the server.
- ◆ **Runaway** - JobID list of Jobs that runaway from the server

To learn more about hung and runaway Jobs, see [Section 6.5.13 Job Hung Limits](#) and [Section 6.5.12 Job Execution Limits](#)

Note

If the --reserved switch is used, a section is displayed for each Job for which resources have been reserved (including the JobID, Job state, and details of the reserved resources).

If the --rusage switch is used, a section is displayed for each Job that is using resources on the Workstation (including the JobID, Job state, and details of the resources used).

Note

If the --parallel switch is used, a list of Parallel Jobs (that is, the JobIDs of Parallel Master and/or Slave Jobs) appears after the list of Running Jobs.

Note

When using --fullid, the Job IDs in the Job section will appear a long format, i.e:

`<poolname>.<jobid>`

**Note**

The load values that are displayed can be either:

- Normalized
- Non-normalized

depending on the way the administrator has configured it.

Use the **--normalize** switch to view the non-default load values (that is, normalized if the default is non-normalized, and non-normalized if the default is normalized).

Interpreting Output with -j and --attributes Options

Using **nbstat** with the **-j** and **--attributes** options displays the following record format for each server.

Each record begins with the keyword **Begin**, ends with the keyword **End**, and is compound of 3 separate sections:

Machine:

- ◆ **Host** - Server name
- ◆ **IP** - Server IP address
- ◆ **Version** - Netbatch version that the server is running
- ◆ **Status** - Server status
- ◆ **Low load threshold** - Load low threshold
- ◆ **High load threshold** - Load high threshold
- ◆ **Interactive users threshold**
- ◆ **Idle** - Idle Thresholds
- ◆ **Selected** - The number of time the machine was selected to execute Jobs
- ◆ **Last time selected** - The last time the machine was selected to execute Jobs
- ◆ **On Since** - Server startup time
- ◆ **Virtual Memory threshold** - Virtual Memory threshold

Attributes:

- ◆ **Host** - Server name
- ◆ **Load** - Current load information



- ◆ **Nload** - Current normalized load information (if Workstation load normalization is enabled)
- ◆ **InteractiveUsers** - Current interactive users information
- ◆ **HW** - Hardware type
- ◆ **OSName** - Operating system name
- ◆ **OSRelease** - Operating system release
- ◆ **OSVersion** - Operating system version and date
- ◆ **Arch** - Architecture type
- ◆ **CPUCount** - Number of CPUs
- ◆ **CPUMhz** - CPU frequency
- ◆ **Classes** - Classes supported by the server
- ◆ **fDS** - Free disk space (in MB) in partitions on which particular configured directories reside
- ◆ **tDS** - Total disk space (in MB) in partitions on which particular configured directories reside
- ◆ **tFDS** - Total free disk space (in MB)
- ◆ **/tmp** - Total free disk space (in MB) in **/tmp**
- ◆ **BuffersRM** - the amount of physical RAM used for file buffers (MB)
- ◆ **CachedRM** - the amount of physical RAM used as cache memory; memory in the pagecache (diskcache) minus SwapCache (the amount of Swap used as cache memory) (MB)
- ◆ **InactiveRM** - the total amount of buffer or page cache memory that is free and available; this is memory that has not been recently used and can be reclaimed for other purposes by the paging algorithm (MB)
- ◆ **fRM** - Free real memory; the amount of physical RAM left unused by the system (MB)
- ◆ **fVM** - Free virtual memory; the total amount of swap memory free (MB)
- ◆ **tRM** - Total real memory; the total usable RAM (i.e. physical memory minus a few reserved bytes and the kernel binary code) (MB)
- ◆ **tVM** - Total virtual memory; the total amount of physical swap memory (MB)

Jobs:



- ◆ **Running** - JobID list of Jobs that are executed on the server
- ◆ **stopped** - JobID list of Jobs that are cancelled on the server
- ◆ **Niced** - JobID list of Jobs that are reniced on the server
- ◆ **Hung** - JobID list of Jobs that are hung on the server
- ◆ **Runaway** - JobID list of Jobs that runaway from the server

Note

*The **BuffersRM**, **CachedRM**, and **InactiveRM** attributes are available only for SUSE Linux Enterprise Server 9 and later.*

Note

*If the **--reserved** switch is used, a section is displayed for each Job for which resources have been reserved (including the JobID, Job state, and details of the reserved resources).*

*If the **--rusage** switch is used, a section is displayed for each Job that is using resources on the Workstation (including the JobID, Job state, and details of the resources used).*

Note

*If the **--parallel** switch is used, a list of Parallel Jobs (that is, the JobIDs of Parallel Master and/or Slave Jobs) appears after the list of Running Jobs.*

Note

*When using **--fullid**, the Job IDs in the Job section will appear a long format, i.e:*

`<poolname>.<jobid>`

Note

The load values that are displayed can be either:

- *Normalized*



- *Non-normalized*

depending on the way the administrator has configured it.

Use the --normalize switch to view the non-default load values (that is, normalized if the default is non-normalized, and non-normalized if the default is normalized).

Diagnostics

The command exit codes are listed below.

Note

Not all error codes are relevant for this command.

Note

These are command exit codes, not Job exit codes.

0

Successful execution

1

Specified Pool Master (-P) does not exist in `pools` file (older commands only (`nbq`, `nbqrm`, `nbstat`, etc.))

2

Unknown host specified (--target for newer commands, -H for older commands)

3

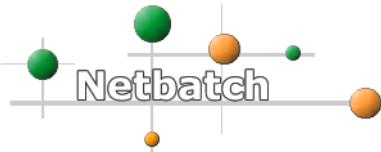
Error occurred on master side (for example, PPM error, Job could not be submitted because of missing permissions, non-existent class, etc.)

4

Problem getting user credentials (most likely an NIS issue on the client machine)

5

PPM timeout (The command waits one minute for a response.)

**212**

Depends on context:

- ◆ CSD error - Job failed
- ◆ Status command - invalid displayed field

213

Depends on context:

- ◆ CSD error - bad parameters
- ◆ Status command - invalid filter

214

Depends on context:

- ◆ CSD error - Workstation not available
- ◆ Status command - invalid grouping

215

Depends on context:

- ◆ CSD error - RG not available
- ◆ Status command - invalid sort

216

CSD error - Qslot not available

221

Interrupted

222

Permission denied

223

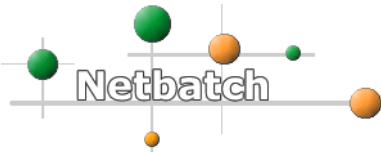
Timeout

224

Unknown error

225

Permissions problem

**229**

The command was run with the **-v** switch (version)

230

The command was run with the **-h** switch (help)

231

Parsing error - mandatory filter not specified

232

Parsing error - invalid value

233

Parsing error - value not specified

234

Parsing error - undefined parameter

235

Parsing error - mandatory parameter not specified

244

No command handler in this context

245

No command handler

248

Communication problem - target service down

249

Communication problem - connection broken during receive stage

250

Server too busy (denial of service response or built-in request limit reached)

251

The server does not support the command.

253

Communication problem - unknown protocol

**254**

Communication problem - no response

255

Master not responding (probably down)

A.20 nbvm

Name

nbvm - instantiate and shutdown virtual machines and clusters of virtual machines via Netbatch.

Description

nbvm is a set of commands to create and remove clusters of virtual machines through Netbatch, and to boot and shutdown these virtual machines, as follows:

- ◆ **nbvm boot** - boot a virtual machine. See [Appendix A.20.1](#).
- ◆ **nbvm cluster-create** - create a cluster of virtual machines. See [Appendix A.20.2](#).
- ◆ **nbvm cluster-remove** - remove a cluster of virtual machines. See [Appendix A.20.3](#).
- ◆ **nbvm shutdown** - shut down a virtual machine. See [Appendix A.20.4](#).



A.20.1 nbvm boot

Name

nbvm boot - boot a virtual machine through Netbatch.

Description

nbvm boot boots a virtual machine from the available OS images on a Workstation in the specified Pool.

Usage

```
nbvm boot [--target <pool_name>|<host_name>]
--image <image> --cores <cores>
--memory <memory> [--cluster-id <cluster_id>]
[--class <class_expression>] [--qslot <qslot>]
```

Target

--target <pool_name>|<host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbjob** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

--class <class_expression>

Specifies the class of machine that the VM will run on. The expression can include static attributes (such as OS type and number of CPUs) and dynamic attributes (such as free virtual memory and load).

--cluster-id <cluster_id>

The ID of the cluster that the VM belongs to. If you do not specify a cluster, your default cluster will be used (and will be created if it does not yet exist). Your default cluster is called

<username>_DEFAULT_CLUSTER.

--cores <cores>

The number of cores required.



--image <image>

The OS image to be used.

--memory <memory>

The amount of memory required (e.g., 4 for 4GB).

--qslot <qslot>

The Qslot to which the "Job" will be submitted. If you omit this switch, it will be submitted to the default Qslot.

A.20.2 nbvm cluster-create

Name

nbvm cluster-create - create a cluster of virtual machines.

Description

nbvm cluster-create creates a cluster of virtual machines. The cluster acts like a mini Netbatch Pool that you can submit Jobs to.

Usage

```
nbvm cluster-create [--target  
<pool_name>|<host_name>]  
--cluster-name <cluster_name>
```

Target

--target <pool_name>|<host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbjob** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

--cluster-name <cluster_name>

Specifies the name of the cluster.



A.20.3 nbvm cluster-remove

Name

nbvm cluster-remove - remove a cluster of virtual machines.

Description

nbvm cluster-create removes a cluster of virtual machines.

Usage

```
nbvm cluster-remove [--target  
<pool_name>|<host_name>]  
--cluster-id <cluster_id>
```

Target

--target <pool_name>|<host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbjob** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

--cluster-id <cluster_id>

Specifies the ID of the cluster to be removed.

A.20.4 nbvm shutdown

Name

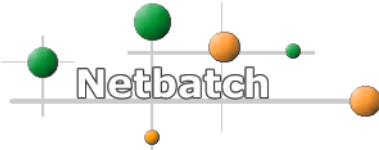
nbvm shutdown - shut down a virtual machine.

Description

nbvm shutdown shuts down a virtual machine.

Usage

```
nbvm shutdown [--target <pool_name>|<host_name>]  
--vmid <vm_id>
```



Target

--target <pool_name> | <host_name>

Specifies the Pool Master, either by Pool name or by Pool Master host name. (If **--target** is omitted, **nbjob** uses the Pool defined in the **NBPOOL** environment variable, or if **NBPOOL** is undefined, the Pool of which the Workstation where the command was run is a member.)

Options

--vmid <vm_id>

Specifies the ID of the virtual machine to be shut down.



