The Enstore Administrator's Guide

November 20, 2019

ABSTRACT

Enstore is the mass storage system implemented at Fermilab as the primary data store for large data sets. Enstore provides access to data on tape or other storage media both local to a user's machine and over networks. It is designed to provide high fault tolerance, availability and scalability sufficient for the current Fermilab and its collaborators needs, as well as easy administration and monitoring. It uses a client-server architecture which provides a generic interface for users and allows for hardware and software components that can be replaced and/or expanded. Enstore is currently integrated with dCache front end which makes up a Hierarchical Data Storage System.

This document describes these tools, how to use them to move data to and from storage media, and how to monitor the progress of jobs through the system.

Revision Record

January 19, 2010	First printing.
April 7, 2011	Updated the migration chapter and fixed typos in the PNFS maintenance chapter.
December 21, 2012	Modified abstract, added hyperlinks to TOC.
March 5, 2013	Rewrote the Metadata Scanning chapter.
January 22, 2014	Added installation chapter.
April 15, 2015	Changed installation chapter. Postgresql installation part.
January 20, 2016	Modified installation chapter.
February 29, 2016	Modified installation chapter.
November 20, 2019	Modified installation chapter.

Disclaimer

Table of Contents

Chapter 1: Enstore Overview

1.1 Introduction

Enstore is the mass storage system implemented at Fermilab as the primary data store for large data sets. The data is stored on different types of magnetic tapes in robotic tape libraries. Enstore is designed to to provide high fault tolerance, availability and scalability sufficient for the current Fermilab and its collaborators needs, as well as easy administration and monitoring.

It uses a client-server architecture which provides a generic interface for users and allows for hardware and software components that can be replaced and/or expanded.

Enstore has two major kinds of software components:

• the Enstore servers, which are software modules that have specific functions, e.g., maintain database of data files, maintain database of storage volumes, maintain configuration, look for error conditions and sound alarms, communicate user requests down the chain to the tape robots, and so on.

• **encp**, a client program for copying files directly to and from the mass storage system.

Enstore can be used directly only from on-site machines. Off-site users are restricted to accessing Enstore via dCache, and in fact on-site users are encouraged to go through dCache as well.

Enstore supports both automated and manual storage media libraries. It allows for a larger number of storage volumes than slots. It also allows for simultaneous access to multiple volumes through automated media libraries. There is no preset upper limit to the size of a data file in the enstore system; the actual size is limited by the physical resources. The lower limit on the file size is zero. The upper limit on the number of files that can be stored on a single volume is about 5000.

Enstore allows users to search and list contents of media volumes as easily as they search native file systems. The stored files appear to the user as though they exist in a mounted UNIX directory. The mounted directory is actually a distributed virtual file system in PNFS namespace containing metadata for each stored file. Enstore eliminates the need to know volume names or other details about the actual file storage.

There are several installed Enstore systems at Fermilab. Currently these include CDFEN for CDF Run II, D0EN for D0 Run II, and STKEN for all other Fermilab users. Web-based monitoring for the Enstore systems is available at http://www-ccf.fnal.gov/enstore/. Currently, all storage libraries are tape libraries. The Computing Division operates and maintains the tape robots, slots, and other tape equipment, but for the present, experiments provide and manage their own volumes.

1.2 Enstore architecture

The Enstore software architecture is presented in Figure 1. Enstore components are:
- A configuration server keeps the system configuration information and provides it to the rest of the system. Configuration is described in an easily maintainable configuration file.
- A volume clerk maintains the volume database and is responsible for declaration of new volumes, assignments of volumes, user quotas, and volume bookkeeping.
- A file clerk maintains the file database, assigns unique bit file IDs and keeps all necessary information about files written into Enstore.
- Info server provides read-only user access to information maintained by file – and volume – clerks.
- Multiple distributed library managers provide queuing, optimization, and distribution of user requests to assigned movers.
- Movers write / read user data to tapes. A mover can be assigned to more than one library manager. A media changer mounts / dismounts tapes in the tape drives at the request of a mover.
- Alarm and log servers generate alarms and log messages from Enstore components correspondingly.
- An accounting server maintains an accounting database containing information about completed and failed transfers and mounts.
- A drivestat server maintains a database with information about tape drives and their usage.
- An inquisitor monitors the state of the Enstore components.
- PNFS namespace server implements a name space that externally looks like a set of Network File Systems.
- Events are used in the Enstore system to inform its components about changes in the configuration, completed and ongoing transfers, states of the servers, etc. An event relay transmits these events to its subscribers.
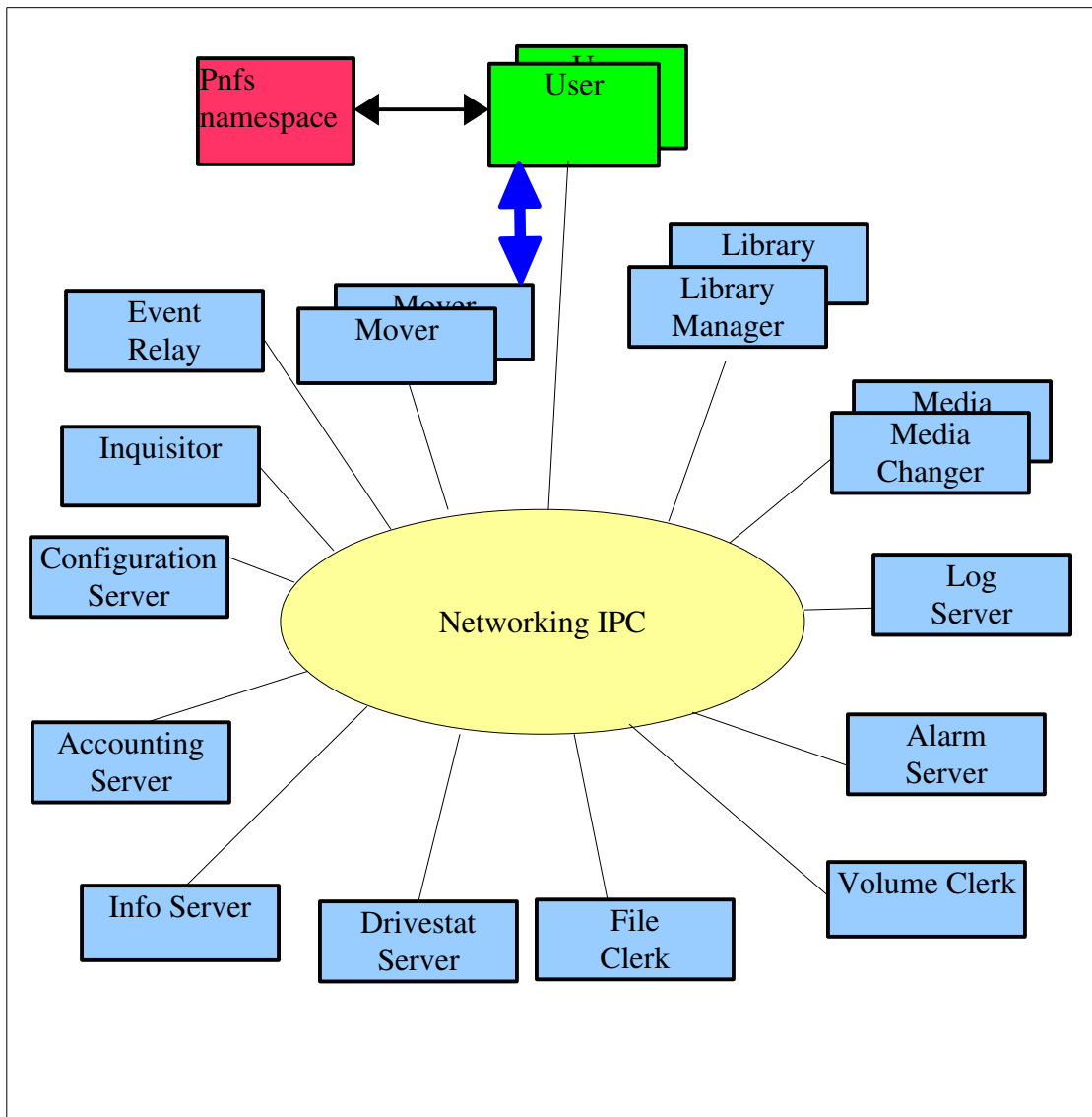
All Enstore components communicate using IPC based on UDP. Great care has been taken to provide reliable communications under extreme load conditions. The user command, encp, retries in case of an internal Enstore error. The number of user computers is not restricted, and Enstore components can be distributed over unlimited number of nodes, tape libraries and tape drives.

1.3 Enstore hardware requirements and configuration

Enstore is a distributed and scalable system. It can be installed on a single Linux node, but for the better performance the following needs to get taken into consideration. Enstore has several databases which are used quite extensively depending on the number of data transfer

requests. Modern tape drives have transfer rates in the order of 100MB/s. Having this in mind the following recommendations for hardware layout and system configuration are suggested.

1.3.1 Hosts

There are no strict requirements for hosts. The general requirements are:

1. dual CPU Intel processor (3.0GHZ or better),
2. 1MB (or more, better 2MB) of RAM
3. 120MB (or more) system disk
4. 1Gb (or more) network adapter for data transfer
5. 100 Mb network adapter - not necessary but it comes with the system anyway and can be used for private LAN connection with robotic library controller
6. Tape drive adapter (whichever is appropriate (SCSI, Fiber Channel) for **mover** node

1.3.2 Additional storage

Additional storage may be needed for large systems to hold databases, system information, log files, etc. It can be any kind of appropriate raid arrays. For better flexibility of the system you may want to considered some kind on NAS, such as Raid arrays connected to hosts via Fiber Channel switch.

1.3.3 Network switch

There no strict requirements for the network switch other than it should provide an adequate connection and transfer rates between enstore components as well as client hosts and tape movers.

1.3.4 System Configuration

We recommend to have the following configuration:
For the small system (one robotic library with one or 2 tape drives and few thousands tapes).
Minimal configuration:

1. host1: pnfs server, Apache web server, configuration_server, log_server, alarm_server, inquisitor, event_relay, ratekeeper, postgres DB server, file_clerk, volume_clerk, info_server, accounting_server, drivestat_server
2. host2: media_changer(s), library_manager(s), mover

*** Note that this configuration may have problems as the number of accesses and their rates increase. It is always better to run one mover
on a separate host

Recommended configuration:

1. host1 : pnfs server
2. host2: Apache web server, configuration_server, log_server, alarm_server, inquisitor, event_relay, ratekeeper
3. host3: media_changer(s), library_manager(s)
4. host4: postgres DB server, file_clerk, volume_clerk, info_server, accounting_server, drivestat_server
5. host5: backups, plots, migration work, etc. This can be done on one of existing hosts but may interfere with operations.
6. one host per mover.

Chapter 2: Enstore Installation

2.1 Introduction

Enstore is written in python and currently requires a slightly modified python language interpreter. Enstore uses a Fermi Tape Tool (FTT) library and set of tools providing access to different types of tape drives. It is included into enstore rpm for the revisions later than 2.0.0-0. This section describes installation and configuration procedures of enstore and all required additional software.

2.1.1 Remote access

Before installing enstore you need to set up all its nodes for remote access. Remote access and product distribution is provided by the following scripts in $ENSTORE_DIR/sbin coming with enstore rpm:

1. enrsh - analog of rsh
2. enrcp - analog of rcp
3. enrsync - analog of rsync

2 options are currently provided for secured access: kerberized rsh and rcp, and ssh. For using either of them you need to setup kerberized or (and) ssh access to all enstore nodes for 'enstore' and 'root' accounts.

The mentioned scripts first check for the presence of the directory containing kerberized utilities. If it is found they assume that krb5 utilities will be used. If $ENSSH or $ENSCP are defined the ssh and scp will be used. As the last resort they try to use non kerberized rsh, rcp, and rsync. Installation and configuration of ssh and kerberized rsh, rcp are out of the scope of this document.

2.1.2 Licensing

Enstore (BSD like):

Copyright (c) 1999-2011, FERMI NATIONAL ACCELERATOR LABORATORY
All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of the FERMI NATIONAL ACCELERATOR LABORATORY, nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

    PNFS:

    Enstore uses a Chimera (old name – PNFS) virtual file system that implements the Enstore namespace. It was written at DESY. The licensing information and code can be obtained at the dcache site. Installation instructions can be found here: http://www-pnfs.desy.de/gettingStarted.html

DCache must be installed at least on the node that will run namespace server. Namespace nfs mount point(s) must be exported to any node transferring data to or from enstore. Some instructions can be found here:
http://www-isd.fnal.gov/ISA/pnfs.html

2.1.3 Enstore rpm

The enstore rpm and additional rpms are located here ftp://ssasrv1.fnal.gov/en/. Currently there are rpms for SL6 and SL7. They are located in slf6x, and slf7x sub directories correspondingly.

To use yum for installing enstore create the following file on all enstore nodes:

/etc/yum.repos.d/enstore.repo containing:

[enstore]
name=Enstore
baseurl=ftp://ssasrv1.fnal.gov/en/slf6x/$basearch
gpgcheck=0
enabled=1

For SLF7 enter *baseurl=ftp://ssasrv1.fnal.gov/en/slf7x/$basearch.*

For other OS versions please check enstore download page or contact enstore mailing list. It is desirable that X-windows is installed at least on head node. This allows to

run entv – application which dynamically shows data transfers.
Installation of enstore always begins on enstore head node - the node where
configuration server will be running.

2.1.4 Installing Enstore.

If you install the new enstore instance (site) the installation must begin on enstore
head node – the node, where **enstore configuration server** will run.
Enstore has 2 kinds of configurations – servers and movers. Mover require less
products installed (minimal installation), but may be installed as Enstore server.

2.1.4.1 Installing postgreSQL DBMS.

PostgreSQL DBMS must be installed on all Enstore nodes. Use the current
postgreSQL documentation for the installation instructions. Alternatively you may
install current PostgreSQL rpms from enstore repository with:
[root@head_node ~]# yum -y install postgresql*
Make sure that initdb, pg_config, postmaster, and psql are in /usr/bin. This can be done by
/usr/sbin/alternatives like:
VER="9.11"
for PROG in "initdb" "pg_ctl" "pg_config" "postmaster" "psql"; do
rm -f /var/lib/alternatives/pgsql-${PROG};
*/usr/sbin/alternatives --install /usr/bin/${PROG} pgsql-${PROG} *
*/etc/alternatives/pgsql-${PROG} 920 --slave /etc/alternatives/pgsql-${PROG} *
pgsql-${PROG} /usr/pgsql-${VER}/bin/${PROG};
rm -f /var/lib/alternatives/pgsql-${PROG}man;
*/usr/sbin/alternatives --install /usr/share/man/man1/${PROG}.1 *
*pgsql-${PROG}man /etc/alternatives/pgsql-${PROG}man 920 --slave *
*/etc/alternatives/pgsql-${PROG}man pgsql-${PROG}man *
/usr/pgsql-${VER}/share/man/man1/${PROG}.1;
done

Set group and user in the postgresql startup script as "enstore".

2.1.4.2 Minimal installation (mover node)

Install the following rpms:mt-st, tcl, tk, enstore. Example:
[root@head_node ~]# yum -y install tcl tk
[root@head_node ~]# yum -y install enstore

2.1.4.3 Complete installation (server node)

For complete installation in addition to the minimal installation install the following
rpms: httpd, ncompress, gnuplot, ImageMagick. Example:

[*root@head_node ~]# yum -y install ncompress gnuplot ImageMagick*

2.1.4.4 Additional packages for Small Files Aggregation

For Small Files Aggregation (SFA) enstore service install additional packages: qpid-python on all nodes using SFA service and additinally qpid-cpp-server on SFA main server. The main SFA server will run qpidd and enstore SFA redirector and dispatcher.

2.1.5 Configuring Enstore

Enstore configuration includes run-time environment, enstore site (instance) configuration – enstore configuration file, and distributed environment configuration. When enstore rpm is installed the user account "enstore" is created and its .bash_profile and .bashrc are modified. The changes are done to the system allowing to start enstore on boot (/etc/rc.d/init.d/enstore-boot, /etc/rc.d/init.d/monitor_server-boot). The /etc/rc.local id also modified to create tape devices on mover nodes.

2.1.5.1 Note about .bashrc and .bash_profile.

In recent enstore rpms .basrhc and .bash_profile will not be installed automatically because they are installed by OS installation and update agents, such as puppet. To install them manyally do:

cp -p /opt/enstore/external_distr/.bashrc ~enstore

cp -p /opt/enstore/etc/enstore.bash_profile ~enstore/.bash_profile

For SL7 remove the following lines from .bashrc:

```
set +u
set -u
```

2.1.5.2 Run-time environment.

Enstore run-time environment can be initially created on Enstore head node and then copied all nodes running enstore servers and movers – all nodes where enstore was installed and will be using this particular enstore instance (site).

2.1.5.2.1 Head node

Assuming that we are configuring the head node, login to head node as user "enstore". Do the following:

[enstore@head_node ~]$ whoami

enstore

[enstore@head_node ~]$ pwd

/home/enstore

[enstore@head_node ~]$ mkdir -p site_specific/config
[enstore@head_node ~]$ cp /opt/enstore/site_specific/config/setup-enstore
site_specific/config
Modify (or add if don't exist) the following lines in site_specific/config/setup-enstore:
export ENSTORE_CONFIG_HOST=   # put here the name of your head node
export ENSTORE_CONFIG_FILE= # put here the path for enstore configuration file
export ENSTORE_MAIL= # put here email address for system administrators
export ENSSH=/usr/bin/ssh # to use ssh instead of rsh
export ENSCP=/usr/bin/scp # to use scp instead of rcp
export FARMLETS_DIR=/usr/local/etc/farmlets # farmlets directory (see below)

2.1.5.2.2 Enstore configuration file.

Enstore configuration file describes configuration of all Enstore servers and movers
as well as additional services used in and by Enstore. The configuration file is defined
by $ENSTORE_CONFIG_FILE in $ENSTORE_HOME/config/setup-enstore file
and is downloaded from $ENSTORE_CONFIG_HOST. We recommend to source
setup-enstore before proceeding with creation of the configuration file (**reminder: do
this on enstore configuration node**).

```
source /usr/local/etc/setups.sh
```

Alternatively if you login as user enstore the enstore environmet will be created
autonmatically.
Enstore configuration file is a python script. It can be created and modified using
Enstore configuration file is a python script. It can be created and modified using
/opt/enstore/etc/enstore_configuration_template or
/opt/enstore/etc/minimal_enstore.conf. Copy the selected file into
ENSTORE_CONFIG_FILE . It is recommended to put enstore configuration file into
$ENSTORE_HOME/site_specific/config directory. Description of configuration file
keys can be found in /opt/enstore/doc/config_params.html.
**You can also use one of the real enstore configuration files, for instance
$ENSTORE_DIR/etc/stk.conf.**
For Fermilab enstore there is a git repository from which all run-time configuration
can be set up. The procedure is described in "Configure enstore configuration git
repostitory for the first time on configuration server".

2.1.5.2.3 Regular node
If enstore instance has multiple nodes copy enstore configuration from head node to
these nodes (considering head_node as head node):
[enstore@new_node ~]$ pwd
/home/enstore

[enstore@new_node ~]$ mkdir -p site_specific/config
[enstore@new_node ~]$ scp head_node:site_specific/config/ site_specific/config*

2.1.5.3 Farmlets

Farmlets are the files containing node names for distributed operations on enstore nodes. These files are used by rgang command (multinode analog of rsh). You need to create initial farmlets and **then add and modify them when adding or removing an enstore node**.

Initially farmlets can be created by /opt/enstore/external_distr/make_farmlets.sh script or manually. You have to create enstore configuration file and start configuration server before running make_farmlets.sh. Otherwise they can be created manually. Farmlets location is defined by $FARMLETS_DIR in ~/site_specific/config/setup-enstore. The farmlet may contain one entry specifiying corresponding hos name or the list of host names to get served, entered one host per line. The mandatory farmlet files are "**enstore**" and "**enstore-down**". These files are used for starting enstore components on all servers and movers. The content of the farmlets reflect the order of nodes to which commands are sent. In "enstore" first node must be the one that runs enstore configuration server (head node), then the rest of the server nodes and movers, "enstoredown"
contains the list of enstore nodes in the order reverse to the order in "enstore".
In the current installation farmlets are in /usr/local/farmlets directory on **each enstore node**.

2.1.5.4 Name Space

Enstore client (encp) uses name space to present files to users. The files are arranged into directories and can be viewed using Unix FS commands such as "ls", "pwd". This name space was initially provided by pnfs, replaced later by chimera. Chimera is part of [dcache](#) project which means that to have chimera dcache rpm needs to get installed form dcache web site. Below is the example of dcache installation and configuration to run chimera name space. Note that the name space service need to get installed only on the node where it will be running. This service can be installed on every server in the framework of Screw Driverless Enstore (SDE) project, which requires the similar installation on all Enstore servers to provide easy replacement of the Enstore services in case of failure.

