

NFT Reference Manual

Table of Contents

Preface	4
Introduction	5
How to Run NFT	6
Basic Execution	6
Basic NFT Features	7
Basic Examples	10
NFT Pathname Syntax	12
NFT Commands Summarized	14
General and Storage-Defaulted Commands	14
Environment-Variable Settings	15
Synchronous and Asynchronous Command Modes	16
Local, Immediate, and Job Commands	17
Local (Client) Commands	17
Server Commands	17
Immediate Commands	17
Job Commands	17
Input and Output Files	19
NFT Output (Log) Files	19
Using LOG and CLOG	19
NFT Logging Techniques and Examples	19
NFT Input Files	21
File Input by Redirection	21
File Input Using SOURCE	21
Logging Jointly with Input Files	22
Job Status and Reporting	23
NFT Job Numbers and Classes	23
Using Job Numbers	23
Job Class Hierarchy	24
Job Reporting with RPT	25
Scope	25
Format and Examples	26
Diagnostic Verbosity	27
NFT Sessions	29
Grouping Jobs by Session	29
One Client, Multiple Sessions	29
Multiple Clients, Multiple Sessions	31
Using NFT in Scripts	32
NFT Command Dictionary	33
Command Syntax Advice	33
ABT (Abort Incomplete Jobs)	34
ASync (Run Jobs in Parallel)	35
BLOCK (Block or Delay Next Command)	36
CD (Change Working Directory)	37

CDUP (Change Working Directory Up)	38
CHGRP (Change Groups)	39
CHKROUTING (Report Routing Availability)	41
CHMOD (Change Permissions)	42
CHOWN (Change Owners)	43
CLOBBER (Enable File Overwriting)	44
CLOG (Close Log File)	45
CLOSE (Restore Remote Host)	45
CLR (Clear Completed Job Reports)	46
CP (Copy/Transfer Files)	48
DELETE (Remove Files)	50
DIR (List Directory Contents, Long)	52
DUALCOPY (Store Dual Copies of Files)	54
ENDGR (End Asynchronous Group)	55
GET (Retrieve Stored Files)	56
GROUP (Begin Asynchronous Group)	58
HELP (Describe NFT Commands)	59
LCD (Change Local Working Directory)	60
LN (Create a Link)	61
LOG (Open Log File)	62
LS (List Directory Contents, Short)	63
MKDIR (Make Directories)	65
NOCLOBBER (Disable File Overwriting)	66
NODUALCOPY (Undo Dual Copy)	66
NOROUTING (Disable Routing)	67
NOTERM (Disable Terminal Output)	68
OPEN (Change Remote Host)	69
PUT (Store Local Files)	71
PWD (Print Working Directory)	73
QUIT (Terminate NFT Client)	74
RENAME (Change File Name)	75
RMDIR (Remove Directories)	76
ROUTING (Use Login Node Jumbo Frames)	77
RPT (Report Job Status)	78
SESSION (Change NFT Sessions)	80
SETCOS (Change Storage Class of Service)	81
SOURCE (Use Command File)	82
STATUS (Report Environment Variables)	83
SYNC (Run Jobs in Series)	84
TERM (Enable Terminal Output)	85
TIME (Report Current Time)	85
VERBOSE (Control State-Change Reports)	86
Disclaimer	88
Keyword Index	89
Alphabetical List of Keywords	91
Date and Revisions	93

Preface

Scope: The NFT Reference Manual describes in detail the syntax, commands, and special features of the NFT (Network File Transport) file-transfer utility. NFT moves files between machines with the help of a dedicated server that provides persistent, passwordless transfers with elaborate job tracking. Extra support for file transfers to or from LC's archival storage system (including jumbo-frame routing where available) is also provided.

For a clear comparison of NFT's features with those of FTP and SCP, and for a concise, task-oriented summary of how to use NFT for ordinary file transfers, see the EZOUTPUT (URL: <https://computing.llnl.gov/LCdocs/ezoutput>) basic file transfer and print guide.

For an overview of archival storage features at LC (open and secure), including access issues, solutions to common storage problems, and NFT's role in using storage, see the EZSTORAGE (URL: <https://computing.llnl.gov/LCdocs/ezstorage>) basic file storage guide.

Availability: NFT runs on LC's open and secure AIX (IBM) and Linux/CHAOS production machines.

Note that besides the open and secure archival storage systems, only those machines that support NFT clients also accept incoming file transfers using NFT. Other hosts (such as FIS) do **not** accept NFT transfers.

Consultant: For help contact the LC customer service and support hotline at 925-422-4531 (open e-mail: lc-hotline@llnl.gov, SCF e-mail: lc-hotline@pop.llnl.gov).

Printing: The print file for this document can be found at:

OCF: <https://computing.llnl.gov/LCdocs/nft/nft.pdf>

SCF: http://www.llnl.gov/LCdocs/nft/nft_scf.pdf

Introduction

NFT (Network File Transport) is a file-transfer utility developed at LC and tailored to local needs. NFT features persistent, passwordless file transfer among worker machines and to the storage system in both the open and secure environments (with automatic routing of storage transfers through fast, jumbo-frame network connections where available). To monitor and confirm file transfers, NFT provides extensive job-tracking aids. It also supports command files and easy use in batch jobs.

This document is the comprehensive reference manual for NFT. Consult [EZOUTPUT](https://computing.llnl.gov/LCdocs/ezoutput) (URL: <https://computing.llnl.gov/LCdocs/ezoutput>) for a concise, task-oriented, introductory treatment of NFT that compares it with FTP. Consult [EZSTORAGE](https://computing.llnl.gov/LCdocs/ezstorage) (URL: <https://computing.llnl.gov/LCdocs/ezstorage>) for an overview of LC's archival storage system and of NFT's role in managing stored files (alternatives to NFT for storage management are also explained). The [FTP Reference Manual](https://computing.llnl.gov/LCdocs/ftp) (URL: <https://computing.llnl.gov/LCdocs/ftp>) tells how standard FTP features, including storage access, are implemented among LC production machines. If your file-transfer needs specifically involve placing many files into or retrieving them from a remote archive (TAR-format library) file, consult the [HTAR Reference Manual](https://computing.llnl.gov/LCdocs/htar) (URL: <https://computing.llnl.gov/LCdocs/htar>) for another LC-designed, locally deployed tool tailored to efficiently managing large archives in storage or on any preauthenticated FTP server. If you prefer a visual interface to NFT (where you select files and target directories with CTRL-CLICK using your mouse, for example), then execute HOPPER on any LC production machine and select NFT from HOPPER's CONNECT menu.

How to Run NFT

Basic Execution

To run NFT on any LC machine where it resides, type

```
nft [options]
```

where NFT's possible execute-line options are:

- i ignores any .nftc file. By default, as soon as it starts to execute, your NFT client reads the file .nftc (if any) in your home directory and immediately processes any NFT commands it finds there, echoing their normal responses (if any) to your terminal. This run-control file typically contains requests for nondefault environment-variable settings (keyword: [environment-variables](#) (page 15)) or for automatic logging of your NFT messages (keyword: [log-files](#) (page 19)).
- n outputs a newline character after each NFT prompt (omitted by default). This is intended to facilitate the handling of PERL scripts, but it is not recommended for ordinary interactive use.

NFT prompts for more input with the string nft> and you terminate your NFT client by typing QUIT (or, less elegantly, CTRL-C). Even after you have stopped running your NFT client, however, the NFT server will persistently execute any jobs (file-transfer requests) that have not yet completed (you can only kill your incomplete NFT jobs by using NFT's ABT command).

Basic NFT Features

DAEMONS.

Although NFT uses its own special server to schedule file-transfer requests and to persistently track them, it uses the standard FTP daemons on the sending and receiving machines to actually carry out your file transfers. Hence, some NFT commands (such as CHGRP or LN) may fail on some machines because the local FTP daemons do not support them there.

BINARY TRANSFER.

All NFT file transfers use the FTP binary (image) mode. You cannot change to ASCII mode with any NFT command.

PASSWORDS.

The NFT server preauthenticates your access to the machines where NFT works. Hence, all NFT file transfers are passwordless.

LIMITS.

The longest pathname that NFT accepts is 1023 characters. (Remember also that many UNIX utilities limit names to 16 characters.) The largest file that you can store using NFT is 10 Tbyte (use HTAR (URL: <https://computing.llnl.gov/LCdocs/htar>) to store and manage very large numbers of related files).

SYNTAX.

To specify the donor and target locations of files to transfer, NFT primarily uses a prefix or sentinel notation, somewhat like that used by SCP, rather than the login-based approach that FTP requires. For details on and examples of this prefix file-specification syntax, see the Syntax section below, keyword: syntax (page 12). See the OPEN (page 69) command for a way to make NFT somewhat mimic the FTP login approach.

SPECIAL CHARACTERS.

Several characters have special roles when used with NFT:

semicolon (:) NFT recognizes the semicolon (;) as a command separator on its execute line or in response to any NFT prompt (e.g., CLOBBER;PWD).

filters (? * [a-b])

The standard UNIX file filter (wild card) characters (? for single characters, * for any string, and [a-b] for end points of a specified range) are all accepted within file names by most NFT commands and can be used literally **only if quoted** (exceptions, where the result would be ambiguous, are noted in the description of specific NFT commands below).

filelists ({a,b,c})

NFT accepts these as a list of itemized file names a, b, and c.

other nonalphanumerics

- # ~

cannot appear as the first character in any file name, but can be used in other positions.

, : }

must be quoted if they appear in any file name (e.g., 'a:b'), because each otherwise has a special meaning for NFT.

quotes (' ")	Quotes have two special roles for NFT:
Quoted commands	Matched quotes surrounding commands only on NFT's execute line allow promptless execution (e.g., see the Input Files (page 21) section). NFT rejects quotes around any commands after its prompt as a syntax error.
Quoted file names	Matched quotes surrounding any file name in an NFT command (e.g., PUT 'a:b') can protect imbedded special characters in the name, allowing them to behave as ordinary alphanumeric characters. But note that this quote protection has two important limits: (1) Quotes do not protect - # ~ in the first-character position. (2) Quotes protect file-filter characters from NFT but not from subsequent special handling by many FTP daemons, who treat them as filters anyway.

COMMANDS.

To request and monitor file transfers, NFT uses interactive commands and environment-variable settings quite like FTP. However, some commands (e.g., GET) have specialized, storage-only roles for NFT that differ from their FTP roles. For details on the NFT commands, see the Command Summary (keyword: [command-summary](#) (page 14)) or the much longer Command Dictionary (keyword: [commands](#) (page 33)). When NFT's interactive commands have multiple, nonexclusive suboptions, you must concatenate all your chosen suboptions with a single hyphen (-) sentinel, not flag each with its own sentinel as UNIX usually allows. Thus, for example, the correct form is

```
dir -FPt
```

FILES.

To handle special situations, NFT can accept its input from files and send its output to files. For usage instructions, see the section on NFT Files (keyword: [files](#) (page 19)).

TRANSFER RATES.

To take advantage of the significantly faster file transfers (to or from storage **only**) that jumbo-frame network connections enable, NFT automatically routes compute-node transfers to/from storage through the login nodes on many LC clusters. For more details, for the implications for LC's parallel file systems, and for ways to check or disable this routing, see NFT's [ROUTING](#) command (page 77) below.

JOB TRACKING.

NFT goes far beyond SCP or FTP in the elaborateness of its job tracking. Relevant features include uniquely numbering each job; commands to report job status before, during, and even after completion; user control of NFT's interactive messages about job progress; and ways to create and monitor "sessions" of related jobs. For details, consult the Job Status and Reporting section (keyword: job-status (page 23)).

CLASSES OF SERVICE (STORED COPIES).

See the SETCOS section of the HPSS Manual (URL: <https://computing.llnl.gov/LCdocs/hpss/index.jsp?show=s1.2.1>) for a detailed discussion of HPSS "class of service" policy. STATUS (page 83) reports the current COS requested, and you can report an already stored file's class of service by using the -h suboption of NFT's DIR command (page 52). NFT handles new and previously stored files differently, however, in regard to COS values.

ERROR MESSAGES.

Most NFT messages about *user* errors (as illustrated in the example (page 10) dialog subsection below) are sequentially numbered along with other NFT responses and begin with the string "error":

```
n.0 error explanation
```

If you transfer many files at once, NFT explicitly distinguishes between "no clobber failures" (overwrites that you prevented) and other failures when it reports errors. Note that, if HPSS storage is down for planned maintenance, then attempts to store files using NFT yield a different error message with this special format:

```
[SCF|OCF] HPSS Storage is down for maintenance.  
*** Try again later.
```

Scripts that execute NFT should check for this special "three-star" error message and avoid needlessly resubmitting NFT jobs (storage requests) once it has been detected.

GRAPHICAL INTERFACE.

If you prefer a graphical interface to (mouse-oriented controller for) NFT, execute HOPPER on any LC production machine and select NFT from HOPPER's CONNECT menu.

Basic Examples

This annotated example shows typical file transfers using NFT.

- GOAL:** To transfer several files among (open) LC machines using NFT without logging on to all of the machines.
- STRATEGY:**
- (1) Start NFT.
 - (2) For convenience, change the working directory on YANA to /p/lscratchb/jfk (you could use pathnames later and skip this step).
 - (3) Transfer (outward copy) local file t1 to /p/lscratchb/jfk/t2 on YANA.
 - (4) Transfer (inward copy) file /p/lscratchb/jfk/t3 from YANA to local file t4.
 - (5) Without logging on to either YANA or ATLAS, transfer (copy) /p/lscratchb/jfk/t3 from YANA to /usr/tmp/t6 on ATLAS.
 - (6) Use storage-default command PUT to transfer file t1 from the client machine (where NFT runs) to storage.llnl.gov as file t2. Note that NO hosts are specified in this command because everything is defaulted.
 - (7) Try to retrieve file t8 from storage to local file t4 using the storage-default GET command. Because NFT's default environment is NOCLOBBER, this attempt fails because t4 already exists as a result of step (4) above. You could use the CLOBBER option next, to allow this overwrite, or...
 - (8) Use GET to retrieve t8 from storage with no name change (and hence no overwriting of t4).

```
nft --- (1)
nft>cd yana:/p/lscratchb/jfk --- (2)
  remote host yana: wd is /p/lscratchb/jfk
nft>cp :t1 yana:t2 --- (3)
  1.0. 95 bytes received in 0.1 seconds
  (0.7 Kbytes/s) from /g/g0/jfk/t1 to /p/lscratchb/jfk/t2
  1 entry copied /g/g0/jfk/t1

nft>cp yana:t3 :t4 --- (4)
  2.0. 98 bytes received in 0.1 seconds
  (1.1 Kbytes.s) from /p/lscratchb/jfk/t3 to /g/g0/jfk/t4
  1 entry copied /p/lscratchb/jfk/t3

nft>cp yana:t3 atlas:/usr/tmp/t6 --- (5)
  3.0. 98 bytes received in 0.4 seconds
  (0.2 Kbytes/s) from /p/lscratchb/jfk/t3 to /usr/tmp/t6
  1 entry copied /p/lscratchb/jfk/t3

nft>put t1 t2 --- (6)
  4.0. 95 bytes sent in 1.0 seconds
  (0.1 Kbytes.s) from /g/g0/jfk/t1 to ~/t2
  1 entry copied /g/g0/jfk/t1
```

```
nft>get t8 t4 ---(7)
  5.0. error. Cannot clobber existing
      sink /g/g0/jfk/t4
```

```
nft>get t8 ---(8)
  6.0. 95 bytes received in 1.8 seconds
      (0.1 Kbytes/s) from ~/t8 to /g/g0/jfk/t8
      1 entry copied ~/t8
nft>quit
```

NFT Pathname Syntax

Because NFT sessions (unlike FTP sessions) do NOT begin with you logging on to a specific remote host, you normally use NFT's pathname syntax to indicate each host (donor and receiver) involved in each NFT command. This section explains that syntax. (NFT does offer an OPEN (page 69) command that somewhat mimics FTP, but its use is atypical.)

An NFT pathname has three parts (some of which may be empty):

PARTS:	prefix	body	tail
EXAMPLE:	atlas:~jfk	/dir1/dir2/	code3*.c

The latter two parts follow the usual UNIX rules for specifying directories, trees of directories, files, and sets of files. File-filter wildcards (such as * and ?) and the special dot directories (. and ..) are allowed in the standard ways.

The first part of the pathname (the prefix) is unique to NFT (though similar to the SCP style). This is where you indicate the location (host) for the directories and files that you want NFT to transfer. There are 12 possible alternative locations, generated by a 3-by-4 matrix of prefix choices, as the left side of this chart reveals:

NFT-Specific Syntax		Usual UNIX Syntax	
Prefix		Body	Tail
-----		----	----
Any	Any	Zero	Zero
One	One	or	or
of	of	More	More
These:	These:	Directories	Files
		(. .. ok)	or
			Filters
:	/		
host:	~		
mt	~user		
	mt		
[mt="empty," NO			
characters here]			

where (1st column)

- :** (colon) indicates the local host (the machine on which you are running the NFT client). For example :test3 indicates that file test3 is in the current working directory on the local host.
- host:** indicates the specific (usually remote) host named. For example, atlas:test3 locates test3 in the current working directory on ATLAS, while yana:/usr/tmp/test3 locates test3 in the directory /usr/tmp on YANA. Only hosts that have NFT clients themselves are allowed here.
- mt** (empty position, no flag) indicates NFT's default host, which is the LC archival storage system (storage.llnl.gov). **WARNING:** This is a significant difference from FTP (e.g., test3 locates file test3 in the current working directory on STORAGE, not on the local host where you are running NFT).

and where (2nd column)

- / (slash) indicates the machine's root directory. For example, `:/test3` locates `test3` in the root directory on the local machine (:).
- ~ (tilde) indicates the user's home directory (must be followed by `/` unless it is last in the pathname). For example, `:~/test3` locates `test3` in the user's home directory on the local machine (:).
- `~user` indicates the home directory of the person with login name `user` (must be followed by `/` unless it is last in the pathname). For example, `:~jfk/test3` locates `test3` in the home directory of user `jfk` on the local machine (:).
- `mt` (empty position, no flag) indicates the current working directory. For example, `test3` locates file `test3` in the (default) current working directory on the (default) host STORAGE.