Appendix B: Precedence of Configuration Directives, Environment Variables, Job Profiles, and Command-Line Switches

Some parameters can be defined in a number of different places. Generally, the precedence for the definition of these parameters is as follows (an item that is further down the list overrides one that is higher up):

1. Directive configured by the Netbatch Administrator
2. **default** directive in a Job Profile (specified switches can be overridden by user-specified options)¹
3. Environment variable
4. User's Job Profile
5. Command-line switches
6. **force** directive in a Job Profile (specified switches **cannot** be overridden by user-specified options)¹
7. **append/prepend** directive in a Job Profile (specified switches are added to those specified by the user)^{1,2}
8. **remove** directive in a Job Profile (any user-specified switches that match those specified here are removed)¹
9. **replace** directive in a Job Profile (replaces part of a parallel reservation string)¹

For Example

If a parameter is defined in an environment variable and by the user on the command line, the command line overrides the environment variable.

¹⁾A Job Profile can be configured at the Pool, Queue, or Qslot level. A directive specified in a Qslot Job Profile overrides the same directive specified at the Queue level, which, in turn, overrides the same directive specified at the Pool level.

²⁾append and prepend directives add arguments either before or after those specified by the user. For example, if a Job Profile containing the directive `append -C "fvm>10"` is applied to a Job called `myJob` that a user submits with the following `nbjob run` command:

```
nbjob run --class myClass myJob
then it is actually submitted with the following switch:
--class "myClass&&fvm>10"
```

**Note**

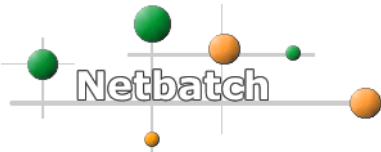
A parameter defined in a Job Profile only applies to Jobs that match the defined specification.

However, there are exceptions. For example, some configuration directives specify limits (a user can specify a value within the defined limits), while others specify default values that the user can override. There are also slightly different precedence rules for certain switches (`--class-reservation` and `--job-constraints`), and when applied using a Job profile—see [Appendix E.5.14, JobProfile](#).

For each parameter that can be defined in more than one place, [Table 5](#), below, explains the specific rules that apply to it.

Table 5: Precedence of Configuration Directives, Environment Variables, Job Profiles, and Command-Line Switches

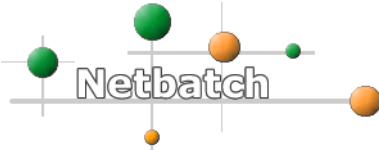
Parameter	Can Be Overridden by the Netbatch Administrator?	Environment Variable	Command-Line Switch	Precedence Rules
Default Pool	Yes	NBPOOL	<code>nbjob run --target</code>	The usual precedence rules apply— <code>nbjob run --target</code> overrides NBPOOL.
Default Queue	Yes	NBQUEUE	<code>nbjob run --queue</code>	The usual precedence rules apply— <code>nbjob run --queue</code> overrides NBQUEUE.
Default Qslot	Yes	NBQSLOT	<code>nbjob run --qslot</code>	<p>The default Qslot defined by the Administrator is only used if a Job is submitted without a Qslot specified (and without NBQSLOT defined).</p> <p>For Jobs submitted with a Qslot specified, the usual precedence rules apply—<code>nbjob run --qslot</code> overrides NBQSLOT.</p>
Default class	Yes	NBCLASS	<code>nbjob run --class</code>	The usual precedence rules apply— <code>nbjob run --class</code> overrides NBCLASS.

**Table 5: Precedence of Configuration Directives, Environment Variables, Job Profiles, and Command-Line Switches**

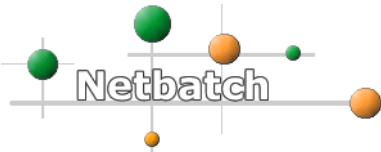
Parameter	Can Be Overridden by the Netbatch Administrator?	Environment Variable	Command-Line Switch	Precedence Rules
Execution limits	Yes	NB_EXEC_LIMITS	nbjob run --exec-limits	Precedence is as follows (an item that is lower on the list overrides a higher one): <ol style="list-style-type: none">1. Pool Job Profile2. Queue Job Profile3. Qslot Job Profile4. NB_EXEC_LIMITS5. User Job profile6. nbjob run --exec-limits7. Gexeclimits
Hung limits	Yes	NB_HUNG_LIMITS	nbjob run --hung-limits	Precedence is as follows (an item that is lower on the list overrides a higher one): <ol style="list-style-type: none">1. Pool Job Profile2. Queue Job Profile3. Qslot Job Profile4. NB_HUNG_LIMITS5. User Job profile6. nbjob run --hung-limits7. Ghunglimits
Resource limits	Yes		nbjob run --rlimits	Limits set by --rlimits are ignored if greater than the default limits (as set by NETSTAR_DEFAULT_STACKSIZE, WSM_ULIMIT, or by the shell in which the Workstation Manager process was started).
Pre-execution program	No	NB_PRE_EXEC	nbjob run --pre-exec	The usual precedence rules apply—nbjob run --pre-exec overrides NB_PRE_EXEC.

**Table 5: Precedence of Configuration Directives, Environment Variables, Job Profiles, and Command-Line Switches**

Parameter	Can Be Overridden by the Netbatch Administrator?	Environment Variable	Command-Line Switch	Precedence Rules
Post-execution program	No	NB_POST_EXEC	nbjob run --post-exec	The usual precedence rules apply—nbjob run --post-exec overrides NB_POST_EXEC.
Job starter	No	NB_JOB_STARTER	nbjob run --job-starter	The usual precedence rules apply—nbjob run --job-starter overrides NB_JOB_STARTER.
License requirements	Yes	NBLICENSES	nbjob run --license	The usual precedence rules apply—nbjob run --license overrides NBLICENSES.
License keyfile location	Yes	NB_LICENSE_FILE	nbjob run --license-keyfile	The usual precedence rules apply—nbjob run --license-keyfile overrides NB_LICENSE_FILE.
Automatic Job requeuing	Yes	NB_ON_JOB_FINISH	nbjob run --on-job-finish	The usual precedence rules apply—nbjob run --on-job-finish overrides NB_ON_JOB_FINISH. --on-job-finish takes precedence over --autoreq.

**Table 5: Precedence of Configuration Directives, Environment Variables, Job Profiles, and Command-Line Switches**

Parameter	Can Be Overridden by the Netbatch Administrator?	Environment Variable	Command-Line Switch	Precedence Rules
Netbatch Feeder limits	No	NBFEED_CONF Specifies a common configuration file.	nbfeeder --conf-file <file> nbfeeder --feeder-file <file> Either file can contain these configuration directives.	Configuration directives defined by the Administrator act as limits —all directives (in a common configuration file or set by the user in the Feeder file or in a separate configuration file) must be within these limits. If a directive exceeds one of these limits, the limit value is used. If NBFEED_CONF is set, the directives in the specified file override any directives in a Feeder file or configuration file specified on the command line.
Use of Process Authentication Groups (PAG)	Yes	—	nbjob run --pag on nosuspend	The usual precedence rules apply. If the user does not specify the --pag switch, Netbatch checks the applicable Job Profile to see whether to apply a PAG. If the Job Profile does not specify whether to add a PAG, Netbatch decides whether to apply a PAG based on how the Workstation that the Job is sent to is configured.

**Table 5: Precedence of Configuration Directives, Environment Variables, Job Profiles, and Command-Line Switches**

Parameter	Can Be Overridden by the Netbatch Administrator?	Environment Variable	Command-Line Switch	Precedence Rules
SIGKILL timeout	No	NB_SIGKILL_TIMEOUT	—	<p>The user can set NB_SIGKILL_TIMEOUT to specify the time that Netbatch waits after sending SIGTERM to a Job before sending SIGKILL (to kill the Job). (GDefSigKillTimeOut specifies the default value.)</p> <p>If this value is greater than the limit specified by GMaxSigKillTimeOut, the limit value is used.</p>
SIGSTOP timeout	No	NB_SIGSTOP_TIMEOUT	—	<p>The user can set NB_SIGSTOP_TIMEOUT to specify the time that Netbatch waits after sending SIGTSTP to a Job before sending SIGSTOP (to suspend the Job). (GDefSigStopTimeOut specifies the default value.)</p> <p>If this value is greater than the limit specified by GMaxSigStopTimeOut, the limit value is used.</p>
Path mapping file	No	NBNTPATH TABLE	—	<p>If Gnbntpathtable defines a global path mapping file, the user can set NBNTPATHTABLE to specify a different path mapping file.</p> <p>The user can also set other environment variables to specify that path mapping be performed selectively.</p>



CONFIDENTIAL

Netbatch
v8.2.2

Revision 4

Precedence of Configuration Directives,



CONFIDENTIAL

Netbatch
v8.2.2

Revision 4

Precedence of Configuration Directives,



Appendix C: The New Netbatch Command Suite

As of version 6.4, Netbatch has a new suite of commands that provide the same functionality as the existing commands, but with a standardized, easy-to-use interface.

The new commands are:

- **nbjob**
- **nbstatus**

The commands are described in the following tables, along with the equivalent existing commands.

Table 7: nbjob

Command	Equivalent Existing Command	Description
nbjob run	nbq	Submits a Job for execution.
nbjob modify	nbqrm -s, -c, -m, --class-reservation, --license-reservation, --license-keyfile, --parallel, --new-priority	Modifies the parameters of a Job that has already been submitted.
nbjob remove	nbqrm [-y]	Removes one or more Jobs that have already been submitted.
nbjob suspend	nbqrm --suspend	Suspends one or more Jobs that have already been submitted.
nbjob resume	nbqrm --resume	Resumes one or more suspended Jobs.
nbjob recall	nbqrm -s	Resubmits one or more Jobs that have already been submitted.
nbjob prun	nbexec	Executes a Parallel Job's Slave Jobs.
nbjob pmodify	N/A	Modifies Parallel Job.

Table 8: nbstatus

Command	Equivalent Existing Command	Description
nbstatus <sub-command> --history	nbquery	Query the Netbatch Tracker database for information.
nbstatus denial-of-service	N/A	Displays the number of requests received from each user and Workstation in the current interval.
nbstatus feeders	nbreport	Displays Feeder information.
nbstatus groups	N/A	Displays information about Netbatch and NIS groups.

**Table 8:** nbstatus

Command	Equivalent Existing Command	Description
nbstatus jobs	nbqstat	Displays Job information.
nbstatus keyfiles	N/A	Displays information about license keyfiles.
nbstatus licenses	nblicense	Displays license information.
nbstatus license-sessions	N/A	Displays information about license sessions.
nbstatus loadbalancer	N/A	Displays information about Load Balancer services.
nbstatus locks	N/A	Displays information about Workstation locks (Workstations that are currently locked or scheduled to be locked).
nbstatus logical-licenses	N/A	Displays information about logical licenses (that is, for all available instances of each license, no matter how many different keyfiles they are listed in).
nbstatus permissions	N/A	Displays information about configured permissions.
nbstatus pools	N/A	Displays Pool information.
nbstatus probes	N/A	Displays information about scheduler probes.
nbstatus qslots	nbqslot	Displays Qslot information.
nbstatus resource-groups	N/A	Displays information about Resource Groups.
nbstatus resources	N/A	Displays information about virtual resources.
nbstatus services	N/A	Displays information about registered services. Currently, registered services include only Netbatch Tracker.
nbstatus wan	N/A	Displays WAN status.
nbstatus workstations	nbstat	Displays Workstation information.