2.1.5.4.1 Installation.

Dcache is implemented in java, so install java (for instance):
[root@gccenmvr16a]# rpm -ivh http://javadl.sun.com/webapps/download/AutoDL?BundleId=78696
Install dcache rpm:
[*root@ns_node]# rpm -ih*
http://www.dcache.org/downloads/1.9/repo/2.2/dcache-2.2.15-1.noarch.rp
Configure dcache to run name server on gccenmvr16a

root@ns_node]# cd /etc/dcache/
[root@ns_node]# cp -p dcache.conf dcache.conf.orig
Modify dcache.conf:
dcache.layout=${host.name}
dcache.namespace=chimera

chimera.db.user=enstore
chimera.db.url=jdbc:postgresql://localhost/chimera?prepareThreshold=3

defaultPnfsServer=gccenmvr16a.fnal.gov
hsmStorageInfoExtractor=org.dcache.chimera.namespace.ChimeraEnstoreStorageInfoExtract
or
dcache.log.mode=new
scripts=${dcache.home}/scripts
scheme=development
dcache.user=root
Create layout:
[root@ns_node dcache]# cd layouts/
Create ns_node.conf:
[dCacheDomain]
[dCacheDomain/poolmanager]
[dCacheDomain/broadcast]
[dCacheDomain/loginbroker]
[dCacheDomain/topo]

[namespaceDomain]
dcache.java.memory.heap=2048m
dcache.java.memory.direct=2048m
[namespaceDomain/pnfsmanager]
pnfsVerifyAllLookups=true
[namespaceDomain/cleaner]

[nfsDomain]
dcache.java.memory.heap=2048m
dcache.java.memory.direct=2048m
[nfsDomain/nfsv41]
cell.name=NFS-${host.name}
nfs.v3=true
nfs.domain=fnal.gov
nfs.idmap.legacy=true

[dirDomain]
[dirDomain/dir]

Create DB directory (this must be on a separate large disk – external RAID):
as "root"
[root@ns_node]# mkdir /chimera
[root@ns_node]# chown enstore.enstore /chimera
as "enstore"
[enstore@ns_node ~]$ /usr/pgsql-9.3/bin/initdb -D /chimera

as "root"
[root@ns_node]# service postgresql-9.3 start
Starting postgresql-9.3 service: [OK]
[root@ns_node]# createdb -U enstore chimera
Add line to /etc/fstab:
localhost:/ /pnfs/fs   nfs
sync,rsize=4096,wsize=4096,user,intr,bg,hard,rw,noac,vers=3,nolock   0   0
Add lines to /etc/export (it must be later extended with names of client machines):
[root@ns_node]# echo "/ localhost(rw,no_root_squash)" >> /etc/exports
[root@ns_node]# echo "/pnfs/fs localhost(rw,no_root_squash)" >> /etc/exports
Create /pnfs/fs:
[root@ns_node]# mkdir -p /pnfs/fs
Start chimera server
[root@ns_node pgsql]# chkconfig --add dcache-server
[root@ns_node pgsql]# chkconfig dcache-server on
[root@ns_node pgsql]# service dcache-server start
Mount /pnfs/fs:
[root@ns_node]# mount /pnfs/fs
Check that chimera works:
[root@ns_node pgsql]# mkdir -p /pnfs/fs/usr/demo_test
[root@ns_node pgsql]# touch /pnfs/fs/usr/demo_test/aaa
[root@ns_node pgsql]# ls /pnfs/fs/usr/demo_test
aaa

2.1.5.4.2 Namespace (pnfs) tags.

 Before staring writing data into enstore you need to nfs mount name space, create
subdirectories, and set pnfs tags for enstore.
Below is some information regarding pnfs taken from "Enstore User Guide".

PNFS Directory Limitations: It is recommended to keep the number of files in any
given PNFS directory under 2000. This is recommended for any NFS based file
system.

Namespace (pnfs) tags.

Before files can be written to tape, Enstore needs to know where and how to write them. Pnfs uses tag files (usually just called tags) in the /pnfs namespace to specify this type of configuration information, and encp transfers this information to Enstore. Tags are associated with directories in the /pnfs namespace, not with any specific file, and thus apply to all files within a given directory (with the exception noted below). When a new directory in the /pnfs namespace is created, it inherits references to the tags of its parent directory. It is a feature of PNFS that a change to a parent directory will also affect its existing subdirectorys' tag references. Manually setting a directory's tags will destroy references to its parent directory's tags. This may be what you want to do, but be aware.

A file gets the tag references of its directory as they exist when the file is written to Enstore, and these are what encp uses to access it. Subsequent changes to a directory's tag references do not affect preexisting files, therefore it is possible to have files in a directory to which the current directory tags do not apply.

Allowable characters within tags are: alphanumeric characters, underscore (_), dash ( ), and slash (/).

Tag listing

The tags include (refer to "Enstore User Guide") :

file_family - this tag determines the file family associated with all files in this directory.

file_family_width - this tag determines the file family width associated with all files in this directory.

file_family_wrapper - this tag determines the file family wrapper associated with all files in this directory. The default is cpio_odc.

library - this tag determines the virtual library (and thus the library manager) associated with all files in this directory.

storage_group - this tag determines the storage group associated with all files in this directory, and shows up as your experiment's top level directory under /pnfs.

Typically, one storage group is associated with an entire experiment. A storage group is assigned to each experiment by the Enstore administrators. Users never change this tag.

Viewing tags

Offsite - users cannot mount pnfs, and therefore cannot see tags.

Onsite - users: to see the values of the tags for a given directory, first setup encp (with qualifier, see, for instance, section 6.1 "Setup encp" of "Enstore User Guide") then cd to the /pnfs subdirectory of interest and enter the command:

% enstore sfs --tags

.(tag)(file_family) = dcache

.(tag)(file_family_width) = 1

.(tag)(file_family_wrapper) = cpio_odc
.(tag)(library) = eagle
.(tag)(storage_group) = test
-rw-rw-r-- 11 xyz sys 6 Jul 26 10:22 .(tag)(file_family)
-rw-rw-r-- 11 xyz sys 1 May 5 2000 .(tag)(file_family_width)
-rw-rw-r-- 11 xyz sys 8 May 5 2000 .(tag)(file_family_wrapper)
-rw-rw-r-- 11 xyz sys 5 May 5 2000 .(tag)(library)
-rw-r--r-- 11 xyz sys 4 Jul 26 10:20 .(tag)(storage_group)
The output first lists the tags and their values, then the tags again in long format to show the owners and
protection modes.

Setting tags

Setting tags
We recommend root 644 permissions for library and storage_group tags. Permissions for other tags can
be 666 to allow everyone to
change file_family, file_family_width, or file_family_wrapper
root owned tags can be changed only on the machine where namespace server runs.
The sfs help command describes what options can be used to get pnfs information or set certain values.
The tags are viewed / set by:
enstore sfs –library
enstoree sfs –storage_group
enstore sfs –file_family
enstore sfs –file_family_width
enstore sfs –file_family_wrapper. For tape there may be 2 kinds of wrapper – "cpio_odc" and "cern"

2.1.6 Finishing installation and configuration

After enstore was installed and configured start enstore configuration server. This is needed to install and configure:
- enstore web site
- enstore cron jobs
- enstore databases

To start enstore configuration server login to enstore head node as user "enstore" and do the following:
[enstore@head_node ~]$ enstore start --just config
Checking configuration_server.
Starting configuration_server
[enstore@head_node ~]$ enstore conf –show

This command will output the enstore configuration described in $ENSTORE_CONFIG_FILE.

Note!!! All enstore commands have –help option. If you do not know what can be done just type **enstore** and investigate what options are available. Also refer to "Enstore Admin Guide" (can be found in /opt/enstore/doc/guides/Enstore_Administrator_Guide.pdf)

2.1.6.1 Enstore web site

After enstore was installed and configuration server started install enstore web site:

[root@head_node]#   source ~enstore/site_specific/config/setup-enstore
[root@head_node]# /opt/enstore/sbin/install_enstore_html

After this step you should be able to access the following web site:

http://www.the_host/enstore/

where the_host is the name of the enstore head node as defined in enstore configuration file. In this particular case it is gccenmvr16a.fnal.gov. Not all web pages are available yet, as they get produced by running enstore servers.

2.1.6.2 Enstore databases

Enstore has 3 databases: file/volume, accounting, and drivestat.

1. enstoredb - file/volume database contains information about all files written to enstore and all volumes (tapes) defined in enstore
2. accounting -contains information about all successful and failed data transfers
3. drivestat database contains information about tape drives configured in the system

There is one more database (operation), which is used only at FNAL. All these databases can be served by separate DB servers or just by one (this is defined in enstore configuration file). Install PosrgresQL on the server(s) defined in the enstore configuration file. The script ($ENSTORE_DIR/sbin/create_database.py), which configures and creates databases uses enstore configuration, but does not consider the host name. The configuration entries for databases are:

      file/volume (enstoredb) – database
      accounting – accounting_server
      drivestat – drivestat_server
      operation – no dedicated enstore server

Described here installation procedure is based on the assumption that the installation starts on the node where **enstore configuration server** will run – enstore head node.

- enstoredb
  check what node enstoredb will be running on:
  [enstore@head_node ~]$ enstore conf --show database
  {'db_dir': '/diskb/enstore/enstore-db-backup',
   'db_host': 'db_node.fnal.gov',
   'db_port': 8888,

```
                'dbarea': '/diskb/enstore/enstore-db',
                'dbhost': 'db_node.fnal.gov',
                'dbname': 'enstoredb',
                'dbport': 8888,
                'dbserverowner': 'products',
                'dbuser': 'enstore',
                'jou_dir': '/diskb/enstore/enstore-journal',
                'status': ('ok', None)}
```
According to the output enstoredb will be running on host db_node.fnal.gov The
following needs to be done as user "root" with enstore environment (which can be
done by sourcing ~enstore/site_specific/config/setup-enstore for instance).
Create and start enstoredb:
 [root@db_node]# python $ENSTORE_DIR/sbin/create_database.py enstoredb

Add entries to dbarea/pg_hba.conf:
```
host enstoredb enstore        <dbhost_IP>/32              trust
host enstoredb enstore_reader <dbhost_IP>/32              trust
```

Update DB :
[root@db_node]# python $ENSTORE_DIR/sbin/update_database_schema.py enstoredb
2013-08-22 13:53:40 : Executing command downloadXml --dbms=postgres
--host=db_node.fnal.gov --dbname=enstoredb --port=8888 --user=enstore
diffxml2ddl --dbms=postgres /opt/enstore/databases/schemas/xml/enstoredb.xml schema.xml
> enstoredb_diff_tmp.sql
2013-08-22 13:53:40 : successfully created diff DDL file: enstoredb_update.sql
2013-08-22 13:53:40 : examine the content of this file
2013-08-22 13:53:40 : after that apply to database like so:
2013-08-22 13:53:40 : psql -h db_node.fnal.gov -p 8888 -U enstore enstoredb -f
enstoredb_update.sql
[root@db_node]# psql -h db_node.fnal.gov -p 8888 -U enstore enstoredb -f enstoredb_update.sql

For accounting, drivestat, operation (just FNAL) repeat steps done for enstoredb,
replacing "enstoredb" with " accounting", "drivestat", "operation".

2.1.6.3 Enstore cron jobs
Install cron jobs defined in enstore configuration file (this must be done on **each**
server node in enstore configuration):
[root@head_node]# $ENSTORE_DIR/tools/install_crons.py

2.1.7 Starting, restarting and stopping enstore

As described in " Configuring Enstore" Enstore starts automatically on boot on each Enstore machine, but they need to get (re)booted in a certain order:

1. Head node.
2. Rest of servers.
3. Movers.

Enstore can also be started, restarted, or stopped by corresponding enstore command in two different ways (NOTE! You must be user "enstore"!):

1. Locally on each enstore server:
   [enstore@head_node ~]$ enstore start
   Outputs actions (if no output there is nothing to start on this node).

2. Remotely on a selected server, group of servers or the whole set of servers. This command uses farmlets:
   [enstore@head_node ~]$ enstore Estart
   This command starts enstore on all nodes in the order, defined in farmlet "enstore"
   [enstore@head_node ~]$ enstore Estart servers
   This command starts enstore on all nodes in the order, defined in farmlet "servers"

To check what enstore related processes are running you can use the following commands:
EPS – to see what entsore related processes run locally
enstore EPS - to see what entsore related processes run on all enstore nodes
enstore EPS servers - to see what entsore related processes run on servers defined in farmlet "servers".

2.1.8 What next?

Note! If you change enstore configuration file you must reload configuration by the following command:
 # enstore config –config $ENSTORE_CONFIG_FILE –load
Note!! All commands must be done as user enstore if not specified differently.
Note!!! All enstore commands have –help option. If you do not know what can be done just type enstore and investigate what options are available. Also refer to "Enstore Admin Guide" (can be found in /opt/enstore/doc/guides/Enstore_Administrator_Guide.pdf)

To start writing data you need to declare volumes in robotic library to enstore. This is done with the following command on enstore Volume Clerk host:
 # enstore vol – add <VOLUME_NAME> <LIBRARY> <STORAGE_GROUP> <FILE_FAMILY> <WRAPPER> <MEDIA_TYPE>

<u>\<VOLUME_BYTE_CAPACITY\></u>
where:
VOLUME_NAME – name of tape volume as known to the robotic library
LIBRARY – name of enstore library (same as namespace library tag)
STORAGE_GROUP – storage group to which this tape is assigned (or none)
 FILE_FAMILY – file family to which this tape is assigned (or none)
WRAPPER - "cpio_odc" or "cern" (or none)
MEDIA_TYPE – type of media as known to the robotic library
VOLUME_BYTE_CAPACITY – capacity of tape in bytes
After volumes are added to enstore you can start writing to them. Please refer to
"Enstore User Guide" and "Enstore Admin Guide"

Chapter 3: Enstore Commands

Enstore provides commands that allow you to communicate with various components of the system. The basic syntax of all Enstore commands is

```
% enstore <command> [--option [argument] ...]
```

All options start with a double dash (`--`). The return codes are 0 (zero) for success, non-zero for failure (currently all failures return number 1).

3.1 enstore info

As of encp v3_2, the command **enstore info** supersedes **enstore file** and **enstore volume**. The developers may decide to remove these latter two commands in future versions of **encp**.

This command communicates with the File Clerk (see section 8.1 *File Clerk*) and the Volume Clerk (see section 8.2 *Volume Clerk*). It returns information about specified file(s) or volume(s).

Syntax:

```
% enstore info [--option [argument] ... ]
```

Options:

-h, --help
Prints the options (i.e., prints this message). Example:

```
    $ enstore info --help

    Usage:
            info [ -h --bfid= --help --list= --ls-active= --usage ]
            --bfid <BFID>          get info of a file
            --file <PATH|PNFSID|BFID|VOL:LOC>  get info on a file
            --find-all-copies <BFID>  find all copies of this file
            --find-copies <BFID>  find the immediate copies of this file
              --find-duplicates <BFID>   find all duplicates related to
    this file
              --find-original <BFID>  find the immediate original of this
    file
              --find-the-original <BFID>  find the very first original of
    this file
            --gvol <VOLUME_NAME>  get info of a volume in human readable
    time
                                   format
       -h, --help                 print this message
            --just <VOLUME_NAME>  used with --pvols to list problem
            --list <VOLUME_NAME>  list the files in a volume
            --ls-active <VOLUME_NAME>  list active files in a volume
            --ls-sg-count         list all sg counts
            --pvols               list all problem volumes
```

```
                   --show-bad         list all bad files
                   --show-copies <BFID>  all copies of a file
                   --show-file <BFID>    show info of a file
                   --usage            print short help message
                   --vol <VOLUME_NAME>   get info of a volume
                   --vols             list all volumes
```

--bfid <BFID>

Returns information (metadata) about the file corresponding to the specified bfid.

You can get the bfid of a file from the **enstore pnfs --bfid <FILE_NAME>** command (section Error: Reference source not found); get the filename from searching PNFS namespace.

Example:

```
   $ enstore info --bfid CDMS105770745000000

      {'bfid': 'CDMS105770745000000',
       'complete_crc': 1191066979L,
       'deleted': 'no',
       'drive': 'stkenmvr7a:/dev/rmt/tps0d1n:4560000022',
       'external_label': 'VO3222',
       'location_cookie': '0000_000000000_0005661',
       'pnfs_mapname': '',
                                                  'pnfs_name0':
      '/pnfs/fs/usr/test/xyz/srmtest/ar017983.0001phys_10',
       'pnfsid': '00050000000000000190EA8',
       'pnfsvid': '',
       'sanity_cookie': (65536L, 3203712884L),
       'size': 197354833L}
```

--find-all-copies <BFID>

Report all the file BFIDs that are duplicates or multiple copies of the specified file BFID, including the specified file itself.

Multiple copies are extra copies of a file written by the same encp process that wrote the original file. Duplicated copies are

multiple copies that were written to another tape by the Enstore administrators some time after the original was written to tape.

Here are two files where the first BFID shone is the original and the second one is a duplicate of the first.

```
$ enstore info --find-all-copies
  CDMS115788240600000

     CDMS115788240600000
     CDMS123800281300002

$ enstore info --find-all-copies
  CDMS123800281300002

     CDMS123800281300002
```

See also **encp--copies**; **enstore info --find-copies**, **--find-duplicates**, **--find-original**, --**find-the-original** and **--show-copies** for more information.

--find-copies <BFID>
Report the file BFIDs that are immediate duplicates or multiple copies of the specified file BFID, including the specified file itself. <u>Multiple copies</u> are extra copies of a file written by the same encp process that wrote the original file. <u>Duplicated copies</u> are multiple copies that were written to another tape by the Enstore administrators some time after the original was written to tape.

Here are two files where the first BFID shone is the original and the second one is a duplicate of the first.

```
$ enstore info --find-copies CDMS115788240600000

     CDMS123800281300002

$ enstore info --find-copies CDMS123800281300002
```

See also **encp--copies**; **enstore info --find-all-copies**, **--find-duplicates**, **--find-original**, --**find-the-original** and **--show-copies** for more information.

--find-duplicates <BFID>
Report the file BFIDs that are associated as a duplicate or multiple copy file of the specified file BFID, including the specified file itself. This command has the same effect as:

$ enstore info --find-all-copies `enstore info --find-the-original <BFID>` <u>Multiple copies</u> are extra copies of a file written by the same encp process that wrote the original file. <u>Duplicated copie</u>s are multiple copies that were written to another tape by the Enstore administrators some time after the original was written to tape.

Here are two files where the first BFID shone is the original and the second one is a duplicate of the first.

```
$ enstore info --find-duplicates
  CDMS115788240600000

     CDMS115788240600000
     CDMS123800281300002

$ enstore info --find-duplicates
  CDMS123800281300002

     CDMS115788240600000
     CDMS123800281300002
```

See also **encp--copies**; **enstore info --find-all-copies**, **--find-copies**, **--find-original**, --**find-the-original** and **--show-copies** for more information.

--find-original <BFID>
Report the file BFIDs that is the immediate original of the specified duplicate or multiple copy file BFID. <u>Multiple copies</u> are extra copies of a file written by the same encp process that wrote the original file. <u>Duplicated copie</u>s are multiple copies that were written to another tape by the Enstore administrators some time after the original was written to tape.

Here are two files where the first BFID shone is the original and the second one is a duplicate of the first.

```
$ enstore info --find-original
  CDMS115788240600000

     None

$ enstore info --find-original
  CDMS123800281300002

     CDMS115788240600000
```

See also **encp--copies**; **enstore info --find-all-copies**, **--find-copies**, **--find-duplicates**, --**find-the-original** and

---

--show-copies for more information.

--find-the-original <BFID>
Report the file BFIDs that is the overall original of the specified duplicate or multiple copy file BFID. <u>Multiple copies</u> are extra copies of a file written by the same encp process that wrote the original file. <u>Duplicated copie</u>s are multiple copies that were written to another tape by the Enstore administrators some time after the original was written to tape.

Here are two files where the first BFID shone is the original and the second one is a duplicate of the first.

```
$ enstore info --find-the-original
  CDMS115788240600000

    CDMS115788240600000

$ enstore info --find-the-original
  CDMS123800281300002

    CDMS115788240600000
```

See also **encp--copies**; **enstore info --find-all-copies**, **--find-copies**, **--find-duplicates**, --**find-original** and**--show-copies** for more information.

This option may be used in any of four ways to return the same information, depending on what information you initially know about the file.

--file <PATH>

--file <BFID> (equivalent to **enstore info --bfid <BFID>**)

--file <PNFSID>

--file <VOLUME:LOCATION>

Returns information on the specified file.

This example uses the path:

```
$ enstore info --file /pnfs/test/NULL/1KB_251

    {'bfid': 'WAMS111453908000000',
     'complete_crc': 0L,
     'deleted': 'no',
     'drive': 'rain:/dev/null:0',
     'external_label': 'NULL01',
     'gid': 6209,
     'location_cookie': '0000_000000000_0000609',
     'pnfs_name0': '/pnfs/test/NULL/1KB_251',
     'pnfsid': '000100000000000000056258',
     'sanity_cookie': (1024L, 0L),
     'size': 1024L,
     'uid': 5744}
```

The file could also be specified as one of the following (BFID, PNFSID or VOLUME:LOCATION (external_label:location_cookie):

```
$ enstore info --file WAMS111453908000000

$ enstore info --file 000100000000000000056258

$ enstore info --file
  NULL01:0000_000000000_0000609
```

--gvol <VOLUME_NAME>

This is just like **enstore info --vol <VOLUME_NAME>**, except that this one prints human-readable time fields (e.g., "declared", "first_access" and "last_access" fields). Example:

```
$ enstore info --gvol VO3332

    {'blocksize': 131072,
     'capacity_bytes': 64424509440L,
     'declared': 'Wed Jan 16 16:13:57 2002',
     'eod_cookie': '0000_000000000_0000044',
     'external_label': 'VO3332',
     'first_access': 'Fri May 10 12:59:35 2002',
     'last_access': 'Mon Oct 27 22:35:45 2003',
     'library': '9940',
     'media_type': '9940',
     'non_del_files': 43,
     'remaining_bytes': 1785262080L,
     'sum_mounts': 234,
     'sum_rd_access': 213,
     'sum_rd_err': 0,
     'sum_wr_access': 43,
     'sum_wr_err': 0,
     'system_inhibit': ['none', 'full'],
     'user_inhibit': ['none', 'none'],
     'volume_family': 'cms.objy_data_files.cpio_odc',
     'wrapper': 'cpio_odc'}
```

--just

Used with **--pvols** to list problem. See **enstore info --pvols**.

--list <VOLUME_NAME>
Lists the files in the specified volume with their volume name, bfid, size, location (file number) on volume, delete flag, and the original filename in pnfs.