This comparative example shows the 12 alternative locations for `dd/ff` that this NFT syntax can specify:

NFT Syntax	Location of dd/ff
<code>:/dd/ff</code>	local host, root
<code>:~/dd/ff</code>	user's home
<code>:~kk/dd/ff</code>	kk's home
<code>:dd/ff</code>	current working dir.
<code>hh:/dd/ff</code>	host hh, root
<code>hh:~/dd/ff</code>	user's home
<code>hh:~kk/dd/ff</code>	kk's home
<code>hh:dd/ff</code>	current working dir.
<code>/dd/ff</code>	storage, root
<code>~/dd/ff</code>	user's home
<code>~kk/dd/ff</code>	kk's home
<code>dd/ff</code>	current working dir.

Some NFT commands (such as `PUT` and `GET`) are dedicated by default to file transfers with `STORAGE` (see keyword: [storage-defaulted](#) (page 14)). Because of this special role, the `storage-defaulted` commands do not require, or even allow, use of the NFT prefix syntax to specify file locations. `Storage-defaulted` NFT commands take standard UNIX pathnames as arguments, but the host on which any file resides is specified solely by the command's definition (usually by the file's position in the argument list), never by using an NFT locational prefix as shown here.

NFT Commands Summarized

NFT commands fall into several different, overlapping groups, based on their scope of operation, their file-transfer roles, and how they are processed during execution.

General and Storage-Defaulted Commands

Most NFT commands are general in scope and apply to any (secure) hosts that NFT serves, including (but not limited to) STORAGE. DIR (to list files) and CP (to copy or transfer them between machines) are examples of general NFT commands.

Some NFT commands are dedicated by default, however, to transferring files to or from STORAGE (such as PUT) or manipulating stored files (such as LN). The storage-defaulted commands exist because STORAGE is the primary file-transfer target for many NFT users and many NFT executions. These special NFT commands

- do not accept (or need) the NFT prefix syntax to specify the location of files to be processed (locations are specified by argument position, as with FTP), and
- can only be used for nonSTORAGE file transfers if you precede their use with an OPEN (page 69) command. NFT's OPEN somewhat mimics FTP's OPEN, but unlike FTP it is not required for most NFT file-transfer operations (and is never required for storage-only operations). Even when you use OPEN, some commands are still limited to storage-only use (as the table below shows).

This table lists the NFT commands by their default scope. Each command has a detailed explanation in the command dictionary later in this manual, keyword: commands (page 33).

General NFT Commands(*)	Storage-Defaulted Commands(+)
cp, delete, ren	get
dir, ls, pwd	put
cd, cdup, lcd	ln
mkdir, rmdir	chown
abt, rpt, clr	
block, group, endgr	
open, close	
help, quit, source, time	
chmod, chgrp	
source, time	

(*)For additional commands (mostly toggles) that control the NFT file-transfer environment, see the next section.

(+)Using OPEN (page 69) lets you "generalize" GET and PUT to other remote hosts, but most hosts other than STORAGE do not allow LN remotely. CHOWN can never be generalized to any host except STORAGE (even after you use OPEN).

Environment-Variable Settings

NFT provides several (pairs of) commands that do not directly perform file transfers or directory changes but rather let you control the file-transfer environment by toggling between alternative states. For each pair, one environment setting is NFT's default, as indicated in the table below (and the most noteworthy default, NOCLOBBER, prevents NFT from overwriting any file with a transferred file of the same name). The STATUS command reports the current settings. Two related environment commands also listed here support many-way choices (not just two-way toggles) among verbosity levels and NFT session numbers. Each command has a detailed explanation in the command dictionary later in this manual, keyword: commands (page 33).

NFT Environment Variables	Toggle Commands
Specify how multiple commands execute	sync (default, serially) async
Resolve file-name conflicts (overwriting)	noclobber (default, no overwrite) clobber
Display NFT output messages	term (default, show) noterm
Log all NFT input and output	clog (default, no log) log <i>filename</i>
Use login-node jumbo frames	routing (default, routes) norouting
Store multiples of mission critical files	nodualcopy (default) dualcopy
Specify HPSS class of service	setcos <i>nnn</i>
Specify verbosity of output	verbose <i>mask</i>
Change NFT session	session <i>nn new</i>
Report current NFT environment	status

Synchronous and Asynchronous Command Modes

The SYNC and ASYNC pair of commands overtly toggles NFT between two modes of executing multiple requests or "jobs," but other ways to change command-execution modes also exist.

By default, NFT commands (and jobs) execute serially (in SYNC or synchronous mode). While commands such as DIR and CD generally use so few resources as to execute (almost) at once, actual file transfers (copies with CP) may be delayed by resource unavailability on the server or client machines. Because it is obviously desirable to make or change directories before starting file transfers that depend on those moves (for example), serial (SYNC) execution is generally quite appropriate.

NFT supports three exceptions to or exemptions from SYNC mode, however, in increasing order of scope:

(1) GET/PUT:

When you use the storage-defaulted commands GET and PUT with file filters (* or ?) or with file lists, NFT always processes the resulting multiple file-transfer requests in parallel (asynchronously), regardless of the current SYNC/ASYNC mode setting.

(2) GROUP/ENDGR:

While remaining in SYNC mode, you can specify a subset of commands to process in parallel (asynchronously among themselves) by preceding them with GROUP and following them with ENDGR. To force all subsequent commands to wait until everything within such a GROUP has executed, use BLOCK after ENDGR and before the next command. For example, the following NFT command sequence first creates directory TEST3, then ASYNCHRONOUSLY copies three files into that directory (since arrival order is often unimportant), then, and only after all three have arrived (BLOCK), issues a report (RPT).

```
mkdir test3
cd test3
group
  cp yana:file1 :file1
  cp atlas:file2 :file2
  cp lucy:file3 :file3
endgr
block
rpt -a
```

(3) ASYNC:

Note: The ASYNC command should be used with caution. Results may not be as expected. The most general way to allow parallel NFT jobs is to use the ASYNC command. ASYNC cancels the default SYNC mode and makes NFT execute ALL subsequent commands in parallel (asynchronously), as soon as resources for them are available. BLOCK has no effect in ASYNC mode.

When you run NFT asynchronously, jobs are scheduled based on availability of the system resources needed to perform them. They may not all start or run immediately. The point of this scheduling is to prevent overloading the NFT server and its network connections.

Most NFT jobs, such as directory listings, directory creation, or reports, require few resources and receive immediate attention. Load balancing really becomes an issue only when you transfer files between machines. Here the NFT scheduling algorithm is quite complex, but it is based on these factors:

- The number of current transfers from the donating host.
- The number of current transfers by the requesting user.
- The size of the file(s) being transferred.
- The supply of resources (e.g., disk space) at the source and sink of each file transfer.

Local, Immediate, and Job Commands

NFT distinguishes between those commands executed locally, by your NFT client, and those executed remotely, by the NFT server. Furthermore, among the server commands, NFT distinguishes between those commands executed immediately and those ("job" commands) queued for persistent, numbered, monitored execution.

Local (Client) Commands

Local (client) commands are those that are executed completely by the NFT client that you run. Because the client, not the server, executes them, local commands have no NFT job numbers assigned to them. The NFT local commands are:

async	block*	clobber	clog
close+	dualcopy	endgr*	group
help	log	noclobber	nodualcopy
norouting	noterm	open+	pwd
quit	routing	setcos	source
status	sync	term	time

(*)These commands do require communication with the server and use job numbers, but the numbers are never seen by the user and no BEGIN, DONE, ERROR, or ACCEPTED messages are returned for them.

(+)The NEXT job (server-executed) command actually and persistently makes the connection requested by an OPEN or CLOSE local command that precedes it.

Server Commands

Immediate Commands

NFT immediate commands execute as soon as the NFT server receives them. They are not placed on the standard job-execution queues as the regular job commands (next section) are. Any job subsequently submitted to the server is affected by the previous immediate commands, but jobs submitted prior to an immediate command will never be affected by it. The immediate server commands are:

abt	clr	rpt
session	verbose	

Job Commands

NFT job commands are server-executed commands each assigned a unique job number that you may use to get information about the job or to abort the job later. These commands are automatically retried by the NFT

server when a temporary failure occurs because of network, host, or communication problems. The NFT job commands are:

cd	cdup	cp
chgrp	chkrouting	chmod
chown	delete	dir
get	lcd	ln
ls	mkdir	put
rename	rmdir	

Input and Output Files

NFT Output (Log) Files

Using LOG and CLOG

NFT allows you to begin at any time recording in a log file all your input to and output messages from the program as it runs. To start a log file, use the NFT command

```
log pathname
```

where *pathname* is usually just the name of a file (e.g., nftlog) that you want NFT to create in the current working directory (where NFT was started) on the machine where you are running the NFT client. If you supply an absolute pathname (such as ~/projects/nftlog or /usr/tmp/testdir5/log5) then NFT creates the log file in the other directory that you specify.

Logging of all NFT input and output continues until you issue the CLOG (close log) command or until you QUIT your current NFT session, whichever comes first.

Issuing a second LOG command with a different pathname will: (1) create a second, independent log file, (2) stop recording messages in the first log file, and (3) start recording (subsequent) messages in the second log file. Issuing a second LOG command with the same pathname will insert an updated time stamp (comment) in the open log file and then simply append all new messages to the old ones. If a file named *pathname* already exists when you first issue a LOG command, NFT opens it, appends a current time stamp, and places all your new messages at the end of that file. Thus, you can jump between multiple log files during an NFT session, channeling messages to one and then another, just by using LOG with each file's name whenever you want to change files.

NFT Logging Techniques and Examples

NFT does not collect a large buffer of messages before logging them; instead, messages are flushed to the log file after every carriage return. Logging starts immediately, and the first line recorded in your log file (after a comment-line date stamp) will be the LOG *pathname* command that requested the file. CLOG will be the last line recorded. As long as you do not change the (default) TERM mode setting with a NOTERM command, all your input and output will continue to display at your terminal while logging occurs.

Every line in an NFT log file is annotated by a prefix that indicates its origin (for easy analysis later). These log-file prefixes are:

indicates a comment (usually the date-time stamp at the start).

usr) indicates input from you at the terminal, for example,

```
usr) get test3
```

src) indicates input from a command file that you had NFT process by issuing the SOURCE command. For example,

```
src) get test3
```

came from a command file, not the keyboard (see below, keyword [source-usage](#) (page 21), for details).

) indicates output from NFT, such as a prompt or response to a command. For example,

```
) 48.0 95 bytes received in 1.8 seconds (0.1 Kbyte/s) test3
```

would be a typical logged response to the GET command shown above.

A sample of a short but typical NFT log file looks like this:

```
#
# NFT log --- Fri Dec 19 13:42:49 2006
#
usr) log nftlog2
) nft>
usr) cd atlas:/usr/tmp
) remote host atlas: wd is /usr/tmp
) nft>
usr) cp :t1 atlas:t2
) 1.0. 95 bytes received in 0.1 seconds
) (0.7 Kbytes/s) from /g/g0/jfk/t1 to /usr/tmp/t2
) 1 entry copied /g/g0/jfk/t1
) nft>
usr) get t6
) 2.0. 95 bytes received in 1.8 seconds
) (0.1 Kbytes/s) from ~/t6 to /g/g0/jfk/t6
) 1 entry copied from ~/t6
) nft>
usr) clog
```

NFT Input Files

File Input by Redirection

NFT accepts input from a command file on its execute line if you use the standard UNIX input redirection symbol (<), as shown here:

```
nft < commandfile
```

Here *commandfile* can be a simple name of a file in the current directory (e.g., infile3), a pathname relative to that directory (such as projects/infile3), or an absolute pathname (e.g., /usr/tmp/projects/infile3).

The command file should contain just the same NFT commands (and arguments) that you might type at your terminal, one per line. Running NFT with a redirected input file alters its usual interactive behavior. NFT executes the commands in the file silently, without echoing them. After all commands have started, NFT automatically ends and you get the usual operating-system prompt. It passes status messages ("remote wd is xxx") to your terminal at once, but any job-numbered transaction messages (about files sent or received) that happen to arrive **after** the NFT client ends are lost (this could be all of them).

Superficially similar to file input by redirection on NFT's execute line is placing a quoted string of (semicolon-delimited) commands on NFT's execute line, as shown here:

```
nft "clobber;time;pwd;put test4"
```

This actually behaves more like file input using SOURCE (next section); however, the quoted commands are not echoed, but the normal status and job-numbered transaction messages are echoed, and afterward NFT does not end but simply prompts for more input (unless the last quoted command is QUIT).

File Input Using SOURCE

NFT also accepts input from a command file during any interactive session if you use the SOURCE command in response to its nft> prompt:

```
source commandfile
```

With SOURCE, as with file redirection, *commandfile* can be an absolute pathname such as /usr/tmp/projects/infile3, as well as a relative pathname or a local file's name. And again (as with file redirection) with SOURCE, status messages continue to appear at your terminal while the commands in the file execute, but you will see no command echoes and only those job-numbered transaction reports (about files sent or received) that happen to arrive before your NFT client ends (and it could end quickly if the last command in your file is QUIT).

Usually the commands in a SOURCE command file are just what you might type at your terminal, one per line. But you can construct condensed and annotated command files by using # as a comment sentinel, semicolon as a command separator, and backslash (\) as a line-continuation flag. Thus, the following two examples are equivalent command files for use with SOURCE.

```
clobber
put test4
get test6

#shows special characters
clobber;put test4;get tes\
t6
```

Logging Jointly with Input Files

You can log NFT output with the LOG and CLOG commands (keyword: [log-files](#) (page 19)) while also providing input from a command file, but the captured results will be different than if you log a normal terminal session.

If, while logging NFT output, you use SOURCE to issue commands from a file, both the SOURCED commands and the system's status messages will be included in the log file, even though the commands would never appear at your terminal. Each is prefixed by src) in the log, but job-numbered transaction reports will only appear in the log file if they would have arrived at the terminal while your NFT client was still running (so often they are missing).

If you use input redirection on the NFT execute line, you can include LOG and CLOG among the commands in the input file to record what happens. However, while your input commands (e.g., PUT test3) and status messages will appear in the log file, only those job-numbered transaction reports that happen to arrive before your client ends are captured in the log. If the redirected command file also contains a SOURCE command within it (to include yet another command file), then each SOURCED input line will appear prefixed by src) in the log file.

Job Status and Reporting

NFT Job Numbers and Classes

Using Job Numbers

Executing NFT begins a session (namely, session 0), and every NFT request (or job) in that session has a unique integer job number, starting with 1 and increasing sequentially. The NFT server remembers your past jobs for up to 4 days depending on NFT traffic (unless you flush their records with CLR or a session has more than 499 job statuses), and during this interval it will not reuse the numbers of remembered jobs. So frequent NFT users may find job numbers incremented by 1 from their most recent job rather than starting at 1.

You can use each job's unique number to get information on its status in case it does not complete immediately (or at all). For example, to request the status of the job numbered 13, use NFT's RPT option:

```
User:  rpt 13
Rtne:  13.0 error. Cannot clobber existing sink
      /g/g0/abc/test4
```

See the RPT section (keyword: [job-reporting](#) (page 25)) for details on other ways to request status reports.

For the special situation where you have jobs in multiple NFT sessions, with multiple job-numbering sequences, see the Sessions section below (keyword: [sessions](#) (page 29)).

Job Class Hierarchy

NFT classifies all jobs into 8 states (or job classes), shown here in their nested hierarchy:

Class	Member Jobs
all	all jobs that NFT remembers, regardless of state(*)
incomplete	jobs waiting to run or not finished running
held	jobs in the scheduling queue
active	jobs currently running
complete	successfully or unsuccessfully completed jobs
okay	successfully completed jobs
error	unsuccessfully completed jobs
aborted	jobs terminated by the user

(*)If you use multiple NFT "sessions," a special practice described in the Sessions section below (keyword: sessions (page 29)), then ALL includes only jobs in the currently selected session, NOT in all the sessions that you have created.

You can use NFT's RPT option to request status information for a whole class of jobs as well as for a specific job number (see the next section, keyword: job-reporting (page 25)). The NFT server remembers which class your jobs are in for about 4 days, but if you submit many jobs and want to monitor only the newest and most interesting ones, you can flush or "clear" NFT job-status records (by job number or for a whole class of jobs) using the CLR option.

Job Reporting with RPT

You can at any time review the status of your NFT requests or jobs by using the RPT (report) option. Two related options let you clear NFT status records no longer of interest (CLR) or abort jobs found to be still incomplete (ABT). The NFT STATUS option reports on your environment-variable settings, not on your file-transfer jobs.

Scope

RPT by default reports the status of your most recent job (in the currently selected NFT session). But you can specify as RPT's argument a specific job number (n) or a range of numbers ($n-m$, using a hyphen, not a comma, as separator) to get reports on just the jobs that you select. The Using Job Numbers section above gives an example (keyword: [job-numbers](#) (page 23)).