For a full explanation of each of these commands, see the following appendixes:

- [Appendix A.2, nbjob](#)
- [Appendix A.3, nbstatus](#)



Appendix D: Log File Contents and Job Exit Codes

D.1 Log File Name and Directory

When a Job ends, Netbatch puts the output file (the user log file) in the directory from which the Job was submitted (or in the file and directory specified by the `--log-file` option of the `nbjob run` command).

Note

If the WSM is down for some reason when the Job ends, writing the log file is not completed, as Netbatch does not have the required information. When the WSM comes back up, writing of the log file is completed.

If Netbatch has a problem writing to this directory, it tries to write the file to the first directory specified in the **NBWD** environment variable. If it cannot write to the first one, it tries the second, and so on, until it finds a directory that it can write to. If Netbatch cannot write to any of the directories specified in **NBWD**, the Job will not run, and an exit status of -31 is recorded.

By default the log file will have a name of the following form:

```
##MMM-dd-hh:mm:ss#<machine name>
```

D.1.1 Log File Contents

The log file is made up of three sections as follows:

- The first and third sections contain Netbatch information
- The second section contains the output (`stdout` and `stderr`) from the Job.

D.1.2 First Section Contents

Table 1 lists the contents of the first section of the log file.

Table 1: Log File Contents, First Section

Name	Description
Log file	The name of the log file.
Job id	The ID number of the Job.
Class	The class of machine that the Job required.
Qslot	The Qslot the Job was submitted to.
Executed on	The name of the machine where the Job was executed.

**Table 1: Log File Contents, First Section**

Name	Description
Pool	The name of the pool that the Job was submitted to.
Queuing time	The time and date that the Job was submitted to the queue.
Starting time	The time and date that the Job started running.
Qwait	The amount of time the Job was waiting in the queue.
Command	The Job command as typed at the command line (excluding nbjob run and its options).

D.1.3 Third Section Contents

Table 2 lists the contents of the third section of the log file.

Table 2: Log File Contents (Third Section)

Name	Description	
Exit status	The exit status of the Job—zero or positive if successful, negative if an error occurred See below for an explanation of the error codes.	
NB	If the Job finished with an exit code other than zero, this line is added to provide additional information.	
Finishing time	The date and time that the Job finished	
CPU time	The amount of CPU time that the Job used. This is broken down into: Usr—CPU time spent running user code Sys—CPU time spent running system calls WC—Wall clock time	
Rusage Stats:	Mem PF CSv Swaps Msg IOops Sigs	Memory resident set size in MB Number of page faults Number of context switches (voluntary/forced) Number of swaps Number of outgoing/incoming IPC messages Number of IO block operations (in/out) Number of signals delivered to user Job



D.2 Exit Status

If the Job's exit status is zero or positive, it means the Job was executed successfully.

A negative exit status means that an error occurred.

Note

If you need to access a Job's exit status without looking at the log file, you can use a post-execution script (by adding the --post-exec switch to nbjob run).

In the post-execution script, Netbatch sets the value of the NB_JOB_EXIT_STATUS environment variable to the Job's exit status.

Table 3: Exit Status Error Codes

Error Name	Exit Status Value	Description
X_EXEC	-4	Job execution failed on remote machine.
X_NOFND	-5	Request unknown.
X_MIDWAY	-6	The Job leader died for unknown reason. Netbatch resubmits such Jobs a limited number of times. This way, if the problem is not Job-related, the Job gets to run, but if the problem <i>is</i> Job-related, not too many resources are wasted.
	-7	The Job was killed by a user request while it was in the wait state queue. This is recorded in the matbatch log file.
X_REMOTE_KILL	-8	Job killed by a specific remote request (nbjob remove).
X_SELF_KILL	-9	Job killed by owner request (user submitted Job with --kill or --resubmit option).
X_RESUBMIT	-10	Job killed by resubmit command (nbjob resubmit).
X_DELFAIL	-11	Job delete request failed.
	-13	Job killed by a specific remote superuser request (nbjob remove).
	-14	The Job was resubmitted by a specific remote request, and returns to the wait queue.
	-15	The Job was resubmitted by a specific remote superuser request, and returns to the wait queue.
	-16	Job was killed because it was preempted.
	-17	Job was preempted, and returns to the wait queue.

**Table 3: Exit Status Error Codes**

Error Name	Exit Status Value	Description
	-18	Netbatch killed the Job because a Workstation threshold was exceeded (e.g., load, number of interactive users, etc.).
	-19	Job was killed by a policy (except a <code>wvmem</code> policy).
SYS2USR	-21 to -29	System to user mode change errors.
	-21	Failed to get entries from password file.
	-22	Failed to set user group ID during Job execution.
	-23	Failed to set user initgroups during Job execution.
	-24	Failed to set user ID during Job execution.
	-25	System to user mode change error.
	-26	System to user mode change error.
	-27	System to user mode change error.
	-28	System to user mode change error.
	-29	System to user mode change error.
OPENLOG	-30 to -39	User log file creation errors.
	-30	Failed to open the log file. No such file or directory. ¹⁾ When using NB Feeder, this can be caused by specifying an invalid working directory.
	-31	Failed to open the log file. Disk quota exceeded. ¹⁾
	-32	Failed to open the log file. Permission denied. ¹⁾
	-33	Failed to open the log file. Other errors. ¹⁾ When using NB Feeder, this can be caused by specifying an invalid working directory.
	-34	Failed to write to the log file. I/O error. ¹⁾
	-35	Failed to write to the log file. Other error. ¹⁾
	-36	Internal interactive socket error ¹⁾
	-37	Internal output pipe error ¹⁾
	-38	Internal error pipe error ¹⁾
	-39	Internal abort pipe error ¹⁾
FORKPIPE	-40 to -49	Errors during <code>fork()</code> and <code>exec()</code> of user Job on remote machine.
	-40	Error during <code>fork()</code> and <code>exec()</code> of user Job on remote machine.
	-41 to -44	UNIX piping between processes failed.

**Table 3: Exit Status Error Codes**

Error Name	Exit Status Value	Description
	-45	Failed to set the process group ID before starting the user Job.
	-46	Cannot fork and run user Job on remote machine at this time.
	-47	Failed to set the process group ID after starting the user Job.
	-48	Failed to change directory (pwd/NBWD) before executing user Job.
	-49	Error during <code>fork()</code> and <code>exec()</code> of user Job on remote machine.
WRITEFAIL	-50	Failed to write to user log file. Disk quota exceeded. ¹
	-51 and below	User Job killed by a signal. To calculate the signal number, subtract 51 from the absolute value of the exit status. (For example, -66 means the Job was killed by signal 15 (66 - 51), SIGTERM.)
CANT_EXEC	-153	Netbatch cannot run the executable.
X_CREATE_INTERACTIVE_IO	-200	Failed to create interactive I/O socket on the WSM.
X_CREATE_INTERACTIVE_ERR	-201	Failed to create interactive err socket on the WSM.
X_NO_INTERACTIVE_PORT	-202	No port is available for creating an interactive socket.
X_LISTEN_INTERACTIVE_IO	-203	Failed to listen on I/O port.
X_LISTEN_INTERACTIVE_ERR	-204	Failed to listen on err port.
X_INTERACTIVE_CONNECT_TIMEOUT	-205	An interactive Job was started on a Workstation but timed out while trying to connect to the <code>nbq</code> command client. (This can happen if the <code>nbq</code> command client is suspended after the Job is submitted, or if there is a firewall between the WSM and the client.)
	-210	Netbatch killed the Job because it had no record of its existence. (This can happen, for example, if persistency for both the WSM and the Pool Master is deleted.) Netbatch only does this to non-interactive Jobs.
	-211	A user-specified wash group is not available for the user on the Workstation.
	-300	Dispatch failed
	-301	Broken dependency

**Table 3: Exit Status Error Codes**

Error Name	Exit Status Value	Description
	-303	The pre-execution script failed.
INVALID_JOB_DATA	-304	An important value (such as CWD or username) is missing.
FORCE_KILL	-305	The Job was removed from the Pool Master without considering Workstation status (for example, the Workstation no longer exists, but on the Pool Master the Job appears to be running). This is the exit status that is assigned when a Job is removed using <code>nbjob remove --force</code> .
	-306	Failure of initial ("in") file copy operation.
	-307	Failure of "out" file copy operation.
	-308	The Job was removed because it was not synchronized between the WSM and the Pool Master.
	-309	A problem occurred while starting the Job process. The Job leader was started, but the Job process could not be started (for at least 12 hours).
	-310	Job failed because Feeder delegate was not available.
	-311	Could not read Job from persistency.
	-312	(For a Job submitted to a Virtual Pool) User does not have an account in the physical Pool that the Job was sent to.
	-313	(For a Job submitted to a Virtual Pool) User does not have permissions in the physical Pool that the Job was sent to.
	-314	(For a Job submitted through a Feeder) The Job did not run because the task that it belongs to was skipped.
	-315	(For a Job submitted through a Feeder) Task setup failed.
	-316	(For a Job submitted through a Feeder) The feeder could not get the Job's real exit status from the Pool. (This can happen if the feeder is down for an hour or more after the Job ends.)
	-321	The checkpoint operation failed.
	-322	The Job was removed because the Workstation it was running on was locked.
	-323	The Job was resubmitted because the Workstation it was running on was locked.
	-3001	The Job has exceeded its hung limit (i.e., it has not used any CPU time in the specified time limit).

**Table 3: Exit Status Error Codes**

Error Name	Exit Status Value	Description
	-3002	The Job has exceeded its execution limit (i.e., it has run for longer than the specified time limit).
	-3004	The Job was killed because the Job was not synchronized between the Workstation and the Pool Master.
	-3005	The Job exceeded its defined Job constraints.
	-3006	The Job was killed because the interactive client is disconnected.
	-3007	The (parallel slave) Job was killed because it could not be dispatched to a Workstation (WSM down or some internal error).
	-3008	The Job was selected to preempt another Job, but did not do so within the timeout period (five minutes), and so was returned to the wait state queue. This exit status is written to the matbatch log file only (and not to the Job log file or Netbatch Tracker, as the Job did not start running).
	-3009	The Job was killed by the Job Failure Detection System.
	-3010	The Job was killed by the WSM auto-tuning service.
	-3011	The Job was resubmitted by the WSM auto-tuning service.
	-3012	The Job was removed because it exceeded a <code>Wvmem</code> limit.
	-3013	The Job was resubmitted because it exceeded a <code>Wvmem</code> limit.
	-3014	The Job was resubmitted by another WSM because of preemption.
	-3015	The Job was killed by another WSM because of preemption.
	-3016	The Job was killed by a policy.
KILL_ON_TASK_CANCEL	-3017	The Job was killed because the task that the Job belongs to was cancelled.
RESUBMIT_ON_THRESHOLD_REACHED	-3019	The Job was resubmitted because it was waiting in a physical Pool for too long (that is, longer than the period specified by the Virtual Pool's <code>ResubmitThreshold</code> directive).
	-3020	(For a Job submitted through a Feeder) The Job (a delegate Job) was killed because there was a problem with it.
	-3021	The Job was killed because it exceeded the parallel Job constraints.