You can get the volume name from the **enstore pnfs** command, using either **--xref** or **--layer** (section Error: Reference source not found), or from the "external_label" field of the **enstore info --bfid <BFID>** command (shown above).

This replaces both **enstore file --list <VOLUME_NAME>** and **enstore volume --list <VOLUME_NAME>**.

Example:

```
$ enstore info --list VO3222

    label  bfid                size   location_cookie      delflag
    original_name

    VO3222 CDMS106503213600000 983803 0000_000000000_0011536 deleted
    /pnfs/fs/usr/eagle/dcache-tests/274.dcache_page_p_27750
```

(This shows one of many lines appearing in the real output, and is reformatted to two lines for readability.)

--ls-active <VOLUME_NAME>
Lists active files in a volume.

You can get the volume name from the **enstore pnfs** command, using either **--xref** or **--layer** (section Error: Reference source not found), or from the "external_label" field of the **enstore info --bfid <BFID>** command (shown above).

Example:

```
$ enstore info --ls-active VO3222

    /pnfs/fs/usr/eagle/dcache-tests/101.dcache_page_a_24401
    /pnfs/fs/usr/eagle/dcache-tests/101.dcache_page_24401
    /pnfs/fs/usr/test/stress-test/myfile1
    /pnfs/fs/usr/test/stress-test/myfile3
    /pnfs/fs/usr/test/stress-test/file128m-11
    ...
```

--ls-sg-count <VOLUME_NAME>
Lists allocated tape counts by library and by storage group. If "storage group" has value "none", the negative number under "allocated" gives the number of tapes that are available in the robot, but not yet assigned to a storage group.
Example:

```
$ enstore info --ls-sg-count VO3332

        library     storage group  allocated
     =======================================
          ...
          9940             ktev        189
          9940             lqcd        150
          9940         miniboone       132
          9940            minos        109
          9940             none        -13
          9940           patriot        20
          9940             sdss        608
          9940             test         28
          9940           theory         70
       CD-9940B             cms        129
          ...
```

--pvols [--just <VOLUME_1> <VOLUME_2> ...]
Without **--just**, this lists all problem volumes. With **--just** followed by a space-separated list of volume names, it lists only the problem volumes among the given list.
The columns returned are: volume name, primary status, primary status time, secondary status, secondary status time. (The time fields are relatively new; not all volumes will display them.)
Example:

```
$ enstore info --pvols

    ==== readonly
    LEGL10           none         *     readonly 0913-1540
    LEGL98           none         *     readonly 0819-2329
    ...
    ==== full
    ...
    VO4845           none         *        full         *
    VO4846           none 1023-1032        full         *
    VO4847           none         *        full         *
    VO4848           none         *        full         *
    VO4849           none         *        full         *
    VO4850           none         *        full 1016-2315
    VO4851           none         *        full 1017-0409
    ...
    $ enstore volume --pvols --just VO3332
```

```
    (no sample output available)
```

--show-bad

Lists all files that are currently unavailable due to media
problems. When a tape problem is discovered, the tape is sent to
the vendor for file recovery. In the interim, a cloned tape is made
available to users, with the bad files marked. This command
option lets you list the bad files. The output lists the tape
number, BFID, file size in bites, and pnfs path of file.
Example:

```
$ enstore info --show-bad

   ...
   VO0053 CDMS105770745000000 95530315
    /pnfs/fs/usr/xyz/my_data/2004-4/.bad.F000xyz43_0000.mdaq.root
   ...
```

We show only one output line, and it is displayed on two lines for
readability. Notice the ".bad." at the front of the filename; this
is how the bad files are marked.

--show-copies <BFID>
Report the file BFIDs that are associated as a duplicate or
multiple copy file of the specified file BFID, including the
specified file itself. This command has a similar effect to:
$ enstore info --find-duplicates <BFID>
but outputs more information. <u>Multiple copies</u> are extra copies of
a file written by the same encp process that wrote the original
file. <u>Duplicated copies</u> are multiple copies that were written to
another tape by the Enstore administrators some time after the
original was written to tape.

Here are two files where the first BFID shone is the original and
the second one is a duplicate of the first. The columns are the
BFID, Storage Group, Library, Media Type, Label, Deleted
Status, Size, CRC, PNFS ID and Original PNFS Path.

```
$ enstore info --show-copies CDMS115788240600000

    CDMS115788240600000        ENDEV       LTO3       LTO3       TEST85
0000_000000000_0000001              no      5290065920       3951343656
00020000000000000003DD20
/pnfs/data2/sl8500/fcdfcaf566/fcdfcaf566_4000_1157882036.data
    CDMS123800281300002        ENDEV       LTO3       LTO3       TEST21
0000_000000000_0000001              no      5290065920       3951343656
0002000000000000000004C4D8
/pnfs/data2/sl8500/fcdfcaf566/fcdfcaf566_4000_1157882036.data

$ enstore info --show-copies CDMS123800281300002

    CDMS115788240600000        ENDEV       LTO3       LTO3       TEST85
0000_000000000_0000001              no      5290065920       3951343656
00020000000000000003DD20
/pnfs/data2/sl8500/fcdfcaf566/fcdfcaf566_4000_1157882036.data
    CDMS123800281300002        ENDEV       LTO3       LTO3       TEST21
0000_000000000_0000001              no      5290065920       3951343656
0002000000000000000004C4D8
/pnfs/data2/sl8500/fcdfcaf566/fcdfcaf566_4000_1157882036.data
```

See also **encp--copies**; **enstore info --find-all-copies**,
--find-copies, **--find-original**, --**find-the-original**, **--show-copies**
and **--show-file** for more information.

--show-file <BFID>
Display information about the file specified by its BFID and
some information about the volume where the file is located. The
columns are the BFID, Storage Group, Library, Media Type,
Label, Deleted Status, Size, CRC, PNFS ID and Original PNFS
Path.

```
$ enstore info --show-file GCMS125028149900000

    GCMS125028149900000          gcc        LTO3       LTO3       TEST11
0000_000000000_0006692             yes               2532       3516567016
```

```
        0001000000000000045BC440
        /pnfs/fnal.gov/testers/NULL/20090814-213844-23461-0.txt
```

See also **enstore info --bfid** and**--show-copies** for more information.

--usage
Prints short help message. Example:

$ enstore info --usage

```
Usage:
        info [ -h --bfid= --help --list= --ls-active= --usage ]
```

--vol <VOLUME_NAME>
Returns detailed information about specified volume
Example:

$ enstore info --vol VO3332

```
{'blocksize': 131072,
 'capacity_bytes': 64424509440L,
 'declared': 1011219237.849051,
 'eod_cookie': '0000_000000000_0000044',
 'external_label': 'VO3332',
 'first_access': 1021053575.259737,
 'last_access': 1067315745.238969,
 'library': '9940',
 'media_type': '9940',
 'non_del_files': 43,
 'remaining_bytes': 1785262080L,
 'sum_mounts': 234,
 'sum_rd_access': 213,
 'sum_rd_err': 0,
 'sum_wr_access': 43,
 'sum_wr_err': 0,
 'system_inhibit': ['none', 'full'],
 'user_inhibit': ['none', 'none'],
 'volume_family': 'cms.objy_data_files.cpio_odc',
 'wrapper': 'cpio_odc'}
```

```
--vols
or
--vols <VOLUME_STATUS>
or
--vols <KEY> <VALUE>
```
Lists all volumes with their available space, the system inhibits, the library, the volume family (period-separated concatenation of storage group, file family and file family wrapper) and any comments.

The VOLUME_STATUS argument is optional. If left off, all volumes are listed. Possible values for this argument include: NOACCESS, NOTALLOWED, full, read_only, migrated.

The KEY option accepts: storage_group, library and media_type

Example:

```
$ enstore info --vols

    label           avail.     system_inhibit     library     vol_family
    comment

    ...

    VO0053          1.19GB     (none      full   )            eagle
    cms.objy_data_files.cpio_odc

    VO0054          0.51GB     (none      full   )            eagle
    cms.objy_data_files.cpio_odc

    VO0055          0.17GB     (none      full   )            eagle
    theory.theory-canopy-C.cpio_odc

    VO0056          0.65GB     (none      full   )            eagle
    theory.theory-canopy-D.cpio_odc

    ...
```

3.2 enstore library

This command communicates with the Library Manager (see section 8.3 *Library Manager*). You can use it to get information pertaining to a particular Library Manager. Use the online monitoring pages (see Chapter 10: *Monitoring Enstore on the Web*) to find the library manager of interest.
Syntax:

```
% enstore library [--option [argument] ... ] <library>
```

The **<library>** argument is required except when using the **--help** option; the "**.library_manager**" portion of the library name is optional.
Options:

-h, --help
Prints this message (i.e., prints the options). Example:

```
$ enstore library --help

    Usage:
            library [OPTIONS]... library


            --get-asserts <library> print sorted lists of pending volume
    asserts
              --get-queue <HOST_NAME>   print queue submitted from the
    specified host.
                                      If empty string specified, print the
    whole queue
            --get-suspect-vols     print suspect volume list
              --get-work-sorted        print sorted lists of pending and
    active requests
        -h, --help               prints this messge
            --usage              prints short help message
```

--get-asserts <LIBRARY>
Prints sorted lists of pending volume asserts for specified library.
Example:

```
$ enstore library --get-asserts
  9940.library_manager

    Pending assert requests: 0
    Active assert requests: 0
    {'status': ('ok', None)}
```

--get-queue <HOST_NAME> <LIBRARY>

Prints queue submitted from the specified **encp** client host. Both arguments are required. If quoted empty string is specified for host name, it prints the whole queue (for all hosts). Examples:

```
$ enstore library --get-queue stkensrv3
  9940.library_manager

    Pending write requests
    Active requests
    Pending read requests:  0
    Pending write requests:  2
    Active read requests:  0
    Active write requests:  0
    {'status': ('ok', None)}
```

The top two lines tell us that there are no pending or active transfers involving stkensrv3 for the 9940 library manager. The 4th line tells us there are 2 pending write requests for this library manager from hosts other than stkensrv3.

If all hosts are specified (the next example), the command returns the fields: host name, library manager, username (of **encp** request), input filename, and output filename for each pending and/or active request (3 shown here), and ends with a summary:

```
$ enstore library --get-queue ""
  9940.library_manager

    Active requests
    fnsimu2 9940.library_manager lixn
       /pnfs/btev/geant2003/xiaonan/dstar_xiaonan_1.evt.gz
       /scr/bphys6/lixn/dstar_xiaonan_1.evt.gz M 9944
    fsgi01 9940.library_manager rschultz
       /usr/bdms/rschultz/fl_066_uplsr7/fl_ed_066_uplsr7.ldhi
       /pnfs/BDMS/lens/fl_066_uplsr7/fl_ed_0663
    fnsfh 9940.library_manager minfarm
       /export/stage02_minos/C00040259_0000.tdaq.root
       /pnfs/minos/caldet_reco/tdaq_data/2002-09/C0004027
    Pending read requests:  0
    Pending write requests:  0
    Active read requests:  1
    Active write requests:  2
    {'status': ('ok', None)}
```

--get-suspect-vols <LIBRARY>

Prints suspect volume list for specified library manager. See 10.6.1 *Suspect Volumes*. Example:

```
$ enstore library --get-suspect-vols
  9940.library_manager

    [{'movers': ['994071.mover'], 'external_label': 'VO4523',
```

```
          'time':    1067290586.907726},    {'movers':    ['994051.mover',
          '994061.mover', ']
```

--get-work-sorted <LIBRARY>
Prints sorted lists of pending and active requests. It sorts by
queue. Example:

> **$ enstore library --get-work-sorted**
> **9940.library_manager**
>
> {'write_queue': [], 'read_queue': [], 'admin_queue': []}
>
> [{'status': ('ok', None), 'vc': {'status': ('ok', None),
>
> 'declared': 1011741604.130481, 'si_time': [1041612783.99499, 0],
> 'blocksiz]

3.3 enstore monitor

This command communicates with the Monitor Server (see Chapter 10:
Monitoring Enstore on the Web) to get network speed information.
On machines with an `enstore.conf` file (see Appendix A: *Network
Control*), the **enstore monitor** command uses the routing already
established there. If **enstore monitor** set up its own, it would interfere
with the routes currently in use.
Syntax:

% enstore monitor [--option [argument] ...]

-h, --help
Prints this message (i.e., prints the options). Example:

```
$ enstore monitor -h
Usage:
        monitor [ -h --help --host= --usage --verbose= ]

   -h, --help              prints this messge
       --host <HOSTIP>     selects a single host
       --port <PORT>       selects a port
       --usage             prints short help message
       --verbose <VERBOSE>  print out information.
```

--host [HOST_NAME or IP_ADDRESS]
Returns network rate for the specified host (Enstore node). If you
don't specify host, it runs the command for all hosts. Example
below shows results for a single host. Example:

```
$ enstore monitor --host stkensrv3

    Trying stkensrv3.fnal.gov
    Network  rate  measured  at  11.33  MB/S  recieving  and  11.1  MB/S
    sending.
```

--port <PORT>
Selects the specified port. If you don't specify port, it runs the
command for the default port.

--verbose <INTEGER_VALUE>
This command is used to help find and fix network problems. It
prints detailed information about actions taken. The higher the
number you give as an argument, the more info displayed.
Example:

```
$ enstore monitor --host stkensrv3 --verbose 20

    6  Tue  Oct  28  10:48:13  2003  msc  called  with  args:  ['monitor',
    '--host', 'stkensrv3', '--verbose=20']
    13 Tue Oct 28 10:48:13 2003 Get monitor_server config info from
    server
    Trying stkensrv0.fnal.gov
    13 Tue Oct 28 10:48:13 2003 Get None config info from server
    13 Tue Oct 28 10:48:13 2003 Get None config info from server
    13 Tue Oct 28 10:48:13 2003 Get log_server config info from server
    13 Tue Oct 28 10:48:13 2003 Get log_server config info from server
    13 Tue Oct 28 10:48:13 2003 Get None config info from server
    13  Tue  Oct  28  10:48:13  2003  Get  alarm_server  config  info  from
    server
    ...
    10 Tue Oct 28 10:48:14 2003 Connecting to monitor server.
    10 Tue Oct 28 10:48:14 2003 Obtaining error status for data socket.
    10 Tue Oct 28 10:48:15 2003 Get the final dialog rate information.
    Network  rate  measured  at  11.34  MB/S  recieving  and  11.23  MB/S
    sending.
```

3.4 enstore pnfs

Enstore has a **pnfs** command that allows you to retrieve a variety of
information, as listed in the option table below. Off-site users cannot
mount /pnfs, and therefore cannot run this command.

☞ Using this command to perform PNFS manipulations and/or change
PNFS tags is restricted to Enstore administrators and/or their designated
gurus, and is covered in Appendix B: *Changing PNFS Tags*.

Syntax:

```
% enstore pnfs [--option [argument] ... ]
```

--help
List the options for the **enstore pnfs** command. Example:

```
% enstore pnfs --help

Usage:
        pnfs [OPTIONS]...

        --bfid <FILENAME>    lists the bit file id for file
        --cat <FILENAME> [LAYER]  see --layer
        --file-family [FILE_FAMILY] gets file family tag, default; sets
   file
                              family tag, optional
         --file-family-width [FILE_FAMILY_WIDTH]  gets file family width
   tag,
                              default; sets file family tag, optional
          --file-family-wrapper [FILE_FAMILY_WRAPPER]  gets file family
   width tag,
                              default; sets file family tag, optional
        --filesize <FILE>    print out real filesize
      -h, --help             prints this messge
        --info <FILENAME>    see --xref
        --layer <FILENAME> [LAYER]  lists the layer of the file
         --library [LIBRARY]   gets library tag, default; sets library
   tag,
                              optional
        --tag <TAG> [DIRECTORY]  lists the tag of the directory
         --tagchmod <PERMISSIONS> <TAG>  changes the permissions for the
   tag; use
                              UNIX chmod style permissions
          --tagchown <OWNER> <TAG>   changes the ownership for the tag;
   OWNER can
                              be 'owner' or 'owner.group'
        --tags [DIRECTORY]   lists tag values and permissions
        --usage              prints short help message
        --xref <FILENAME>    lists the cross reference data for file
```

--bfid <FILE_NAME>
Returns the BFID of the file; select file name to specify from within
pnfs space and use relative/absolute path as needed.
Example:

```
$ enstore pnfs --bfid /pnfs/mist/zuu/100MB_002

    WAMS104102942800000
```

--cat <PATH_TO_FILE> [LAYER]
--cat is an alias for --layer; see --layer.

--file-family

Prints the file family name associated with the current pnfs directory. Example:

```
$ enstore pnfs --file-family

    dcache
```

--file-family-width

Prints the file family width associated with the current pnfs directory. Example:

```
$ enstore pnfs --file-family-width

    1
```

--file-family-wrapper

Prints the file family wrapper associated with the current pnfs directory. Example:

```
$ enstore pnfs --file-family-wrapper

    cpio_odc
```

--filesize <PATH_TO_FILE>

Prints the real filesize in bytes; useful for files of size greater than (2G-1) bytes, since PNFS stores file size as 1 in this case. Example:

```
$ enstore pnfs --filesize a01

    24198
```

--info <PATH_TO_FILE>

Prints information about the file, this is an alias for the --xref option. See --xref.

--layer <PATH-TO-FILE> <LAYER>

Prints information about the file. Layer 0 is used internally by pnfs and it can't be viewed. Layer 1, the default, gives the file's BFID. Layer 2 is used by dCache. Layers 3, 5, 6, 7 are not currently used. Layer 4 produces output equivalent to --xref, but returns info without field labels.

The option --cat is an alias for this option.

Examples:

Layer 1 gives BFID (default):

```
$ enstore pnfs --layer a01

    CDMS105889726300000

$ enstore pnfs --layer a01 1

    CDMS105889726300000
```

Layer 2 is used for dCache:

```
$ enstore pnfs --layer a01 2

  2,0,0,0.0,0.0
  :c=1:d15ef6a3;l=554423;
  w-fcdfdata018-1
```

The file has a version1 crc of c=1:d15ef6a3, it has a length l=554423, and it is in pool w-fcdfdata018-1.

```
$ enstore pnfs --layer a01 2

  2,0,0,0.0,0.0
  :
```

Layer 4 gives --xref output (see --xref):

```
$ enstore pnfs --layer a01 4

    VO3222
    0000_000000000_0006264
    24198
    dcache
    /pnfs/fs/usr/test/xyz/srmtest/test_1_1/a01

    0005000000000000000191030

    CDMS105889726300000
    stkenmvr5a:/dev/rmt/tps3d1n:4560000022
```

--tags [DIRECTORY]
List the tag values of specified PNFS directory (if no directory
argument, it lists tags for current working directory (cwd or pwd)).
Example:

$ pwd

```
/pnfs/test/xyz/srmtest/test_1_1
```

$ enstore pnfs --tags

```
.(tag)(file_family) = dcache
.(tag)(file_family_width) = 1
.(tag)(file_family_wrapper) = cpio_odc
.(tag)(library) = 9940
.(tag)(storage_group) = test
-rw-rw-r--   11 root     sys            6 Jul 26  2001
                 /pnfs/test/xyz/srmtest/test_1_1/.(tag)(file_family)
-rw-rw-r--   11 root     sys            1 May  5  2000
                            /pnfs/test/xyz/srmtest/test_1_1/.(tag)
(file_family_width)
-rw-rw-r--   11 root     sys            8 May  5  2000
                            /pnfs/test/xyz/srmtest/test_1_1/.(tag)
(file_family_wrapper)
-rw-rw-r--   11 root     sys            4 Jul  3 10:59
                 /pnfs/test/xyz/srmtest/test_1_1/.(tag)(library)
-rw-r--r--   11 root     sys            4 Jul 26  2001
                 /pnfs/test/xyz/srmtest/test_1_1/.(tag)(storage_group)
```

(minor reformatting done to enhance readability)

--xref <FILE_NAME>
List cross-reference information (metadata) for specified file. (--info is an alias for --xref.) The information includes:

- volume: storage media volume label
- location cookie: file position on tape (the number of the file on tape)
- size: file size in bytes
- file family: file family
- original name: original name in /pnfs before any move/copy command issued; i.e., the destination filename given in the encp command used to copy the file to Enstore
- map file: obsolete, but some older files may have a value here
- pnfsid file: unique id for the file as assigned by PNFS
- pnfsid map: obsolete, but some older files may have a value here
- bfid: unique id for the file as assigned by Enstore (matches layer 1)
- origdrive: id of drive used when file was written to media (files generated prior to 10/2000, encp v2_5 or earlier, will not have a value here)
- crc: CRC of the file (appears for files after 10/2003, using encp v3_1 or greater)

Example:

```
$ enstore pnfs --xref a01

    volume: VO3222
    location_cookie: 0000_000000000_0006264
    size: 24198
    file_family: dcache
    original_name: /pnfs/fs/usr/test/xyz/srmtest/test_1_1/a01
    map_file:
    pnfsid_file: 000500000000000000191030
    pnfsid_map:
    bfid: CDMS105889726300000
    origdrive: stkenmvr5a:/dev/rmt/tps3d1n:4560000022
    crc: unknown
```

--library
Returns the value of the library tag (the virtual library associated with files in the directory) for the current pnfs directory. Example:

```
$ enstore pnfs --library

    9940
```

3.5 enstore file (deprecated)

This command has been deprecated for users as of **encp** v3_2, and (along with `enstore volume`) replaced with `enstore info` (see section Error: Reference source not found).

This command communicates with the File Clerk (see section 8.1 *File Clerk*). It returns information about a specified file or files on a specified volume.