In addition, RPT (with CLR and ABT) understands the 8 classes of NFT jobs described above (keyword: [job-classes](#) (page 24)). Using a one-letter code for each job class, you can request status information on all your jobs that belong to the class you select:

Job Class	RPT	ABT	CLR
all	-a		
incomplete	-i	-i	
held	-h	-h	
active	-x	-x	
complete	-c		-c
okay	-o		-o
error	-e		-e
aborted	-k		-k

Note that while you can report (RPT) on any class of job, you can only abort (ABT) incomplete jobs (held or active) and you can only clear (CLR) the status records of completed jobs (okay, error, or aborted). See the next section for a sample report (keyword: [rpt-examples](#) (page 26)).

Besides the job number(s) or job class(es) you select, three other factors are relevant to the scope of RPT status reports:

(1) RECORD PERSISTENCE.

The NFT server remembers your jobs, their numbers, and their status for up to 4 days before purging its records. So frequent NFT users will get reports on all members of a job class throughout this time range, not just on the jobs started with their currently running NFT client. If this is a problem, use CLR to overtly delete the old(er) records that are no longer of interest.

(2) SESSIONS.

Most NFT uses have only one NFT session, and RPT reports on the jobs (job class members) in that session. If you start multiple sessions, RPT reports only on the jobs in your currently selected session, ignoring all your jobs in other sessions. This can give you more control of your status reports or lead to confusion, depending on your awareness of the sessions you have started. For details on the effect of multiple sessions, see the Sessions section below (keyword: [sessions](#) (page 29)).

(3) VERBOSITY.

The NFT VERBOSE option (keyword: verbose-levels (page 27)) lets you specify which state changes NFT reports interactively as it runs (e.g., when jobs begin as well as when they end). VERBOSE does not, however, affect which jobs (or job class members) NFT includes in RPT status reports, nor the amount of detail provided on each job's status line. Thus, VERBOSE does not change RPT's scope at all, although it does change general NFT dialog.

Format and Examples

Most RPT status reports consist of one line per job reported, with the format

```
      jnumFrnum.  status  info  
Example:  
      12.0.      done.      ~/nfttest/test4
```

where

- jnum* is the unique integer number NFT assigned to this job.
- F* is a flag that indicates either a primary job (.) or a secondary job (/). A primary, user-submitted job such as GET * may generate many secondary jobs, one for each file retrieved.
- rnum* is NFT's retry count, which always starts at 0.
- status* is a standard status-reporting term (begin, done, error, start, accept).
- info* characterizes your specific job, for example, by giving the pathname of the file retrieved or of the directory listed.

To illustrate a typical RPT job-status report on a class of jobs, assume that you have started an NFT client and issued these commands:

```
cd nfttest  
dir  
get test4  
clobber  
get test4
```

Then, if you request a status report on all of these jobs by using

```
rpt -a
```

NFT's response might look like this:

```
11.0. done. ~/nfttest  
12.0. done. ~/nfttest  
13.0. error. Cannot clobber existing sink  
      /g/g0/abc/test4  
14.0. done. /g/g0/abc/test4
```

Note that your CLOBBER command, which does not involve any information transfer between machines, is not treated as a numbered job by NFT and so is omitted from RPT's status report. Invoking RPT's -v ("verbose") option inserts on each reported line the command (action) used as well as the usual status information. For example:

```
14.0. done. get /g/g0/abc/test4
```

Diagnostic Verbosity

NFT jobs pass through several states from submittal to completion, and you can control how finely NFT reports on these changes of state by using its VERBOSE option. By default, NFT passes along transfer statistics from the FTP daemon that actually moves files at NFT's request, as well as sending error and abort diagnostic messages if a job completes unsuccessfully. But there are other state changes, too, and you can request messages about any or all of them by using the appropriate argument for VERBOSE. (VERBOSE does not change the scope of jobs covered in status reports from RPT (page 78), which has its own verbose -v option, nor the environment-variable setting reports from STATUS.)

Each possible state change for an NFT job corresponds to one bit in a (32-bit) mask that VERBOSE sets. You request diagnostic messages about a state change by setting its bit in the mask, and you set each bit by using the decimal value shown in the table below. To request a combination of reports, ADD the corresponding decimal values and use the sum as the argument for VERBOSE (for example, the default combination of diagnostic messages corresponds to the sum $4+8+64=76$).

State Change	Decimal Value	Diagnostic Meaning
Begin	1	Client has submitted job
Done	2	Job has completed successfully
Error(*)	4	Job has failed (unsuccessfully completed)
Abort(*)	8	Job was killed by user
Accepted	16	Job was received by server
Reserved	--	--
Transfer stats(*)	64	FTP transfer amount and rate
Start	128	Server has started job execution
Progress errors	256	Immediately reports in-progress errors in secondary jobs
Reserved	--	--

(*)Default verbosity (combination 76)

To see how changing the VERBOSE value changes the grain size of state-change reports during NFT dialogs, compare this default-value exchange (VERBOSE 76)

```
User: get test4
Rtne: 14.0. 95 bytes received in 1.3 seconds (0.1 Kbytes/s)
      from ~/test4 to /tmp/jfk/test4
      14.0. 1 entry copied ~/test4
      nft>
```

with this maximum-value exchange for the same job (VERBOSE 479):

```
User: get test4
Rtne: 14.0. accept.
      14.0. begin ~/test4
      14.0. start ~/test4
      14.0. 95 bytes received in 1.3 seconds (0.1 Kbytes/s)
      from ~/test4 to /tmp/jfk/test4
```

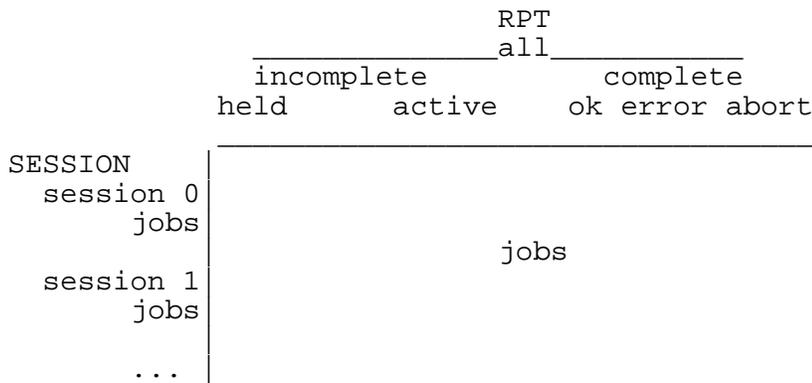
```
14.0. 1 entry copied ~/test4
14.0. done. /tmp/jfk/test4
nft>
```

NFT Sessions

Grouping Jobs by Session

Whenever you start an NFT client you start some NFT "session," with a unique (integer) session identifier. (This is true even when you use quoted execute-line commands or input from a file, although the absence of a prompt or echo can hide NFT's use of a session here.) By default your session is 0. The NFT server associates the current session identifier with every job you submit during that session as part of its persistent job tracking and execution. As a result, sessions provide another dimension along which you can group jobs, independent of their job class (keyword: `job-classes` (page 24)), for monitoring and analyzing their status. An NFT session is a logical set of jobs sharing the same session number. It is not the same as an NFT client (one client can use several sessions), nor a sequence of jobs (some session jobs may be asynchronous and mixed with other-session jobs), nor a time block (two sessions may overlap in time).

This diagram shows the relationship between NFT job classes (used by the RPT job-status report option) and job sessions:



The diagram also suggests the prime reasons for invoking multiple sessions:

- To subdivide the jobs in a class (all complete jobs, all error jobs, etc.) between sessions for separate status monitoring where you need such detail or extra control.
- To allow separate RPT reports, not intermixed, on two (or more) sets of secondary jobs. Thus running a large GET * request in one session and a large PUT * request in another session would let you generate separate RPT reports on the many subordinate file transfers of each without interference with the other.
- To let you recover and inspect the RPT status reports on all the jobs in each session (perhaps including jobs from several classes) independently of one another, even after your NFT client ends.

One Client, Multiple Sessions

TECHNIQUES.

You can start a new session at any time while running NFT by typing

```
session n
```

where *n* is an integer from 1 to 99 inclusive. This command closes your former NFT session (session 0 is the default) and opens a new one, in which the new session number *n* is associated with all your subsequent NFT

commands. NFT numbers the jobs in each session with an increasing sequence of integers that ignores all other sessions (so, e.g., each session may have an unrelated job numbered 13).

You can reopen a former session by using its session number in this same command (e.g., `SESSION 0` reopens that session and closes session 1 if issued while you are in session 1). Reopening a session lets you check on its jobs with `RPT` or start more jobs associated with it.

NFT gives no overt confirmation when you close one session and open another (just the usual prompt for input). And `RPT` status reports do not reveal which session they cover. So to discover which session is now open (or to confirm a requested change of session) use NFT's `STATUS` command, which reports the current session number along with other NFT environment settings.

Because NFT remembers your session numbers for up to 4 days and because picking a session number already in use reopens that session rather than creates a new one, you may sometimes want NFT to open a new session by automatically picking a number for it that is guaranteed to be unused. To guarantee a fresh session number, use

```
session new
```

to which NFT responds with the message

```
New Session: n
```

where n is the next unused (by you) session number available. As before, your previous session is now closed and the n th session is now open.

EFFECTS.

NFT sessions group together jobs, not log messages or environment variable settings.

RPT scope RPT reports on the status of only those NFT jobs associated with the session open when you issue the `RPT` command. To report on jobs in other sessions, you must first reopen the session of interest, then type `RPT`. `RPT` does not accept session-number arguments and its reports do not state which session number they cover (use `STATUS`). Remember also that you may have several jobs with the same job number, one in each of several independent sessions.

LOG scope If you start recording your NFT interactions with `LOG logfile`, then *logfile* will continue to collect messages in an unbroken sequence even if you open new sessions or reopen old ones. You can change log files if you wish (keyword: [log-files](#) (page 19)), but changing sessions does NOT automatically change or close log files, and you cannot permanently assign separate log files to separate sessions. Sessions divide NFT jobs while log files divide NFT messages.

environment-variable scope

Environment-variable settings (e.g., `SYNC/ASYN` or `NOCLOBBER/CLOBBER`) persist independently of NFT session changes. You cannot permanently assign some settings to one session and different settings to another. Thus you cannot simply declare one `NOCLOBBER GET` session and another `CLOBBER PUT` session, for example, because each time you set that variable all (incomplete) jobs in all sessions will be affected. Only by using separate NFT clients (not multiple sessions with one client) can you have two sets of environment-variable settings for two sets of jobs at once. See the next section for details.

Multiple Clients, Multiple Sessions

You can run several NFT clients at once (each in its own window, for example), but to do so you must compensate for the way NFT manages sessions when the same user runs more than one client.

```
Client 0   Client 1
-----   -----
Default:  session 0 "steals"
          session 0
To regain,
you type: session 0 session 1
```

By default, your first NFT client (e.g., client 0) opens session 0. Executing a second NFT client (e.g., client 1) automatically transfers that open session to the second client. One result of this "session theft" affects output messages, all of which are diverted from the first to the second NFT client. Another result affects subsequent input (commands) to the first client. Attempts to execute more commands (other than SESSION) yield a fatal error, terminating the NFT client:

```
***Panic. Another application instance
has opened this session.
```

You can overcome this session theft and use both NFT clients with the help of NFT's SESSION command, used either interactively or in batch runs:

INTERACTIVELY.

Issue SESSION 1 to the second client (client 1) to open a new, independent session for it. Then issue SESSION 0 to the first client (client 0) to reopen or recover the lost session. Now both NFT clients will separately accept and process input, keep logs, and report job status with RPT. They will also maintain different NFT environment variable settings (e.g., NOCLOBBER for one, CLOBBER for the other) if you wish. In fact, only by using such separate NFT clients, each associated with a different NFT session, can you have two sets of environment-variable settings for two sets of NFT jobs at once.

BATCH RUNS.

The above technique is not very practical if you want to execute several batch scripts at the same time, each involving NFT, yet keep their NFT transactions separately recorded. Instead, invoke SESSION NEW when you first run NFT in each script, for example,

```
nft "session new;log nftlog;status;clobber;put abc;quit"
```

This requests (and reports to your log file) a previously unused session number, so that your file-transfer records end up segregated in that uniquely identified session. You can then reopen that same session (with the SESSION *n* command) to accumulate records for every subsequent NFT execution in the same script, or repeatedly use SESSION NEW for each NFT run.

Multiple clients, each with their own session, are usually used to divide NFT job streams so they can be processed or recorded differently. But, if this is not your goal, you can pass NFT sessions back and forth repeatedly between clients at any time simply by typing SESSION *n* to cause the current client to open and feed its jobs into the *n*th session. NFT numbers jobs sequentially by session, NOT by client, so exchanging sessions between clients in this way will allow both clients to contribute jobs, with one sequence of job numbers, to one accumulating status record (per session) that RPT will report. See the Grouping Jobs by Session section (keyword: [session-scope](#) (page 29)) for more details on how NFT sessions and job-status reports interact.

Using NFT in Scripts

Certain techniques and features scattered among various places in this manual are especially helpful if you want to execute NFT within a script (for use as a batch job, for example) and then feed NFT a series of (normally interactive) commands. This section summarizes these techniques for easy comparison, roughly in order of increasing complexity, and offers cross reference links when details are available elsewhere.

QUOTED COMMANDS.

Perhaps the easiest way to pass a series of commands to NFT noninteractively within a script is to (1) separate the commands with semicolons and (2) quote the entire command sequence on NFT's execute line. For example,

```
nft "clobber;cd dir3;put test3;quit"
```

enables overwriting, changes storage directories to DIR3, saves file TEST3, and terminates NFT. You must include QUIT at the end of such quoted command sequences to prevent NFT from prompting interactively after it executes the other commands. For technical details, see the [NFT Features](#) (page 7) and [NFT Input Files](#) (page 21) sections.

HERE FILE.

Very like the foregoing technique is the use of a UNIX "here file" of command lines imbedded within your larger shell script, initiated by << and delimited by a string of your choice (such as EOF). This example "here file" duplicates the behavior of the quoted NFT commands in the previous paragraph:

```
nft <<EOF
clobber
cd dir3
put test3
quit
EOF
```

VARIABLE EVALUATION.

For more elaborate situations you may prefer to assign NFT command sequences to a script variable and then evaluate that variable on the NFT execute line. For example, a PERL script might contain this fragment (using the same NFT commands as before):

```
$inn = "clobber;cd dir3; " .
      "put test3;quit";
@out = 'nft "$inn"';
```

FILE REDIRECTION.

NFT provides two ways to execute already existing separate command files:

- (1) the usual UNIX redirection of input, and
- (2) NFT's own SOURCE (read an input file) command. This approach allows you to reuse elaborate, comment-annotated NFT command files, but those files are not part of the invoking script itself, posing a possible file-management problem. For a comparison of the implementation details of these two external-file approaches, see the [NFT Input Files](#) (page 21) section.

If actually completing (not just launching) all your NFT commands before starting your script's next step is important to you, consider using NFT's [BLOCK](#) (page 36) command before you QUIT. And if running several scripts with NFT commands at the same time is likely, or if you typically rerun a script many times close together, then consider using NFT's SESSION NEW feature to help manage your NFT record keeping. Review the [NFT Sessions](#) (page 29) section for a discussion of the relevant problems and possible solutions.

NFT Command Dictionary

Command Syntax Advice

The other subsections of this command dictionary explain each of NFT's (over 40) interactive commands in alphabetical order. Comparisons and cross references indicate when several alphabetically scattered commands are closely related to each other in function.

- See the much shorter Command Summary (page 14) for a concise overview of NFT commands grouped by function instead of by name.
- See the How To Run NFT (page 6) section for instructions on how to start and operate the program.
- See the Basic NFT Features (page 7) section for length limits and special-character rules that apply within the NFT commands you use.

NFT has several special syntactical features that affect how you can use its interactive commands:

CASE.

You can type all NFT commands in either lowercase or uppercase (e.g., status or STATUS). Of course, command suboptions (such as -R) remain case sensitive as is typical of UNIX software.

CONCATENATION.

When NFT's interactive commands have multiple, nonexclusive suboptions, you **MUST** concatenate all your chosen suboptions with a single hyphen (-) sentinel, not flag each with its own sentinel as UNIX usually allows. Thus, for example, the correct form is

```
dir -FPt
```

ESCAPE.

While running NFT, you can use the exclamation mark (!) as an escape character (prefix) to execute any ordinary UNIX shell command. Thus typing LN executes NFT's own LN command (on the STORAGE system) but typing !LN executes the regular LN utility on the local (client) machine instead. Naturally, these !-escaped commands get no NFT job numbers and are not persistently executed by the NFT server.

ABT (Abort Incomplete Jobs)

SYNTAX:

```
abt [[n[-m]] | [-opt]]
```

ROLE:

Aborts (immediately ends) your most recent NFT request ("job") by default, or aborts your specific job with unique job number *n* (an integer), or the range of jobs with numbers *n-m* inclusive (using a hyphen, not a comma, as separator), or all members of the job class specified by any one job-class option *opt* listed below. You can only abort incomplete jobs (those either still held for scheduling or actively running). Attempting to abort completed jobs has no effect, but will return the warning that "completed jobs were ignored."

You can use NFT's RPT (report) option to discover the job numbers of still-incomplete jobs that you might want to abort. On the other hand, you need not wait for an RPT report, or even for the nft> input prompt, to use ABT to prevent a just-issued erroneous command from completing. You can type ABT immediately after starting some undesired NFT job, even before getting an input prompt, and (often) abort that current job quickly.

ABT does not stop your current NFT client from running, just as QUITing or killing the client does not stop the NFT server from completing any of your already submitted jobs.