**Table 3: Exit Status Error Codes**

Error Name	Exit Status Value	Description
	-3022	The parallel slave Job was killed because the master Job exited.
	-3023	The Job was killed by the feeder.
	-3024	The delegate was waiting but has no Jobs to run.
	-3025	The Job was killed during nbfeeder kill.
	-3026	The parallel Job was resubmitted because a slave Job was resubmitted.
	-3027	The Job was killed because the delegate Job was killed.
	-3028	The parallel slave Job was killed because the master Job does not exist,
	-3029	The Job was preempted by an interactive process.
	-3030	The Job was denied by Job profile on resubmit.
	-3031	Unknown
	-3032	Internal non-reportable
	-4010	Interactive (ION) session could not be dispatched.
	-4011	Interactive (ION) session was removed by the nbsession remove command.
	-4012	Scheduler timeout while trying to schedule an interactive (ION) session.
	-5000	The Job was removed by NBCalendar on session expiration.

¹⁾ To pinpoint the source of the problem, check the relevant entry in the /tmp/netbatch_internal_error.log file on the Workstation.



Appendix E: Environment Variables

Netbatch uses a number of environment variables, which fall into three groups:

- Those set by the user (or for the user by the Netbatch Administrator) to specify defaults
- Those used to ensure that Jobs that will/may be dispatched to a Windows Workstation run correctly
- Those set by Netbatch when a Job is run or after it is run. (The latter are available during the post-execution stage.)

These are described in [Appendix E.1](#), [Appendix E.2](#), and [Appendix E.3](#), below, and are summarized in [Table 1](#), below.

Note

An `nbjob run` command-line option usually overrides an environment variable that has been set to specify the same parameter.

In most cases (but not all), the environment variable overrides any default values that set for the parameter.

See [Section B: Precedence of Configuration Directives, Environment Variables, Job Profiles, and Command-Line Switches](#).

Table 1: Environment Variables

Name	Set by	Available in post-exec?	Description
<code>__NB_CLASS</code>	Netbatch	Yes	Contains the Job's class requirement expression (in the Job's environment).
<code>__NB_CLASSRESERVATION</code>	Netbatch	Yes	Contains the Job's class reservation requirement expression (in the Job's environment).
<code>__NB_CMD_LINE</code>	Netbatch	Yes	Contains the Job's full command line.
<code>__NB_DELEGATE_ID</code>	Netbatch	No	When running Jobs through a Feeder and using delegate Jobs, this contains the delegate Job's Job ID (in the Pool).
<code>__NB_FEEDER_HOST</code>	Feeder	No	The Feeder sets this (in the Job's environment) to the host that the Feeder is running on.
<code>__NB_FEEDER_PORT</code>	Feeder	No	The Feeder sets this (in the Job's environment) to the port that the Feeder listens on.
<code>__NB_FEEDER_TASK</code>	Feeder	No	The Feeder sets this (in the Job's environment) to the name of the task that the Job belongs to.

**Table 1: Environment Variables**

Name	Set by	Available in post-exec?	Description
<code>__NB_FINISH_TIME</code>	Netbatch	Yes	The time at which the Job finished executing
<code>__NB_JOBID</code>	Netbatch	Yes	Netbatch sets this (in the Job's environment) to: <code><pool name>. <jobID></code> When running Jobs through a Feeder, when using delegates: <ul style="list-style-type: none">• In the Pool, this is the Job ID of the delegate Job.• In the Feeder, this is the ID of the Job.• In the Pool, <code>__NB_DELEGATE_ID</code> contains the delegate Job's Job ID.
<code>__NB_LOG_FILE</code>	Netbatch	Yes	Contains the location of the Job log file (in the Job's environment). Note that during the pre-exec stage, this only contains a valid value if the <code>--log-file (nbjob run)</code> option was used.
<code>__NB_PARALLEL_POOL_HOST</code>	Netbatch	No	Contains the name of the Pool Master to which the hosts listed in <code>NB_PARALLEL_JOB_HOSTS</code> belong.
<code>__NB_POOL</code>	Netbatch	Yes	Contains the name of the Pool that the Job was submitted to (in the Job's environment).
<code>__NB_QSLOT</code>	Netbatch	Yes	Contains the name of the Qslot that the Job was submitted to (in the Job's environment).
<code>__NB_QUEUE</code>	Netbatch	Yes	Contains the name of the Queue that the Job was submitted to (in the Job's environment).
<code>__NB_RUNNING_ON_DYNAMIC_WSM</code>	Netbatch		Has a value of <code>true</code> if the Job is running on a dynamic WSM (that is, in a virtual machine), or <code>false</code> if it is not.
<code>__NB_RUSAGE</code>	Netbatch	Yes	Contains information about the Job's resource usage.
<code>__NB_SANDBOX_DIR</code>	Netbatch	Yes	Contains the path of the Job's secure directory (which is created by Netbatch and whose permissions are set to 700).
<code>__NB_START_TIME</code>	Netbatch	Yes	The time at which the Job started executing
<code>__NB_SUBMIT_PWD</code>	Netbatch	Yes	Contains the value of the PWD environment variable when the user submitted the Job (that is, the user's working directory when the Job was submitted).
<code>__NB_TIMES_RESTARTED</code>	Netbatch	Yes	The number of times that the Job was restarted
<code>_NBNTNOCMD</code>	User	No	Specifies that path translation is not to be performed for commands. (Only used when submitting Jobs that will/may be executed on a Windows Workstation.)

**Table 1: Environment Variables**

Name	Set by	Available in post-exec?	Description
_NBNTNOCMDARG	User	No	Specifies that path translation is not to be performed for command arguments. (Only used when submitting Jobs that will/may be executed on a Windows Workstation.)
_NBNTNOCWD	User	No	Specifies that path translation is not to be performed for the user's current working directory. (Only used when submitting Jobs that will/may be executed on a Windows Workstation.)
_NBNTNOENV	User	No	Specifies that path translation is not to be performed for environment variables. (Only used when submitting Jobs that will/may be executed on a Windows Workstation.)
_NBNTNOJOB	User	No	Specifies that path translation is not to be performed for the Job log file (as specified with nbjob run's --log-file switch). (Only used when submitting Jobs that will/may be executed on a Windows Workstation.)
NB_EXEC_LIMITS	User	No	(DEPRECATED) Specifies the execution limits to be used for every Job that the user submits.
NB_HUNG_LIMITS	User	No	(DEPRECATED) Specifies the hung limits to be used for every Job that the user submits.
NB_JOB_EXIT_STATUS	Netbatch	Yes	In the post-execution script (specified with nbjob run's --post-exec switch), Netbatch sets the value of this environment variable to the exit status of the Job.
NB_JOB_STARTER	User	No	Specifies a Job starter program that performs certain operations and then runs the Job.
NB_LICENSE_FILE	User	No	Specifies the full path and filename of the license keyfile(s) where the feature(s) that the user's Jobs require can be found.
NB_NOMOUNT	User	No	Specifies drives that should not be mapped. Mapping drives can take a long time, and is usually not required. (Only used when submitting Jobs that will/may be executed on a Windows Workstation.)
NB_ON_JOB_FINISH	User	No	Specifies that if Job ends with a certain exit code or if a specified expression is true for it, the specified action is performed.
NB_PARALLEL_JOB_HOSTS	Netbatch	No	Lists the hosts on which an execution slot has been allocated for executing a Parallel Job.
NB_POOLS	User	No	Specifies the Physical Pool(s) in the Virtual Pool that you want to use or exclude.
NB_PROPERTIES	User		Specifies properties for the Job (like adding --properties to nbjob run).
NB_PRE_EXEC	User	No	Specifies a pre-execution program that is executed immediately before every Job.

**Table 1: Environment Variables**

Name	Set by	Available in post-exec?	Description
NB_POST_EXEC	User	No	Specifies a post-execution program that is executed immediately after every Job. The post-execution program is executed even if the Job is killed or suspended.
NB_PROFILE_DIR	User	No	Specifies the location of the .nbqargs file , which specifies sets of Job options for specific commands (Jobs).
NB_PROFILE_FILE	User	No	Deprecated
NB_SIGKILL_TIMEOUT	User	No	Specifies the time that Netbatch waits before sending the SIGKILL signal to a Job to kill it (after the SIGTERM signal is sent). If this variable's value is greater than the default maximum set by the Netbatch Administrator, then the default maximum value is used.
NB_SIGSTOP_TIMEOUT	User	No	Specifies the time that Netbatch waits before sending the SIGSTOP signal to a Job to suspend it (after the SIGTSTP signal is sent). If this variable's value is greater than the default maximum set by the Netbatch Administrator, then the default maximum value is used.
NB_USER_LOG_FILE	Netbatch	No	Contains the path to the user log file (in the Job's environment).
NB_WASH_ENABLED	User	Yes	Specifies that wash groups (as defined in NB_WASH_GROUPS) should be used for every Job that the user submits. (This is the same as adding the --wash switch to nbjob run.)
NB_WASH_GROUPS	User	Yes	Specifies the NIS groups (up to 15) that should be applied to the Job (as a comma-separated list). (Used when the user belongs to more than 16 such groups.)
NBAUTOREQ	User	No	Specifies that a Job should be resubmitted a specified number of times if it returns a specific exit status code(s). Deprecated —use NB_ON_JOB_FINISH instead.
NBCLASS	User	No	Specifies the class of machine that Jobs need to run on.
NBCONF	User	No	Specifies the directory where the pools and nbgroup files are located.
NBCONFSERVER	Netbatch	No	For Windows Workstations, specifies the hostname of the machine where the configuration service is running. (Windows Workstations use the configuration service to get Pool information at runtime.)
NBFEED_CONF	Administrator	No	Specifies the common Netbatch Feeder configuration file to be used.



Table 1: Environment Variables

Name	Set by	Available in post-exec?	Description
NBLICENSES	User	No	Specifies the license(s) that the user's Jobs require.
NBMAILONERR	User	No	If set, the user receives email notification about Jobs that do not succeed.
NBNTAUTOEXEC	User	No	The value of this variable is prepended to the command. This is similar to using a Job starter, and is provided for backward compatibility.
NBNTDOMAINNAME	User	No	Specifies the login to be used (instead of the primary login). (Only used when submitting Jobs that will/may be executed on a Windows Workstation.)
NBNTNOTTRANSLATE	User	No	Specifies that data path translation is to be skipped for colons or for any specified environment variables. (Only used when submitting Jobs that will/may be executed on a Windows Workstation.)
NBNTPATHTABLE	User	No	Specifies the location of the path mapping file. If not set, the global path mapping file is used. (Only used when submitting Jobs that will/may be executed on a Windows Workstation.)
NBPOOL	User	No	Specifies the Pool that Netbatch uses.
NBPROJ	User	No	Specifies the name of the project that Jobs belong to.
NBQSLT	User	No	Specifies the Qslot that Netbatch sends Jobs to.
NBQUEUE	User	No	Specifies the Queue that Netbatch sends Jobs to.
NBTASK	User	No	Specifies a task name that is assigned to all Jobs (like specifying --task <name> with nbjob run).
NBUMASK	User	No	Sets the permissions for the output file.
NBWD	User (but if not set by user, Netbatch sets it)	No	Specifies one or more backup directories where Netbatch sends Job output if it cannot access the user's working directory when the Job ends. (If NBWD contains multiple directories, and Netbatch cannot write to the first one, it tries the second, and so on, until it finds a directory that it can write to.) If not set by the user, Netbatch sets NBWD at run time to the directory where the nbjob run command was run.
NETBATCH_WORKING_DIR	Administrator	Yes	Specified the Netbatch working directory. The Pool Master uses this directory to store libraries and jar files, debug log files, and persistency. It is defined in pool.properties and replaces NS_DEBUG_LOG, NS_PERSISTENCY, NS_PERSISTENCY_POOL, NS_LOCAL_DIR, and NS_CRED_DIR.