Syntax:

```
% enstore file [--option [argument] ... ]
```

Options:

-h, --help
Prints the options (i.e., prints this message). Example:

```
$ enstore file --help

    Usage:
            file [ -h --bfid= --help --list= --ls-active= --usage ]

            --bfid <BFID>        get info of a file
        -h, --help               print this message
            --list <VOLUME_NAME>  list the files in a volume
            --ls-active <VOLUME_NAME>  list active files in a volume
            --show-bad           lists all bad files
            --usage              print short help message
```

--bfid <BFID>
Returns information (metadata) about the file corresponding to the specified bfid.
You can get the bfid of a file from the `enstore pnfs --bfid <FILE_NAME>` command (section Error: Reference source not found); get the filename from searching PNFS namespace.
Example:

```
$ enstore file --bfid CDMS105770745000000

    {'bfid': 'CDMS105770745000000',
     'complete_crc': 1191066979L,
     'deleted': 'no',
     'drive': 'stkenmvr7a:/dev/rmt/tps0d1n:4560000022',
     'external_label': 'VO3222',
     'location_cookie': '0000_000000000_0005661',
     'pnfs_mapname': '',
                                            'pnfs_name0':
    '/pnfs/fs/usr/test/xyz/srmtest/ar017983.0001phys_10',
     'pnfsid': '000500000000000000190EA8',
     'pnfsvid': '',
```

```
        'sanity_cookie': (65536L, 3203712884L),
        'size': 197354833L}
```

--list <VOLUME_NAME>

Lists the files in the specified volume with their volume name, bfid, size, location (file number) on volume, delete flag, and the original filename in pnfs.

You can get the volume name from the **enstore pnfs** command, using either **--xref** or **--layer** (section Error: Reference source not found), or from the "external_label" field of the **enstore file --bfid <BFID>** command (shown above).

The **enstore info --list <VOLUME_NAME>** is an alias for this command.

Example:

```
$ enstore file --list VO3222

    label  bfid              size   location_cookie      delflag
    original_name

    VO3222 CDMS106503213600000 983803 0000_000000000_0011536 deleted
    /pnfs/fs/usr/eagle/dcache-tests/274.dcache_page_p_27750
```

(This shows one of many lines appearing in the real output, and is reformatted to two lines for readability.)

--ls-active <VOLUME_NAME>

Lists active files in a volume.

You can get the volume name from the **enstore pnfs** command, using either **--xref** or **--layer** (section Error: Reference source not found), or from the "external_label" field of the **enstore file --bfid <BFID>** command (shown above).

Example:

```
$ enstore file --ls-active VO3222

    /pnfs/fs/usr/eagle/dcache-tests/101.dcache_page_a_24401
    /pnfs/fs/usr/eagle/dcache-tests/101.dcache_page_24401
    /pnfs/fs/usr/test/stress-test/myfile1
    /pnfs/fs/usr/test/stress-test/myfile3
    /pnfs/fs/usr/test/stress-test/file128m-11
    ...
```

--show-bad

Lists all files that are currently unavailable due to media problems. When a tape problem is discovered, the tape is sent to the vendor for file recovery. In the interim, a cloned tape is made available to users, with the bad files marked. This command option lets you list the bad files. The output lists the tape number, BFID, file size in bites, and pnfs path of file.

Example:

```
$ enstore info --show-bad

    ...
    VO0053 CDMS105770745000000 95530315
     /pnfs/fs/usr/xyz/my_data/2004-4/.bad.F000xyz43_0000.mdaq.root
    ...
```

We show only one output line, and it is displayed on two lines for readability. Notice the ".bad." at the front of the filename; this is how the bad files are marked.

Replaced by: **enstore info --show-bad**.

--usage

Prints short help message. Example:

```
$ enstore file --usage

    Usage:
            file [ -h --bfid= --help --list= --ls-active= --usage ]
```

3.6 enstore volume (deprecated)

This command has been deprecated for users as of **encp** v3_2, and replaced (along with **enstore file**) with **enstore info** (see section Error: Reference source not found).

This command communicates with the Volume Clerk (see section 8.2 *Volume Clerk*) to return information on data volumes.

Syntax:

```
% enstore volume [--option [argument] ... ]
```

-h, --help
Prints this message (i.e., prints the options). Example:

```
$ enstore volume --help

Usage:
        volume [OPTIONS]...


        --gvol <VOLUME_NAME>  get info of a volume in human readable
time
                              format
    -h, --help                prints this messge
        --just <VOLUME_NAME>  used with --pvols to list problem
        --list <VOLUME_NAME>  list the files in a volume
        --ls-active <VOLUME_NAME>  list active files in a volume
        --ls-sg-count         list all sg counts
        --pvols               list all problem volumes
        --usage               prints short help message
        --vol <VOLUME_NAME>   get info of a volume
        --vols                list all volumes
```

--gvol <VOLUME_NAME>
This is just like **enstore volume --vol
<VOLUME_NAME>**, except that this one prints human-readable
time fields (e.g., "declared", "first_access" and "last_access"
fields). Example:

```
$ enstore volume --gvol VO3332

    {'blocksize': 131072,
     'capacity_bytes': 64424509440L,
     'declared': 'Wed Jan 16 16:13:57 2002',
     'eod_cookie': '0000_000000000_0000044',
     'external_label': 'VO3332',
     'first_access': 'Fri May 10 12:59:35 2002',
     'last_access': 'Mon Oct 27 22:35:45 2003',
     'library': '9940',
     'media_type': '9940',
     'non_del_files': 43,
     'remaining_bytes': 1785262080L,
     'sum_mounts': 234,
     'sum_rd_access': 213,
     'sum_rd_err': 0,
     'sum_wr_access': 43,
     'sum_wr_err': 0,
     'system_inhibit': ['none', 'full'],
     'user_inhibit': ['none', 'none'],
     'volume_family': 'cms.objy_data_files.cpio_odc',
     'wrapper': 'cpio_odc'}
```

--just
Used with **--pvols** to list problem. See **enstore
volume --pvols**.

--list <VOLUME_NAME>
This is an alias for the **enstore info --list
<VOLUME_NAME>** command. See section Error: Reference
source not found.

--ls-active <VOLUME_NAME>
Lists original file names of active files in a volume. Example:

```
$ enstore volume --ls-active VO3332

    /
    pnfs/cms/UserFederation/data/jetmet_production/data/Collections/jm_
    Hit601_g125_UCSD/jm02_qqh120_ll/EVD0.jet0102.DB
    /
    pnfs/cms/UserFederation/data/jetmet_production/data/TAssoc/jm_2x103
    3PUjm602_TkMu_g125_UCSD/jm02_hlt15-20/EVD11.jet0102.DB
    /
    pnfs/cms/UserFederation/data/jetmet_production/data/Digis/jm_2x1033
    PUjm602_TkMu_g125_UCSD/jm02-hlt0-15/EVD12.jet0102.DB
```

```
        /
        pnfs/cms/UserFederation/data/jetmet_production/data/Hits/jm_Hit601_
        g125_UCSD/jm02_hlt230-300/EVD12.jet0102.DB
        ...
```

--ls-sg-count <VOLUME_NAME>

Lists allocated tape counts by library and by storage group. If
"storage group" has value "none", the negative number under
"allocated" gives the number of tapes that are available in the
robot, but not yet assigned to a storage group.

Example:

```
    $ enstore volume --ls-sg-count VO3332

         library    storage group  allocated
        =======================================
              ...
           9940            ktev       189
           9940            lqcd       150
           9940       miniboone       132
           9940           minos       109
           9940            none       -13
           9940          patriot        20
           9940            sdss       608
           9940            test        28
           9940          theory        70
         CD-9940B           cms       129
              ...
```

--pvols [--just <VOLUME_1> <VOLUME_2> ...]
Without **--just**, this lists all problem volumes. With
--just followed by a space-separated list of volume names, it
lists only the problem volumes among the given list.
The columns returned are: volume name, primary status, primary
status time, secondary status, secondary status time. (The time
fields are relatively new; not all volumes will display them.)
Example:

```
$ enstore volume --pvols

    ==== readonly
    LEGL10          none        *    readonly 0913-1540
    LEGL98          none        *    readonly 0819-2329
    ...
    ==== full
    ...
    VO4845          none        *       full       *
    VO4846          none 1023-1032      full       *
    VO4847          none        *       full       *
    VO4848          none        *       full       *
    VO4849          none        *       full       *
    VO4850          none        *       full 1016-2315
    VO4851          none        *       full 1017-0409
    ...
    $ enstore volume --pvols --just VO3332
    (no sample output available)
```

--vol <VOLUME_NAME>
Returns detailed information about specified volume
Example:

```
$ enstore volume --vol VO3332

    {'blocksize': 131072,
     'capacity_bytes': 64424509440L,
     'declared': 1011219237.849051,
     'eod_cookie': '0000_000000000_0000044',
     'external_label': 'VO3332',
     'first_access': 1021053575.259737,
     'last_access': 1067315745.238969,
     'library': '9940',
     'media_type': '9940',
     'non_del_files': 43,
     'remaining_bytes': 1785262080L,
     'sum_mounts': 234,
     'sum_rd_access': 213,
     'sum_rd_err': 0,
     'sum_wr_access': 43,
     'sum_wr_err': 0,
     'system_inhibit': ['none', 'full'],
     'user_inhibit': ['none', 'none'],
     'volume_family': 'cms.objy_data_files.cpio_odc',
     'wrapper': 'cpio_odc'}
```

--vols
Lists all volumes with their available space, the system inhibits,
the library, the volume family (period-separated concatenation of
storage group, file family and file family wrapper) and any
comments.
Example:

```
$ enstore volume --vols

    label          avail.    system_inhibit    library    vol_family
    comment
    ...
    VO0053          1.19GB    (none    full   )           eagle
    cms.objy_data_files.cpio_odc
    VO0054          0.51GB    (none    full   )           eagle
    cms.objy_data_files.cpio_odc
    VO0055          0.17GB    (none    full   )           eagle
    theory.theory-canopy-C.cpio_odc
    VO0056          0.65GB    (none    full   )           eagle
    theory.theory-canopy-D.cpio_odc
    ...
```

Chapter 4: Enstore Administrator Commands

Enstore provides commands that allow you to communicate with various components of the system. The basic syntax of all Enstore commands is

`% enstore <command> [--option [argument] ...]`

All options start with a double dash (--). The return codes are 0 (zero) for success, non-zero for failure (currently all failures return number 1.)

The switches listed here are additional to those defined in the "The Enstore and dCache User's Guide."

4.1 enstore alarm

Syntax:

`% enstore configuration [ --option [argument] ...]`

Options:

```
-h, --help
Prints the options (i.e. Prints this message).  Example:

$ enstore configuration --help

Usage:
      alarm [OPTIONS]...

  -a, --alive              prints message if the server is up or down.
      --client-name <CLIENT_NAME>  set alarm client name
      --condition <CONDITION>  condition used when generating a remedy ticket
      --do-alarm <DO_ALARM>  turns on more alarms
      --do-log <DO_LOG>      turns on more verbose logging
      --do-print <DO_PRINT>  turns on more verbose output
      --dont-alarm <DONT_ALARM>  turns off more alarms
      --dont-log <DONT_LOG>  turns off more verbose logging
      --dont-print <DONT_PRINT>  turns off more verbose output
      --dump                 print (stdout) alarms the alarm server has in
                             memory
  -h, --help                 prints this messge
      --message <MESSAGE>    message along with raise option
      --raise                raise an alarm
      --remedy_type <REMEDY_TYPE>  type used when generating a remedy ticket
      --resolve <KEY>        resolve the previously raised alarm whose key
                             matches the entered value
      --retries <ALIVE_RETRIES>  number of attempts to resend alive requests
      --root-error <ROOT_ERROR>  error which caused an alarm to be raised [D:
                             UNKONWN]
      --severity <SEVERITY>  severity of raised alarm (E, U, W, I, M, C)[D:
W]
      --timeout <SECONDS>    number of seconds to wait for alive response
      --usage                prints short help message
```

--client-name When issuing an alarm from the command line this switch allows the user to specify the name of the client. This overrides the default of "ALARM_CLIENT."
--dump Dump the current list of alarms. This output goes to the file that stdout has been redirected to for the alarm_server; not the stdout for the alarm_server client. ```$ enstore alarm --dump```
--message <MESSAGE> Include the message in the alarm. This is intended to be the long description of the problem. Used with --raise. See --raise for an example.

--remedy-type <REMEDY_TYPE>
This is an FNAL specific switch. Possible values are:
   ● 'STK Enstore'
   ● 'D0 Enstore'
   ● 'CDF Enstore'
for the remedy category MSS.

--rasise
Raise an alarm from the command line. See --message, --root-error and --severity.
 	
```$ enstore alarm --raise --severity E --root-error \```
```"permission denied" --message \```
```"Unable to query tape robot via media changer stk."```

--resolve <KEY> Remove the alarm with key KEY from the Enstore Active Alarms web page. The key can be obtained from the Enstore Active Alarms web page or from --dump. ```$ enstore alarm --resolve 1044893163.67```
--root-error <ROOT_ERROR> Include text that gives the reason for the error. This is intended to be a short string. Used with --rasise. The default string is "UNKNOWN." See --raise for an example.

--severity <SEVERITY>

The severity needs to be one of the following letters:
- E - Error.
- U – User Error.
- W – Warning. This is the default.
- I – Informational. Use of this severity is discouraged. Alarms are supposed to indicate real problems.
- M – Miscellaneous. Use of this severity is discouraged. Alarms are supposed to indicate real problems.
- C – E-mailable error. Some errors are only resolvable by the administrators of the client machines (typically those that run encp). This option will send them the alarm via e-mail. See the alarm server configuration section 1.4.1 for configuration. If the error cannot be e-mailed it defaults to placing it on the alarms page.

4.2 enstore backup

This command initiates the backup of the Enstore Database. The backup is dumped into the directory specified in the configuration file in **database|db_dir** and copied to the backup host defined in **crons|backup_node** to the backup directory defined in **crons|backup_dir**.

Syntax:

```
% enstore backup
```

4.3 enstore configuration

Syntax:

```
% enstore configuration [ --option [argument] ...]
```

Options:

-h, --help
Prints the options (i.e. Prints this message). Example:

```
$ enstore configuration --help

Usage:
        conf [OPTIONS]...

   -a, --alive                 prints message if the server is up or down.
       --config-file <CONFIG_FILE>  config file to load
       --do-alarm <DO_ALARM>   turns on more alarms
       --do-log <DO_LOG>       turns on more verbose logging
       --do-print <DO_PRINT>   turns on more verbose output
       --dont-alarm <DONT_ALARM>  turns off more alarms
       --dont-log <DONT_LOG>   turns off more verbose logging
       --dont-print <DONT_PRINT>  turns off more verbose output
       --file-fallback         return configuration from file if configuration
                               server is down
   -h, --help                  prints this messge
       --list-library-managers  list all library managers in configuration
       --list-media-changers  list all media changers in configuration
       --list-movers           list all movers in configuration
       --load                  load a new configuration
       --print                 print the current configuration
       --retries <ALIVE_RETRIES>  number of attempts to resend alive requests
       --show                  print the current configuration in python format
       --summary               summary for saag
       --threaded-impl <THREADED_IMPL>  Turn on / off threaded implementation
       --timeout <SECONDS>     number of seconds to wait for alive response
       --timestamp             last time configfile was reloaded
       --usage                 prints short help message
```

--config-file <CONFIG_FILE>
Used with –load to tell the configuration server the configuration file to load.
For an example see --load.

--file-fallback
If the configuration server is not available (by default after 9 seconds), then
the configuration client will return the contents of the configuration file
located in the $ENSTORE_CONFIG_FILE environmental variable. Use
--retries and--timeout to override the 9 second default. This switch is
expected to be used by Enstore installation scripts that may need to be run
before Enstore is started.
Must be used with either --show or --print.

```
$ enstore configuration --file-fallback --show crons html_dir

    /srv/enstore/www
```

--list-library-managers
Output to standard out the list of all configured library managers.

```
$ enstore configuration --list-library-managers

            library manager              host
        9940.library_manager stkensrv4.fnal.gov
         dlt.library_manager stkensrv4.fnal.gov
        test.library_manager stkensrv4.fnal.gov
       null1.library_manager stkensrv4.fnal.gov
     CD-9940B.library_manager stkensrv4.fnal.gov
CD-LTO3_test.library_manager stkensrv4.fnal.gov
      CD-LTO3.library_manager stkensrv3.fnal.gov
```

--list-media-changers
Output to standard out the list of all configured media changers.

```
$ enstore configuration --list-media-changers

           media changer              host              type
      null1.media_changer stkensrv4.fnal.gov     RDD_MediaLoader
     SL8500.media_changer stkensrv4.fnal.gov     STK_MediaLoader
        stk.media_changer stkensrv4.fnal.gov     STK_MediaLoader
```

--list-movers
Output to standard out the list of all configured movers.

```
$ enstore configuration --list-movers

         mover              host mc_device    driver        library
  9940B27.mover      stkenmvr27a  0,1,10,2  FTTDriver CD-9940B.library_manager
  9940B40.mover      stkenmvr40a  0,1,10,7  FTTDriver CD-9940B.library_manager
  LTO3_13.mover     stkenmvr113a  0,3,1,12  FTTDriver CD-LTO3.library_manager
  LTO3_06.mover     stkenmvr106a   0,0,1,3  FTTDriver CD-LTO3.library_manager
  9940B16.mover      stkenmvr16a 0,0,10,18  FTTDriver CD-9940B.library_manager
  LTO3_14.mover     stkenmvr114a  0,3,1,13  FTTDriver CD-LTO3.library_manager
  9940B26.mover      stkenmvr26a  0,0,10,7  FTTDriver CD-9940B.library_manager
  9940B11.mover      stkenmvr11a 0,0,10,16  FTTDriver CD-9940B.library_manager
  9940B21.mover      stkenmvr21a  0,1,10,4  FTTDriver CD-9940B.library_manager
   994052.mover       stkenmvr5a  0,0,10,9  FTTDriver 9940.library_manager
  LTO3_12.mover     stkenmvr112a   0,1,1,1  FTTDriver CD-LTO3.library_manager
  LTO3_08.mover     stkenmvr108a  0,2,1,13  FTTDriver
['CD-LTO3.library_manager', 'CD-LTO3_test.library_manager']
```

--load
Tell the configuration server to reload the configuration file.

```
$ enstore configuration --load -config-file \
 $ENSTORE_CONFIG_FILE
```

--print
Output to standard out the current Enstore configuration in a script friendly format. If one or more KEYs are present then just that subsection of the configuration is printed. See--show for a different output format. Example:

```
$ enstore conf --print crons html_dir
```

```
crons.html_dir:/local/ups/prd/www_pages/enstore/
```

--show [KEY1 [KEY2] ...]]
Output to standard out the current Enstore configuration in a native python format. If one or more KEYs are present then just that subsection of the configuration is printed. See--show for a different output format. Example:

```
$ enstore conf --show blocksizes

{'8MM': 131072,
 '9840': 131072,
 '9940': 131072,
 '9940B': 131072,
 'DECDLT': 131072,
 'LTO3': 131072,
 'null': 131072,
 'status': ('ok', None)}
```

--threaded-impl <THREADED_IMPL>
Parts of the configuration server have been made multi threaded for performance reasons. Early on this proved to be unstable. THREADED_IMPL should be set to 1 (default) for turning on the threading or 0 for turning it off.

```
$ enstore configuration --threaded-impl 1
```

--timestamp
Displays the date and time that the configuration was las (re)loaded into the configuration server.

```
$ enstore configuration --timestamp
Thu Jan 10 17:43:25 2008
```

4.4 enstore event_relay

Syntax:
```
% enstore event_relay [--options [arguments] ...]
```
Options:

--dump
Tells the event_relay to dump to contents of the list of processes that have subscribed for event_relay messages. This output goes to the file that stdout has been redirected to for the event_relay; not the stdout for the event_relay client.

```
$ enstore event_relay --dump
```

4.5 enstore file

Syntax:
```
% enstore file [--options [arguments] ...]
```

Options:

```
-h, --help
Prints the options (i.e. Prints this message).  Example:

$ enstore file --help

Usage:
        file [OPTIONS]...

        --add <BFID>              add file record (dangerous! don't try this at
                                  home)
   -a, --alive                    prints message if the server is up or down.
        --backup                  backup file journal -- part of database backup
        --bfid <BFID>             get info of a file
        --bfids <VOLUME_NAME>     list all bfids on a volume
        --deleted <YES/NO>        used with --bfid to mark the file as deleted
        --do-alarm <DO_ALARM>     turns on more alarms
        --do-log <DO_LOG>         turns on more verbose logging
        --do-print <DO_PRINT>     turns on more verbose output
        --dont-alarm <DONT_ALARM>  turns off more alarms
        --dont-log <DONT_LOG>     turns off more verbose logging
        --dont-print <DONT_PRINT>  turns off more verbose output
        --find-all-copies <FILE>   find all copies of this file
        --find-copies <FILE>       find the immediate copies of this file
        --find-duplicates <FILE>   find all duplicates related to this file
        --find-original <FILE>     find the immediate original of this file
        --find-the-original <FILE>  find the very first original of this file
        --get-crcs <BFID>         get crc of a file
   -h, --help                     prints this messge
        --list <VOLUME_NAME>      list the files in a volume
        --ls-active <VOLUME_NAME>  list active files in a volume
        --mark-bad <PATH> [BFID] Mark the file with the given filename as bad.
                                  Include the bfid only if the file is a multiple
                                  copy file.
        --modify <BFID>           modify file record (dangerous!)
        --recursive               restore directory
        --restore <BFID> [UID[:GID]]  restore a deleted file with optional
                                  uid:gid
        --retries <ALIVE_RETRIES>  number of attempts to resend alive requests
        --set-crcs <SET_CRCS>     set CRC of a file
        --show-bad                list all bad files
        --show-state              show internal state of the server
        --timeout <SECONDS>       number of seconds to wait for alive response
        --unmark-bad <PATH> [BFID]  Unmark the file with the given filename as
                                  bad.  Include the bfid only if the file is a
                                  multiple copy file.
        --usage                   prints short help message
```

--add <BFID> <key1>=<value1> <key2>=<value2> ...
Add file record to the Enstore DB. <BFID> must fit the bfid string pattern.
If <BFID> is "None", then a new bfid will be assigned. The bfid is always
printed to the terminal; this is important for the cases where a new bfid is
assigned. If the bfid you want already exists use **--modify** instead. There
is no normal use case for this switch.

**$ enstore file --add None pnfs_name0=/pnfs/blah complete_crc=1 size=42 **
 **pnfsid=000D000000000023 deleted=y drive= external_label=TEST00 **
 location_cookie=0000_00000000_000001 "sanity_cookie=(42,1)"

bfid = WAMS121457893800000
To confirm that everything is accurate:

$ enstore file --bfid WAMS121457893800000

{'bfid': 'WAMS121457893800000',
 'complete_crc': 1L,
 'deleted': 'yes',
 'drive': '',
 'external_label': 'TEST01',
 'gid': -1,
 'location_cookie': '0000_00000000_000001',
 'pnfs_name0': '/pnfs/mist/blah',
 'pnfsid': '000D000000000023',
 'sanity_cookie': (42L, 1L),
 'size': 42L,
 'uid': -1,
 'update': None}
Note: The fields shone in this example are the required fields. The **gid** and
uid fields are defaulted to -1 in this case.
Note: It is important to make sure that the volume is defined in Enstore
before using this command. If the volume does not exist, then commands
like "**enstore info --bfid <bfid>**" will fail with "NO SUCH
FILE/BFID" because of the use of outer joins between the file and volume
tables. This also means that **--modify** will not work, requiring this to be
fixed at the SQL level.

--backup
Backup the file journal.

$ enstore file --backup

--bfids <VOLUME_NAME>
Deprecated. See **enstore info--bfids** . This file command has been modified
to point to the information server instead of the file clerk.

--deleted <yes | no>
Used with --bfid to mark a file as deleted.