RPT (page 78) (report) and CLR (page 46) (clear) are NFT commands closely related to ABT. The NFT STATUS (page 83) command reports on your environment-variable settings, not on your file-transfer jobs.

OPTIONS:

ABT accepts several job-class options *opt* to specify which set of your NFT jobs you want aborted. But ABT expects you to use these options ONE at a time: combined options do not yield combined job classes to be terminated. Possible stand-alone job-classes on which you can apply ABT include:

- a all jobs that NFT remembers, regardless of state. [If you use multiple NFT sessions (page 29), a rare practice, then -a selects only jobs in the current session, NOT in all the sessions that you have created, and the other job-class options behave likewise.]
- i incomplete jobs, waiting to run or not finished running.
- h held jobs, incomplete jobs in the scheduling queue.
- x active jobs, incomplete jobs currently running.

EXAMPLE:

A typical use of ABT to stop an inadvertantly started file transfer (in this case, immediately after it was started, even before NFT returns another input prompt) is:

```
User: get test4
      abt
Rtne: 5.0 aborted /g/g0/jfk/test4
```

ASYNC (Run Jobs in Parallel)

SYNTAX:

async

ROLE:

Begins asynchronous mode. Because SYNC is the default setting whenever you run NFT, the ASYNC command serves to cancel this default or any previous SYNC (page 84) command. In ASYNC mode, NFT executes all your subsequent commands (jobs) in parallel, allowing any job to run in any order as soon as resources are available. **Note: The ASYNC command should be used with caution. Results may not be as expected.** ASYNC should **not** be used with the belief that file transfers will occur more quickly, nor should it be used with multiple recursive PUTs.

Asynchronous execution on a limited scale also occurs for multi-file transfers using GET, PUT, or CP, and for command sets flanked by GROUP (page 58) and ENDGR. The BLOCK (page 36) command has no effect in ASYNC mode. See the command-sequencing (page 16) section above for a comparative analysis of the three cases where NFT allows asynchronous command execution.

The SYNC (page 84) command cancels ASYNC mode. ASYNC has no options and returns no mode confirmation, but you can use NFT's STATUS (page 83) command at any time to discover your current SYNC/ASYNC setting, which persists even across logical NFT "sessions (page 29)."

BLOCK (Block or Delay Next Command)

SYNTAX:

block

ROLE:

Prevents NFT from executing any more commands until all previously entered synchronous jobs have completed. Thus BLOCK;QUIT will end NFT only after all pending jobs are done rather than QUITing immediately (the default).

Synchronous (SYNC (page 84), sequential) command execution is the default whenever you run NFT, but NFT does support these three exceptions to or exemptions from SYNC mode:

- multiple-file transfers using GET, PUT, or CP,
- command sets flanked by GROUP and ENDGR, and
- commands following ASYNC.

BLOCK has no effect in ASYNC mode, but using BLOCK immediately after any of these asynchronous episodes ends will prevent the next command (which may depend on previous job completions for success) from executing prematurely, as the example below shows. Also, if you start a new NFT session (page 29), BLOCK is an easy, harmless way to confirm that the new job-sequence numbering has started.

BLOCK has no options and returns no confirmation.

EXAMPLE:

While remaining in SYNC mode, you can specify a subset of commands to process in parallel (asynchronously among themselves) by preceding them with GROUP and following them with ENDGR. To force all subsequent commands to wait until everything within such a GROUP has executed, use BLOCK after ENDGR and before the next command. For example, the following NFT command sequence first creates directory TEST3, then ASYNCHRONOUSLY copies three files into that directory (since arrival order is often unimportant), then, and only after all three have arrived (BLOCK), issues a report (RPT).

```
mkdir test3
cd test3
group
  cp yana:file1 :file1
  cp atlas:file2 :file2
  cp lucy:file3 :file3
endgr
block
rpt -a
```

CD (Change Working Directory)

SYNTAX:

```
cd [host][pathname]
```

ROLE:

Changes the current working directory on the specified *host* to the specified *pathname*. Here

host is the NFT host-specifying prefix that defaults to the STORAGE (not current client) machine, as explained above (keyword: prefix (page 12)). If you precede CD with an OPEN (page 69) command, then *host* defaults to the machine that OPEN specified.

pathname is a standard UNIX path that defaults to your home directory.

So CD used with no arguments (and no OPEN) changes your current STORAGE directory to your home directory. Use PWD (page 73) to discover the name of your current working directory, CDUP (page 38) to move up one directory level, and LCD (page 60) to change local (client-machine) directories. These commands are somewhat redundant, so that the following are exactly equivalent, and all require using NFT's own prefix syntax for hosts.

```
CD :..
CDUP :
LCD ..
```

CD takes no options, and it confirms each requested directory change.

WARNING:

If you specify a nonexistent pathname for the UNIX CD command, you get an immediate error message and no directory change. If you specify a nonexistent pathname for NFT's CD command, however, you get the usual confirmation message before receiving the string "no such file or directory." And subsequent use of PWD will also report the nonexistent directory, without complaint. Of course attempts to then use DIR or actual file-transfer commands will fail. You must overtly reset the working directory with another use of CD to a real location to overcome this error and continue transferring files.

EXAMPLE:

In response to each CD command, NFT reports both the machine involved and the new working directory (wd), where the default remote machine is always STORAGE.

```
R/Us:  nft> cd nftttest
Rtne:   remote wd is ~/nftttest           [on STORAGE]
R/Us:  nft> cd :~
Rtne:   local wd is /g/g0/jfk            [on client machine]
```

CDUP (Change Working Directory Up)

SYNTAX:

```
cdup [host]
```

ROLE:

Changes the current working directory on the specified *host* to the parent directory (one level up), where *host* is the NFT host-specifying prefix that defaults to the STORAGE (not current client) machine, as explained above (keyword: prefix (page 12)). If you precede CDUP with an OPEN (page 69) command, then *host* defaults to the machine that OPEN specified.

So CDUP used with no arguments (and no OPEN) moves your current STORAGE directory one level up. Use PWD (page 73) to discover the name of your current working directory, CD (page 37) to change directories generally, and LCD (page 60) to change local (client-machine) directories. These commands are somewhat redundant, so that the following are exactly equivalent, and all require using NFT's own prefix syntax for hosts.

```
CD :..
CDUP :
LCD ..
```

CDUP takes no options, and it confirms each requested directory change.

EXAMPLE:

In response to each CDUP command, NFT reports both the machine involved and the new working directory (wd), where the default remote machine is always STORAGE.

```
R/Us:  nft> cdup
Rtne:   remote wd is ~                [on STORAGE]
R/Us:  nft> cdup :
Rtne:   local wd is /g/g0/jfk         [on client machine]
```

CHGRP (Change Groups)

SYNTAX:

```
chgrp [-R] group [host]filelist
```

ROLE:

changes to *group* the group membership of the files or directories specified by *filelist*, a standard UNIX pathname or file filter, on the specified *host* (an NFT host-specifying prefix (page 12) that defaults to the STORAGE (not current client) machine).

CHGRP can change the group for link targets, but not for the link itself. If you change the group for a directory, however, CHGRP does not recursively change groups for all the contained files (although you can change them all at once from within the directory by using the * filter if you wish or by invoking the -R option).

DEFAULTS:

When you store a file with NFT, its default group is the same as that of the storage directory that receives it (not necessarily the group it belonged to on the donor machine). So changing a storage directory's group will change the default group of all files subsequently stored within it (but already stored files will retain their original group, even if they are overwritten later). If you try to assign a file to a nonexistent group or one to which you do not belong, NFT returns an error message (exact text varies depending on your target machine). You can discover the groups you belong to on the local (client) machine by taking advantage of NFT's ! escape syntax and typing

```
!groups youruserid
```

but you may not belong to the same groups on the STORAGE machine, and NFT does not support any equivalent to GROUPS for use on STORAGE. See the "Sharing Stored Files" section of the EZSTORAGE (URL: <https://computing.llnl.gov/LCdocs/ezstorage>) guide for how to use LDAPSEARCH to discover your storage groups. Contact the LC user hotline (at lc-hotline@llnl.gov or lc-hotline@pop.llnl.gov) to obtain the forms needed to create new groups on STORAGE.

CHMOD (page 42) is a similar NFT command that changes file permissions. You can use DIR (page 52) to reveal the current group to which each stored file or directory belongs.

CHGRP confirms changes made with a summary message that gives the count of files or directories whose group it has changed.

OPTIONS:

CHGRP takes a single option:

-R recursively changes the group membership of every child of the directory that you specify. NFT ignores soft links to subdirectories.

EXAMPLE:

If group BIG exists on the STORAGE machine and if you belong to it, you can assign stored file TEST2 to it with the line below; if these conditions are not met (e.g., group XXX) you get the error message shown (you get a "group name invalid" message on machines other than STORAGE).

```
R/Us: nft>chgrp big test2
Rtne: 1.0 1 entry changed ~/nfttest/test2
R/Us: nft>chgrp xxx test2
```

```
Rtne: 2.0 error. storage.llnl.gov: 451 could not  
get user id from registry. ~/nfttest/test2
```

(You can also change storage groups, even recursively, by using a dedicated tool called CHGRPSTG, which is explained and illustrated in EZSTORAGE (URL: <https://computing.llnl.gov/LCdocs/ezstorage>).

CHKROUTING (Report Routing Availability)

SYNTAX:

chkrouting [[*host*]:]*pathname*

ROLE:

Reveals whether or not job routing is currently available for PUTs from or GETs to the specified *pathname*. Since routing occurs between *pathname* and storage (see the [ROUTING](#) (page 77) section for a full explanation), it is never useful to insert a storage directory or a stored file name as the argument of **CHKROUTING**. Instead, use NFT's standard colon-based [syntax](#) (page 12) to specify a *pathname* on the client host (:) or on another nonstorage machine (*host*:).

When routing is available, NFT performs it automatically, so most users should never need **CHKROUTING**. The commands [ROUTING](#) (page 77) and [NOROUTING](#) (page 67) respectively enable and disable routing if necessary. Reasons why **CHKROUTING** might report "no job routing" include:

- (a) Routing is not supported for *pathname*'s underlying file system (ordinary NFS-mounted file systems, such as those for the common home directories, have no routing).
- (b) Routing has been disabled by a previous **NOROUTING** command during this NFT session.
- (c) The specified *pathname* contains at least one link as one of its directories or as its terminal file (NFT does not both follow links and perform routing).

CHKROUTING often seems to be an [immediate](#) (page 17) NFT command. But it must perform FTP I/O to parse *pathname* and to check symbolic links, so it is really a job command (and, as the example below shows, [VERBOSE 78](#) (page 86) will reveal its sequential job numbers).

EXAMPLE:

This dialog shows both local (:) and remote (up:) reporting cases, as well as **CHKROUTING**'s latent job numbers.

```
R/Us:  nft> chkrouting :/usr/tmp
Rtne:  no job routing
R/Us:  nft> chkrouting :/p/lscratchb
Rtne:  job routing
R/Us:  nft> chkrouting up:/p/gscratcha
Rtne:  no job routing
R/Us:  nft> verbose 78
R/Us:  nft> chkrouting :/p/lscratchb
Rtne:  job routing
21.0.  done /p/lscratchb
```

CHMOD (Change Permissions)

SYNTAX:

```
chmod [-R] rights [host]filelist
```

ROLE:

changes to *rights* the access rights or "mode" of the files or directories specified by *filelist*, a standard UNIX pathname or file filter, on the specified *host* (an NFT host-specifying prefix (page 12) that defaults to the STORAGE (not current client) machine.

DEFAULTS:

When you transfer a file with NFT, its default rights (mode) on the sink machine seldom agree with its original rights on the source machine. For example, NFT changes rights for files going to STORAGE using umask (octal subtraction) 027 on OCF but 077 on SCF. Hence, you usually need to invoke CHMOD to explicitly restore the original permissions after transfer if you want them to persist.

WARNING:

While the UNIX CHMOD utility lets you specify rights both symbolically (with a syntax such as g+w) and octally (by ORing, or adding, the octal numbers of the rights to assign), the NFT CHMOD command accepts the octal format only. Attempts to assign rights symbolically yield an error message. Consult EZFILES (URL: <https://computing.llnl.gov/LCdocs/ezfiles>) for a summary of the octal numbers associated with each combination of access rights that CHMOD accepts.

CHGRP (page 39) is a similar NFT command that changes file groups. You can use DIR (page 52) to reveal the current rights that each stored file or directory has.

CHMOD confirms changes made with a summary message that gives the count of files or directories whose rights have changed.

OPTIONS:

CHMOD takes a single option:

-R recursively changes the rights (mode) of every child of the directory that you specify. NFT ignores soft links to subdirectories.

EXAMPLE:

You can use octal mode 765 to assign a chosen set of rights to stored file TEST2, then confirm the assignment by using DIR.

```
R/Us: nft>chmod 765 test2
Rtne: 1.0 1 entry changed ~/nfttest/test2
R/Us: nft>dir test2
Rtne: -rwxrw-r-x 1 jfk jfk 6229 Aug16 14:23 test2
```

(You can also change storage rights, even recursively and symbolically, by using a dedicated tool called CHMODSTG, which is explained and illustrated in EZSTORAGE (URL: <https://computing.llnl.gov/LCdocs/ezstorage>).

CHOWN (Change Owners)

SYNTAX:

```
chown [-R] owner [host]filelist
```

ROLE:

(Storage only) changes to *owner* the official owner of the files or directories specified by *filelist*, a standard UNIX pathname or file filter. NFT reports "command not available on specified host" if you try to use CHOWN on files located anywhere except STORAGE. You need to use the *host* specifier *storage:* if and only if your current remote host is not STORAGE.

DEFAULTS:

When you store a file with NFT, its default owner is the same as that of the storage directory that receives it (not necessarily the owner it had on the donor machine). So changing a storage directory's owner will change the default owner of all files subsequently stored within it (but already stored files will retain their original owner, even if they are overwritten later). And storing a file owned by another user into your storage directory will change its ownership to you.

CHMOD (page 42) is a similar NFT command that changes file permissions, while CHGRP (page 39) changes file groups. You can use DIR (page 52) to reveal the current owner for each stored file or directory.

CHOWN confirms changes made with a summary message that gives the count of files or directories whose group it has changed.

OPTIONS:

CHOWN takes a single option:

-R recursively changes the owner of every child of the directory that you specify. NFT ignores soft links to subdirectories.

EXAMPLE:

Currently only privileged users (system administrators) can change the ownership of stored files with CHOWN.

CLOBBER (Enable File Overwriting)

SYNTAX:

clobber

ROLE:

Causes NFT to handle file-name conflicts by allowing an incoming file to overwrite any file of the same name in the receiving (usually the current working) directory on the target host. By default, NFT prevents such file overwriting and instead returns a warning when file-name conflicts occur (NOCLOBBER). So CLOBBER overrides NFT's default NOCLOBBER behavior.

CLOBBER and NOCLOBBER are mutually exclusive alternative settings for an NFT environment variable that preserves your choice of behavior until you overtly change it (or terminate your NFT client). Only by using separate NFT clients (not multiple sessions with one client) can you have two sets of environment variable settings for two sets of NFT jobs at once. See the "sessions (page 29)" section for details.

STATUS (page 83) reports your current choice of CLOBBER or NOCLOBBER settings. CLOBBER takes no options and returns no confirmation message.

EXAMPLE:

Trying to store (an updated version of) an already stored file with NFT is a typical situation where the default NOCLOBBER behavior needs to be changed with CLOBBER to allow the desired overwriting to occur, as shown here:

```
R/Us:  nft> put test2
Rtne:  12.0. error. cannot clobber existing sink ~/nfttest/test2
R/Us:  nft> clobber
R/Us:  nft> put test2
Rtne:  13.0. 95 bytes sent in 1.0 seconds (0.1Kbytes/s) test2
```

CLOG (Close Log File)

SYNTAX:

clog

ROLE:

Closes the NFT log file most recently opened with the LOG *pathname* command. QUITting your NFT client also closes its open log file, if any, but changing sessions does not. As the LOG section (page 62) explains, you can create more than one log file, but only one can be in use (open) at any time.

CLOG takes no options or arguments.

EXAMPLE:

NFT uses prefixes to reveal the source of every log-file line. For an explanation of these prefixes and a typical sample NFT log file, see the section on NFT Logging Techniques (keyword: log-examples (page 19)). Also, using input files changes the way NFT logs its interactions (keyword: file-interactions (page 22)), but using sessions does not (keyword: session-usage (page 29)).

CLOSE (Restore Remote Host)

SYNTAX:

close

ROLE:

After any OPEN (page 69), restores NFT's remote host to its default value (namely, STORAGE). As with OPEN, NFT sends no confirmation or reminder of which host is currently open after a CLOSE, and you must use STATUS to disclose the current remote host. OPEN (page 69) and CLOSE exist to supplement NFT's native host-specifying colon syntax (page 12) and to somewhat mimic the behaviour of FTP. But unlike FTP, NFT lets you use several OPENs sequentially without requiring a corresponding paired CLOSE after each one. CLOSE used with no preceding OPEN changes nothing and issues no error message.

EXAMPLE:

See the OPEN (page 69) section for an example of restoring NFT's remote host to STORAGE by using CLOSE following several OPEN commands.

CLR (Clear Completed Job Reports)

SYNTAX:

```
clr [[n[-m]] | [-opt]]
```

ROLE:

Clears (deletes the job-status record on) your most recent NFT request ("job") by default, or clears your specific job with unique job number *n* (an integer), or the range of jobs with numbers *n-m* inclusive (using a hyphen, not a comma, as separator), or all members of the job class specified by any one job-class option *opt* listed below.