E.1 Environment Variables Set by the User

E.1.1 NB_EXEC_LIMITS

Note

The NB_EXEC_LIMITS environment variable is deprecated. Use Job constraints instead. See --job-constraints in [nbjob run](#).

This specifies the execution limits to be used for every Job that the user submits. It is the same as adding `--exec-limits <execLimits>` to every `nbjob run` command.

Note

Execution limits are rounded up to the nearest minute.

E.1.2 NB_HUNG_LIMITS

Note

The NB_HUNG_LIMITS environment variable is deprecated. Use Job constraints instead. See --job-constraints in [nbjob run](#).

This specifies the hung limits to be used for every Job that the user submits. It is the same as adding `--hung-limits <hungLimits>` to every `nbjob run` command.

Note

Hung limits are rounded up to the nearest minute.

E.1.3 NB_JOB_STARTER

This specifies a Job starter program that performs certain operations and then runs the Job.

E.1.4 NB_LICENSE_FILE

This specifies the location (full path and filename) of the license keyfile(s) where the feature(s) that the user's Jobs require can be found.



If more than one keyfile path is specified, they are delimited by colons (:).

Note

*Users can set **NB_LICENSE_FILE** to define multiple keyfiles (which can be on multiple license servers). Netbatch searches for licenses in the locations specified in **NB_LICENSE_FILE** in the order in which they are specified there.*

E.1.5 **NB_NOMOUNT**

This specifies the drives that should not be mapped on a Windows Workstation when Netbatch logs on to it to run a Job. Mapping drives can take a long time, and in many cases is not necessary.

Its value is either a comma-delimited list of drive letters or * (no drives should be mapped).

E.1.6 **NB_ON_JOB_FINISH**

This specifies that if Job ends with a certain exit code or if a specified expression is true for it, the specified action is performed.

It is typically used:

- To protect Jobs from blackhole machines
- When sending Jobs to a Virtual Pool, to prevent a Job being sent back to a physical Pool on which it failed (for example, because the Job does not have access to the files that it needs in that Pool)

E.1.7 **NB_POOLS**

This specifies the physical Pools to be included or excluded for Jobs that are submitted to Virtual Pools. The syntax is the same as for the **nbjob run --remote-pools** switch.

E.1.8 **NB_POST_EXEC**

This specifies a post-execution program that is executed immediately after every Job.

**Note**

The post-execution program is executed even if the Job is killed or suspended.

E.1.9 NB_PRE_EXEC

This specifies a pre-execution program that is executed immediately before every Job.

E.1.10 NB_PROFILE_DIR

Specifies the location of the `.nbqargs` file (previously called `.NetbatchJobArgsMapping`), which specifies sets of Job options for specific commands (Jobs). If the `NB_PROFILE_DIR` file is not set, Netbatch looks in the user's home directory to find the `.nbqargs` file.

Note

The `NB_PROFILE_FILE` environment variable is deprecated and its use is not encouraged. It is still supported in the current release, but at some point it will cease to be supported.

Note

Originally, there were two mapping files—`.NetbatchJobArgsMapping` (previously called `.NetStarTask`) and `.NetbatchLicenseMapping`.

`.NetbatchJobArgsMapping` has been renamed `.nbqargs`.

`.NetbatchLicenseMapping` is no longer supported.

`.NetStarTask` is deprecated and its use is not encouraged. It is still supported in the current release, but at some point it will cease to be supported.

E.1.11 NB_PROFILE_FILE

The `NB_PROFILE_FILE` environment variable is deprecated and its use is not encouraged. It is still supported in the current release, but at some point it will cease to be supported.

Use `NB_PROFILE_DIR` instead. See [Section E.1.10](#), above.



E.1.12 NB_SIGKILL_TIMEOUT

This specifies the time that Netbatch waits before sending the SIGKILL signal to a Job to kill it (after the SIGTERM signal is sent).

A user might want to do this to allow Jobs more time to perform cleanup tasks (such as closing files) than that allowed by the default set by the Netbatch Administrator.

If this variable's value is greater than the default maximum set by the Netbatch Administrator, then the default maximum value is used.

E.1.13 NB_SIGSTOP_TIMEOUT

This specifies the time that Netbatch waits before sending the SIGSTOP signal to a Job to suspend it (after the SIGTSTP signal is sent). If this variable's value is greater than the default maximum set by the Netbatch Administrator, then the default maximum value is used.

E.1.14 NB_WASH_ENABLED

This specifies that the specified groups should be applied to every Job that the user submits. (This is the same as adding the `--wash` switch to `nbjob run`.)

Note

A Job's owner must belong to all of its wash groups according to the group map.

E.1.15 NB_WASH_GROUPS

This specifies the NIS groups (up to 15) to be applied to the user's Jobs (as a comma-separated list).

Note

A Job's owner must belong to all of its wash groups according to the group map.



E.1.16 NBAUTOREQ

Note

*Deprecated—use **NB_ON_JOB_FINISH** instead.*

This specifies that a Job should be resubmitted a specified number of times if it returns a specific exit status code(s). Users should set this variable so that Jobs are not lost to blackholes.

Syntax:

```
setenv NBAUTOREQ "<#_attempts>:<exit_code>
[ ... ]"
```

where:

- **#_attempts** specifies the number of times to resubmit the Job.
- **exit_code** can be any integer (error code) or one or more of the following macros:
 - ◆ **NBErr**—all Netbatch errors except -8, -9, -10, and -11
 - ◆ **Nexit**—all negative Job exit values
 - ◆ **Pexit**—all positive Job exit values
 - ◆ **NZexit**—all non-zero Job exit values

If more than one **exit_code** is specified, they are delimited by spaces.

or:

```
setenv NBAUTOREQ "attempts=<#_attempts>:<expression>"
```

where:

- **<#_attempts>** specifies the number of times to resubmit the Job.
 - **expression** is an arbitrarily complex boolean expression that can consist of any number of terms of the form **exit <relational_operator> <exit_code>**, separated by the logical operators **&&** (AND) and **||** (OR), and together with **!** (NOT). **relational_operator** can be **>**, **<**, **>=**, **<=**, or **==**.
- exit_code** can be any integer (error code) or one or more of the following macros:
- ◆ **NBErr**—all Netbatch errors except -8, -9, -10, and -11



- ◆ **Nexit**—all negative Job exit values
- ◆ **Pexit**—all positive Job exit values
- ◆ **NZexit**—all non-zero Job exit values

E.1.17 NBCLASS

This specifies the class of machine that Jobs need to run on. It is the same as adding `--class <ClassName>` to every `nbjob run` command.

E.1.18 NBCONF

This specifies the directory where the `pools` and `nbgroup` files are located. It overrides the default location `/usr/local/lib/Netbatch/`.

E.1.19 NBFEED_CONF

The Netbatch Administrator sets this for the user to specify a common configuration file that will be used for all Netbatch Feeders that the user starts.

E.1.20 NBLICENSES

This specifies the license(s) that the user's Jobs require. It is the same as adding `--license <licenses>` to every `nbjob run` command. `<licenses>` is a valid boolean expression

Note

*Setting **NBLICENSES** does not have any effect unless **NB_LICENSE_FILE** is correctly set.*

Note

*Setting **NBLICENSES** causes the same license requirements to be specified for every Job.*

Using Job Profiles is a much more flexible way of specifying license requirements, as it allows the user to specify different requirements for each Job type. This is described in detail in the Netbatch User Guide.

E.1.21 NBMAILONERR

Users can set **NBMAILONERR** to receive email notification about Jobs that do not succeed.



NBMAILONERR is very useful if a user is running thousands of regression Jobs and is interested in knowing only failed cases. Users should set this environment variable before queuing any Job.

E.1.22 NBPOOL

This specifies the Pool that Netbatch uses. It is the same as adding `--target <pool_name>` to every Netbatch command.

E.1.23 NBPROJ

This specifies the name of the project that Jobs belong to.

E.1.24 NBQSLOT

This specifies the Qslot that Netbatch sends Jobs to. It is the same as adding `--qslot <qslot_name>` to every Netbatch command where it applies, except `nbqrm`.

E.1.25 NBQUEUE

This specifies the Queue that Netbatch sends Jobs to. It is the same as adding `--queue <QueueName>` to every Netbatch command where it applies.

E.1.26 NBTASK

This specifies a task name that is assigned to all Jobs. It is the same as adding `--task <name>` to every Job.

E.1.27 NBUMASK

This sets the permissions for the output file. Users can set it so that other members of their groups can overwrite their output files.

If **NBUMASK** is not set, the default mask is 022 (`-rwxr-xr-x`).



E.1.28 NBWD

Specifies one or more backup directories where Netbatch sends Job output if it cannot access the user's working directory when the Job ends. (If **NBWD** contains multiple directories, and Netbatch cannot write to the first one, it tries the second, and so on, until it finds a directory that it can write to.)

Note

*If the user has not set **NBWD**, Netbatch automatically sets it (when a Job is run, in the Job's own environment) to the directory where the **nbjob run** command was run.*

Note

*On Windows, **NBWD** must contain a single Windows path, and nothing else. Multiple values are not supported.*

E.2 Environment Variables for Netbatch on Windows

The following environment variables are used to enable Netbatch Jobs to run correctly on Windows Workstations:

E.2.1 NBNTAUTOEXEC

Netbatch prepends the value of this variable to the Job command. It is similar to using a Job starter (the recommended approach) and is provided for backward compatibility.

E.2.2 NBNTDOMAINNAME

Specifies that Netbatch should use a login that the user entered when running **nbauth** other than the one specified as the primary login.

E.2.3 NBNTNOTTRANSLATE

NBNTNOTTRANSLATE Netbatch translates the user's environment from UNIX format to Windows format (i.e., it changes the path separator from : to ;, and slashes (/) to backslashes (\)).

The user can set **NBNTNOTTRANSLATE** to skip data path translation for colons or for any specified environment variables.



E.2.4 **NBNTPATHTABLE**

NBNTPATHTABLEIf the user is using a user-specific (local) path mapping file or a local copy of the global path mapping file, **NBNTPATHTABLE** specifies the location of the file. This can be a UNC or Samba path.

If **NBNTPATHTABLE** is not set, the global path mapping file is used.

E.2.5 **_NBNTNOCMD**

_NBNTNOCMD**_NBNTNOCMD**If **_NBNTNOCMD** is set to 1, path translation is not performed for commands.

E.2.6 **_NBNTNOCMDARG**

_NBNTNOCMDARG**_NBNTNOCMDARG**If **_NBNTNOCMDARG** is set to 1, path translation is not performed for command arguments.