```
$ enstore info --bfid WAMS118954675400000 | grep deleted
 'deleted': 'no',
$ enstore file --bfid WAMS118954675400000 --deleted yes

$ enstore info --bfid WAMS118954675400000 | grep deleted
 'deleted': 'yes',
```

--find-all-copies <BFID>
List all the BFIDs that are multiple copies or originals of the specified BFID.

```
$ enstore file --find-all-copies WAMS115254898100000

WAMS115254898100000
WAMS115254898700000
```

--find-copies <BFID>
List all the immediate copies of the specified BFID.

```
$ enstore file --find-copies WAMS115254898100000

WAMS115254898700000

$ enstore file --find-copies WAMS115254898700000

```

--find-duplicates <BFID>
List the original and all copies of the specified BFID.

```
$ enstore file --find-duplicates WAMS115254898700000

WAMS115254898100000
WAMS115254898700000

$ enstore file --find-duplicates WAMS115254898100000

WAMS115254898100000
WAMS115254898700000
```

--find-original <BFID>
List the immediate original to this BFID.

```
$ enstore file --find-original WAMS115254898700000

WAMS115254898100000
```

--find-the-original <BFID>
List the master original to this BFID.

```
$ enstore file --find-original WAMS115254898700000

WAMS115254898100000
```

--get-crcs <BFID>

Prints to standard out the crc information for the specified file.

```
$ enstore file --get-crcs WAMS115254898700000
```

```
bfid 'WAMS115254898700000': sanity_cookie (10241L, 1458501222L), complete_crc
1458501222L
```

--mark-bad <PATH> [BFID]

Renames the file specified by <PATH> in PNFS to be <DIRNAME>/.bad.<BASENAME>. This must be the current path in PNFS, and not the original path stored in the Enstore DB. The Enstore database is also updated to include a record that this file has been marked bad, but does not also modify the original path.

Files should be marked bad when the file has been determined to be unreadable from tape, but other files on the tape continue to read without problems. If there is a problem with reading every file on the tape or the tape itself then set the tape NOTALLOWED with the enstore volume--not-allowed command in section 1.24. See also--unmark-bad and --show-bad.

```
$ ls /pnfs/test/my_dir/1KB_147
```

```
/pnfs/test/NULL/1KB_147
```

```
# enstore file --mark-bad  /pnfs/test/my_dir/1KB_147
```

```
$ ls /pnfs/mist/my_dir/.bad.1KB_147
```

```
/pnfs/test/NULL/.bad.1KB_147
```

If the bad file is a multiple copy of the primary copy of the file, then we need to specify the multiple copy"s BFID on the command line.

```
# enstore file --mark-bad \
/pnfs/test/my_dir/1KB_147 CDMS121891419900000
```

--modify <BFID>

Modify the database entry for the specified BFID. This command can create many problems if not used with the utmost care. The quoting shown below is not always necessary, however there are some PNFS ids that happen to look like exponential numbers to python. The quoting shown overrides the default attempt to interpret them as floating point numbers and instead treats them as strings.

```
$ enstore file --bfid WAMS100888634200000

{'bfid': 'WAMS100888634200000',
 'complete_crc': 0L,
 'deleted': 'no',
 'drive': 'rain:/dev/null:0',
 'external_label': 'NULL01',
 'gid': -1,
 'location_cookie': '0000_000000000_0000142',
 'pnfs_name0': '/pnfs/mist/NULL/1KB_100',
 'pnfsid': '000100000000000000006350',
 'sanity_cookie': (1024L, 0L),
 'size': 1024L,
 'uid': -1,
 'update': None}

$ enstore file --modify WAMS100888634200000 "gid=1530" \
 "uid=9276" "pnfsid='000100000000000000006350'"

bfid = WAMS100888634200000

$ enstore file --bfid WAMS100888634200000

{'bfid': 'WAMS100888634200000',
 'complete_crc': 0L,
 'deleted': 'no',
 'drive': 'testnode:/dev/null:0',
 'external_label': 'NULL01',
 'gid': 1530,
 'location_cookie': '0000_000000000_0000142',
 'pnfs_name0': '/pnfs/test/NULL/1KB_100',
 'pnfsid': '000100000000000000006350',
 'sanity_cookie': (1024L, 0L),
 'size': 1024L,
 'uid': 9276,
 'update': None}
```

--recursive

--restore <BFID> [UID[:GID]]

This command will restore a file that has been removed from PNFS and has been marked as deleted in the Enstore database. PNFS must be mounted and the current effective user id must have the necessary permissions to touch/open the file in the PNFS namespace. For all practical purposes, this command will almost always be run as user root on the PNFS server itself; though that is not a requirement.

```
$ enstore info --bfid WAMS120224553900000

{'bfid': 'WAMS120224553900000',
 'complete_crc': 0L,
 'deleted': 'yes',
 'drive': 'testnode:/dev/null:0',
 'external_label': 'NULL10',
 'gid': 1530,
 'location_cookie': '0000_000000000_0000005',
 'pnfs_name0': '/pnfs/test/NULL/restore_example_file',
 'pnfsid': '00010000000000000007FF08',
 'sanity_cookie': (12559L, 0L),
 'size': 12559L,
 'uid': 9276,
 'update': None}

$ ls /pnfs/test/NULL/restore_example_file

ls: /pnfs/test/NULL/restore_example_file: No such file or directory

$ enstore file --restore WAMS120224553900000

$ ls /pnfs/test/NULL/restore_example_file

/pnfs/test/NULL/restore_example_file
```

--set-crcs

--show-bad

List all files that are marked bad. See also--mark-bad and --unmark-bad.

```
$ enstore file --show-bad

NULL01 WAMS103661998200000 1024 /pnfs/test/NULL/.bad.1KB_147
```

--unmark-bad <PATH> [BFID]

Unmark a file that has previously been marked as bad. This renames the file in PNFS back to its original name and removes the file from the marked bad list in the Enstore database. See also--mark-bad and--show-bad.

```
$ ls /pnfs/test/my_dir/1KB_147
```

`ls: /pnfs/test/NULL/1KB_147: No such file or directory`

```
$ ls /pnfs/test/my_dir/.bad.1KB_147
```

`/pnfs/test/NULL/.bad.1KB_147`

It is important to note in the following example command that the basename of the file starts with .bad. and is not the original filename given to the --mark-bad command.

```
# enstore file --unmark-bad  /pnfs/test/my_dir/.bad.1KB_147

$ ls /pnfs/test/my_dir/1KB_147
```

`/pnfs/test/NULL/1KB_147`

If the bad file is a multiple copy of the primary copy of the file, then we need to specify the multiple copy's BFID on the command line.

For multiple copies, the path is not changed when the --mark-bad is done. Do not include the .bad. at the beginning of the file like primary files are done.

```
# enstore file --unmark-bad \
/pnfs/test/my_dir/1KB_147 CDMS121891419900000
```

4.6 enstore info

As of encp v3_2, the user level command **enstore info** supersedes **enstore file** and **enstore volume**. The admin level commands **enstore file** and **enstore volume** are still used for modifying commands.

Syntax:

```
% enstore info [--option [argument] ...]
```

Options:

-h, --help
Prints the options (i.e. Prints this message). Example:

```
$ enstore file --help

Usage:
        info [OPTIONS]...

   -a, --alive                prints message if the server is up or down.
        --bfid <BFID>         get info of a file
        --bfids <VOLUME_NAME>  list all bfids on a volume
        --check <VOLUME_NAME>  check a volume
        --do-alarm <DO_ALARM>  turns on more alarms
        --do-log <DO_LOG>      turns on more verbose logging
        --do-print <DO_PRINT>  turns on more verbose output
        --dont-alarm <DONT_ALARM>  turns off more alarms
        --dont-log <DONT_LOG>  turns off more verbose logging
        --dont-print <DONT_PRINT>  turns off more verbose output
        --file <PATH|PNFSID|BFID|VOL:LOC>  get info of a file
        --find-all-copies <BFID>  find all copies of this file
        --find-copies <BFID>  find the immediate copies of this file
        --find-duplicates <BFID>  find all duplicates related to this file
        --find-original <BFID>  find the immediate original of this file
        --find-same-file <BFID>  find a file of the same size and crc
        --find-the-original <BFID>  find the very first original of this file
        --get-sg-count <LIBRARY> <STORAGE_GROUP>  check allocated count for
                              lib,sg
        --gvol <VOLUME_NAME>  get info of a volume in human readable time
                              format
   -h, --help                prints this messge
        --history <VOLUME_NAME>  show state change history of volume
        --just                used with --pvols to list problem
        --labels              list all volume labels
        --list <VOLUME_NAME>  list the files in a volume
        --ls-active <VOLUME_NAME>  list active files in a volume
        --ls-sg-count         list all sg counts
        --pvols               list all problem volumes
        --query <QUERY>       query database
        --retries <ALIVE_RETRIES>  number of attempts to resend alive requests
        --show-bad            list all bad files
        --show-copies <BFID>  all copies of a file
        --show-file <BFID>    show info of a file
        --timeout <SECONDS>   number of seconds to wait for alive response
        --usage               prints short help message
        --vol <VOLUME_NAME>   get info of a volume
        --vols                list all volumes
        --write-protect-status <VOLUME_NAME>  show write protect status
```

--bfids <VOLUME_NAME>
List all of the BFIDs on the specified volume.

```
$ enstore info --bfids TEST01

WAMS100456552800000
WAMS100456552900000
WAMS100456553000000
WAMS100456553100000
WAMS100456563700000
WAMS100463404700000
WAMS100463422700000
WAMS100463427300000
WAMS100463493300000
WAMS100463735500000
```

--check <VOLUME_NAME>

Prints a synopsis of the volume status.

```
$ enstore info --check TEST01

TEST01        0.52GB ['none', 'full'] ['none', 'none']
```

--find-same-file <BFID>

Lists all files in the Enstore DB with the same CRC and size as the file with the specified BFID.

```
$ enstore info --find-same-file WAMS115254898100000

TEST01  WAMS115401377100000     10241 0000_000000000_0000005 deleted
/pnfs/test/10KB_002
TEST01  WAMS115401417000000     10241 0000_000000000_0000007 deleted
/pnfs/test/10KB_002
TEST01  WAMS115254898100000     10241 0000_000000000_0000010 active
/pnfs/test/10KB_002
```

--get-sg-count <LIBRARY> <STORAGE_GROUP>

Lists the number of volumes belonging to the indicated library and storage group pairing.

```
$ enstore info --get-sg-count LTO3 test

        rain            zee         3
```

--gvol <VOLUME_NAME>

Same as --vol except that time values are displayed in human readable format. See also--vol.

```
$ enstore info --gvol TST001

{'blocksize': 131072,
 'capacity_bytes': 858993459200L,
 'comment': '',
 'declared': 'Fri Dec 21 14:31:20 2007',
 'eod_cookie': '0000_000000000_0000639',
 'external_label': 'TST001',
 'first_access': 'Fri Dec 21 16:03:09 2007',
 'last_access': 'Sun Jan 20 10:33:53 2008',
 'library': 'LTO4',
 'media_type': 'LTO4',
 'remaining_bytes': 719404544L,
 'si_time': ('Wed Dec 31 18:00:00 1969', 'Wed Dec 26 14:57:03 2007'),
 'sum_mounts': 369,
 'sum_rd_access': 707,
 'sum_rd_err': 1,
 'sum_wr_access': 638,
 'sum_wr_err': 0,
 'system_inhibit': ['none', 'full'],
 'user_inhibit': ['none', 'none'],
 'volume_family': 'SSA.ssa-test.cpio_odc',
 'wrapper': 'cpio_odc',
 'write_protected': 'n'}
```

--history <VOLUME_NAME>
Prints a report of the status changes to the volume.

```
$ enstore info --history TST001

2007-12-26 14:57:03.083077   system_inhibit[1]    full
```

--just <problem_type_1> <problem_type_2> ...
When used with--pvols to display one type of problem volumes. The most common values for problem types include (but not limited to):
1. migrated
2. readonly
3. full
4. NOACCESS
5. NOTALLOWED
6. duplicated

```
$ python info_client.py --pvols --just migrated

==== migrated
VO2140             0.90GB    (none       0509-1704 migrated 0811-0008)   9940
sdss.sdss_mt.cpio_odc                => VO8585
VO2141             0.90GB    (none       0509-1704 migrated 0811-0722)   9940
sdss.sdss_mt_2c.cpio_odc             => VO8585 VOB250
```

This is really just a sorting of system inhibits that are not 'none'. Full volumes are not considered problem volumes unless another condition (i.e. NOACCESS) is also there.

--labels
List all the volume names.

```
$ enstore info -labels

TEST00
TEST00.deleted
TEST01
TEST02
```

--query <SQL query>
Allows the user to issue an sql command to the database. Use of this command requires knowledge of the database scheme.

```
$ enstore info --query "select label from volume limit 10;"

                               label
---------------------------------------
                             JL4742
                     JL4742.deleted
rain:zee.shortcut_test.null:1116865327606
                             NULL02
                             NULL03
                             NULL04
                             NULL05
                             STORM1
                     STORM4.deleted
                             UPB018
```

--write-protect-status <VOLUME_NAME>
Prints in the write protect status is OFF, ON or UNKNOWN.

```
$ enstore info --write-protect-status TST001
```
```
TST001 write-protect OFF
```

4.7 enstore inquisitor

The **enstore inquisitor** commands have **enstore sched** as an alias.

Syntax:

```
% enstore inquisitor [--options [arguments] ...]
```

Options:

-h, --help
Prints the options (i.e. Prints this message). Example:

```
$ enstore file --help

Usage:
        inquisitor [OPTIONS]...

   -a, --alive                prints message if the server is up or down.
        --do-alarm <DO_ALARM>  turns on more alarms
        --do-log <DO_LOG>      turns on more verbose logging
        --do-print <DO_PRINT>  turns on more verbose output
        --dont-alarm <DONT_ALARM>  turns off more alarms
        --dont-log <DONT_LOG>  turns off more verbose logging
        --dont-print <DONT_PRINT>  turns off more verbose output
        --down <SERVER[,SERVER]>  servers to mark down
        --dump                 print (stdout) state of servers in memory
        --get-last-alive <SERVER[,SERVER]>  return the last time a heartbeat
                               was received by the inquisitor for the listed
                               servers
        --get-max-encp-lines   return number of displayed lines on the encp
                               history web page
        --get-refresh          return the refresh interval for inquisitor
                               created web pages
        --get-update-interval  return the interval between updates of the
                               server system status web pages
   -h, --help                 prints this messge
        --is-up <SERVER>       check if <server> is up
        --max-encp-lines <NUM_LINES>  set the number of displayed lines on the
                               encp history web page
        --nooutage <SERVER[,SERVER]>  remove the outage check from the SAAG
                               page for the specified servers
        --nooverride <SERVER[,SERVER]>  do not override the status of the
                               specified servers
        --outage <SERVER[,SERVER]>  set the outage check on the SAAG page for
                               the specified servers
        --override <SERVER[,SERVER]>  override the status of the specified
                               servers with the saagstatus option
        --reason <STRING>      information associated with a server marked down
                               or with an outage
        --refresh <SECONDS>    set the refresh interval for inquisitor created
                               web pages
        --retries <ALIVE_RETRIES>  number of attempts to resend alive requests
        --saagstatus <STATUS>  status to use for override
        --show                 print (stdout) the servers scheduled down, known
                               down, seen down and overridden
        --subscribe            subscribe the inquisitor to the event relay
        --time <STRING>        information associated with a server marked down
                               or with an outage
        --timeout <SECONDS>    number of seconds to wait for alive response
        --up <SERVER[,SERVER]>  servers to mark up
        --update               update the server system status web page
        --update-and-exit      update the server system status web page and
                               exit the inquisitor
        --update-interval <SECONDS>  set the interval between updates of the
                               system servers status web page
        --usage                prints short help message
```

--down <SERVER[,SERVER]>
Mark a server as known down. Valid names include all entities that appear on the Mass Storage Status at-a-Glance. See also--reason, --up and --show.

```
$ enstore inquisitor --down LTO3.library_manager
```

--dump
This output goes to the file that stdout has been redirected to for the inquisitor; not the stdout for the inquisitor client.

```
$ enstore inquistor --dump
```

--get-last-alive <SERVER[,SERVER]>
Show the last time the requested server(s) sent a heartbeat message.

```
$ enstore inquisitor --get-last-alive rain.library_manager
rain.library_manager   Wed Mar 26 16:14:25 2008
```

--get-max-encp-lines
Show the number of lines displayed on the encp History web page. See --max-encp-lines to set this value.

```
$ enstore inquisitor --get-max-encp-lines
250
```

--get-refresh
Show the refresh interval in seconds for inquisitor created web pages. See--refresh to set this value.

```
$ enstore inquisitor --get-refresh
3600
```

--get-update-interval
Show the update interval in seconds between updates of the server system status web pages. See also --update-interval.

```
$ enstore inquisitor --get-update-interval
20
```

--is-up <SERVER>
Show if a server is up. This switch is script friendlier than --show.

```
$ enstore inquisitor --is-up LTO4_0.mover
no
```

--max-encp-lines <NUM_LINES>
Sets the maximum number of lines on the encp History web page. See also --get-max-encp-lines.

```
$ enstore inquisitor --get-max-encp-lines
250

$ enstore inquisitor --max-encp-lines 251

$ enstore inquisitor --get-max-encp-lines

251
```

--nooutage <SERVER[,SERVER]>
Removes the specified servers from the list of scheduled down servers. See also --outage.

```
$ enstore inquisitor --nooutage LTO4_0.mover
```

--nooverride <SERVER[,SERVER]>
See also --override.

--outage <SERVER[,SERVER]>
Adds the specified servers to the list of scheduled down servers. See also --nooutage.

```
$ enstore inquisitor --outage LTO4_0.mover
```

--override <SERVER[,SERVER]>
See also --nooverride.

--reason <STRING>
Adds a comment to explain why a server is down or has a scheduled outage. See also --up, --outage and--show.

```
$ enstore inquisitor -down LTO4_0.mover -reason "tape drive
investigation"
```

--refresh <SECONDS>
Sets the number of seconds between updates on inquisitor created web pages. See also--get-refresh.

```
$ enstore inquisitor --get-refresh
3600

bash-3.00$ enstore inquisitor --refresh 5000

bash-3.00$ enstore inquisitor --get-refresh
5000
```

--saagstatus <STATUS>

--show
Show what Enstore severs are down or scheduled to be down.

```
$ enstore inquisitor --show


 Enstore Items Scheduled To Be Down
 ----------------------------------


 Enstore Items Known Down
 ------------------------
   DLT1.mover : fan to noisy


 Enstore Items Down and the Number of Times Seen Down
 ----------------------------------------------------
   Drivestat Server : 34853
   DLT1.mover : 61020
```

--subscribe
Subscribe the inquisitor to the event_relay.

```
$ enstore inquisitor --subscribe
```

--time <STRING>

--up <SERVER[,SERVER]>
Mark an Enstore server as being up. Valid names include all entities that
appear on the Mass Storage Status-at-a-Glance. See also--reason, --down and
--show.

```
$ enstore inquisitor --down LTO3.library_manager
```

--update
Tells the inquisitor to update the Enstore Server Status web page now instead
of waiting for the next update interval. See also --update-and-exit.

```
$ enstore inquisitor --update
```

--update-and-exit
Tells the inquisitor to update the Enstore Server Status web page now instead
of waiting for the next update interval. After the page is updated the
inquisitor quits. See also --update.