You can only clear job-status information for completed NFT jobs (those that either ran successfully to completion, ended with an error, or were aborted by you). Attempting to clear incomplete jobs or jobs whose records were already cleared has no effect, and will often return an error message such as "no completed jobs in range." You can use **ABT** to abort an incomplete job before clearing its record, and you can use **RPT** to discover the job numbers of your completed NFT jobs.

By default, jobs are remembered by the NFT server for up to 4 days after they complete. This can be a convenience for reviewing your work, but frequent NFT users may find long accumulated **RPT** status reports confusing and may want to prune no-longer-interesting job records from them by astute use of **CLR**.

RPT (page 78) (report) and **ABT** (page 46) (abort) are NFT commands closely related to **CLR**. The **NFT STATUS** (page 83) command reports on your environment-variable settings, not on your file-transfer jobs.

OPTIONS:

CLR accepts several job-class options *opt* to specify which set of your NFT job records you want cleared. But **CLR** expects you to use these options ONE at a time: combined options do not yield combined job classes to be cleared. Possible stand-alone job-classes on which you can apply **CLR** include:

- a all jobs that NFT remembers, regardless of state. [If you use multiple **NFT sessions** (page 29), a rare practice, then -a selects only jobs in the current session, NOT in all the sessions that you have created, and the other job-class options behave likewise.]
- c complete jobs, that have successfully or unsuccessfully completed running.
- o okay jobs, those that have successfully completed running.
- e error jobs, that have unsuccessfully completed running.
- k aborted jobs, those that were terminated by the user with **ABT**.

EXAMPLE:

CLR returns no confirmation of success when you delete one or more job-status records, just the usual prompt for next input:

```
User: clr 4  
Rtne: nft>
```

CP (Copy/Transfer Files)

SYNTAX:

```
cp [-dR] [host]sourcepath [host][sinkpath]
```

```
cp [-dR] [host]{f11,f12,...} [host][{flx,fly,...}]
```

ROLE:

Transfers (copies) the file specified by *sourcepath* from the first host into the file specified by *sinkpath* on the second host. Alternatively, CP transfers (copies) each file in the ordered list {*f11,f12...*} on the first host into the corresponding file in the second ordered list {*flx,fly...*} on the second host (each file list must have the same number of members). Here

- host* is the NFT host-specifying prefix that defaults to the STORAGE (not current client) machine, as explained above (keyword: prefix (page 12)).
- sourcepath* and *sinkpath* are standard UNIX pathnames that default to the current working directory on each host (specified by LCD (page 60) or CD (page 37)). You can use standard UNIX file filters in *sourcepath* (to copy many files with one command) if *sinkpath* is a directory. File filters are never allowed in *sinkpath* itself.
- f11,f12,...* and *flx,fly,...* must be surrounded by the braces { } shown above. If both lists are present they must have an equal number of members and no file filters. If you omit the second list you can include file filters among the entries in the first.

DEFAULTS:

CP always transfers files using FTP binary mode (you must run FTP itself, not NFT, for ASCII-mode file transfers). By default, NFT does NOT overwrite existing files with incoming files of the same name (NO CLOBBER), but you can enable overwrites with the CLOBBER (page 44) command. CP commands that include file filters, to transfer many files at once, are always processed asynchronously (subordinate jobs run in parallel, in any order) regardless of your current SYNC/ASYNC (page 35) setting, so never use filters with CP if arrival order is crucial.

Because NFT uses overt host prefixes to specify the source and sink machines for a file copy (not login, as with FTP), you can use CP to transfer files between two machines even when you are running the NFT client on a third machine (third-party transfers, see example below).

NFT's GET (page 56) and PUT (page 71) commands behave like CP but by default only work with the LC STORAGE system, not between any two hosts. (You can "generalize" them if you use OPEN (page 69) first.) Hence, CP is really NFT's clearest equivalent of the well-known FTP general GET and PUT commands.

OPTIONS:

CP accepts two options:

- d destroys each source file after the transfer (copy) is successful.
- R recursively copies source subdirectories (however, soft links to subdirectories are not followed).

EXAMPLE:

(1) Transfer (copy) file t1 from the local machine (:) into file /usr/tmp/t2 on YANA.

```
nft>cp :t1 yana:/usr/tmp/t2
1.0. 95 bytes received in 0.1 seconds
(0.7 Kbytes/s) from /g/g0/jfk/t1 to /usr/tmp/t2
1.0 1 entry copied /g/g0/jfk/t1
```

(2) Transfer (copy) file /usr/tmp/t3 from YANA into file /usr/tmp/t6 on ATLAS, even if running NFT on a third machine.

```
nft>cp yana:/usr/tmp/t3 atlas:/usr/tmp/t6
3.0. 98 bytes received in 0.4 seconds
(0.2 Kbytes/s) from /usr/tmp/t3 to /usr/tmp/t6
3.0 1 entry copied /g/g0/jfk/t3
```

(3) Use the file filter * to transfer (copy) all files whose names begin with LOG from YANA to ATLAS. Note that these files did not copy in numerical order because NFT processes such multiple-file jobs asynchronously (in parallel), even if SYNC mode is enabled, and that the final transaction summary (here 11.0) has the *lowest* job number of the jobs reported.

```
nft>cp yana:/usr/tmp/log* atlas:/usr/tmp
12.0. 914 bytes received in 0.1 seconds (7.4 Kbytes/s)
from /usr/tmp/log1 to /usr/tmp/log1
14.0. 288 bytes received in 0.3 seconds (0.9 Kbytes/s)
from /usr/tmp/log3 to /usr/tmp/log3
13.0. 178 bytes received in 0.1 seconds (1.9 Kbytes/s)
from /usr/tmp/log2 to /usr/tmp/log2
11.0. 3 entries copied (aggregate 3.4 Kbytes/s) /usr/tmp/log*
```

DELETE (Remove Files)

SYNTAX:

```
del[ete] [-R] [host]pathname
```

```
del[ete] [-R] [host]{file1,file2,...}
```

ROLE:

Deletes the file specified by *pathname* from the specified *host*. Alternatively, DELETE removes each file in the ordered list {*file1,file2...*} from the specified host. Here

host is the NFT host-specifying prefix that defaults to the STORAGE (not current client) machine, as explained above (keyword: prefix (page 12)). If you precede DELETE with an OPEN (page 69) command, then *host* defaults to the machine that OPEN specified.

pathname is a standard UNIX pathname, often a file in the current working directory (specified by CD (page 37)). To delete many files with one command, use a standard UNIX file filter at the end of *pathname*. At LC, HPSS interprets DELETE * to remove all files from *only* your current working directory, *not* from any of its child directories (this is an important difference from FTP's MDELETE *, which behaves recursively in STORAGE).

file1,file2,... must be surrounded by the braces {} shown above. File filters are allowed in any list member.

Remember that by default deletions occur among your stored files, not your local files (on the client machine), which you must overtly specify with the colon (:) prefix. To delete (empty) directories, use RMDIR (page 76) instead of DELETE. NFT's DIR (page 52) command lists your files. DELETE reports each single file it removes, but not multiples (from filter use). NFT has no separate MDELETE command (but see -R below).

OPTIONS:

DELETE accepts one option:

-R recursively deletes all subdirectories and their contents. NFT ignores soft links to subdirectories. (NFT's DELETE and RMDIR commands are equivalent when you invoke the -R option.)

EXAMPLE:

To DELETE local file test8 you must use NFT's colon (:) prefix, or else NFT will try to delete a file of that name from your current STORAGE working directory instead:

```
nft>delete test8
8.0. error. storage: 562
No such file or directory ~/test8
nft>delete :test8
9.0 1 entry deleted /g/g0/jfk/test8
```

DIR (List Directory Contents, Long)

SYNTAX:

dir [-opts] [host][pathname]

ROLE:

Lists the contents of a directory in a long (detailed) format, with entries alphabetical by file (or subdirectory) name in the ASCII collating sequence that puts symbols first, then uppercase letters, then lowercase letters. Here

- host* is the NFT host-specifying prefix that defaults to the STORAGE (not current client) machine, as explained above (keyword: prefix (page 12)). If you precede DIR with an OPEN (page 69) command, then *host* defaults to the machine that OPEN specified.
- pathname* is a standard UNIX path that defaults to the current working directory.
- opts* control the format of DIR's display, but only those options supported by the FTP daemon on the target host actually work, while others will fail, usually with an error message of the form

nnn.0 error: Syntax error: Invalid command options specified
Different FTP daemons support different sets of display *opts* and those not supported by LC's STORAGE machine(s) are noted in the option list below.

LS (page 63) is a similar NFT command with different default output.

OPTIONS:

- a lists all files, including dot (.), dot-dot (..), and the others beginning with a period.
- b displays nonprintable characters in the octal *\ddd* notation.
- c (not supported by STORAGE) sorts the listing by time of last change.
- d lists the directory entry itself (i.e., the single entry for .), not the contents of the directory.
- e (not supported by STORAGE) lists the security level (a digit).
- f (not supported by STORAGE) forces each argument to be treated as a directory.
- g (not supported by STORAGE) includes group ownership in the listing.
- h lists each file's class of service (COS) integer (in the third column, instead of its owner). See the SETCOS section of the HPSS Manual (URL: <https://computing.llnl.gov/LCdocs/hpss/index.jsp?show=s2.1.4>) for details about COS policy issues. NFT's own SETCOS command (page 81) may be used to specify COS before you store a file using NFT.
- i (not supported by STORAGE) prints the i-node number before each list entry.

- l lists permissions, owner, group, and date with each list entry.
- n lists the owner's UID and group's GID numbers, rather than the associated character strings.
- o (not supported by STORAGE) includes file ownership in the listing.
- p (not supported by STORAGE) displays each directory name with an appended slash.
- q (not supported by STORAGE) prints nondisplayable characters in file names as a question mark (?).
- r (not supported by STORAGE) reverses the (default alphabetical) order of the listing.
- s (not supported by STORAGE) reports the file size in sectors (instead of the default of blocks).
- t (not supported by STORAGE) sorts the listing by time last modified (most recent first).
- u (not supported by STORAGE) sorts the listing by time last accessed (most recent first).
- A (not supported by STORAGE) same as option -a except that dot (.) and dot-dot (..) are not listed.
- F appends to each directory name a slash (/), to each executable file an asterisk (*), and to each soft link an at-sign (@).
- L (not supported by STORAGE) lists the target of each symbolic link rather than the link itself.
- P (not supported by STORAGE) lists the account identifier with each entry.
- R provides a recursive listing of the contents of all subdirectories too (may run very slowly). NFT ignores soft links to subdirectories.

EXAMPLE:

A typical default (no *-opts* used) response to NFT's DIR command has this form (the meaning of each field is indicated below it), with a similar line for each file reported:

```
-rwxr-xr-x      2      jfk  doc   2048 Sep30 16:04 test1
[permission  links  owner group size  date      name]
```

DUALCOPY (Store Dual Copies of Files)

SYNTAX:

dualcopy

ROLE:

Causes NFT to write dual copies of Mission Critical files to HPSS Storage.

Newly written archival data is currently stored in HPSS via one of five classes of service (COS), informally called Small, Medium, Large, Jumbo, and Mission Critical. By default, files of Small and Medium COS are dual copied. (This is subject to change without notice.) Files of Large and Jumbo COS are not dual copied because dual copy is cost prohibitive.

Utilization of the Mission Critical COS is monitored, and users should limit their use of the service for truly irreplaceable data.

You can check the current DUALCOPY or NODUALCOPY setting for your NFT session with STATUS (page 83). You can discover the COS with which already stored files have been stored by using NFT's DIR (page 52) command with the -h option (COS then appears in the third column in DIR's report). Use the NODUALCOPY (page 66) command to store a file to the HPSS default COS (which may or may not be dual-copied based upon file size). See also the detailed COS discussion in LC's HPSS Reference Manual (URL: <https://computing.llnl.gov/LCdocs/hpss/index.jsp?show=s1.2.1>).

EXAMPLE:

To select the dual-copy COS to store file1 using the NFT interface:

```
R/Us: nft> dualcopy
R/Us: nft> put file1
```

To then store file2 to the HPSS default COS:

```
R/Us: nft> nodualcopy
R/Us: nft> put file2
R/Us: nft> quit
```

ENDGR (End Asynchronous Group)

SYNTAX:

endgr

ROLE:

Ends the scope of an asynchronous group of NFT commands (jobs). See the GROUP (page 58) command for usage and examples.

ENDGR has no options and returns no confirmation.

GET (Retrieve Stored Files)

SYNTAX:

```
get [-dR] sourcepath [sinkpath]
```

```
get [-dR] {file1,file2,...} [{filex,filey,...}]
```

ROLE:

(Storage defaulted) transfers (copies) the file specified by *sourcepath* from the LC STORAGE system into the file specified by *sinkpath* on the local machine (where you are running your NFT client). Alternatively, GET transfers (copies) each file in the ordered list {*file1,file2...*} on the STORAGE system into the corresponding file in the second ordered list {*filex,filey...*} on the local machine (each file list must have the same number of members). If you precede GET with an OPEN (page 69) command, you can also retrieve files from a specified host other than STORAGE. Here

sourcepath and *sinkpath* are standard UNIX pathnames. You can use standard UNIX file filters in *sourcepath* (to transfer many files with one command) if *sinkpath* is a directory. File filters are never allowed in *sinkpath* itself. If *sourcepath* is just a simple file name, omitting *sinkpath* causes NFT to put the retrieved file into the current local working directory (which you can specify with LCD (page 60)). If *sourcepath* is a longer pathname, then omitting *sinkpath* causes NFT to try to duplicate *sourcepath* on the local machine, but if the appropriate subdirectories do not already exist the transfer fails with an error.

file1,file2,... and *filex,filey,...* must be surrounded by the braces {} shown above. If both lists are present they must have an equal number of members and no file filters. If you omit the second list you can include file filters among the entries in the first.

DEFAULTS:

Unlike NFT's CP (copy) command and FTP's general GET command, NFT's GET transfers files by default only from LC's STORAGE system and only to the local (client) machine. Hence, no host-specifying prefixes are allowed when you use GET, and third-party GETs between remote machines are not supported. You can, however, use OPEN (page 69) to reset NFT's remote host and then GET files from a nonSTORAGE source machine.

GET always transfers files using FTP binary mode (you must run FTP itself, not NFT, for ASCII-mode file transfers). By default, NFT does NOT overwrite existing files with incoming files of the same name (NOLOBBER), but you can enable overwrites with the CLOBBER (page 44) command. GET commands that include file filters, to transfer many files at once, are always processed asynchronously (subordinate jobs run in parallel, in any order) regardless of your current SYNC/ASYNC (page 35) setting, so never use filters with GET if arrival order is crucial.

Use CP (page 48) (copy) for all general file transfers with NFT, and use PUT (page 71) (transfer to STORAGE) to move files in the opposite direction from GET.

OPTIONS:

- d destroys the source file (on STORAGE) after transfer to the local machine is successful.
- R recursively copies all children in subdirectories of *sourcepath*, if it is a directory. NFT ignores soft links to subdirectories.

EXAMPLE:

By default NFT prevents overwriting an existing file (here t4) whenever you use GET to retrieve (here t6) from STORAGE. You can retrieve to a nonconflicting file name (as shown here) or use NFT's CLOBBER command to enable overwriting.

```
nft>get t6 t4
5.0. error. Cannot clobber existing
sink /g/g0/jfk/t4

nft>get t6
6.0. 95 bytes received in 1.8 seconds
(0.1 Kbytes/s) from ~/t6 to /g/g0/jfk/t6
1 entry copied ~/t6
```

NOTE:

If you want to GET a few files from inside a stored TAR-format archive file *without* first GETting the whole (large) archive to your local machine, use HTAR instead of NFT. Using HTAR's -F option, you can even extract files from within a remote TAR-format archive file on *any* LC machine that has a preauthenticated FTP server. Consult the [HTAR Reference Manual](https://computing.llnl.gov/LCdocs/htar) (URL: <https://computing.llnl.gov/LCdocs/htar>) for instructions and examples.

GROUP (Begin Asynchronous Group)

SYNTAX:

group

ROLE:

Begins the scope of an asynchronous group of NFT commands (that will be closed by an ENDGR command).

All commands between a GROUP/ENDGR pair are executed asynchronously, in parallel, as resources become available, even though the default for NFT command execution is synchronous (serial). The whole group is scheduled serially, as usual, so that all previously entered synchronous commands will execute before any commands within the group execute.

Users interested in running NFT asynchronously should consult the SYNC (page 84) command section as well as the command-sequencing (page 16) discussion for a comparative analysis of the three ways that NFT supports parallel jobs.

OPTIONS:

GROUP has no options, but you must use ENDGR to close a command group started with GROUP.

EXAMPLE:

While remaining in SYNC mode, you can specify a subset of commands to process in parallel (asynchronously among themselves) by preceding them with GROUP and following them with ENDGR. To force all subsequent commands to wait until everything within such a GROUP has executed, use BLOCK after ENDGR and before the next command. For example, the following NFT command sequence first creates directory TEST3, then ASYNCHRONOUSLY copies three files into that directory (since arrival order is often unimportant), then, and only after all three have arrived (BLOCK), issues a report (RPT).

```
mkdir test3
cd test3
group
  cp yana:file1 :file1
  cp atlas:file2 :file2
  cp lucy:file3 :file3
endgr
block
rpt -a
```

HELP (Describe NFT Commands)

SYNTAX:

help [*command*]

ROLE:

Lists the available NFT command names, in broad functional groups, or, if you supply a specific command's name, describes that command. Here

command identifies the NFT option on which you want specific help. NFT returns a MAN-page-like syntax summary briefly covering uses and suboptions, followed by a few examples. Help messages for commands with many suboptions, such as DIR, are abbreviated.