E.2.7 **_NBNTNOCWD**

_NBNTNOCWD**_NBNTNOCWD**If **_NBNTNOCWD** is set to 1, path translation is not performed for the user's current working directory.

E.2.8 **_NBNTNOENV**

_NBNTNOENV**_NBNTNOENV**If **_NBNTNOENV** is set to 1, path translation is not performed for environment variables.

E.2.9 **_NBNTNOJOB**

_NBNTNOJOB**_NBNTNOJOB**If **_NBNTNOJOB** is set to 1, path translation is not performed for the Job output file (as specified with **nbjob run**'s **--log-file** option).



E.3 Environment Variables Set by Netbatch when a Job Is Run

E.3.1 **__NB_CLASS**

Netbatch sets this (in the Job's environment) to the Job's class requirement expression.

E.3.2 **__NB_CLASSRESERVATION**

Netbatch sets this (in the Job's environment) to the Job's class reservation requirement expression.

E.3.3 **__NB_CMD_LINE**

Netbatch sets this to the Job's full command line.

E.3.4 **__NB_DELEGATE_ID**

When running Jobs through a Feeder and using delegate Jobs, this contains the delegate Job's Job ID (in the Pool).

E.3.5 **__NB_FEEDER_HOST**

When a Feeder dispatches a Job to a Pool, it sets this (in the Job's environment) to the host that the Feeder is running on.

E.3.6 **__NB_FEEDER_PORT**

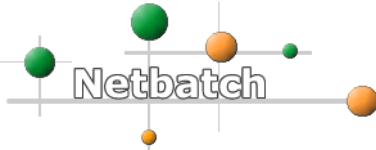
When a Feeder dispatches a Job to a Pool, it sets this (in the Job's environment) to the port that the Feeder listens on.

E.3.7 **__NB_FEEDER_TASK**

When a Feeder dispatches a Job to a Pool, it sets this (in the Job's environment) to the name of the task that the Job belongs to.

E.3.8 **__NB_FINISH_TIME**

Netbatch sets this to the time at which the Job finished executing.



E.3.9 __NB_JOBID

__NB_JOBID When Netbatch or a Feeder dispatches a Job for execution, it sets **__NB_JOBID** (in the Job's environment) to:

```
<pool name>.<jobID>
```

where:

- **pool name** is the name of the Pool that the Job was submitted to.
- **jobID** is the Job's ID.

When running Jobs through a Feeder, when using delegates:

- In the Pool, this is the Job ID of the delegate Job.
- In the Feeder, this is the ID of the Job.
- In the Pool, **__NB_DELEGATE_ID** contains the delegate Job's Job ID.

E.3.10 __NB_LOG_FILE

Netbatch sets this (in the Job's environment) to the path of the Job log file.

Note

Before the Job is dispatched for execution, this only contains a valid value if the --log-file (nbjob run) option was used.

E.3.11 __NB_PARALLEL_POOL_HOST

__NB_PARALLEL_POOL_HOST contains the name of the Pool Master to which the hosts listed in **NB_PARALLEL_JOB_HOSTS** belong.

The **nbexec** command sends a Slave Job (part of a Parallel Job) to the Pool Master specified in **__NB_PARALLEL_POOL_HOST** for validation before it is executed on the specified host.

Caution

*Do not set **__NB_PARALLEL_POOL_HOST** yourself.
Netbatch sets it as part of the scheduling process to ensure system consistency.*



E.3.12 __NB_POOL

Netbatch sets this (in the Job's environment) to the name of the Pool that the Job was submitted to.

E.3.13 __NB_QSLOT

Netbatch sets this (in the Job's environment) to the name of the Qslot that the Job was submitted to.

E.3.14 __NB_QUEUE

Netbatch sets this (in the Job's environment) to the name of the Queue that the Job was submitted to.

E.3.15 __NB_RUNNING_ON_DYNAMIC_WSM

This has a value of `true` if the Job is running on a dynamic WSM (that is, in a virtual machine), or `false` if it is not.

E.3.16 __NB_RUSAGE

This contains information about the Job's resource usage.

E.3.17 __NB_START_TIME

Netbatch sets this to the time at which the Job started executing.

E.3.18 __NB_SUBMIT_PWD

__NB_SUBMIT_PWD contains the value of the PWD environment variable when the user submitted the Job (that is, the user's working directory when the Job was submitted).

This is useful for ClearCase Jobs, where the directory does not exist until after the view is set. (See [Section 5.3.18, Setting Up a Queue/Qslot for ClearCase Users](#).)

E.3.19 __NB_TIMES_RESTARTED

Netbatch sets this to the number of times that the Job was restarted, for whatever reason.



E.3.20 NBWD

If the user has not set **NBWD**, Netbatch automatically sets it (when a Job is run, in the Job's own environment) to the directory where the `nbjob run` command was run.

E.3.21 NBCONFSERVER

For Windows Workstations, **NBCONFSERVER** specifies the hostname of the machine where the configuration service is running. (Windows Workstations use the configuration service to get Pool information at runtime.)

E.3.22 NB_JOB_EXIT_STATUS

In the post-execution script (specified with `nbjob run`'s `--post-exec` switch), Netbatch sets the value of this environment variable to the exit status of the Job.

E.3.23 NB_PARALLEL_JOB_HOSTS

NB_PARALLEL_JOB_HOSTS lists the hosts on which an execution slot has been allocated for executing a Parallel Job. The host names are used with `nbexec` to dispatch the Slave Jobs.

Note

If two or more execution slots are used on the same Workstation, its name appears the corresponding number of times in the list.

*This list does **not** include the host that runs the Master Job.*

The hosts are delimited by spaces.



Appendix F: Smart Classes Reference

This appendix lists the features of the language that is used to build the requirements of the Smart Classes.

F.1 Syntax

F.1.1 Literals

- **Boolean:** true, false
- **Integer:** e.g., 7, -33, 0666, 0x1a
- **Real:** e.g., 3.14, -2.134, 1e+5
- **String:** e.g., "aaa", "\t"

F.1.2 Operators

- **Arithmetic:** +, -, *, /, %, unary +, unary -
- **Logical:** &&, ||, !
- **Bitwise:** &, |, ~, ^, <<, >>, >>>
- **Comparison:** <, <=, >, >=, !=, ==
- **Conditional:** ?, :
- **Non-Strict operators:** "is" , "isnt"

F.1.3 Functions

- "`startswith(<field_name>,'<string>')`"
`startswith()` matches fields whose value starts with the specified string.
- "`glob(<field_name>,'<expression>')`"
`glob()` matches fields that match `expression`. `expression` can contain:
 - Any character
 - . - matches any character
 - \w - matches any letter (upper- or lower-case)
 - \d - matches any digit
 - \s - matches any space character
 - * - zero or more of the previous character



- ? - exactly one of the previous character

F.2 Attributes

You can use any `nbstatus workstations` field in a class expression. These are listed in [Appendix A.3.32, nbstatus workstations](#).

F.3 Examples

For Example

To specify that a Job must run on a Workstation running Linux that has at least 500MB of free real memory, submit it as follows:

```
nbjob run --target <pool> [other options]
--class "OSName=='Linux'&&fRM>=1000" <command>
```

For Example

To specify that a Job must run on a Workstation that has at least 2GB of free space on /tmp or that has at least 10GB of total free disk space, submit it as follows:

```
nbjob run --target <pool> [other options]
--class "fDS('/tmp')>=2000|tFDS>=10000"
<command>
```

Note

The value of the `fds('<path>')` attribute is zero unless Netbatch has been configured to monitor the specified directory.



Appendix G: Regex Replace Functions

The regex replace functions can be used to make changes to an expression. They can be used in any boolean expression, including those in:

- `--on-job-finish` (`njob run` and `nbq`)—in a `Modify` operation
- A `JobProfile`—to make changes to a switch's arguments
- The filter expression in `nbstatus` commands

The functions are:

- `$(regexReplace(<input>, <n-group-pattern>, <replacement>[, ...]))`
- `$(iRegexReplace(<input>, <n-group-pattern>, <replacement>[, ...]))`

These are identical except that in `iRegexReplace()`, `pattern` is not case-sensitive.

In both functions:

- `input` is the item to which the match is applied (usually the name of a Job field).
- `n-group-pattern` is the regular expression that specifies what is matched. Within each match, if part of the pattern is enclosed in parentheses, the value of the matching part of input is assigned to a variable (`VAR1`). The value of each additional matching part is assigned to an additional variable (`VAR2`, `VAR3`, etc.).

For example:

```
regexReplace("2,4", "(\d+)", "toInt(number(VAR1)*2)")
```

returns:

```
4,8
```

Here there are two matches. In each, there is a single expression in parentheses, the value of which is assigned to `VAR1`.

Another example:

```
regexReplace("2,3;4:5",
"(\d+)[,:;](\d+)", "number(VAR1)*2",
"toInt(number(VAR1)*2)")
```

returns:

```
4.0,6;8.0:10
```



Here, there are two matches (2, 3 and 4:5). In each, there are two expressions in parentheses, which are assigned to **VAR1** and **VAR2** respectively. (In the first match, **VAR1=2** and **VAR2=3**. In the second, **VAR1=4** and **VAR2=5**.) The final two expressions specify the changes that are made to these variables.

For a complete list of the special characters and constructs that can be used here, see [this page](#).

- **replacement** is what a matching part is replaced by. It can include functions to manipulate variables assigned by pattern (see above), for example, **number()** to convert a string to a number and **toInt()** to convert a number to an integer.

Escaping Quotes

If you need to include three levels of nested quotes, use escaped double-quotes for the most deeply-nested ones. The way you do this depends on the shell you are using:

- For bash, use `\"`.
- For tcsh, use `"\""`.

In a [JobProfile](#), use the bash form.

Examples

To resubmit a Job after it ends with double the memory requirement in its **--class** switch, type (in bash):

```
nbjob run --class "2G && SLES10 && 1C" --on-job-finish
"ExitStatus==0:Modify('class=regexreplace(class,\"(.*"
)(\d+)(G.*)\",\"VAR1\",\"toInt(number(VAR2"
)*2)\",\"VAR3\")'):Requeue(1)" sleep 10
```

In tcsh, the same command looks like this:

```
nbjob run --class "2G && SLES10 && 1C" --on-job-finish
"ExitStatus==0:Modify('class=regexreplace(class,""""
(.*)(\d+)(G.*)""","""VAR1""","""toInt(number(VAR2"
)*2)""","""VAR3""")'):Requeue(1)" sleep 10
```



To modify a Job in a **JobProfile** such that a class reservation is added which includes the (class) memory requirement from its **--class** switch both as a class requirement and an **frm** requirement (which is half of the class memory requirement), plus a cores requirement, add a line like this:

```
force --class-reservation
"memory=$(regexReplace(class, '(.*)_((\d+)(G.*))',
'"',
'toInt(number(VAR2))','\"'),cores=1,frm=$(regexRep
lace(class, '(.*)_((\d+)(G.*))', '\"',
'toInt(number(VAR2)*1024*0.5)','\"'))"
```



CONFIDENTIAL

Netbatch
v8.2.2

Revision 4

Regex Replace Functions



Appendix H: Functions

The functions listed here can be used in **nbstatus** commands as follows:

- In a filter (specification) expression, to specify which records are displayed
- In the argument of the **--fields** switch to add a new field (e.g., **--fields "GB (RM)"**)

Some functions (as noted in their descriptions, below) can be used in the argument of the **--fields** switch (**func** in **--fields <field>[:<func>][:<size>]**) to perform manipulations on the values of specific fields.