```
$ enstore inquisitor --update-and-exit
```

> --update-interval <SECONDS>
> Sets the number of seconds between updates of the Enstore Server Status web page. See also --get-update-interval.
>
> ```
> $ enstore inquisitor --get-update-interval
>
> 20
>
> $ enstore inquisitor --update-interval 30
>
> $ enstore inquisitor --get-update-interval
>
> 30
> ```

4.8 enstore library

Syntax:
```
% enstore library [--options [arguments] ...] \
<library>
```
Options:

> --delete-work <UNIQUE_ID>
> Remove the request with the specified unique ID from the queue.
>
> ```
> $ enstore library --get-queue '' stk.library_manager
>
> Pending write requests
> testnode.fnal.gov stk.library_manager enstore /home/enstore/testfile
> /pnfs/test/xyz/testfile P 0      FF zoo FF_W 1
> Pending read requests:  0
> Pending write requests:  1
> Active read requests:  0
> Active write requests:  0
> {'status': ('ok', None)}
>
> $ enstore library --delete-work \
> testnode.fnal.gov-1202315917-17575-0 stk
>
> {'status': ('ok', 'Work deleted')}
>
> $ enstore library --get-queue '' stk
>
> Pending read requests:  0
> Pending write requests:  0
> Active read requests:  0
> Active write requests:  0
> {'status': ('ok', None)}
> ```

> --priority <UNIQUE_ID> <PRIORITY>
> Set the priority for the request with the specified unique id.
>
> ```
> $ enstore library --priority \
> testnode.fnal.gov-1202315917-17575-0 5 stk.library_manager
>
> {'status': ('ok', 'Priority changed')}
> ```

--rm-active-vol <VOLUME>
This command removes the volume from the list of active volumes.

This command should only be used when it is known that the tape is not located in the assigned tape drive. This command is usually used after an error occurred and the tape needed to be ejected manually from the tape drive. The library manager will not schedule requests for a different tape drive for a tape while it believes the tape is still in a drive.

```
$ enstore library --rm-active-vol TEST01L1 stk.library_manager
```

--rm-suspect-vol <VOLUME>
This command removes the volume from the suspect volume list. This list keeps track of volume with mover combinations that result in media errors. The library manager will not assign a suspect volume to its matching suspect movers.

```
$ enstore library --rm-suspect-vol TST001 stk.library_manager
```

--start-draining <LIBRARY STATE>
Tell the library to stop assigning requests to movers. The normal state is unlocked, however it can be set to:

- **lock** – Stop assigning any new requests to movers. If a new encp request is received return an error.
- **ignore** - Stop assigning any new requests to movers. If a new encp request is received return a success message to encp, but do not insert the request into the queue.
- **pause** - Stop assigning any new requests to movers. If a new encp request is received return a success message to encp and put the request at the end of the queue.
- **noread** – If a read request is received an error is returned to encp. If a write request is received it is processed normally.
- **nowrite** – If a write request is received an error is returned to encp. If a read request is received it is processed normally.

See --stop-draining and --status.

```
$ enstore library --status aml2.library_manager
LM state:unlocked
$ enstore library --start-draining lock aml2

$ enstore library --status aml2
LM state:locked
```

--status
Print the current state of the library manager: It will be one of: unlocked, locked, pause, ignore, noread or nowrite. See--start-draining and--stop draining for more information and examples.

--stop-draining
Set the state of the library manager back to its default state. Most often this is the unlocked state. A different default state can be set in the configuration; see section 5.4. If this is set in this libraries configuration, then it will be set back to that value.

```
$ enstore library --status SL8500.library_manager

LM state:nowrite

$ enstore library --stop-draining SL8500

$ enstore library --status SL8500

LM state:unlocked
```

--vols
Reports information on the current state of the active volumes and their currently associated movers.

```
$ enstore library --vols CDF-LTO3

label       mover              tot.time status                 system_inhibit
rq. host         updated           volume family
IAD655     LTO3_24.mover     28        IDLE            (0    ) (none
full) fcdfdata107     02-06-08 14:24:31 cdf.cdfpstn.cpio_odc
IAB451     LTO3_01.mover     286       HAVE_BOUND      (30   ) (none
full) fcdfdata098     02-06-08 14:24:54 cdf.cdfptnt.cpio_odc
IAD726     LTO3_31.mover     664       ACTIVE-READ     (4    ) (none
full) fcdfdata105     02-06-08 14:24:50 cdf.cdfpstn.cpio_odc
```

4.9 enstore log

Syntax:
```
% enstore log [--options [arguments] ...]
```
Options:

-h, --help
Prints the options (i.e. Prints this message). Example:

```
$ enstore log --help

Usage:
      log [OPTIONS]...

  -a, --alive                prints message if the server is up or down.
      --client-name <CLIENT_NAME>  set log client name
      --do-alarm <DO_ALARM>  turns on more alarms
      --do-log <DO_LOG>      turns on more verbose logging
      --do-print <DO_PRINT>  turns on more verbose output
      --dont-alarm <DONT_ALARM>  turns off more alarms
      --dont-log <DONT_LOG>  turns off more verbose logging
      --dont-print <DONT_PRINT>  turns off more verbose output
      --get-last-logfile-name  return the fname of yesturdays log file
      --get-logfile-name     return the name of the current log file
      --get-logfiles <PERIOD>  return the last 'n' log file names (today,
                             week, month, all)
  -h, --help                 prints this messge
      --message <MESSAGE>    log a message
      --retries <ALIVE_RETRIES>  number of attempts to resend alive requests
      --timeout <SECONDS>    number of seconds to wait for alive response
      --usage                prints short help message
```

--client-name
When used with--message this switch will set the client name recorded in the log file. See--message for an example. The default name, when this switch is not used is, LOG_CLIENT.

--get-last-logfile-name
Return the name of yesterdays log file.

```
$ enstore log --get-last-logfile-name

/diska/enstore-log/LOG-2008-02-05
```

--get-logfile-name
Return the name of todays log file.

```
$ enstore log --get-logfile-name

/diska/enstore-log/LOG-2008-02-06
```

--get-logfiles

--message <MESSAGE>
Send a message to the log server to be placed in the log file. See --client-name.

```
$ enstore log --message "This is a test message." --client-name
"MY_SCRIPT"
```

4.10 enstore media

Syntax:

**% enstore media [--options [arguments] ...] \
<media_changer>**

Options:

-h, --help
Prints the options (i.e. Prints this message). Example:

```
$ enstore media --help

Usage:
        media [OPTIONS]...

    -a, --alive                prints message if the server is up or down.
        --dismount <EXTERNAL_LABEL> <DRIVE>
        --do-alarm <DO_ALARM>  turns on more alarms
        --do-log <DO_LOG>      turns on more verbose logging
        --do-print <DO_PRINT>  turns on more verbose output
        --dont-alarm <DONT_ALARM>  turns off more alarms
        --dont-log <DONT_LOG>  turns off more verbose logging
        --dont-print <DONT_PRINT>  turns off more verbose output
        --get-work
    -h, --help                 prints this messge
        --list                 list all media changers in configuration
        --list-clean           List cleaning volumes.
        --list-drives          List all drives.
        --list-slots           List all slot counts.
        --list-volumes         List all volumes.
        --max-work <MAX_WORK>
        --mount <EXTERNAL_LABEL> <DRIVE>
        --retries <ALIVE_RETRIES>  number of attempts to resend alive requests
        --show
        --show-drive <DRIVE>
        --show-robot
        --show-volume <VOLUME> <MEDIA_TYPE>  Returns information about a
volume.
        --timeout <SECONDS>    number of seconds to wait for alive response
        --usage                prints short help message
```

--dismount
Move a volume from the mouth of a drive to a slot in the media library. See
also --mount.

```
$ enstore media --dismount TST001 LTO-800G stk.media_changer
```

--get-work

Print various pieces of information about the current list of outstanding media relocation requests.

```
$ enstore media --get-work SL8500.media_changer

{'max_work': 10,
 'status': ('ok', 0, None),
 'worklist': [('mount', 'VOF764', '0,3,1,1'),
             ('mount', 'VOG033', '0,2,1,12'),
             ('mount', 'VOG586', '0,1,1,12'),
             ('dismount', 'VOG014', '0,1,1,1'),
             ('mount', 'VOG365', '0,0,1,3'),
             ('dismount', 'VOG727', '0,3,1,13')]}
```

-list-clean

Report on the number of cleanings remaining for cleaning tapes.

```
$ enstore media --list-clean SL8500G1.media_changer
```

volume	type	max	current	remaining
CLN006	LTO-CLNU	50	50	0
CLN007	LTO-CLNU	50	50	0
CLN008	LTO-CLNU	50	50	0
CLN009	LTO-CLNU	50	45	5
CLN010	LTO-CLNU	50	28	22
CLN011	LTO-CLNU	50	1	49
CLN013	LTO-CLNU	50	50	0
CLN014	LTO-CLNU	50	16	34
CLN015	LTO-CLNU	50	20	30

--list-drives

List the available drives and some information about them.

```
$ enstore media --list-drives stk.media_changer
```

name	state	status	type	volume
0,0,10,0	online	available	T9940B	
0,0,10,1	online	available	T9940B	
0,0,10,2	online	available	T9940A	
0,0,10,4	online	available	T9940B	
0,0,10,5	online	available	T9940B	
0,0,10,6	online	in use	T9940B	VOB915
0,0,10,7	online	in use	T9940B	VO2688

--list-slots

Report on the current number of media slots that are used and empty.

```
$ enstore media --list-slots aml2.media_changer
```

location	media type	total	free	used	disabled
ST01	DECDLT	4320	4294	26	0
ST02	3480	4800	1603	3197	0
ST03	3480	4800	82	4718	0

--list-volumes

List the available volumes and some information about them. This command can take a while (sometimes several minutes) to complete.

```
$ enstore media --list-volumes aml2.media_changer

          volume        type              state              location
          270PROL1      3480                O
          797PROL1      3480                O
          835PROL1      3480                O
          836PROL1      3480                O
          899PROL1      3480                O
           CA2530      DECDLT               O
           CA2531      DECDLT               O
           CA2532      DECDLT               O
```

For the AML2 media changers:
- The location column is left blank.
- Some tapes are reported backwards. 270PROL1 from above is really PRO270L1, but CA2530 really is CA2530.

--max-work <MAX_WORK>

Override the default max work parameter with the user supplied value. See also --get-work.

```
$ enstore media --get-work test.media_changer

{'status': ('ok', 0, None), 'max_work': 7, 'worklist': []}

bash-3.00$ enstore media --max-work 10 test.media_changer

bash-3.00$ enstore media --get-work test.media_changer

{'status': ('ok', 0, None), 'max_work': 10, 'worklist': []}
```

--mount <EXTERNAL_LABEL> <DRIVE>

Move a volume from the its media slot and insert it into a drive. See also --dismount.

```
$ enstore media -mount TST001 0,5,1,0 SL8500.media_changer
```

--show

Determine if the robot is available and report the status back to the user.

```
$ enstore media --show SL8500.media_changer

2008-02-06:16:27:21 4.320000 ('ok', 0, 'query server => 0,run          ,
4.320000')
```

--show-drive <DRIVE>
Show information about a specific drive.

```
$ enstore med --show-drive DG4A aml2

        name          state         status            type    volume
        DG4A             up               0       6380/7480
```
For the AML2 media changer:
- The (letter) O status means the tape is **O**ccupied in its slot.
- The **M** status means that it is mounted in a drive.

For the STK media changer:
- **home** status means that it is located in it slot.
- **In use** status means that is is mounted in a drive.

--show-robot
Reserved to show the state of the robot in a more generic format than --show.
Currently is an alias to --show.

--show-volume
Show information about a specific volume.

```
$ enstore med --show-volume PRO270L1 DECDLT aml2.media_changer

          volume        type                 state             location
        PRO270L1        3480                    O
```
For the AML2 media changers:
- The location column is left blank.
- The (letter) O status means the tape is **O**ccupied in its slot.
- The **M** status means that it is mounted in a drive.

For the STK media changer:
- **home** status means that it is located in it slot.
- **In use** status means that is is mounted in a drive.

4.11 enstore monitor

Without any switches or arguments, **enstore monitor** contacts all
Enstore machines to run a simple network rate test. The results are then sent
to the monitor server running on the same machine as the configuration server
and web server. If the monitor server on the configuration server and web
server node is not running then this command will eventually timeout, but will
run very slowly.
Syntax:
% enstore monitor [--options [arguments] ...]
Options:

-h, --help
Prints the options (i.e. Prints this message). Example:

```
$ enstore monitor --help

Usage:
      mon [OPTIONS]...

  -a, --alive              prints message if the server is up or down.
  -h, --help               prints this messge
      --host <HOSTIP>      selects a single host
      --html-gen-host <HTML_GEN_HOST>  ip/hostname of the html server
      --port <PORT>        selects a port
      --retries <ALIVE_RETRIES>  number of attempts to resend alive requests
      --summary            summary for saag
      --timeout <SECONDS>  number of seconds to wait for alive response
      --usage              prints short help message
      --verbose <VERBOSE>  print out information.
```

--host <HOSTIP>
Specify a single host to run the rate test with. By default all Enstore hosts are contacted. HOSTIP is allowed to be a host name or IP address.

```
$ enstore monitor --host stkensrv2.fnal.gov

Trying stkensrv2.fnal.gov
Network rate measured at 11.16 MB/S recieving and 11.25 MB/S sending.
```

--html-gen-host <HTML_GEN_HOST>
This switch allows for the user to override the default hostname of the primary monitor server. Normally the rate results are sent to the monitor server running on the same node as the configuration server and the web server.

```
$ enstore monitor --host stkensrv2.fnal.gov --html-gen-host
gccensrv1.fnal.gov

Trying stkensrv2.fnal.gov
Network rate measured at 11.19 MB/S recieving and 11.24 MB/S sending.
```

--port
This switch is needed to use communicate with a monitor server that was started using a different port number than the default 7499.

```
$ enstore monitor --host stkensrv2.fnal.gov --port 7499

Trying stkensrv2.fnal.gov
Network rate measured at 11.18 MB/S recieving and 11 MB/S sending.
```

--summary
This switch outputs a python dictionary that is used internally by the **enstore network** command.

4.12 enstore mover

Syntax:

```
% enstore mover [--options [arguments] ...] <mover>
```
Options:

-h, --help

Prints the options (i.e. Prints this message). Example:

```
$ enstore mover --help

Usage:
      mov [OPTIONS]... mover_name


  -a, --alive              prints message if the server is up or down.
      --clean-drive        clean tape drive
      --do-alarm <DO_ALARM>  turns on more alarms
      --do-log <DO_LOG>      turns on more verbose logging
      --do-print <DO_PRINT>  turns on more verbose output
      --dont-alarm <DONT_ALARM>  turns off more alarms
      --dont-log <DONT_LOG>  turns off more verbose logging
      --dont-print <DONT_PRINT>  turns off more verbose output
      --down               set mover to offline state
      --dump               get the tape drive dump (used only with M2
                           movers)
  -h, --help               prints this messge
      --list               list all movers in configuration
      --mover_dump         send mover internals to stdout
      --notify <E_MAIL_ADDRESS>  send e-mail.  Used with --dump option only
      --offline            set to offline state
      --online             set to online state
      --retries <ALIVE_RETRIES>  number of attempts to resend alive requests
      --sendto <E_MAIL_ADDRESS>  send e-mail.  Used with --dump option only
      --status             print mover status
      --timeout <SECONDS>  number of seconds to wait for alive response
      --up                 set mover to online state
      --usage              prints short help message
      --warm-restart       gracefully restart the mover
```

--clean-drive

Clean the drive associated with the mover.

```
$ enstore mover --clean-drive 9940B27.mover
```

--down

Set the mover to the offline state. This will prevent the mover from
requesting additional work items from the library manager. This is the same
command as --offline.

```
$ enstore mover --down LTO3_06.mover
```

--dump

Output the tape drive dump. This output goes to the file that stdout has been
redirected to for the mover; not the stdout for the mover client. Only works
with Mammoth 2 tape drives.

```
$ enstore mover --dump M201.mover
```

--mover_dump
Output the mover internals dump. This output goes to the file that stdout has been redirected to for the mover; not the stdout for the mover client.

$ enstore mover --mover-dump 9940B27.mover

--notify <E_MAIL_ADDRESS>

--offline
Set the mover to the offline state. This will prevent the mover from requesting additional work items from the library manager. This is the same command as --down.

$ enstore mover -offline LTO3_06.mover

--online
Set the mover to the online state. This will allow the mover to begin requesting additional work items from the library manager. This is the same command as --up.

$ enstore mover -online 9940B27.mover

--sendto <E_MAIL_ADDRESS>

--status

Prints in native python format current status and/or state information about the mover.

```
$ enstore mov --status LTO3_06.mover

{'buffer': 'Buffer 67108864  79691776  2621440000',
 'buffer_max': 2621440000L,
 'buffer_min': 67108864L,
 'bytes_buffered': 79691776,
 'bytes_read': 347280136L,
 'bytes_read_last': 347280136L,
 'bytes_to_transfer': 2440004730L,
 'bytes_written': 267386880L,
 'client': '131.225.189.74',
 'current_location': 6,
 'current_volume': 'VOI177',
 'default_dismount_delay': 20,
 'drive_id': 'ULTRIUM-TD3',
 'drive_sn': '1210116666',
 'drive_vendor': 'IBM',
 'files':
('cmsstor74.fnal.gov:/storage/data1/write-pool-1/data/00070000000000000FBD84E
8',

'/pnfs/fnal.gov/usr/cms/WAX/11/store/mc/2008/2/6/TaS-W0jet-alpgen-Skim_01_AOD
SIM/0003/64C34427-68D5-DC11-AF3B-0018F3D09702.root'),
 'last_error': ('ok', None),
 'last_location': 34,
 'last_volume': 'VOE330',
 'max_dismount_delay': 600,
 'mode': 'WRITE',
 'rate of network': 54970350.576162718,
 'rate of tape': 71208384.297553152,
 'state': 'ACTIVE',
 'status': ('ok', None),
 'successful_writes': 56,
 'time_in_state': 6.3608489036560059,
 'time_stamp': 1202403301.7609749,
 'transfers_completed': 189,
 'transfers_failed': 0}
```

--up

Set the mover to the online state. This will allow the mover to begin requesting additional work items from the library manager. This is the same command as --online.

```
$ enstore mover -up 9940B27.mover
```

--warm-restart

Using this switch will instruct the mover to restart. If a file transfer is currently in progress, the restart is delayed until the transfer is completed, the volume is ejected and the volume is put away.

```
$ enstore mover -warm-restart --M201.mover
```

4.13 enstore network

Syntax:

% enstore network [--options [arguments] ...]

Options:

-h, --help

Prints the options (i.e. Prints this message). Example:

$ enstore network --help

```
Usage:
       network [ -h --help --html-gen-host= --usage ]

  -h, --help                 prints this messge
       --html-gen-host <NODE_NAME>  ip/hostname of the html server
       --usage               prints short help message
```

--html_gen_host

See section 1.11 for the **enstore monitor --html-gen-host** command.

4.14 enstore pnfs

Syntax:

% enstore pnfs [--options [arguments] ...]

Options:

-h, --help
Prints the options (i.e. Prints this message). Example:

```
$ enstore pnfs --help

Usage:
      pnfs [OPTIONS]...

      --bfid <FILENAME>     lists the bit file id for file
      --cat <FILENAME> [LAYER]  see --layer
      --const <FILENAME>
      --counters <FILENAME>
      --countersN <DBNUM>    (must have cwd in pnfs)
      --cp <UNIXFILE> <FILENAME> <LAYER>  echos text to named layer of the
                            file
      --cursor <FILENAME>
      --database <FILENAME>
      --databaseN <DBNUM>    (must have cwd in pnfs)
      --down <REASON>       creates enstore system-down wormhole to prevent
                            transfers
      --dump                dumps info
      --duplicate [FILENAME] [DUPLICATE_FILENAME]  gets/sets duplicate file
                            values
      --echo <TEXT> <FILENAME> <LAYER>  sets text to named layer of the file
      --file-family [FILE_FAMILY]  gets file family tag, default; sets file
                            family tag, optional
      --file-family-width [FILE_FAMILY_WIDTH]  gets file family width tag,
                            default; sets file family width tag, optional
      --file-family-wrapper [FILE_FAMILY_WRAPPER]  gets file family wrapper
                            tag, default; sets file family wrapper tag,
                            optional
      --filesize <FILE>     print out real filesize
  -h, --help                prints this messge
      --id <FILENAME>       prints the pnfs id
      --info <FILENAME>     see --xref
      --io <FILENAME>       sets io mode (can't clear it easily)
      --layer <FILENAME> [LAYER]  lists the layer of the file
      --library [LIBRARY]   gets library tag, default; sets library tag,
                            optional
      --ls <FILENAME> [LAYER]  does an ls on the named layer in the file
      --mount-point <FILENAME>  prints the mount point of the pnfs file or
                            directory
      --nameof <PNFS_ID>    prints the filename of the pnfs id (CWD must be
                            under /pnfs)
      --parent <PNFS_ID>    prints the pnfs id of the parent directory (CWD
                            must be under /pnfs)
      --path <PNFS_ID>      prints the file path of the pnfs id (CWD must be
                            under /pnfs)
      --position <FILENAME>
      --rm <FILENAME> <LAYER>  deletes (clears) named layer of the file
      --showid <PNFS_ID>    prints the pnfs id information
      --size <FILENAME> <FILESIZE>  sets the size of the file
      --storage-group [STORAGE_GROUP]  gets storage group tag, default; sets
                            storage group tag, optional
      --tag <TAG> [DIRECTORY]  lists the tag of the directory
      --tagchmod <PERMISSIONS> <TAG> changes the permissions for the tag;use
                            UNIX chmod style permissions
      --tagchown <OWNER> <TAG>  changes the ownership for the tag; OWNER can
                            be 'owner' or 'owner.group'
      --tagecho <TEXT> <TAG>  echos text to named tag
      --tagrm <TAG>         removes the tag (tricky, see DESY documentation)
      --tags [DIRECTORY]    lists tag values and permissions
      --up                  removes enstore system-down wormhole
      --usage               prints short help message
      --xref <FILENAME>     lists the cross reference data for file
```

--const <FILENAME>

Outputs constant information from pnfs for a target file or directory in PNFS.

```
$ enstore --const /pnfs/test/mydir
```

```
MD2_P_VERSION=30109
MD2_RECORD_LENGTH=1012
MD_MAX_NAME_LENGTH=200
MD_HASH_SIZE=128
MAX_BODY_SIZE=928
MAX_TAG_NAME_SIZE=62
TAG_DATA_SIZE=762
HASH_HANDLES=66
HASH_POINTERS=77
DATA_POINTERS=77
DIR_ITEMS=3
DATA_UNITS=928
```

This shell command has the same result.