HELP takes no options and requests for help about nonexistent NFT commands return a syntax error.

LCD (Change Local Working Directory)

SYNTAX:

```
lcd [pathname]
```

ROLE:

Changes the current working directory on the local machine (where you are running your NFT client) to the specified *pathname*, which is a standard UNIX path that defaults to your home directory.

Use CD (page 37) to change working directories generally, PWD: (page 73) to discover the name of your current local working directory, and CDUP (page 38) to move up one directory level. These commands are somewhat redundant, so that the following are exactly equivalent, and all require using NFT's own prefix syntax (page 12) for hosts.

```
CD :..
CDUP :
LCD ..
```

LCD takes no options, and it confirms each requested directory change.

WARNING:

If you specify a nonexistent pathname for the UNIX CD command, you get an immediate error message and no directory change. If you specify a nonexistent pathname for NFT's LCD command, however, you get the usual confirmation message before receiving the string "no such file or directory." And subsequent use of PWD will also report the nonexistent directory, without complaint. Of course attempts to then use DIR or actual file-transfer commands will fail. You must overtly reset the local working directory with another use of LCD to a real location to overcome this error and continue transferring files.

EXAMPLE:

In response to each LCD command, NFT reports both the machine involved and the new working directory (wd).

```
R/Us:  nft> lcd /usr/tmp/stuff
Rtne:   local wd is /usr/tmp/stuff      [on client machine]
```

LN (Create a Link)

SYNTAX:

```
ln -s sourcepath linkname
```

ROLE:

(Storage only) creates a new directory entry called *linkname* that points to the file or directory specified in *sourcepath* on the STORAGE system. Here

- s specifies a soft link (hard links on the STORAGE system are not allowed, and using LN without this option returns an error message that says so).
- sourcepath* is a standard UNIX path that leads to the original file or directory, to which you want to link.
- linkname* is a UNIX path that leads to where the link (pointer) will be placed. If *linkname* ends in a directory, then the children of *sourcepath* will be duplicated there as links back to their counterparts. If *linkname* ends in a file, then the new link will be given the file name at the end of that path.

NFT's LN command works only on the STORAGE system, not on any other hosts. (Technically, using OPEN (page 69) should allow you to use LN on the nonSTORAGE host you have OPENed. In practice, however, the only LC FTP daemon that supports remote links is the one on STORAGE, both open and secure.) But you can use the escape syntax !LN to issue a link command that executes on the local (client) machine (though not on other remote machines).

EXAMPLE:

To create a link (pointer) in your storage directory NFTTEST called T4 that points back to actual file T1 in your home storage directory, use the LN command shown here.

```
nft>ln -s ~/t1 ~/nfttest/t4
3.0. 1 entry linked ~/t1
```

LOG (Open Log File)

SYNTAX:

`log pathname`

ROLE:

Begins recording in a log file all your input to and output messages from NFT as it runs. Here

pathname is usually just the name of a file (e.g. nftlog) that you want NFT to create in the current working directory (where NFT was started) on the machine where you are running the NFT client. If you supply an absolute pathname (such as ~/projects/nftlog or /usr/tmp/testdir5/log5) then NFT creates the log file in the other directory that you specify.

Logging of all NFT input and output continues until you issue the CLOG (page 45) (close log) command or until you QUIT your current NFT session, whichever comes first.

Issuing a second LOG command with a *different* pathname will (1) create a second, independent log file, (2) stop recording messages in the first log file, and (3) start recording (subsequent) messages in the second log file. Issuing a second LOG command with the *same* pathname will insert an updated time stamp (comment) in the open log file and then simply append all new messages to the old ones. If a file named *pathname* already exists when you first issue a LOG command, NFT opens it, appends a current time stamp, and places all your new messages at the end of that file. Thus you can jump between multiple log files during an NFT session, channeling messages to one and then another, just by using LOG with each file's name whenever you want to change files.

NFT does not collect a large buffer of messages before logging them; instead, messages are flushed to the log file after every carriage return. Logging starts immediately, and the first line recorded in your log file (after a comment-line date stamp) will be the LOG *pathname* command that requested the file. CLOG will be the last line recorded. As long as you do not change the (default) TERM mode setting with a NOTERM (page 68) command, all your input and output will continue to display at your terminal while logging occurs.

DEFAULTS:

LOG takes no options and there is no default log-file name.

EXAMPLE:

NFT uses prefixes to reveal the source of every log-file line. For an explanation of these prefixes and a typical sample NFT log file, see the section on NFT Logging Techniques (keyword: log-examples (page 19)). Also, using input files changes the way NFT logs its interactions (keyword: file-interactions (page 22)), but using sessions does not (keyword: session-usage (page 29)).

LS (List Directory Contents, Short)

SYNTAX:

```
ls [-opts] [host][pathname]
```

ROLE:

Lists the contents of a directory in a short format (names only by default, unless you add detail with options), with entries alphabetical by file (or subdirectory) name in the ASCII collating sequence that puts symbols first, then uppercase letters, then lowercase letters. Here

- host* is the NFT host-specifying prefix that defaults to the STORAGE (not current client) machine, as explained above (keyword: prefix (page 12)). If you precede LS with an OPEN (page 69) command, then *host* defaults to the machine that OPEN specified.
- pathname* is a standard UNIX path that defaults to the current working directory.
- opts* control the format of LS's display, but only those options supported by the FTP daemon on the target host actually work, while others will fail, usually with an error message of the form

```
nnn.0. error: Syntax error: Invalid command options specified
Different FTP daemons support different sets of display opts and those not supported by
LC's STORAGE machine(s) are noted in the option list below.
```

DIR (page 52) is a similar NFT command with different default output.

OPTIONS:

- a lists all files, including dot (.), dot-dot (..), and the others beginning with a period.
- b displays nonprintable characters in the octal *\ddd* notation.
- c (not supported by STORAGE) sorts the listing by time of last change.
- d lists the directory entry itself (i.e., the single entry for .), not the contents of the directory.
- e (not supported by STORAGE) lists the security level (a digit).
- f (not supported by STORAGE) forces each argument to be treated as a directory.
- g (not supported by STORAGE) includes group ownership in the listing.
- h lists each file's class of service (COS) identifier (instead of its owner). See the SETCOS section of the HPSS Manual (URL: <https://computing.llnl.gov/LCdocs/hpss>) for class of service details.
- i (not supported by STORAGE) prints the i-node number before each list entry.
- l lists permissions, owner, group, and date with each list entry.

- m (not supported by STORAGE) lists file names horizontally, each separated by one comma and one blank space from the next.
- n lists the owner's UID and group's GID numbers, rather than the associated character strings.
- o (not supported by STORAGE) includes file ownership in the listing.
- p (not supported by STORAGE) displays each directory name with an appended slash.
- q (not supported by STORAGE) prints nondisplayable characters in file names as a question mark (?).
- r (not supported by STORAGE) reverses the (default alphabetical) order of the listing.
- s (not supported by STORAGE) reports the file size in sectors (instead of the default of blocks).
- t (not supported by STORAGE) sorts the listing by time last modified (most recent first).
- u (not supported by STORAGE) sorts the listing by time last accessed (most recent first).
- x (not supported by STORAGE) lists file names horizontally in columns.
- A (not supported by STORAGE) same as option -a except that dot (.) and dot-dot (..) are not listed.
- C (not supported by STORAGE) formats the listing in multiple columns.
- F appends to each directory name a slash (/), to each executable file an asterisk (*), and to each soft link an at-sign (@).
- L (not supported by STORAGE) lists the target of each symbolic link rather than the link itself.
- P (not supported by STORAGE) lists the account identifier with each entry.
- R provides a recursive listing of the contents of all subdirectories too (may run very slowly). If you use -R together with any multicolumn option (-m, -x, -C), the multicolumn option is ignored. NFT ignores soft links to subdirectories.

EXAMPLE:

A typical default (no *-opts* used) response to NFT's LS command is a one-column list of file and directory names. See the [DIR](#) (page 52) section above for an annotated explanation of the more elaborate output that using LS with options can yield.

MKDIR (Make Directories)

SYNTAX:

mkdir [*host*]*pathname*

ROLE:

Creates the specified directory on the specified host. Here

host is the NFT host-specifying prefix that defaults to the STORAGE (not current client) machine, as explained above (keyword: prefix (page 12)). If you precede MKDIR with an OPEN (page 69) command, then *host* defaults to the machine that OPEN specified.

pathname specifies where to put the new directory. If this is a simple directory name, then NFT makes the directory in the current working directory (specified by CD (page 37)). If this is a relative or absolute pathname all of whose other directories already exist, then NFT makes the new directory as a child of the last directory in the path.

MKDIR takes no options and (at default reporting levels) does not confirm the creation of the directory you requested. However, NFT's DIR (page 52) command lists your files and directories for confirmation.

NOCLOBBER (Disable File Overwriting)

SYNTAX:

noclobber

ROLE:

Causes NFT to handle file-name conflicts by preventing an incoming file from overwriting any file of the same name in the receiving (current working) directory on the target host. Instead NFT returns a warning when file-name conflicts occur. NOCLOBBER is NFT's default behavior, so you would normally need to use the overt NOCLOBBER command only to reverse your previous use of the CLOBBER command, which enables file overwriting.

CLOBBER and NOCLOBBER are mutually exclusive alternative settings for an NFT environment variable that preserves your choice of behavior until you overtly change it (or terminate your NFT client). Only by using separate NFT clients (not multiple sessions with one client) can you have two sets of environment variable settings for two sets of NFT jobs at once. See the "[sessions](#) (page 29)" section for details.

[STATUS](#) (page 83) reports your current choice of CLOBBER or NOCLOBBER settings. NOCLOBBER takes no options and returns no confirmation message.

EXAMPLE:

NOCLOBBER (no overwriting) is NFT's default behavior; consult the [CLOBBER](#) (page 44) section for an example of how to reverse this behavior.

NODUALCOPY (Undo Dual Copy)

SYNTAX:

nodualcopy

ROLE:

Causes NFT to write files to HPSS storage in their default class of service (COS).

You can check the current DUALCOPY or NODUALCOPY setting for your NFT session with [STATUS](#) (page 83). You can discover the COS with which already stored files have been stored by using NFT's [DIR](#) (page 52) command with the -h option (COS then appears in the third column in DIR's report). Use the [DUALCOPY](#) (page 54) command to store a Mission Critical file to the HPSS. See also the detailed COS discussion in LC's [HPSS Reference Manual](#) (URL: <https://computing.llnl.gov/LCdocs/hpss/index.jsp?show=s1.2.1>).

NOROUTING (Disable Routing)

SYNTAX:

norouting

ROLE:

Disables the normally automatic ROUTING (page 77) of NFT file transfers (to or from storage) from compute nodes to login nodes to take advantage of the latter's jumbo-frame network connections.

On many LC production machines, the login nodes have jumbo-frame network connections to storage, enabling large data blocks to transfer more quickly than with standard, smaller frames. Compute nodes lack these jumbo-frame connections because of their higher cost. When appropriate, NFT now routes file-transfer requests (to or from storage) from their originating compute nodes to the login nodes on the same cluster to take advantage of available faster jumbo-frame transfer rates.

NOROUTING disables such routing (usually needed only for timing or other special tests). Use CHKROUTING (page 41) to see if routing is currently enabled or disabled (or otherwise not available) between a specified *pathname* and storage. STATUS also reveals whether routing is on (yes) or off (no). See the ROUTING (page 77) section below for more details on where routing is available and when it is beneficial.

EXAMPLE:

Routing is on by default, but you can disable it with NOROUTING if you wish.

```
R/Us: nft> norouting
R/Us: nft> status
Rtne: Connected to storage as jfk.
      . . .
      Routing:      no
```

NOTERM (Disable Terminal Output)

SYNTAX:

noterm

ROLE:

Causes NFT to prevent terminal display of all output from its executed commands, and to stop offering its interactive nft> prompt as well. So NOTERM overrides NFT's default TERM behavior. If you have enabled an NFT log file with the LOG (page 62) command, however, all normal output and prompts continue to collect in that file even after you use NOTERM.

TERM and NOTERM are mutually exclusive alternative settings for an NFT environment variable that preserves your choice of behavior until you overtly change it (or terminate your NFT client). Only by using separate NFT clients (not multiple sessions with one client) can you have two sets of environment variable settings for two sets of NFT jobs at once. See the "sessions (page 29)" section for details.

STATUS (page 83) normally reports your NFT environment variable settings, but of course NOTERM hides all STATUS output as well as other output. NOTERM takes no options and returns no confirmation message, but the absence of NFT prompts betrays its use.

OPEN (Change Remote Host)

SYNTAX:

open *host*

ROLE:

Changes from STORAGE (the default) to *host* (which you must specify) the remote host with which NFT interacts and on which it reports. Actually, OPEN itself is a local command that only changes the remote host reported by the STATUS command in its "connected to *host* as *yyy*" output. Only after you attempt a specific interaction with that host (such as CD or PUT) does the hidden NFT server try to connect to it (persistently). For OPEN, *host* must be a domain name (e.g., ATLAS or ATLAS.LLNL.GOV), not a numerical IP address.

When using OPEN remember that:

- (1) NFT sends no clarification or reminder of which host is currently open, and no altered prompt reveals the current target (unlike FTP). You must use STATUS to disclose the current remote host.
- (2) ONLY machines offering NFT clients can really be OPENed for NFT file transfers. You can request OPENS of other hosts (such as FIS) without receiving any immediate error message, and even with a successful reset of the STATUS report (see example below). But attempts to actually change directories, move files, etc., will fail, yielding "invalid source" or "invalid sink" messages.

OPEN exists to supplement NFT's native host-specifying colon syntax (page 12) and to somewhat mimic the behavior of FTP. Open lets you use PUT and GET (but not the other "storage-defaulted" commands) with remote hosts other than the LC storage system (because OPEN resets NFT's default remote host). Unlike FTP, however, NFT lets you use subsequent OPENS to repeatedly reset the current remote host without requiring a paired CLOSE after each OPEN.

EXAMPLE:

Using OPEN lets you

- (1) change NFT's remote host from STORAGE to ATLAS,
- (2) confirm the change with STATUS, and
- (3) successfully transfer a file to ATLAS with PUT, which is reserved for storage-only use if you omit the preceding OPEN. But note that you can also
- (4) apparently change remote hosts to FIS, and
- (5) apparently confirm that change with STATUS even though
- (6) all actual file-transfer attempts fail with an error message (because FIS is not a host known to the NFT server).
- (7) CLOSE restores NFT's remote host to STORAGE.

```
nft>open atlas          ---(1)
nft>status              ---(2)
  connected to atlas as jfk...
nft>put test3 /usr/tmp/test3a  ---(3)
  1.0 95 bytes sent in 0.1 seconds
  (0.7 Kbytes/s) from /g/g0/jfk/test3
  to /usr/tmp/test3a
nft>open fis           ---(4)
nft>status              ---(5)
  connected to fis as jfk...
```

```
nft>put test3 ---(6)
  2.0 error Invalid host
  specified. ~/test3
nft>close ---(7)
nft>status
  connected to storage as jfk...
```

PUT (Store Local Files)

SYNTAX:

```
put [-dR] sourcepath [sinkpath]
```

```
put [-dR] {file1,file2,...} [{filex,filey,...}]
```

ROLE:

(Storage defaulted) transfers (copies) the file specified by *sourcepath* from the local machine (where you are running your NFT client) into the file specified by *sinkpath* on the LC STORAGE system. Alternatively, PUT transfers (copies) each file in the ordered list {*file1,file2...*} on the local machine into the corresponding file in the second ordered list {*filex,filey...*} on the STORAGE system (each file list must have the same number of members). If you precede PUT with an OPEN (page 69) command, you can also deliver files to a specified host other than STORAGE. Here

sourcepath and *sinkpath* are standard UNIX pathnames. You can use standard UNIX file filters in *sourcepath* (to transfer many files with one command) if *sinkpath* is a directory. File filters are never allowed in *sinkpath* itself. If *sourcepath* is just a simple file name, omitting *sinkpath* causes NFT to put the stored file into the current STORAGE working directory (which you can specify with CD (page 37)). If *sourcepath* is a longer pathname, then omitting *sinkpath* causes NFT to try to duplicate *sourcepath* on the STORAGE system, but if the appropriate subdirectories do not already exist the transfer fails with an error.

file1,file2,... and *filex,filey,...* must be surrounded by the braces { } shown above. If both lists are present they must have an equal number of members and no file filters. If you omit the second list you can include file filters among the entries in the first.

DEFAULTS:

Unlike NFT's CP (copy) command and FTP's general PUT command, NFT's PUT transfers files by default only from the local (client) machine and only to the STORAGE system. Hence, no host-specifying prefixes are allowed when you use PUT, and third-party PUTs between remote machines are not supported. You can, however, use OPEN (page 69) to reset NFT's remote host and then PUT files to a nonSTORAGE target machine.

PUT always transfers files using FTP binary mode (you must run FTP itself, not NFT, for ASCII-mode file transfers). By default, NFT does NOT overwrite existing files with incoming files of the same name (NO CLOBBER), but you can enable overwrites with the CLOBBER (page 44) command. PUT commands that include file filters, to transfer many files at once, are always processed asynchronously (subordinate jobs run in parallel, in any order) regardless of your current SYNC/ASYNCR (page 35) setting, so never use filters with PUT if arrival order is crucial.

Use CP (page 48) (copy) for all general file transfers with NFT, and use GET (page 56) (transfer from STORAGE) to move files in the opposite direction from PUT.

OPTIONS:

-d destroys the source file (on the local machine) after transfer to the STORAGE system is successful.

-R recursively transfers (copies) files in subdirectories if *sourcepath* is a directory. NFT ignores soft links to subdirectories.