Note

*The **regexReplace** and **iRegexReplace** functions are explained in a separate appendix—[Appendix G](#):*

Note

*You can see which functions are available for each **nbstatus** command (and also in a **preemption** block or an **nbjob run** Job constraints expression) by using **nbstatus functions**. Adding the **--context** switch shows additional functions for the following contexts:*

- **nbsession status**
- **nbstatus qslots**
- **nbstatus license-allocation**
- **nbstatus workstations**
- **nbstatus jobs**
- **preemption block**
- **jobConstraints**

*A context containing a space must be enclosed in quotes. For example, **--context "nbstatus jobs"**.*

Note

*To see the data type of any **nbstatus** field, use **--format typed-csv**. This displays the following for each field:*



- *Type (string, long, map, etc.)*
- *Description*
- *Whether it is displayed by default*
- *Display field name*
- *Display width*

By default, this only shows the default fields. Use --fields all to display all fields.

Note

The Netbatch-specific functions that appear in the table below cannot be used with --history. This is because --history uses a different backend data structure.

Table 2: Functions and Their Descriptions

Function Name/Usage	Scope	Description
!	Any nbstatus command	NOT—returns true if the argument is false, and vice-versa.
!=	Any nbstatus command	Not equal to—compares two arguments and returns true if they are not equal.
!~	Any nbstatus command	Inverse pattern-matching operator—returns true if what is on the right does not match the pattern on the left. Note: The syntax is the same as for glob(), below. The difference is that !~ can be used with --history, while glob() cannot.
&&	Any nbstatus command	AND—returns true if the expressions on either side are both true. It can be used in --fields to combine multiple boolean fields into a single field (e.g., --fields "Delegate&&EnvSubmitted").
*	Any nbstatus command	Multiplication operator It can be used in --fields to multiply one numeric field by another, combining them into a single field (e.g., --fields "utime*CurrentCost").
+	Any nbstatus command	Addition operator It can be used in --fields to add one numeric field to another, combining them into a single field (e.g., --fields "utime+stime").

**Table 2: Functions and Their Descriptions**

Function Name/Usage	Scope	Description
-	Any nbstatus command	Subtraction operator It can be used in --fields to subtract one numeric field from another, combining them into a single field (e.g., --fields "wtime -utime").
/	Any nbstatus command	Division operator It can be used in --fields to divide one numeric field by another, combining them into a single field (e.g., --fields "wtime/utime").
<	Any nbstatus command	Less than operator
<=	Any nbstatus command	Less than or equal to operator
=	Any nbstatus command	Assignment operator—assigns the value on the right to the variable on the left.
==	Any nbstatus command	Equals operator
=~	Any nbstatus command	Pattern-matching operator—returns true if what is on the right matches the pattern on the left. Note: The syntax is the same as for glob(), below. The difference is that =~ can be used with --history, while glob() cannot. Note that with --history, there are a few slight differences in the syntax, due to the different back-end data structure. For example, with --history, you must use * as a wildcard instead of .* without --history.
>	Any nbstatus command	Greater than operator
>=	Any nbstatus command	Greater than or equal to operator
ActualClassReservation('attribute')	nbstatus jobs, nbsession status	Returns the actual amount of the specified attribute that is reserved for the Job/session. For example: ActualClassReservation('cores')
AvgOverTime('<time_interval>', '{RM VM RM+VM}')	Job constraints	Returns the average of the sampled values of the specified parameter in time_interval, where time_interval is of the form <number>[{s m h}] (seconds, minutes, or hours). The default is minutes (so 60 means 60 minutes).

**Table 2: Functions and Their Descriptions**

Function Name/Usage	Scope	Description
CamaCacheMatch()	WSRank expression	Returns the percentage match of the cached datasets on the specific Workstation compared to the datasets required by the Job.
CanJobPreempt(<full_job_ID>)	nbstatus jobs, nbsession status	Returns true if the Job belongs to a Qslot that can currently preempt the Qslot of the specified Job.
CanQslotPreempt('<qslot>')	nbstatus qslots	Returns true if the current Qslot's Jobs can preempt those in the specified Qslot (the argument).
CanRent()	nbstatus jobs	Returns true if the Job is able to borrow resources that are reserved for a bigger Job, but that are currently unused. (This is true if the Job was submitted with the --reservation-condition switch and if that switch's specified expression is true.)
Concurrency()	Workstation policy	Returns the number of Jobs that can run on the Workstation.
DeltaOverTime()	Job constraints	Returns the difference between the maximum and minimum values of the specified parameter in time_interval, where time_interval is of the form <number>[{s m h}] (seconds, minutes, or hours). The default is minutes (so 60 means 60 minutes).
DynamicEquals(<object>, <object>)	nbstatus workstations	Compares the current values of the specified items instead of the cached values. (Netbatch caches dynamic class values for Workstations.)
GB(<attribute>)	Any nbstatus command	Converts the specified amount to gigabytes.
GetFieldFromModificationHistory('<queue_regex>', '<field>', '<default>')	nbstatus jobs, nbsession status	Returns the first entry from the ModificationHistory field for which the Queue matches queue_regex and the field matches field. If no match is found, default is returned.
MillisToSeconds('<time_field>')	Any nbstatus command	Converts the argument from milliseconds to seconds.
PB(<attribute>)	Any nbstatus command	Converts the specified amount to petabytes.
TB(<attribute>)	Any nbstatus command	Converts the specified amount to terabytes.
TimeInDays(<attribute>)	Any nbstatus command	Returns the specified time value in days.
TimeInHours(<attribute>)	Any nbstatus command	Returns the specified time value in hours.
TimeInMinutes(<attribute>)	Any nbstatus command	Returns the specified time value in minutes.

**Table 2: Functions and Their Descriptions**

Function Name/Usage	Scope	Description
TimeInSeconds(<attribute>)	Any nbstatus command	Returns the specified time value in seconds.
TimeInState(<state>)	nbstatus jobs, nbsession status, Job constraints	Returns the time the Job is in its current state (in milliseconds). Valid states are: send, resched, approving, susp, resub, del, wait remote, wait, run, and disc.
ToDate(<time>)	Any nbstatus command	Returns the specified Unix time as a date/time.
WSRank(['<attribute>'])	nbstatus workstations	Returns the Workstation's rank (availability), according to the specified availability calculation scheme. If you call it without any parameters, it returns the Workstation's rank for all configured availability calculation schemes.
abs(<attribute>)	Any nbstatus command	Returns the absolute value of the argument (that is, if it is negative, this function converts it to positive and returns it, or if it is positive, it returns it unchanged).
acceptingMachinesAfterHost(<hostname>[...])	nbstatus workstations	This function is specific to Network Topology Awareness (NTA). It returns a score for the specified machine according to its proximity to other available machines. The machine with the most available adjacent machines gets the highest score. hostname can be an actual hostname (in which case it must be enclosed in quotes) or a field containing a hostname. Returns
acceptingMachinesBeforeHost(<hostname>[...])	nbstatus workstations	Returns
actualClassReservation('<key>')	nbstatus jobs	Returns the value of the specified individual class reservation (the value corresponding to an individual key from the the actualClassReservation field's multiple key/value pairs). For example: actualClassReservation('cores')
atan(<attribute>)	Any nbstatus command	Returns the arc tangent of the argument.
avg(<double>[,...])	Any nbstatus command	Returns the average of the specified values. If used in --fields in the <field>:<function> form, the results are grouped and the field to which avg was applied shows the average of the values of that field for all the records.

**Table 2: Functions and Their Descriptions**

Function Name/Usage	Scope	Description
avgcollection('<collection>')	Any nbstatus command	Returns the average of the values that make up the specified collection.
avgovertime('<time_interval>', '{RM VM RM+VM}')	Job constraints expression	Returns the average over time of the usage of the specified memory type. <code>time_interval</code> is of the form <code><number>[{s m h}]</code> (seconds, minutes, or hours). The default is minutes (so 60 means 60 minutes).
belongsToResourceGroup('<resource_group>')	nbstatus workstations	Returns true if the Workstation belongs to the specified Resource Group.
camawsrankinput('<dataset_list>')	nbstatus workstations	Used internally by <code>CamaCacheMatch()</code> (see above). Returns the number of matched datasets on the Workstation from the specified list.
ceiling('<attribute>')	Any nbstatus command	Rounds the argument up to the nearest integer.
concat('<arg>[,...]')	Any nbstatus command	Returns a single string containing all the arguments concatenated together. If used in --fields in the <code><field>:<function></code> form, the results are grouped and the field to which <code>concat</code> was applied shows the concatenated string containing the values for all the records.
conjunction('<collection>, <collection>')	Any nbstatus command	Compares two collection fields and only returns the items that appear in both. This only works for fields that have multiple values (e.g., <code>WaitingVirtualResources</code> in nbstatus jobs).
contains('<object>, <collection>')	Any nbstatus command	Returns true if the specified object exists in the collection (where <code>collection</code> can be an nbstatus field).
count('<arg>[,...]')	Any nbstatus command	Returns the number of items in the specified list of items. If used in --fields in the <code><field>:<function></code> form, the results are grouped and the field to which <code>count</code> was applied shows the number of records.
countMatches('<string>', '<regex>')	Any nbstatus command	Returns the number of times the specified regular expression matches the specified string. (<code>string</code> can be replaced, for example, by the name of a field.)
count_notempty('<object>[,...]')	Any nbstatus command	Returns 1 if the value for the specified field is the field's default value, or 0 if it is not. If you specify multiple fields, this returns a single number that is the sum of the return values for all the specified fields.