```
$ cat "/pnfs/test/.(const)(NULL)"
```

--counters <FILENAME>

Outputs constant information from pnfs for a target file or directory in PNFS.

```
$ enstore pnfs --counters /pnfs/test/mydir
```

```
mist=1
time=1202415188
getroot=0
get_record=87
getattr=162567
lookup=53634
mkdir=0
setattr=269
rmdir=0
readdir=438
mkfile=63
rmfile=66
rename=35
mklink=0
readlink=42
readdata=4089
writedata=66
setsize=0
setperm=0
truncate=0
rmfromdir=32
addtodir=32
chparent=0
delobject=0
forcesize=0
NULL=0
looponly=32
command=0
get_chain=0
find_id=64
mod_link=64
setattrs=0
rmfromdirpos=0
mod_flags=0
```

This shell command has the same result.

```
$ cat "/pnfs/test/mydir/.(get)(counters)"
```

--countersN <DBNUM>

--cp <UNIXFILE> <FILENAME> <LAYER>

Copies a text file into the named layer of the specified file. In the following example, the contents of /tmp/L4 are copied into layer 4 of the file /pnfs/test/mydir/somefile. This is similar to what --**echo** does, but makes it easier to write multi line output to the layer.

To see what we currently have and what we believe we will want to replace the layer information with. Note the difference between the volume name and the location.

```
$ cat /tmp/L4
TEST01
0000_000000000_0000397
1024
my_file_family
/pnfs/test/mydir/somefile

000100000000000000018820

WAMS103661998200000
testnode:/dev/null:0
$ enstore pnfs --layer /pnfs/test/mydir/somefile 4
TEST42
0000_000000000_0000402
1024
my_file_family
/pnfs/test/mydir/somefile

000100000000000000018820

WAMS103661998200000
testnode:/dev/null:0
```

Now update the layer 4 information.

```
$ enstore pnfs --cp /tmp/L4 /pnfs/test/mydir/somefile 4
```

Verify that the contents are what we expect.

```
$ enstore pnfs --xref /pnfs/test/mydir/somefile
volume: NULL01
location_cookie: 0000_000000000_0000397
size: 1024
file_family: my_file_family
original_name: /pnfs/test/mydir/somefile
map_file:
pnfsid_file: 000100000000000000018820
pnfsid_map:
bfid: WAMS103661998200000
origdrive: testnode:/dev/null:0
crc: unknown
```

--cursor <FILENAME>
Outputs the PNFS specific cursor information to standard out about the specified directory or parent directory of the file specified.

```
$ enstore pnfs --cursor /pnfs/test
```

```
dirID=0001000000000000000010A8
dirPerm=0000001400000020
mountID=0001000000000000000001060
```
This shell command has the same result.

```
$ cat "/pnfs/test/.(get)(cursor)"
```

--database <FILENAME>
Output the PNFS database information for the specified file or directory.

```
$ enstore pnfs --database /pnfs/data1
```

```
data1:1:r:enabled:/diska/pnfsdb/pnfs/databases/data1
```
This shell command has the same result.

```
$ cat "/pnfs/data1/.(get)(database)"
```

--databaseN

--dump
After the command is completed, additional diagnostic information is also printed to standard out.

--echo <TEXT> <FILENAME> <LAYER>
Writes the contents of TEXT into the named layer of the specified file. This switch is most useful in writing information to layer 1 of a file.

Look at the current layer 1 information.

```
$ enstore pnfs --layer /pnfs/test/mydir/somefile 1
```

```
WAMS103667492900000
```
Update the layer 1 information.

```
$ enstore pnfs --echo \
WAMS103661998200000 /pnfs/test/mydir/somefile 1
```

Verify that the value is now correct.

```
$ enstore pnfs --bfid /pnfs/test/mydir/somefile
```

```
WAMS103661998200000
```

--id <FILENAME>

Outputs the pnfsid of the specified filename or directory.

```
$ enstore pnfs --id /pnfs/test/mydir/somefile
```
```
000100000000000000018820
```
This shell command has the same result.

```
$ cat "/pnfs/test/mydir/.(id)(somefile)"
```

--io <FILENAME>

Feature not yet implemented.

--ls

Forks an ls on the named layer of a file.

```
$ enstore pnfs --ls /pnfs/test/mydir/fname
```
```
-rw-r-----  1 enstore g023 127 Nov  6  2002 /pnfs/test/mydir/.(use)(4)(fname)
```
This shell command has the same result.

```
$ ls -l "/pnfs/test/mydir/.(use)(4)(fname)"
```

--mount-point <FILENAME>

Outputs the directory that is the mount point for the mounted PNFS file
system that the specified file or directory belongs to.

```
$ enstore pnfs --mount-point /pnfs/test/mydir/somefile
```
```
/pnfs/test
```

--nameof <PNFS_ID>

Output the name of the file or directory with the specified pnfsid.

```
$ enstore pnfs --nameof 000100000000000000018820
```
```
somefile
```
This shell command has the same result.

```
$ cat "/pnfs/test/.(nameof)(000100000000000000018820)"
```

--parent <PNFS_ID>

Output the pnfsid parent directory of the file or directory with the specified
pnfsid.

```
$ enstore pnfs --parent 000100000000000000018820
```
```
0001000000000000000010A8
```
```
$ enstore pnfs --nameof 0001000000000000000010A8
```
```
mydir
```
This shell command has the same result.

```
$ cat "/pnfs/test/.(parent)(000100000000000000018820)"
```

--path <PNFS_ID>
Outputs the full path of the file with the specified pnfsid.
WARNING: This command must be used with care. It launches a linear
search through the entire PNFS database for each component in the full file
path. Abuse of this command will result incredibly slow response times from
PNFS.
Note: If multiple PNFS servers are mounted on a single machine it is possible
that multiple matches are found. When this happens; the error message is
sent to standard error, a non-success exit status is returned and all of the
matched paths are printed to standard out.

```
$ enstore pnfs --path 00010000000000000000010A8
```

```
/pnfs/test/mydir
```

```
$ enstore pnfs --path 000100000000000000018820
```

```
/pnfs/test/mydir/somefile
```
A similar command is the pathfinder PNFS utility.

```
# pathfinder 00010000000000000000E468
```

```
00010000000000000000E468 testfile6
00010000000000000000010A8 mydir
00010000000000000001060 test
0000000000000000001080 usr
0000000000000000001040 fs
0000000000000000001020 root
0000000000000000001000 -
0000000000000000000100 -
0000000000000000000000 -
/root/fs/usr/test/mydir/testfile6
```

--position <DIRECTORY>
Outputs the PNFS specific position information to standard out about the
specified directory.

```
$ enstore pnfs -postion /pnfs/test
```

```
dirID=00010000000000000000010A8
dirPerm=0000001400000020
mountID=00010000000000000001060
```
This shell command has the same result.

```
$ cat "/pnfs/test/.(get)(position)"
```

--rm <FILENAME> <LAYER>
Feature not yet implemented.

--showid <PNFS_ID>

Outputs PNFS information about the specified pnfsid.

```
$ enstore pnfs -id /pnfs/test/mydir/somefile

000100000000000000018820

$ enstore pnfs -showid 000100000000000000018820

 -------------------------------------------------------
 ID             : 000100000000000000018820
 Type           : --I---r----
 next ID        : 000000000000000000000000
 base ID        : 000000000000000000000000
 parent ID      : 0001000000000000000010A8
 creation time  : Wed Nov  6 15:59:39 2002
 modif. time    : Thu Feb  7 14:40:18 2008
 Type           : Regular ( Inode )
 Info bytesPerBlock : 928
 Info blocksPerhash : 77
 mst_dev        : 1
 mst_ino        : 16877600
 mst_mode       : 100640
 mst_nlink      : 1
 mst_uid        : 9276
 mst_gid        : 1530
 mst_rdev       : 100
 mst_size       : 1024
 mst_atime      : Thu Nov 29 10:37:31 2007
 mst_mtime      : Wed Nov  6 15:59:42 2002
 mst_ctime      : Wed Nov  6 15:59:39 2002
 mst_blksize    : 512
 mst_blocks     : 0
 Entries(0)     : 0
 Chain(0)       : 000000000000000000000000
 Group(0)       : 0
 Entries(1)     : 19
 Chain(1)       : 000100000000000000018830
 Group(1)       : 0
 Entries(2)     : 0
 Chain(2)       : 000000000000000000000000
 Group(2)       : 0
 Entries(3)     : 0
 Chain(3)       : 000000000000000000000000
 Group(3)       : 0
 Entries(4)     : 127
 Chain(4)       : 000100000000000000018838
 Group(4)       : 0
 Entries(5)     : 0
 Chain(5)       : 000000000000000000000000
 Group(5)       : 0
 Entries(6)     : 0
 Chain(6)       : 000000000000000000000000
 Group(6)       : 0
 Entries(7)     : 0
 Chain(7)       : 000000000000000000000000
 Group(7)       : 0
```

This shell command has the same result.

```
cat "/pnfs/test/.(showid)(000100000000000000018820)"
```

--size <FILENAME> <SIZE>

Sets the file size as seen by the C library stat() function call.

```
$ ls -l /pnfs/test/mydir/zerofile
```

```
-rw-rw-r-- 1 enstore g023 0 Feb  7 15:52 /pnfs/test/mydir/zerofile
```

```
$ enstore pnfs --size /pnfs/test/mydir/zerofile 123456
```

```
$ ls -l /pnfs/test/mydir/zerofile
```

```
-rw-rw-r-- 1 enstore g023 123456 Feb  7 15:55 /pnfs/test/mydir/zerofile
```

--storage-group [STORAGE_GROUP]

Similar to --file-family, --file-family-width, --file-family-wrapper and --library. See the User's Guide for descriptions to those switches. All storage group tag files should be owned by root; thus requiring the effective user id to be root in order to modify the storage group. The current working directory must be in the PNFS directory whose tags need to be viewed or modified. See also --tagecho.

```
$ enstore pnfs --storage-group
```

```
old_storage_group
```

```
# enstore pnfs --storage-group new_storage_group
```

```
$ enstore pnfs --storage-group
```

```
new_storage_group
```

--tagecho <TEXT> <TAG>

Write the contents of TEXT to the named tag.

```
# enstore pnfs --tagecho new_storage_group storage_group
```

See also --file-family, --file-family-width, --file-family-wrapper and--library in the User's Guide and --storage-group. The current working directory must be in the PNFS directory whose tags need to be viewed or modified.

--tagrm <TAG>

Feature not yet implemented.

4.15 enstore pnfs_agent

Syntax:
```
% enstore pnfs_agent [--options [arguments] ...]
```
Options:

```
-h, --help
Prints the options (i.e. Prints this message).  Example:

$ enstore pnfs_agent --help

Usage:
        pnfs_agent [OPTIONS]...

   -a, --alive                  prints message if the server is up or down.
        --do-alarm <DO_ALARM>   turns on more alarms
        --do-log <DO_LOG>       turns on more verbose logging
        --do-print <DO_PRINT>   turns on more verbose output
        --dont-alarm <DONT_ALARM>   turns off more alarms
        --dont-log <DONT_LOG>   turns off more verbose logging
        --dont-print <DONT_PRINT>   turns off more verbose output
   -h, --help                   prints this messge
        --retries <ALIVE_RETRIES>   number of attempts to resend alive requests
        --status                print pnfs_agent status
        --timeout <SECONDS>     number of seconds to wait for alive response
        --usage                 prints short help message
```

4.16 enstore quota

Syntax:
% enstore quota [--options [arguments] ...]
Options:

```
-h, --help
Prints the options (i.e. Prints this message).  Example:

$ enstore quota --help

Usage:
        quota [OPTIONS]...

        --create <LIBRARY> <STORAGE_GROUP> [REQUESTED] [AUTHORIZED] [QUOTA]
create quota for (library, storage_group
                                )
        --delete <LIBRARY> <STORAGE_GROUP>  delete (library, storage_group)
        --disable               disable quota
        --enable                enable quota
   -h, --help                   prints this messge
        --set-authorized <LIBRARY> <STORAGE_GROUP> <NUMBER>  set authorized
                                number for (library, storage_group)
        --set-quota <LIBRARY> <STORAGE_GROUP> <NUMBER>  set quota for
(library,
                                storage_group)
        --set-requested <LIBRARY> <STORAGE_GROUP> <NUMBER>  set requested
number
                                for (library, storage_group)
        --show [LIBRARY] [STORAGE_GROUP]  show quota
        --show-by-library       show quota by the libraries
        --usage                 prints short help message
```

--create <LIBRARY> <STORAGE_GROUP> [REQUESTED] [AUTHORIZED] [QUOTA]
--delete <LIBRARY> <STORAGE_GROUP>
--disable
--enable
--set-authorized <LIBRARY> <STORAGE_GROUP> <NUMBER>
--set-quota <LIBRARY> <STORAGE_GROUP> <NUMBER>
--set-requested <LIBRARY> <STORAGE_GROUP> <NUMBER>
--show [LIBRARY] [STORAGE_GROUP]
--show-by-library

4.17 enstore ratekeeper

Syntax:

% enstore ratekeeper [--options [arguments] ...]

Options:

-h, --help
Prints the options (i.e. Prints this message). Example:

$ enstore ratekeeper --help

```
Usage:
      ratekeeper [ -ha --alive --help --retries= --timeout= --usage ]

  -a, --alive               prints message if the server is up or down.
  -h, --help                prints this messge
      --retries <ALIVE_RETRIES>  number of attempts to resend alive requests
      --timeout <SECONDS>   number of seconds to wait for alive response
      --usage               prints short help message
```

4.18 enstore restart

Without any switches or arguments, this restarts most Enstore processes on the current host. See also **enstore start** and **enstore stop**.
Syntax:
% enstore restart [--options [arguments] ...]
Options:

-h, --help
Prints the options (i.e. Prints this message). Example:
 	
$ enstore restart --help
 	
``
Usage:
        restart [-h --all --help --just= --usage]
 	
        --all   specify all servers
   -h, --help   prints this messge
        --just <SERVER NAME>   specify single server
        --usage   prints short help message
``

--all Restart all Enstore processes on the current host. This includes the **monitor server** and **pnfs_agent** excluded by the default list without any switches or arguments.
--just <SERVER_NAME> Restart just the server specified. **$ enstore restart --just log_server** ``` Checking log_server. Stopping log_server: 131.225.84.1:7504 Stopped log_server. Starting log_server: 131.225.84.1:7504 ```

4.19 enstore scan

At minimum, the **--type** argument is required. Its value may be **forward** or **reverse**, with the latter pending implementation.
Syntax:
% enstore scan --type=SCAN_TYPE [OPTIONS]
Options:

```
-h, --help
```

Prints the options.

```
$ enstore scan --help

Usage: enstore scan -t SCAN_TYPE [OPTIONS]

Options:
  -h, --help            show this help message and exit
  -t SCAN_TYPE, --type=SCAN_TYPE
                        (forward, reverse) scan type
  -d FS_ROOT, --directory=FS_ROOT
                        (for forward scan only) absolute path of directory to
                        scan recursively (recommended default is
                        /pnfs/fs/usr)
                        (not recommended to be specified for large nested
                        directories)
  -o OUTPUT_FILE, --output_file=OUTPUT_FILE
                        absolute path to output file for notices (default is
                        dynamic, e.g. /home/enstore/sca
n_20130301T180002.log)
                        (appended if exists)
  -p PRINT, --print=PRINT
                        (checks, notices) for the specified scan type, print
                        all checks and their overviews or all notice
                        templates, and exit
  -r, --resume          for specified output file (per -o), resume scan where
                        aborted (default is to restart scan) (use with same
                        database only)
  -s STATUS_INTERVAL, --status_interval=STATUS_INTERVAL
                        max status output interval in seconds (default
                        approaches 600)
```

Refer to the **Metadata Scanning** chapter for additional usage information.

4.20 enstore schedule

See enstore inquisitor command.

4.21 enstore start

Without any switches or arguments, this starts most Enstore processes on the current host. See also **enstore stop** and **enstore restart**.
Syntax:
% enstore start [--options [arguments] ...]
Options:

-h, --help
Prints the options (i.e. Prints this message). Example:

```
$ enstore start --help

Usage:
      restart [ -h --all --help --just= --usage ]

      --all                  specify all servers
  -h, --help                 prints this messge
      --just <SERVER NAME>   specify single server
      --nocheck              do not check if server is already running.
      --usage                prints short help message
```

--all
Start all Enstore processes on the current host. This includes the **monitor server** and **pnfs_agent** excluded by the default list without any switches or arguments.

--just <SERVER_NAME>
Start just the server specified.

```
$ enstore start --just log_server

Checking log_server.
Starting log_server: 131.225.84.1:7504
```

--nocheck
Assumes that the servers are already down. This speeds startup if it is already known that the servers will not still be running. Used by **enstore restart**.

4.22 enstore stop

Without any switches or arguments, this stops most Enstore processes on the current host. See also **enstore start** and **enstore restart**.
Syntax:
% enstore stop [--options [arguments] ...]
Options:

-h, --help
Prints the options (i.e. Prints this message). Example:

```
$ enstore stop --help

Usage:
      restart [ -h --all --help --just= --usage ]

      --all                  specify all servers
  -h, --help                 prints this messge
      --just <SERVER NAME>   specify single server
      --usage                prints short help message
```

--all
Stop all Enstore processes on the current host. This includes the **monitor server** and **pnfs_agent** excluded by the default list without any switches or arguments.

--just <SERVER_NAME>
Stop just the server specified.

```
$ enstore stop --just log_server

Checking log_server.
Stopping log_server: 131.225.84.1:7504
Stopped log_server.
```

4.23 enstore system

Updates the Status at a Glance page and System Summary web pages.
Syntax:
`% enstore system [--options [arguments] ...]`
Options:

-h, --help
Prints the options (i.e. Prints this message). Example:

```
$ enstore system --help

Usage:
        system [ -h --help --usage ]

  -h, --help                prints this messge
        --usage             prints short help message
```

4.24 enstore up_down

Syntax:
`% enstore up_down [--options [arguments] ...]`
Options:

-h, --help
Prints the options (i.e. Prints this message). Example:

```
$ enstore up_down --help

Usage:
        up_down [ -h --help --make-html --no-mail --summary --usage ]

  -h, --help                prints this messge
        --make-html         format output as html
        --no-mail           do net send e-mail in case of errors
        --summary           print (stdout) server states
        --usage             prints short help message
```

--make-html
--no-mail
--summary

4.25 enstore volume

Syntax:
% enstore up_down [--options [arguments] ...]
Options:

-h, --help
Prints the options (i.e. Prints this message). Example:

```
$ enstore volume --help

Usage:
      vol [OPTIONS]...

      --VOL1OK              reset cookie to '0000_000000000_0000001'
      --add <VOLUME_NAME> <LIBRARY> <STORAGE_GROUP> <FILE_FAMILY> <WRAPPER>
          <MEDIA_TYPE> <VOLUME_BYTE_CAPACITY> [REMAINING_BYTES]  declare a
                            new volume
  -a, --alive               prints message if the server is up or down.
      --all                 used with --restore to restore all
      --assign-sg <STORAGE_GROUP> <VOLUME_NAME>  reassign to new storage
group
      --backup              backup voume journal -- part of database backup
      --bypass-label-check  skip syntatical label check when adding new
                            volumes
      --check <VOLUME_NAME>  check a volume
      --clear <VOLUME_NAME>  clear a volume
      --clear-sg            used with recycle to clear storage group
      --decr-file-count <COUNT>  decreases file count of a volume
      --delete <VOLUME_NAME>  delete a volume
      --do-alarm <DO_ALARM>  turns on more alarms
      --do-log <DO_LOG>      turns on more verbose logging
      --do-print <DO_PRINT>  turns on more verbose output
      --dont-alarm <DONT_ALARM>  turns off more alarms
      --dont-log <DONT_LOG>  turns off more verbose logging
      --dont-print <DONT_PRINT>  turns off more verbose output
      --erase <VOLUME_NAME>  erase a volume
      --export <VOLUME_NAME>  export a volume
      --forget-all-ignored-storage-groups  clear all ignored storage groups
      --forget-ignored-storage-group <STORAGE_GROUP>  clear a ignored
storage
                            group
      --full <VOLUME_NAME>  set volume to full
      --get-sg-count <LIBRARY> <STORAGE_GROUP>  check allocated count for
                            lib,sg
      --gvol <VOLUME_NAME>  get info of a volume in human readable time
                            format
  -h, --help                prints this messge
      --history <VOLUME_NAME>  show state change history of volume
      --ignore-storage-group <STORAGE_GROUP>  ignore a storage group. The
                            format is "<library>.<storage_group>"
      --import <EXPORTED_VOLUME_OBJECT>  import an exported volume object.
                            The file name is of the format
"vol.<volume_name>.obj"
      --just                used with --pvols to list problem
      --keep-declaration-time  keep declared time when recycling
      --labels              list all volume labels
      --list <VOLUME_NAME>  list the files in a volume
      --ls-active <VOLUME_NAME>  list active files in a volume
      --ls-sg-count         list all sg counts
      --migrated <VOLUME_NAME>  set volume to MIGRATED
      --modify <VOLUME_NAME>  modify a volume record -- extremely dangerous
      --new-library <LIBRARY>  set new library
      --no-access <VOLUME_NAME>  set volume to NOTALLOWED
      --not-allowed <VOLUME_NAME>  set volume to NOTALLOWED
      --pvols               list all problem volumes
      --read-only <VOLUME_NAME>  set volume to readonly
      --rebuild-sg-count    rebuild sg count db
      --recycle <VOLUME_NAME>  recycle a volume
      --reset-lib <LIBRARY>  reset library manager
      --restore <VOLUME_NAME>  restore a volume
```

--VOL1OK Used in conjunction with --add. This modifier lets Enstore know that the volume will have an ANSI VOL1 label already written on the media. If there is not a VOL1 label already on the volume do not use this switch.
--add <VOLUME_NAME> <LIBRARY> <STORAGE_GROUP> \ <FILE_FAMILY> <WRAPPER> \ <MEDIA_TYPE> <VOLUME_BYTE_CAPACITY> \ [REMAINING_BYTES] Used to declare a new volume to Enstore. The VOLUME_NAME, LIBRARY, and MEDIA_TYPE need to be set when this command is executed. The LIBRARY, STORAGE_GROUP, FILE_FAMILY AND WRAPPER can in any combination be set to their correct value or the string "none" (without the double quotes). The LIBRARY value, when specified, needs to match the short name (without the trailing .library_manager) of the library it will belong to. The MEDIA_TYPE must match a media type defined in the **blocksizes** section of the configuration file (see section ?). `$ enstore volume --add TS0001 none none none none LTO 800G` See --vol or --gvol for the example to show the meta data for TS0001.
--all

--assign-sg <STORAGE_GROUP> <VOLUME_NAME>
Specify a new storage group for this volume. See also --new-library.
 	
`$ enstore volume --assign-sg new_sg NULL01`
 	
`BAD STATUS ('V-ERROR', 'can not reassign from existing storage group zee')`

--backup Backup the volume journal. `$ enstore volume --backup`

--bypass-label-check
Used with --add to skip the check if a volume name is valid. The format of a valid ANSI label in this exact order is:
    1. 2 Alphabetical characters
    2. 2 Alphanumeric characters
    3. 2 Numeric characters
    4. For LTO tapes in ADIC robots only, an appended L1 or L2.
The following example shows an 'illegal' label name being defined; the fifth character in the label is alphabetical not numeric.
 	
`$ enstore volume -add STORM1 none none none none LTO 100G \`
`--bypass-label-check`

--check <VOLUME_NAME>
Output the current status of the volume. This output puts everything on one line, instead of multiple lines like --vol, --gvol does.