EXAMPLE:

To store local file t1 into a file called t2 in the current STORAGE working directory, use this command:

```
nft>put t1 t2
4.0. 95 bytes sent in 1.0 seconds
(0.1 Kbytes.s) from /g/g0/jfk/t1 to ~/t2
1 entry copied /g/g0/jfk/t1
```

NOTES:

(a) If you want to PUT a large TAR-format archive file into storage but don't have the space (or time) to build it first on your local machine, use HTAR instead of NFT (HTAR will actually build the archive *directly in storage* as member files arrive). With HTAR's -F option, you can similarly transfer files directly into a *remote* TAR-format archive file on any LC machine that has a preauthenticated FTP server. Consult the HTAR Reference Manual (URL: <https://computing.llnl.gov/LCdocs/htar>) for details and examples.

(b) Since a common PUT error is to omit CLOBBER (page 44) if you wish to overwrite a stored file, NFT now explicitly distinguishes between "no clobber failures" and other failures when it reports user errors.

PWD (Print Working Directory)

SYNTAX:

```
pwd [-a] | [host]
```

ROLE:

Prints (reports the name of) the current working directory on the specified *host*, where *host* is the NFT host-specifying prefix that defaults to the STORAGE (not current client) machine, as explained above (keyword: prefix (page 12)). If you precede PWD with an OPEN (page 69) command, then *host* defaults to the machine that OPEN specified.

So by default PWD used with no arguments reports your current STORAGE directory. Use CD (page 37) to change directories generally, CDUP (page 38) to move one directory level up, and LCD (page 60) to change local (client-machine) directories. These commands are somewhat redundant, so that the following are exactly equivalent, and all require using NFT's own prefix syntax for hosts.

```
CD :..
CDUP :
LCD ..
```

WARNING:

If you specify a nonexistent directory with CD or LCD, then subsequent use of PWD will report that nonexistent directory with no complaint, even though attempts to use DIR or actual file-transfer commands will fail. You must specify a real location with another CD or LCD command to overcome this error.

OPTIONS:

-a reports the current working directory on both the local (NFT-client) and the STORAGE machines. This replaces a host specification.

EXAMPLE:

Remember that PWD reports on the STORAGE machine by default.

```
R/Us: nft> pwd
Rtne:  remote wd is ~/nfttest           [on STORAGE]
R/Us: nft> pwd :
Rtne:  local wd is /g/g0/jfk/stuff      [on client machine]
R/Us-a nft> pwd -a
Rtne:  local wd is /g/g0/jfk/stuff
       remote wd is ~/nfttest
```

QUIT (Terminate NFT Client)

SYNTAX:

quit

ROLE:

Terminates your current interactive NFT client and closes any open log file, but does NOT stop your previously submitted file-transfer jobs from continuing to execute. Indeed, job persistence even after your client ends is an NFT safety feature. To wait until all jobs complete before QUITing, use the combination command BLOCK (page 36);QUIT.

To discover if any incomplete file-transfer jobs remain, even from previous NFT runs, use NFT's RPT (page 78) command. To terminate specific file-transfer jobs (as opposed to terminating the client that submits them), use NFT's ABT (page 34) (abort) command.

RENAME (Change File Name)

SYNTAX:

```
ren[ame] [host]sourcepath [host]sinkpath
```

```
ren[ame] [host]{file1,file2,...} [host]{filex,filey,...}
```

ROLE:

Renames the file specified by *sourcepath* to the name specified by *sinkpath*. Alternatively, RENAME changes the name of each file in the ordered list *{file1,file2...}* into the corresponding file name in the second ordered list *{filex,filey...}* (each file list must have the same number of members). Here

host is the NFT host-specifying prefix that defaults to the STORAGE (not current client) machine, as explained above (keyword: prefix (page 12)). For RENAME (unlike CP), if you use a *host* prefix, it must be the SAME for both *sourcepath* and *sinkpath*. You can thus rename files on any single remote machine, but you can NOT rename files "across machines." Use CP (page 48) to move files between machines and simultaneously change their names.

sourcepath and *sinkpath* are standard UNIX pathnames that default to the current working directory (specified by CD (page 37)). Because of the obvious ambiguity that would result, you cannot use file filters.

file1,file2,... and *filex,filey,...* must be surrounded by the braces *{ }* shown above, and both lists must have an equal number of members and no file filters.

Some FTP daemons do not support renaming directories. To actually transfer files between machines, use NFT's CP (copy) (page 48), GET (from storage) (page 56), or PUT (to storage) (page 71) commands instead of RENAME. To confirm name changes, use DIR (page 52). RENAME takes no options and reports (only) the old name of each file it changes.

EXAMPLE:

To RENAME file test7 to test8 on the local (client) machine, use the syntax shown here:

```
nft>rename :test7 :test8
11.0 1 entry renamed /usr/tmp/test7
```

RMDIR (Remove Directories)

SYNTAX:

```
rmdir [-R] [host]pathname
```

```
rmdir [-R] [host]{dir1,dir2,...}
```

ROLE:

Removes the (empty) directory specified by *pathname* from the specified *host*. Alternatively, RMDIR removes each (empty) directory in the ordered list {*dir1,dir2...*} from the specified host. Here

host is the NFT host-specifying prefix that defaults to the STORAGE (not current client) machine, as explained above (keyword: prefix (page 12)). If you precede RMDIR with an OPEN (page 69) command, then *host* defaults to the machine that OPEN specified.

pathname is a standard UNIX pathname that ends in a directory, often a child of the current working directory (specified by CD (page 37)). To delete many directories with one command, use a standard UNIX file filter at the end of *pathname*.

dir1,dir2,... must be surrounded by the braces {} shown above. File filters are allowed in any list member.

Remember that by default removals occur among your storage directories, not your local directories (on the client machine), which you must overtly specify with the colon (:) prefix. To delete files instead of directories, use DELETE (page 50) instead of RMDIR. NFT's DIR (page 52) command lists your files and directories.

OPTIONS:

-R (uppercase are) recursively deletes the files and subdirectories in a directory before deleting the directory itself. Without -R, you must empty each directory before you can remove it with RMDIR. (The DELETE and RMDIR commands are equivalent when you invoke the -R option.) NFT ignores soft links to subdirectories.

ROUTING (Use Login Node Jumbo Frames)

SYNTAX:

routing

ROLE:

Performs file transfers (to or from storage) on login nodes with jumbo-frame network connections when available (on by default).

On many LC production machines, the login nodes have jumbo-frame network connections to storage, enabling large data blocks to transfer more quickly than with standard, smaller frames. Compute nodes lack these jumbo-frame connections because of their higher cost. When appropriate, NFT now routes file-transfer requests (to or from storage) from their originating compute nodes to the login nodes on the same cluster to take advantage of available faster jumbo-frame transfer rates.

NFT routes storage transfers (as explained above) by default. Use NOROUTING (page 67) to disable automatic routing (perhaps for timing or other tests). Use CHKROUTING (page 41) to see if routing is currently enabled or disabled (or otherwise not available) between a specified *pathname* and storage. STATUS also reveals whether routing is on (yes) or off (no). Routing will *not* speed NFT transfers between file systems one of which is not storage (between different parallel file systems, for example). Also, routing seldom benefits file transfers to or from regular NFS-mounted file systems (such as the common home directories). Routing primarily speeds transfers between storage and Lustre parallel file systems (either direction) on Linux/CHAOS clusters. It may later similarly benefit transfers between storage and GPFS on IBM/AIX machines (where now the high ratio of compute nodes to login nodes poses possible contention problems).

EXAMPLE:

Routing is on by default, but you can restore it with the ROUTING command if needed.

```
R/Us: nft> routing
R/Us: nft> status
Rtne: Connected to storage as jfk.
      . . .
      Routing:   yes
```

RPT (Report Job Status)

SYNTAX:

```
rpt [[n[-m]] | [-opt]]
```

ROLE:

Reports the current status of your most recent NFT request ("job") by default, or the status of the specific job with unique job number *n* (an integer), or the range of jobs with numbers *n-m* inclusive (using a hyphen, not a comma, as separator), or all members of the job class specified by any one job-class option *opt* listed below.

Besides the job number(s) or job class you select, three other factors are relevant to the scope of RPT status reports:

Record persistence

The NFT server remembers your jobs, their numbers, and their status for up to 4 days before purging its records. So frequent NFT users will get reports on all members of a job class throughout this time range, not just on the jobs started with their currently running NFT client. If this is a problem, use CLR to overtly delete the old(er) records that are no longer of interest.

Sessions

Most NFT uses have only one NFT session, and RPT reports on the jobs (job class members) in that session. If you start multiple sessions, RPT reports only on the jobs in your currently selected session, ignoring all your jobs in other sessions. This can give you more control of your status reports or lead to confusion, depending on your awareness of the sessions you have started. For details on the effect of multiple sessions, see the Sessions section (keyword: [sessions](#) (page 29)).

Verbosity

The NFT VERBOSE command (keyword: [verbose-levels](#) (page 27)) lets you specify which state changes NFT reports interactively as it runs (e.g., when jobs begin as well as when they end). VERBOSE does NOT, however, affect which jobs (or job class members) NFT includes in RPT status reports, nor the amount of detail provided on each job's status line. Thus VERBOSE does not change RPT's scope at all, although it does change general NFT dialog. See the -v option below for a different way to add detail to each line that RPT reports.

[ABT](#) (page 34) (abort) and [CLR](#) (page 46) (clear) are NFT commands closely related to RPT. The NFT [STATUS](#) (page 83) command reports on your environment-variable settings, not on your file-transfer jobs.

OPTIONS:

RPT accepts many job-class options *opt* to specify which set of your NFT jobs you want reported. But RPT expects you to use these options ONE at a time: combined options (such as -ek or -ke) yield INcomplete status reports or sometimes return just the warning message

```
job class? choose from -aichxoek. command rejected.
```

Possible stand-alone job-classes on which you can request reports include:

- a all jobs that NFT remembers, regardless of state. [If you use multiple NFT [sessions](#) (page 29), a rare practice, then -a selects only jobs in the current session, NOT in all the sessions that you have created, and the other job-class options behave likewise.]

- i incomplete jobs, waiting to run or not finished running.
- h held jobs, incomplete jobs in the scheduling queue.
- x active jobs, incomplete jobs currently running.
- c complete jobs, that have successfully or unsuccessfully completed running.
- o okay jobs, those that have successfully completed running.
- e error jobs, that have unsuccessfully completed running.
- k aborted jobs, those that were terminated by the user with ABT.
- v output that has the same scope as other RPT reports but with more detail on each line (adds the command used or action taken to the usual status message).

EXAMPLE:

A typical annotated example of output from RPT, with the format of each line explained, appears in part of the Job Reporting section above (keyword: [rpt-examples](#) (page 26)).

SESSION (Change NFT Sessions)

SYNTAX:

```
session nn | new
```

ROLE:

Closes your former NFT session (session 0 is the default) and opens a new one, in which the new session number *nn* is associated with all your subsequent NFT commands. NFT numbers the jobs in each session with an increasing sequence of integers that ignores all other sessions (so, e.g., each session may have an unrelated job numbered 13).

You can reopen a former session by using its session number in this same command (e.g., SESSION 0 reopens that session and closes session 1 if issued while you are in session 1). Reopening a session lets you check on its jobs with [RPT](#) (page 78) or start more jobs associated with it. Here

- | | |
|------------|--|
| <i>nn</i> | is an integer from 0 through 99 inclusive that uniquely identifies your session. |
| <i>new</i> | causes NFT to choose a unique, unused session number for you, open that session, and report the identifier chosen. Because NFT remembers your session numbers for up to 4 days and because picking a session number already in use reopens that session rather than creates a new one, you may sometimes want NFT to open a new session by automatically picking a number for it that is guaranteed to be unused. To guarantee a fresh session number, use |

```
session new
```

NFT gives no overt confirmation when you close one session and open another (just the usual prompt for input). And RPT status reports do not reveal which session they cover. So to discover which session is now open (or to confirm a requested change of session) use NFT's [STATUS](#) (page 83) command, which reports the current session number along with other NFT environment settings.

See the [Sessions](#) (page 29) section above for a complete analysis of the implications of using multiple NFT sessions, especially if combined with multiple NFT clients or in batch jobs.

SETCOS (Change Storage Class of Service)

SYNTAX:

```
setcos nnn
```

ROLE:

Sets your class of service (COS) for files subsequently transferred to storage to *nnn*. See the SETCOS discussion in LC's HPSS Reference Manual (URL: <https://computing.llnl.gov/LCdocs/hpss/index.jsp?show=s2.1.4>) for allowed values of *nnn*.

You can check the current COS setting for your NFT session with STATUS (page 83). STATUS dutifully reports any integer *nnn* that you supply with SETCOS, even if not allowed by HPSS. And when STATUS reports a COS of 0 (zero), HPSS automatically sets each stored file's COS based on its size according to the default behavior.

You can discover the COS with which already stored files have been stored by using NFT's DIR (page 52) command with the -h option (COS then appears in the third column in DIR's report). See also the detailed class of service discussion in LC's HPSS Reference Manual (URL: <https://computing.llnl.gov/LCdocs/hpss/index.jsp?show=s1.2.1>).

EXAMPLE:

STATUS here reveals the change (from the default situation) after you use SETCOS.

```
R/Us:  nft> status
Rtne:  Connected to storage as jfk.
      . . .
      Cos:      0
R/Us:  nft> setcos 120
R/Us:  nft> status
Rtne:  Connected to storage as jfk.
      . . .
      Cos:      120
```

SOURCE (Use Command File)

SYNTAX:

source *pathname*

ROLE:

Reads and executes all the NFT commands contained in the text file located at *pathname*, where

pathname is usually just the name of a file (e.g., extracom) that you want NFT to read from the current working directory on the machine where you are running the NFT client. If you supply an absolute pathname (such as ~/projects/extracom or /usr/tmp/testdir5/input) then NFT uses that location instead.

When you use SOURCE NFT's normal response messages continue to appear at your terminal while the commands in the file execute. (Note that the SOURCED commands themselves do NOT echo at the terminal: thus PUT test3 will NOT appear but NFT's response when test3 is stored will appear. This can make some responses hard to interpret.)

Usually the commands in a SOURCE file are just what you might type at your terminal, one per line. But you can construct condensed, annotated command files by using # as a comment sentinel, semicolon as a command separator, and backslash (\) as a line-continuation flag (example below).

You can achieve an effect rather similar to using SOURCE by instead using [file redirection](#) (page 21) on NFT's execute line when you first start your NFT client, but neither the commands in the input file nor responses that arrive after your client ends will echo at your terminal with this alternative approach.

DEFAULTS:

SOURCE takes no options and there is no default input file.

EXAMPLE:

This example shows a simple 3-line command file for use with SOURCE, and a version altered (in appearance but not effect) using three sentinel characters mentioned above. These are equivalent command files for use with SOURCE.

```
clobber
put test4
get test6

#shows special characters
clobber;put test4;get tes\
t6
```

STATUS (Report Environment Variables)

SYNTAX:

status

ROLE:

Reports a list of the current values of NFT's environment variables, including the current session number and apparent security level. Most of the commands that toggle the values of these variables do not report the result of their own action, so using STATUS is the best way to confirm changes you have made.

If you need information on the status of file transfers you have requested, use NFT's RPT (page 78) command rather than STATUS.

EXAMPLE:

Here is a typical NFT response to a STATUS command.

```
R/Us:  nft> status
Rtne:  Connected to storage as jfk.
      Session:                0.
      Verbose:                 76 (decimal), 4c (hexidecimal).
      Clobber:                 no.
      Routing:                 yes.
      Cos:                     0.
      Dualcopy:                no.
      Job Execution mode:      Synchronous.
      Group construct:         closed.
      Input from:              standard-in.
      Output to standard-out:  yes.
      Output to log file:      no.
```

SYNC (Run Jobs in Series)

SYNTAX:

sync

ROLE:

Begins synchronous mode. Since SYNC is the default setting whenever you run NFT, the SYNC command serves chiefly to cancel a previous ASYNC (page 35) command. In SYNC mode, NFT executes all your subsequent commands (jobs) strictly in series, always preserving the order in which you submitted them (with three exceptions).

NFT supports these three exceptions to or exemptions from SYNC mode:

- multiple-file transfers using GET or PUT,
- command sets flanked by GROUP and ENDGR, and
- commands following ASYNC.

See the command-sequencing (page 16) section above for a comparative analysis of these SYNC exceptions.

The ASYNC, (page 35) GROUP, (page 58) and BLOCK (page 36) commands can all be used to influence how NFT sequences its jobs. SYNC has no options and returns no mode confirmation, but you can use NFT's STATUS (page 83) command at any time to discover your current SYNC/ASYNC setting, which persists even across logical NFT "sessions (page 29)."

TERM (Enable Terminal Output)

SYNTAX:

term

ROLE:

Causes NFT to display at your terminal output from its executed commands, as well as its interactive prompt `nft>` for more input. TERM is NFT's default behavior, so you would normally need to use the overt TERM command only to reverse your previous use of the NOTERM (page 68) command, which disables terminal output.

TERM and NOTERM are mutually exclusive alternative settings for an NFT environment variable that preserves your choice of behavior until you overtly change it (or terminate your NFT client). Only by using separate NFT clients (not multiple sessions with one client) can you have two sets of environment variable settings for two sets of NFT jobs at once. See the "sessions (page 29)" section for details.

TERM takes no options and returns no confirmation message, but its use restores the interactive `nft>` prompts that are absent after NOTERM.

TIME (Report Current Time)

SYNTAX:

time

ROLE:

Reports the current system time (day, date, hour, minutes, seconds) on the machine where you are running the NFT client.