**Table 2: Functions and Their Descriptions**

Function Name/Usage	Scope	Description
countjobs(<expression>)	nbstatus workstations, WSRank expression, nbjob run --class expression	Returns the number of Jobs that match the specified expression (where expression is an arbitrarily complex boolean expression that can include any nbstatus jobs field).
countrunning(<object>)	Preemption block	Returns the number of running Jobs that were dispatched from the Qslot.
countsheets(<expression>)	nbstatus workstations, WSRank expression, nbjob run --class expression	Returns the number of execution slots on the Workstation that match expression (where expression is an arbitrarily complex boolean expression that can include any nbstatus jobs field).
deltaovertime('<time_interval>', '{utime stime utime+stime}')	Job constraints expression	Returns the delta over time of the specified usage time argument. time_interval is of the form <number>[{s m h}] (seconds, minutes, or hours). The default is minutes (so 60 means 60 minutes).
equals(<field_name>, '<value>')	Any nbstatus command	Case-sensitive equals. Returns true if the value of the specified field is the same as value. Use this instead of == if you need value to be case-sensitive. (== is not case-sensitive.)
fds('<path>')	nbstatus workstations, workstation policy	Returns the amount of free disk space for the specified path, in MB. Note that this only works for paths that the Workstation is configured to monitor. For paths that are not monitored, it returns zero.
filterbykey(map, collection)	Any nbstatus command	Returns map with only items that match those in collection, and all others removed.
floor('<attribute>')	Any nbstatus command	Rounds the argument down to the nearest integer.
getAllocation([<level>])	nbstatus jobs, nbsession status	Returns the configured allocation of the Qslot that the Job was dispatched from, or of a Qslot above it. For example, if the Job was dispatched from /mmg/iil_vp/tvpv_tq, specifying a level of: <ul style="list-style-type: none">• 1 gives the allocation for mmg• 2 gives the allocation for iil_vp• 3 gives the allocation for tvpv_tq

**Table 2: Functions and Their Descriptions**

Function Name/Usage	Scope	Description
getMaxCapabilityFromModificationHistory('<capability>',<default_value>)	nbstatus jobs	Returns the highest value of the capability from all of the Job's iterations (from its modification history). To find the default value, use an expression like: Number(MapValue(ActualClassReservation,'memory'))
getqueue('<object>')	Preemption block	Returns the Queue.
getrusage('<string>')	nbstatus jobs, nbsession status	Returns the rusage value for the specified attribute (e.g., wtime, utime, etc.).
glob(<field_name>,<expression>')	Any nbstatus command	Matches items for which the specified field matches the specified regular expression. expression can contain: <ul style="list-style-type: none">• Any character• . - matches any character• \w - matches any letter (upper- or lower-case)• \d - matches any digit• \s - matches any space character• * - zero or more of the previous character• ? - exactly one of the previous character
hasPermissions('<user>')	nbstatus qslots, nbstatus license- allocation	Returns true if the specified user has permissions to submit Jobs to the Qslot (nbstatus qslots) or to use the license in the Qslot (license-allocation).
hasPermissionsOfType('<user>','<type>')	nbstatus qslots, nbstatus license- allocation	Returns true if the specified user has permissions of the specified type. type is one of the following: <ul style="list-style-type: none">• JobSubmit• JobModify• JobRemove• JobAdmin• QueueOpen• QueueClose• QueueActivate• QueueInactivate• QueueAdmin

**Table 2: Functions and Their Descriptions**

Function Name/Usage	Scope	Description
<code>hassubword('string', 'string')</code>	Any nbstatus command	Returns true if the second string appears in the first string's space-delimited list. For example: <code>hassubword('hello there', 'hello')</code> returns true, but: <code>hassubword('hello', 'hell')</code> returns false.
<code>inlist('<search_term>', '<object>[, ...])'</code>	Any nbstatus command	Returns true if <code>search_term</code> matches any of the subsequent arguments.
<code>iregexReplace(<string>, <string>, <string> [, ...])</code>	Any nbstatus command	Uses a regular expression to make changes to an expression (not case-sensitive). See Appendix G:
<code>is('<object>', '<object>')</code>	Any nbstatus command	Returns true if the two objects are the same.
<code>isEmpty('<string>')</code>	Any nbstatus command	Returns true if <code>string</code> is an empty string.
<code>isNull('<object>')</code>	Any nbstatus command	Returns true only if the value of the specified field is null.
<code>isResource('<queue>'[, '<qslot>'])</code>	nbstatus workstations	Returns true if the Workstation is a resource that can be used by the specified Queue/Qslot.
<code>keys('<string>', '<pair_delimiter>', '<key_value_delimiter>')</code>	Any nbstatus command	Converts the specified string into a collection, using the two specified delimiters. The first delimiter is the one that separates the key/value pairs. The second is the one that separates the key from the value.
<code>lcase('<string>')</code>	Any nbstatus command	Returns string, converted to all lower-case. For example: <code>lcase('HELLO')</code> returns hello.
<code>list('<string>'[, '<delimiter>'])</code>	Any nbstatus command	Converts the specified string into a list, by splitting it at the specified delimiter. If no delimiter is specified, a comma (,) is used.
<code>listvalue(<collection>, n)</code>	Any nbstatus command	Returns the nth element in collection.
<code>mapvalue(<map>, '<key>')</code>	Any nbstatus command	Returns the value corresponding to the specified key in map.

**Table 2: Functions and Their Descriptions**

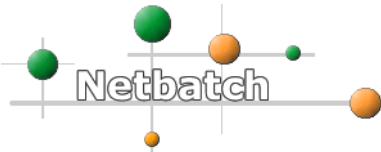
Function Name/Usage	Scope	Description
<code>max(<double>[,...])</code>	Any nbstatus command	Returns the highest of the specified values. If used in --fields in the <field>:<function> form, the results are grouped and the field to which <code>max</code> was applied shows the highest value for that field out of all the records.
<code>member('<field_name>', '<string>')</code>	Any nbstatus command	Matches items where string is one of the values of the specified field. This only works for fields that have multiple values (e.g., <code>WaitingVirtualResources</code> in nbstatus jobs).
<code>min(<double>[,...])</code>	Any nbstatus command	Returns the lowest of the specified values. If used in --fields in the <field>:<function> form, the results are grouped and the field to which <code>min</code> was applied shows the lowest value for that field out of all the records.
<code>now()</code>	Any nbstatus command	Returns the current Unix time.
<code>numallocated('<object>')</code>	Premption block	Returns the number of allocated units for the Qslot.
<code>number(<object>)</code>	Any nbstatus command	Returns the specified object, converted to a number. For example: <code>number(TimeInRunning)</code> returns the time a Job has been running in milliseconds (instead of something like 1h:13m:20s).
<code>parallel('<property>')</code>	nbstatus jobs, nbsession status	Returns the value of the specified parallel property (from the Job's parallel request).
<code>random()</code>	Any nbstatus command	Returns a random number between zero and one.
<code>rankMachineForEmulationNTA(<hostname>)</code>	nbstatus workstations	This function is specific to Network Topology Awareness (NTA). It returns a score for the specified machine according to its proximity to other available machines. The machine with the most available adjacent machines gets the highest score. <code>hostname</code> can be an actual hostname (in which case it must be enclosed in quotes) or a field containing a hostname.
<code>regexReplace(<string>, <string>, <string> [,...])</code>	Any nbstatus command	Uses a regular expression to make changes to an expression (case-sensitive). See Appendix G .
<code>replace(<string>, <string>, <string> [,...])</code>	Any nbstatus command	Similar to <code>regexReplace()</code> , but more simple.

**Table 2: Functions and Their Descriptions**

Function Name/Usage	Scope	Description
<code>reservation('<capability>')</code>	Job constraints expression	Returns the value of the Job's reservation of the specified Workstation capability. (For example, if a Job reserved two of the capability called <code>cores</code> , <code>reservation("cores")</code> returns 2 for that Job.)
<code>round('<attribute>')</code>	Any nbstatus command	Rounds the argument to the nearest integer.
<code>runtimeExcludePools(<string>, <double>)</code>	nbstatus jobs, nbsession status	Returns a list of remote Pools for which the predicted runtime upper bound exceeds the specified number for the specified percentage.
<code>runtimePrediction(<string>[,...])</code>	nbstatus jobs, nbsession status	For the given percentage, returns the calculated runtime prediction range. For a virtual pool, this returns the range for each remote pool.
<code>runtimePredictionUB(<runtime_predicted_UB>, <time>, <list_of_pools>)</code>	nbstatus jobs, nbsession status	Returns a comma-separated list of excluded pools from the supplied list, that is, ones for which <code>runtime_predicted_UB</code> is greater than <code>time</code> .
<code>size(<collection>)</code>	Any nbstatus command	Returns the size (number of items) of the specified collection. This only works for fields that have multiple values (e.g., <code>WaitingVirtualResources</code> in nbstatus jobs).
<code>split(<string>,<string>)</code>	Any nbstatus command	Returns a collection of strings that results from splitting the first argument at the delimiter specified by the second argument. If the second argument contains multiple characters, all are used as delimiters.
<code>sqrt(<double>)</code>	Any nbstatus command	Returns the square root of the argument.
<code>startswith((<field_name>,'<string>'))</code>	Any nbstatus command	Matches items for which the specified field starts with the specified string.
<code>strcat(<string>[,...])</code>	Any nbstatus command	Returns the specified strings concatenated together into a single string.

**Table 2: Functions and Their Descriptions**

Function Name/Usage	Scope	Description
<code>stringmapvalue(<string>, <key_val_delimiter>, <pair_delimiter>, <key>)</code>	Any nbstatus command	<p>Converts the input string into a map using the specified key/value delimiter (that is, the delimiter between the key and the value) and the specified pair delimiter (that is, the delimiter that separated the key/value pairs), and then returns the value for the specified key from the map.</p> <p>For example:</p> <pre>stringmapvalue('a=1:b=2', ':', '=' , 'b')</pre> <p>returns 2.</p>
<code>substring(<string>, <string>[, <n>])</code>	Any nbstatus command	<p>Returns the beginning of the first string up to (but not including) the n+1th occurrence of the second string.</p> <p>If n is not specified (or if it is zero), the beginning of the first string up to the first occurrence of the second string is returned.</p> <p>If there are fewer than n+1 occurrences, the function returns NA.</p> <p>For example:</p> <pre>substring('hello there', 'e', 2)</pre> <p>returns hello ther.</p>
<code>sum(<double>[, . . .])</code>	Any nbstatus command	<p>Returns the sum of the specified values.</p> <p>If used in --fields in the <field>:<function> form, the results are grouped and the field to which sum was applied shows the sum of the values of that field for all the records.</p>
<code>tds('<path>')</code>	nbstatus workstations, workstation policy	<p>Returns the total amount of disk space for the specified path, in MB.</p> <p>Note that this only works for paths that the Workstation is configured to monitor. For paths that are not monitored, it returns zero.</p>
<code>toInt(<double>)</code>	Any nbstatus command	Converts the specified double to an int. The result is rounded down.
<code>tomap(<object>)</code>	Any nbstatus command	Casts the specified object to a map.
<code>toString(<object>)</code>	Any nbstatus command	Takes an object and converts it to a string.

**Table 2: Functions and Their Descriptions**

Function Name/Usage	Scope	Description
<code>uniq(<double>[, . . .])</code>	Any nbstatus command	Returns the number of unique values from a list of values. If used in --fields in the <field>:<function> form, the results are grouped and the field to which uniq was applied shows the number of unique values of that field for all the records.
<code>uniq_list(<object>[, . . .])</code>	Any nbstatus command	Returns a list of the unique values from a list of values. If used in --fields in the <field>:<function> form, the results are grouped and the field to which uniq_list was applied shows a list of the unique values of that field for all the records.
<code>values(<map>)</code>	Any nbstatus command	Takes a map and returns a collection containing just the values (without their keys). For example: <code>values(RemoteData)</code>
<code> </code>	Any nbstatus command	OR—returns true if either of the expressions on either side are true. It can be used in --fields to combine multiple boolean fields into a single field (e.g., --fields "Delegate EnvSubmitted").



CONFIDENTIAL

Netbatch
v8.2.2

Revision 4

Functions

Copyright © 2015 Intel Corporation. Intel Confidential.