VERBOSE (Control State-Change Reports)

SYNTAX:

verbose *mask*

ROLE:

Specifies which changes of state for each file-transfer job NFT will report to you, where

mask is a 32-bit mask each of whose bits toggles the reporting of one kind of state change. The mask's default value is decimal 76. To set the bits, see the table below.

NFT jobs pass through several states from submittal to completion, and you can control how finely NFT reports on these changes of state by using its VERBOSE option. By default, NFT passes along transfer statistics from the FTP daemon that actually moves files at NFT's request, as well as sending error and abort diagnostic messages if a job completes unsuccessfully. But there are other state changes too, and you can request messages about any or all of them by using the appropriate argument for VERBOSE. (VERBOSE does not change the detail level covered in status reports from RPT (page 78), nor the environment-variable setting reports from STATUS (page 83). See instead RPT's -v option.)

Each possible state change for an NFT job corresponds to one bit in a (32-bit) mask that VERBOSE sets. You request diagnostic messages about a state change by setting its bit in the mask, and you set each bit by using the decimal value shown in the table below. To request a combination of reports, ADD the corresponding decimal values and use the sum as the argument for VERBOSE (for example, the default combination of diagnostic messages corresponds to the sum $4+8+64=76$).

State Change	Decimal Value	Diagnostic Meaning
Begin	1	client has submitted job
Done	2	job has completed successfully
Error(*)	4	job has failed (unsuccessfully completed)
Abort(*)	8	job was killed by user
Accepted	16	job was received by server
Reserved	--	--
Transfer stats(*)	64	FTP transfer amount and rate
Start	128	server has started job execution
Progress errors	256	immediately reports in-progress errors in secondary jobs
Reserved	--	--

(*)Default verbosity (combination 76)

VERBOSE does not confirm your requests for different state-change reporting, so you must use NFT's STATUS (page 83) command to verify the current NFT verbosity setting.

EXAMPLE:

To see how changing the VERBOSE value changes the grain size of state-change reports during NFT dialogs, compare this default-value exchange (VERBOSE 76)

```
User: get test4
Rtne: 14.0. 95 bytes received in 1.3 seconds (0.1 Kbytes/s)
      from ~/test4 to /tmp/jfk/test4
      14.0. 1 entry copied ~/test4
      nft>
```

with this maximum-value exchange for the same job (VERBOSE 479):

```
User: get test4
Rtne: 14.0. accept.
      14.0. begin ~/test4
      14.0. start ~/test4
      14.0. 95 bytes received in 1.3 seconds (0.1 Kbytes/s)
      from ~/test4 to /tmp/jfk/test4
      14.0. 1 entry copied ~/test4
      14.0. done. /tmp/jfk/test4
      nft>
```

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government thereof, and shall not be used for advertising or product endorsement purposes.

(C) Copyright 2007 The Regents of the University of California. All rights reserved.

Keyword Index

To see an alphabetical list of keywords for this document, consult the [next section](#) (page 91).

Keyword	Description
-----	-----
entire	This entire document.
title	The name of this document.
scope	Topics covered in this document.
availability	Where this program runs.
who	Who to contact for assistance.
introduction	Overview of NFT and this manual.
execute-line	How to run NFT.
nft-usage	Basic NFT execution tips, options.
nft-features	NFT features, limits, assumptions.
examples	Basic NFT sample sessions.
syntax	Specifying files, hosts with path prefixes.
prefix	Specifying files, hosts with path prefixes.
command-summary	NFT commands compared, grouped.
option-summary	NFT commands compared, grouped.
storage-defaulted	General vs. storage-defaulted commands.
environment-variables	Setting NFT envr. variables.
command-sequencing	Series vs. parallel execution.
command-remoteness	Local vs. server execution.
local-commands	Locally (client) executed commands.
server-commands	NFT server executed commands.
immediate-commands	Commands executed without scheduling.
job-commands	Commands scheduled by NFT server.
files	NFT input and output files.
log-files	Capturing (logging) NFT output.
output-files	Capturing (logging) NFT output.
log-usage	Using LOG and CLOG.
log-examples	Logging techniques and samples.
input-files	Inputting commands from files.
redirection-input	Command file input by redirection.
source-usage	Command file input using SOURCE.
file-interactions	Effect of input files on log files.
job-status	Job status, reporting, grouping.
job-identifiers	NFT job numbers and classes.
job-numbers	How NFT uniquely numbers jobs.
job-classes	Hierarchy of NFT job classes.
job-reporting	Making RPT job-status reports.
rpt-scope	Changing scope of RPT reports.
rpt-examples	RPT report format and examples.
verbose-levels	Change-of-state verbosity control.
sessions	Using NFT job sessions.
session-scope	How sessions affect RPT report scope.
session-usage	Multi-session techniques, effects.
session-transfer	Sessions with multiple clients.
scripts	Using NFT within batch scripts.
commands	Dictionary of NFT commands.
options	Dictionary of NFT commands.
command-syntax	NFT command syntax advice.
abt	Abort Incomplete Jobs (command)
async	Run Jobs in Parallel (command)
block	Block or Delay Next Command (command)
cd	Change Working Directory (command)
cdup	Change Working Directory Up (command)

<u>chgrp</u>	Change Groups (command)
<u>chkrouting</u>	Report Routing Availability (command)
<u>chmod</u>	Change Permissions (command)
<u>chown</u>	Change Owners (command)
<u>clobber</u>	Enable File Overwriting (command)
<u>clog</u>	Close Log File (command)
<u>close</u>	Restore Remote Host (command)
<u>clr</u>	Clear Completed Job Reports (command)
<u>cp</u>	Copy/Transfer Files (command)
<u>delete</u>	Remove Files (command)
<u>dir</u>	List Directory Contents, Long (command)
<u>dualcopy</u>	Store Multiples of Mission Critical Files (command)
<u>endgr</u>	End Asynchronous Group (command)
<u>get</u>	Retrieve Stored Files (command)
<u>group</u>	Begin Asynchronous Group (command)
<u>help</u>	Describe NFT Commands (command)
<u>lcd</u>	Change Local Working Directory (command)
<u>ln</u>	Create a Link (command)
<u>log</u>	Open Log File (command)
<u>ls</u>	List Directory Contents, Short (command)
<u>mkdir</u>	Make Directories (command)
<u>noclobber</u>	Disable File Overwriting (command)
<u>nodualcopy</u>	Undo Dual Copy (command)
<u>norouting</u>	Disable Routing (command)
<u>noterm</u>	Disable Terminal Output (command)
<u>open</u>	Change Remote Host (command)
<u>put</u>	Store Local Files (command)
<u>pwd</u>	Print Working Directory (command)
<u>quit</u>	Terminate NFT Client (command)
<u>rename</u>	Change File Name (command)
<u>rmdir</u>	Remove Directories (command)
<u>routing</u>	Use Login Node Jumbo Frames (command)
<u>rpt</u>	Report Job Status (command)
<u>session</u>	Change NFT Sessions (command)
<u>setcos</u>	Change Storage Class of Service (command)
<u>source</u>	Use Command File (command)
<u>status</u>	Report Environment Variables (command)
<u>sync</u>	Run Jobs in Series (command)
<u>term</u>	Enable Terminal Output (command)
<u>time</u>	Report Current Time (command)
<u>verbose</u>	Control State-Change Reports (command)
<u>index</u>	The structural index of keywords.
<u>a</u>	The alphabetical index of keywords.
<u>date</u>	The latest changes to this document.
<u>revisions</u>	The complete revision history.

Alphabetical List of Keywords

Keyword	Description
<u>a</u>	The alphabetical index of keywords.
<u>abt</u>	Abort Incomplete Jobs (command)
<u>availability</u>	Where this program runs.
<u>async</u>	Run Jobs in Parallel (command)
<u>block</u>	Block or Delay Next Command (command)
<u>cd</u>	Change Working Directory (command)
<u>cdup</u>	Change Working Directory Up (command)
<u>chgrp</u>	Change Groups (command)
<u>chkrouting</u>	Report Routing Availability (command)
<u>chmod</u>	Change Permissions (command)
<u>chown</u>	Change Owners (command)
<u>clobber</u>	Enable File Overwriting (command)
<u>clog</u>	Close Log File (command)
<u>close</u>	Restore Remote Host (command)
<u>clr</u>	Clear Completed Job Reports (command)
<u>command-remoteness</u>	Local vs. server execution.
<u>command-sequencing</u>	Series vs. parallel execution.
<u>command-summary</u>	NFT commands compared, grouped.
<u>command-syntax</u>	NFT command syntax advice.
<u>commands</u>	Dictionary of NFT commands.
<u>cp</u>	Copy/Transfer Files (command)
<u>date</u>	The latest changes to this document.
<u>delete</u>	Remove Files (command)
<u>dir</u>	List Directory Contents, Long (command)
<u>dualcopy</u>	Store Multiples of Mission Critical Files (command)
<u>endgr</u>	End Asynchronous Group (command)
<u>entire</u>	This entire document.
<u>environment-variables</u>	Setting NFT envr. variables.
<u>examples</u>	Basic NFT sample sessions.
<u>execute-line</u>	How to run NFT.
<u>file-interactions</u>	Effect of input files on log files.
<u>files</u>	NFT input and output files.
<u>get</u>	Retrieve Stored Files (command)
<u>group</u>	Begin Asynchronous Group (command)
<u>help</u>	Describe NFT Commands (command)
<u>immediate-commands</u>	Commands executed without scheduling.
<u>index</u>	The structural index of keywords.
<u>input-files</u>	Inputting commands from files.
<u>introduction</u>	Overview of NFT and this manual.
<u>job-classes</u>	Hierarchy of NFT job classes.
<u>job-commands</u>	Commands scheduled by NFT server.
<u>job-identifiers</u>	NFT job numbers and classes.
<u>job-numbers</u>	How NFT uniquely numbers jobs.
<u>job-reporting</u>	Making RPT job-status reports.
<u>job-status</u>	Job status, reporting, grouping.
<u>lcd</u>	Change Local Working Directory (command)
<u>ln</u>	Create a Link (command)
<u>local-commands</u>	Locally (client) executed commands.
<u>log</u>	Open Log File (command)
<u>log-examples</u>	Logging techniques and samples.
<u>log-files</u>	Capturing (logging) NFT output.
<u>log-usage</u>	Using LOG and CLOG.
<u>ls</u>	List Directory Contents, Short (command)
<u>mkdir</u>	Make Directories (command)

<u>nft-features</u>	NFT features, limits, assumptions.
<u>nft-usage</u>	Basic NFT execution tips, options.
<u>noclobber</u>	Disable File Overwriting (command)
<u>nodualcopy</u>	Undo Dual Copy (command)
<u>norouting</u>	Disable Routing (command)
<u>noterm</u>	Disable Terminal Output (command)
<u>open</u>	Change Remote Host (command)
<u>option-summary</u>	NFT commands compared, grouped.
<u>options</u>	Dictionary of NFT commands.
<u>output-files</u>	Capturing (logging) NFT output.
<u>prefix</u>	Specifying files, hosts with path prefixes.
<u>put</u>	Store Local Files (command)
<u>pwd</u>	Print Working Directory (command)
<u>quit</u>	Terminate NFT Client (command)
<u>redirection-input</u>	Command file input by redirection.
<u>rename</u>	Change File Name (command)
<u>revisions</u>	The complete revision history.
<u>rmdir</u>	Remove Directories (command)
<u>routing</u>	Use Login Node Jumbo Frames (command)
<u>rpt</u>	Report Job Status (command)
<u>rpt-examples</u>	RPT report format and examples.
<u>rpt-scope</u>	Changing scope of RPT reports.
<u>scope</u>	Topics covered in this document.
<u>scripts</u>	Using NFT within batch scripts.
<u>server-commands</u>	NFT server executed commands.
<u>session</u>	Change NFT Sessions (command)
<u>session-scope</u>	How sessions affect RPT report scope.
<u>session-transfer</u>	Sessions with multiple clients.
<u>session-usage</u>	Multi-session techniques, effects.
<u>sessions</u>	Using NFT job sessions.
<u>setcos</u>	Change Storage Class of Service (command)
<u>source</u>	Use Command File (command)
<u>source-usage</u>	Command file input using SOURCE.
<u>status</u>	Report Environment Variables (command)
<u>storage-defaulted</u>	General vs. storage-defaulted commands.
<u>sync</u>	Run Jobs in Series (command)
<u>syntax</u>	Specifying files, hosts with path prefixes.
<u>term</u>	Enable Terminal Output (command)
<u>time</u>	Report Current Time (command)
<u>title</u>	The name of this document.
<u>verbose</u>	Control State-Change Reports (command)
<u>verbose-levels</u>	Change-of-state verbosity control.
<u>who</u>	Who to contact for assistance.

Date and Revisions

Revision Date	Keyword Affected	Description of Change
-----	-----	-----
04Sep08	<u>dualcopy</u> <u>nodualcopy</u> <u>environment-variables</u> <u>index</u>	New command added, explained New command added, explained New variable toggles added. New keywords for new commands.
17Apr07	<u>routing</u> <u>norouting</u> <u>chkrouting</u> <u>setcos</u> <u>rpt</u> <u>status</u> <u>dir</u> <u>nft-features</u> <u>environment-variables</u> <u>local-commands</u> <u>examples</u> <u>index</u>	New command added, explained. New command added, explained. New command added, explained. New command added, explained. New -v verbosity option added. Now reports ROUTING, SETCOS too. DIR -h reveals SETCOS effect. Routing and COS details added. Some error messages expanded. New variable toggles added. New commands added. All GPS cases replaced, updated. New keywords for new commands.
12Apr06	<u>introduction</u> <u>nft-features</u>	LANL, Sandia users can now run NFT. LANL, Sandia users can now run NFT.
19Jul05	<u>introduction</u> <u>nft-features</u>	HOPPER interface to NFT noted. HOPPER interface to NFT noted.
12Oct04	<u>commands</u> <u>index</u> <u>storage-defaulted</u> <u>job-commands</u> <u>introduction</u>	Five ACL commands deleted. Keywords for ACL commands hidden. ACL commands deleted. ACL commands deleted. Note about discontinued commands.
01Jun04	<u>nft-features</u>	Two error-message formats noted.
20Jan04	<u>nft-features</u> <u>examples</u> <u>commands</u> <u>cp</u> <u>chgrp</u> <u>chmod</u> <u>dir</u> <u>ls</u> <u>acladd</u> <u>aclclear</u> <u>aclremove</u> <u>aclreplace</u> <u>aclshow</u> <u>index</u>	Many details revised. Output messages changed here, elsewhere. Seven commands add -R option. Added -d option. No longer limited to storage. No longer limited to storage. Many new format options. Many new format options. New command added, shown. New keywords for new commands.
29Jul03	<u>introduction</u> <u>get</u>	HTAR comparative role expanded. HTAR now gets from nonstorage archives.

	<u>put</u>	HTAR now puts into nonstorage archives.
09Oct02	<u>availability</u>	NFT on Linux, Furnace replaces Forest.
20Jun02	<u>nft-features</u> <u>delete</u> <u>setlev</u>	Class of service, stored copies added. Contrast with FTP's MDELETE noted. Class of service, stored copies added.
14Feb02	<u>entire</u>	Names updated in all examples.
27Aug01	<u>introduction</u> <u>get</u> <u>put</u>	HTAR role and manual noted. When to use HTAR instead of GET. When to use HTAR instead of PUT.
09Nov00	<u>availability</u> <u>introduction</u> <u>commands</u>	DEC becomes Compaq/DEC. All CRAY-based examples replaced. All CRAY-based examples replaced.
24May00	<u>scope</u> <u>introduction</u> <u>chgrp</u> <u>chmod</u>	EZSTORAGE relevance noted. EZSTORAGE relevance noted. CHGRPSTG in EZSTORAGE noted. CHMODSTG in EZSTORAGE noted.
03Aug98	<u>availability</u> <u>introduction</u> <u>dir</u> <u>ls</u>	NFT now on open DEC's, open IBM. Section expanded, clarified. Four suboptions disabled. Four suboptions disabled.
20Nov97	<u>availability</u> <u>nft-features</u> <u>storage-defaulted</u> <u>redirection-input</u> <u>session-transfer</u> <u>command-syntax</u> <u>open</u> <u>close</u> <u>commands</u> <u>scripts</u>	Only client hosts accept NFT transfers. Special characters added, limits revised. Keyword changed, OPEN role noted. Quoted commands explained, compared. SESSION NEW for batch noted. Cross references expanded. Undocumented command added. Undocumented command added. Many cross refs to OPEN added. New script-use tips added.
29Sep97	<u>availability</u>	NFT now on SCF IBM SP (SKY).
02Sep97	<u>availability</u>	NFT now on all SCF DEC's.
18Aug97	<u>availability</u>	NFT now on OAK (SCF DEC) too.
03Jul97	<u>execute-line</u> <u>examples</u> <u>setlev</u> <u>status</u> <u>redirection-input</u> <u>environment-variables</u>	-S security option persists disabled. -S option deleted from sample. Command persists but role disabled. Sec. level reported but disabled. -S option deleted from example. SETLEV role changed.
21Nov96	<u>commands</u> <u>nft-usage</u>	Detailed NFT command dictionary added. More details added.

nft-features Suboption use clarified.
command-sequencing
source-usage Example revised.
Example revised.

17Oct96 entire First edition of NFT reference manual.

EJG (04Sep08)

UCRL-WEB-201529

LLNL Privacy and Legal Notice (URL: <http://www.llnl.gov/disclaimer.html>)

EJG (04Sep08) Contact on the OCF: lc-hotline@llnl.gov, on the SCF: lc-hotline@pop.llnl.gov