```
$ enstore vol --check TEST01

TEST01        60.14GB ['NOTALLOWED', 'none'] ['none', 'none']
```
The contents of the output are:
1. The volume name.
2. The unused bytes renaming on the volume.
3. The pair of system inhibits.
4. The pair of user inhibits.

--clear <VOLUME_NAME> [system_inhibit | user_inhibit <1 | 2>]
Clear the any system inhibit on the volume. By default "system_inhibit 1" is assumed if not specified.
Example to clear NOTALLOWED or NOACESS system inhibits on the volume.

```
$ enstore vol--check TEST01

TEST01        60.14GB ['NOTALLOWED', 'none'] ['none', 'none']

$ enstore vol --clear TEST01


$ enstore vol --check TEST01

TEST01        60.14GB ['none', 'none'] ['none', 'none']
```
Example to clear full or readonly system inhibits on the volume:

```
bash-3.00$ enstore vol --check TEST04

TEST04        59.82GB ['none', 'full'] ['none', 'none']

bash-3.00$ enstore vol --clear TEST04 system_inhibit 2

bash-3.00$ enstore vol --check TEST04

TEST04        59.82GB ['none', 'none'] ['none', 'none']
```

--clear-sg
Used with --recycle to clear the storage group. This will allow the volume to be placed into the common blank pool for any experiment to use.

--decr-file-count <COUNT>

--delete <VOLUME_NAME>
Delete a volume. The volume must contain only files that are marked deleted. If the volume contained zero files on it, the volume is totally removed from the system. If the volume contained files, then the volume is renamed to have .deleted appended to it. To remove a .deleted file see --erase.
First the empty volume example:

```
$ enstore volume --check TEST01

TEST01     1000.00GB ['none', 'none'] ['none', 'none']

$ enstore volume --delete TEST01

$ enstore volume --check TEST01

BAD STATUS ('NO SUCH VOLUME', 'Info Clerk: no such volume TEST01')

$ enstore volume --check TEST01.deleted

BAD STATUS ('NO SUCH VOLUME', 'Info Clerk: no such volume TEST01.deleted')
```

Here is the non-empty volume example:

```
$ enstore volume --check TEST02

TEST02     1000.00GB ['none', 'none'] ['none', 'none']

$ enstore volume --delete TEST02

$ enstore volume --check TEST02.deleted

TEST02.deleted   60.32GB ['DELETED', 'none'] ['none', 'none']
```

--erase <VOLUME_NAME>
Remove a .deleted volume from the system. This command is used after --delete or --recycle.

```
$enstore volume -check TEST02.deleted

TEST02.deleted   60.32GB ['DELETED', 'none'] ['none', 'none']

$ enstore volume -erase TEST02.deleted

$ enstore volume --check TEST02.deleted

BAD STATUS ('NO SUCH VOLUME', 'Info Clerk: no such volume TEST02.deleted')
```

--export <VOLUME_NAME>

--forget-all-ignored-storage-groups

--forget-ignored-storage-group <STORAGE_GROUP>

--full <VOLUME_NAME>

Set the specified volume's status to full.

```
$ enstore volume --check STORM3

STORM3        59.82GB ['none', 'none'] ['none', 'none']

$ enstore volume --full STORM3


$ enstore volume --check STORM3

STORM3        59.82GB ['none', 'full'] ['none', 'none']
```

--get-sg-count <LIBRARY> <STORAGE_GROUP>

Lists the number of volumes belonging to the indicated library and storage group pairing. This as the the same functionality as **enstore info** **--get-sg-count**.

```
$ enstore volume -get-sg-count LTO3 test

        rain            zee         3
```

--gvol <VOLUME_NAME>

Similar to --vol, but the time values are converted to human readable strings instead of seconds from the beginning of the epoch.

```
$ enstore volume --gvol VO2345

{'blocksize': 131072,
 'capacity_bytes': 214748364800L,
 'comment': '',
 'declared': 'Fri Aug  3 08:42:35 2007',
 'eod_cookie': '0000_000000000_0000130',
 'external_label': 'VO2345',
 'first_access': 'Tue Aug 28 07:09:39 2007',
 'last_access': 'Tue Aug 28 09:29:07 2007',
 'library': 'CD-9940B',
 'media_type': '9940B',
 'remaining_bytes': 134423138304L,
 'si_time': ('Wed Dec 31 18:00:00 1969', 'Tue Jul 17 02:56:42 2007'),
 'sum_mounts': 51,
 'sum_rd_access': 0,
 'sum_rd_err': 0,
 'sum_wr_access': 129,
 'sum_wr_err': 0,
 'system_inhibit': ['none', 'none'],
 'user_inhibit': ['none', 'none'],
 'volume_family':
'minos.mcout_cedar_phy_near_daikon_00_L010170N_cand.cpio_odc',
 'wrapper': 'cpio_odc',
 'write_protected': 'n'}
```

--history <VOLUME_NAME>
Output the inhibit changes for the volume.

```
$ enstore volume --history BZG023

2008-02-11 13:19:17.892302    system_inhibit[0]    none
2008-02-11 13:18:17.22456     system_inhibit[1]    full
2007-09-18 14:17:05.716276    system_inhibit[0]    none
2007-07-18 15:16:37.713169    system_inhibit[0]    NOACCESS
2007-01-31 10:25:22.996649    system_inhibit[0]    none
2007-01-31 10:15:26.246705    system_inhibit[0]    NOACCESS
```

--ignore-storage-group <STORAGE_GROUP>

--import <EXPORTED_VOLUME_OBJECT>

--just

--keep-declaration-time
Keep the current declaration time when recycling a volume. See also
--recycle.

--labels
List all volume names.

```
$ enstore volume --labels | head

NUL001
NUL002
NUL003
NUL004
NUL005
NUL006
NUL007
NUL008
NUL009
NUL010
```

--migrated <VOLUME_NAME>

--modify <VOLUME_NAME>
Powerful command to modify all database fields (except volume names).
See --vol or --gvol for examples of outputting the contents of these changes.
Example to reset the comment field:

```
$ enstore volume --modify TEST03 'comment=""'
```

Example to reset the system inhibit:

```
$ enstore volume –modify TEST03 \
"system_inhibit=['none', 'full']"
```

--new-library <LIBRARY>
--no-access <VOLUME_NAME> Synonym for --not-allowed.

--not-allowed <VOLUME_NAME>
Set the volume system inhibit to NOTALLOWED.
 	
$ enstore volume --check TZ4562
 	
`TZ4562      59.82GB ['none', 'full'] ['none', 'none']`
 	
bash-3.00$ enstore volume --not-allowed TZ4562
 	
bash-3.00$ enstore volume --check TZ4562
 	
`TZ4562      59.82GB ['NOTALLOWED', 'full'] ['none', 'none']`

--read-only <VOLUME_NAME>
Set the specified volume's status to readonly.
 	
$ enstore volume --check TZ4562
 	
`TZ4562      59.82GB ['none', 'none'] ['none', 'none']`
 	
$ enstore volume --full TZ4562
 	
$ enstore volume --check TZ4562
 	
`TZ4562      59.82GB ['none', 'readonly'] ['none', 'none']`

--rebuild-sg-count
--recycle <VOLUME_NAME> Rename the current volume by VOLUME_NAME to VOLUME_NAME.deleted. Then declare a new VOLUME_NAME volume while preserving information like mount and access counts. See also --clear-sg and --keep-declaration-time. **$enstore volume --recycle JK4562**
--reset-lib <LIBRARY>
--restore <VOLUME_NAME>

--set-comment \<COMMENT\> \<VOLUME_NAME\> Set the comment field for a volume. `$ enstore volume --vol GR4444	grep comment`  `'comment': '',`  `$ enstore volume --set-comment "Possible issue at location` `42." GR4444`  `$ enstore volume --vol GR4444	grep comment`  `'comment': 'Possible issue at location 42.',` To reset the comment back to being emtpy some care needs to be used to avoid the shell eating the double quotes.  `$ enstore volume --set-comment \"\" GR4444`
--set-sg-count \<LIBRARY\> \<STORAGE_GROUP\> \<COUNT\>
--show-ignored-storage-groups
--show-quota
--show-state Show the internal state of the volume clerk.
--trim-obsolete \<VOLUME_NAME\>
--write-protect-off \<VOLUME_NAME\> Set the write protect tab state to off for the specified volume. See also --write-protect-on and --write-protect-status.
--write-protect-on \<VOLUME_NAME\> Set the write protect tab state to on for the specified volume. See also --write-protect-off and --write-protect-status.

--write-protect-status <VOLUME_NAME>
Show the write protect tab state for the specified volume. See also
--write-protect-off and --write-protect-on.

```
$ enstore volume --write-protect-status GAME42
GAME42 write-protect OFF
$ enstore volume --write-protect-on GAME42

$ enstore volume --write-protect-status GAME42
GAME42 write-protect ON
$ enstore volume --write-protect-off GAME42

$ enstore volume --write-protect-status GAME42
GAME42 write-protect OFF
```

Chapter 5: Migration and Duplication

Migration is the task of reading files from one tape then writing them onto another tape with the intend of replacing the original. Usually this is to put them on newer tapes, typically using newer tape technology.

Duplication reads original files and instead of replacing the old copies makes the newly written copies a duplicate[1] of the original.

Cloning via migration occurs when the source media type and destination media type are the same.

There are four stages recorded in the progress of file migration or duplication. These states are recored in the **migration** table in the Enstore DB.
1. **Copied** – the original file(s) has been read from the source tape and written to the new tape.
2. **Swapped**
    a) Migration – The meta data has been swapped. This means that the new copy is the one users see from PNFS.
    b) Duplication – The new file's meta data has be recorded as a duplicate of the original file.
3. **Checked** – The new copy on the new tape has been read to verify that everything went correctly with reading and writing the file.
4. **Closed** – The file copying and verification is done.

See the **--status** switch for information on these states. If the file copying becomes interrupted; it can safely be restarted and it will pick up where it left off.

There are three times that a volume's meta data are modified:
1. The original volume is set to **migrating**, **duplicating** or **cloning** when the first file is attempted to be migrated, duplicated or cloned via migration. This is to prevent additional files being written to the volume while it is being copied.
2. After all files on a original volume are copied and swapped:
    a) The original volume's meta data has its system inhibit set to **migrated**, **duplicated** or **cloned**.
    b) The original volume's comment is set to record the volumes the files were migrated or duplicated onto.
3. After all the files on the destination volume are copied, swapped,

---

1 Encp can make multiple copies (A.K.A. duplicates) of files using the **--copies** switch. Duplication, as described here, mimics this feature of encp.

checked and closed:

    a) The comment of the new tape is set to recored the volume list that the files were migrated or duplicated from.

    b) The **migration_history** is updated to record the original volume, destination volume and the time stamp this was completed.

    c) For migration only: The file_family is set to the non-migration mangled file family. Duplication leaves this value alone because it has already been set to the correct multiple copy mangled file family.

There are three modes to migration or duplication:

1) Copying a list of files. Internally, it is converted to a list of bfids.
2) Copying a list of bfids.
3) Copying all the files on a list of volumes.

Migrating or duplicating an entire volume's worth of files at one time is usually more efficient.

5.1 Preparation

Before migrating or duplicating files the following steps are recommended.

1. Check that **/pnfs/fs/** is mounted on the node the migration or duplication will be run on. If multiple PNFS installations are present, make sure the correct one is currently mounted.
2. Check that the pnfs tags in the **/pnfs/fs/usr/Migration** directory contain the correct values. It is strongly recommended that this directory be its own PNFS database.
3. Use the **enstore info --show-bad** command to identify any files that will be expected to be unreadable.
4. Use the **enstore scan** command to check the meta data of the source bfids, files or volumes.

5.2 Migration

5.2.1 Reasons for migration

Three main reasons exist for migrating data to new tapes.

1. Using denser media with faster tape drives.
2. Data compaction or "squeezing" of active data files to free up space occupied by deleted files.
3. The tape a file(s) is currently on is no longer reliable.

5.2.2 Migration command

Syntax, as root:

```
# migrate [ --option [argument] ...] [target_list]
# migrate [ --option [argument] ...] [media_type
[library [storage_group [file_family [wrapper]]]]]
# migrate --restore [target_list]
# migrate --scan [target_list]
```

Syntax, as any user:

```
% migrate --status [target_list]
% migrate --migrated-from <volume_list>
% migrate --migrated-to <volume_list>
```

The target list can be volume labels, BFIDs or file names in any order in any combination. Internally, file names are converted to BFIDs. All BFIDs are processed before volumes.

Here is an example of the primary use of this command to migrate three volumes worth of files to new media.[2]

```
# migrate --spool-dir /data/data2/Migration_Spool VO0000
VO0001 VO0002
```

A return code of zero (0) indicates success migrating all files (or volumes) in the argument list.

A volume that has been migrated will have its status set to reflect that it has been migrated. The following shows that OLD000 has been migrated to NEW001. **Note:** The comment contains the string "<=" or "=>". This indicates migration was done. If duplication were performed, this string would be "->" or "<-" instead. **Note**: A volume's comment field can be changed at any time; especially if a problem with the tape arises. This clobbering means that this can not be the only means used to determine if a volume contains migrated (or duplicated) files.

```
$ enstore info --vol OLD000 | egrep "comment|system_inhibit"

 'comment': '=> NEW001',
 'system_inhibit': ['none', 'migrated'],

$ enstore info --vol NEW001 | egrep "comment|system_inhibit"

 'comment': '<= OLD000',
 'system_inhibit': ['none', 'none'],
```

A log of the migration, everything set to standard out, is also sent to:

---

2   The sample command has been split onto two lines with the standard shell continuation character (\).

/var/migration/MigrationLog@<timestamp>#<pid>

Options:

-h, --help
Prints the options (i. e. Prints this message). Example:

```
$ migrate --help
Usage:
      migrate [OPTIONS]... [bfid1 [bfid2 [bfid3 ...]]] | [vol1 [vol2
[vol3 ...]]] | [file1 [file2 [file3 ...]]] | [vol1:lc1 [vol2:lc2
[vol3:lc3 ...]]]
      migrate [OPTIONS]... [media_type [library [storage_group [file_family
[wrapper]]]]]
      migrate [OPTIONS]... --restore [bfid1 [bfid2 [bfid3 ...]] | [vol1
[vol2 [vol3 ...]]] | [file1 [file2 [file3 ...]]] | [vol1:lc1 [vol2:lc2
[vol3:lc3 ...]]]
      migrate [OPTIONS]... --scan [bfid1 [bfid2 [bfid3 ...]] | [vol1 [vol2
[vol3 ...]]] | [file1 [file2 [file3 ...]]] | [vol1:lc1 [vol2:lc2
[vol3:lc3 ...]]]
      migrate [OPTIONS]... --migrated-from <vol1 [vol2 [vol3 ...]]>
      migrate [OPTIONS]... --migrated-to <vol1 [vol2 [vol3 ...]]>
      migrate [OPTIONS]... --status [bfid1 [bfid2 [bfid3 ...]]] | [vol1
[vol2 [vol3 ...]]] | [file1 [file2 [file3 ...]]] | [vol1:lc1 [vol2:lc2
[vol3:lc3 ...]]]
      migrate [OPTIONS]... --show <media_type> [library [storage_group
[file_family [wrapper]]]]


      --destination-only    Used with --status to only list output assuming
                             the volume is a destination volume.
      --file-family <FILE_FAMILY>  Specify an alternative file family to
                             override the pnfs file family tag.
  -h, --help                 prints this message
      --infile <INFILE>      Read target list of bfids, volumes,
                             volume:location_cookie pairs or paths from file.
                              Types can be intermixed.
      --library <LIBRARY>    Specify an alternative library to override the
                              pnfs library tag.
      --migrated-from        Report the volumes that were copied to this
                              volume.
      --migrated-to          Report the volumes that were copied from this
                              volume.
      --migration-only       Used with --status to only list output assuming
                              the target is not orhas not a multiple copy.
      --multiple-copy-only   Used with --status to only list output assuming
                              the target is orhas a multiple copy.
      --priority <PRIORITY>  Sets the initial job priority.  Only
                              knowledgeable users should set this.
      --read-to-end-of-tape  Read to end of tape before starting to write.
      --restore              Restores the original file or volume.
      --scan                 Scan completed volumes or individual bfids.
      --show <MEDIA_TYPE> [LIBRARY] [STORAGE_GROUP] [FILE_FAMILY] [WRAPPER]
                              Report on the completion of volumes.
      --skip-bad             Skip bad files.
      --source-only          Used with --status to only list output assuming
                              the volume is a source volume.
      --spool-dir <SPOOL_DIR>  Specify the directory to use on disk.
      --status               Report on the completion of a volume.
                              S = State of duplication:
                                  P = Primary/original copy; duplication
```

```
                              C = Muliple copy; duplication
                              O = Original/primary copy
                              M = Multiple copy
                      D = Deleted state:
                         N = Not deleted
                         Y = Yes deleted
                         U = Unknown; failed write
                      B = Bad file
                         B = Bad file
                         E = Empty metadata fields
    --usage               prints short help message
    --use-disk-files      Skip reading files on source volume, use files
                           already on disk.
    --use-volume-assert   Use volume assert when scanning destination
                           files.
    --with-deleted        Include deleted files.
    --with-final-scan     Do a final scan after all the files are recopied
                           to tape.
```

--destination-only
Used with --status to report the requested targets as source volumes or
BFIDs. See also --source-only and --status.

--duplication-only
Used with --status to report duplication only. See also --duplication-only,
--multiple-copy-only and --status.

--file-family <FILE_FAMILY>
This switch overrides the original file family. This is typically used to
combine a number of smaller file families into one while squeezing a few
small capacity tapes onto new larger capacity tapes. This can help improve
file density by not wasting space on tapes.

--library <LIBRARY>
This switch overrides the PNFS library tag when writing the new copy.

It is permissible to specify a comma separated list of libraries, without any
whitespace. This will make additional multiple copies using the **encp**
--copies functionality.

--migrated-from <VOLUME_LIST>
Report on the list of volumes whose files to which files were migrated to the
requested volumes. This information does not come from the volumes
comment field; it is determined from the Enstore DB.

```
$ migrate --migrated-from PSA423 PSA403
```

```
PSA423 <= PRU482L1 PRV397L1
PSA403 <= PRV033L1 PRV034L1
```
See **--migrated-to** for information in the other direction.

--migrated-to <VOLUME_LIST>
Report on the list of volumes whose files to which files were migrated from
the requested volumes. This information does not come from the volumes

comment field; it is determined from the Enstore DB.

```
$  migrate --migrated-to PRV397L1 PRV034l1
```

```
PRV397L1 => PSA423
PRV03411 =>
```
See **--migrated-to** for information in the other direction.

--migration-only
Used with --status to report migration and cloning only. See also
--destination-only, --multiple-copy-only and --status.

--multiple-copy-only
Used with --status to report copies made by **encp** only and from duplication.
See also --destination-only, --migration-only and --status.

 --priority <PRIORITY>
The default priority for a migration **encp** is zero. This is lower than the
default priority of one for a typical **encp**. Use of this switch allows the user
to override the default to give this migration a higher priority.

--read-to-end-of-tape
Do not start writing files to the destination volume, until all source files have
been read from the source volume.

--restore <VOLUME_LIST> | <BFID_LIST> | <FILENAME_LIST>
After the swap step, the PNFS meta data points to the newly written copy of
the file. This command resets the PNFS meta data to point to the original
copy on the original tape, mark the old copy as undeleted, mark the new
copy as deleted and removes the temporary PNFS file from underneath
/pnfs/fs/usr/Migration/. If a volume was the target, then it also
resets the original volumes comment and system inhibit values.

```
# migrate --restore TEST01 TEST02
```

...

The sequence of commands to undo a volume migration are:
Undo the metadata swap.

```
# migrate --restore <VOLUME_LIST>
```

*Purge the BFIDs of the new tape copies from the Enstore (file & volume)
database.*
　　1.　*If all files on the new tape are from volumes that have just been
　　　　restored.*

```
$ enstore volume --modify <VOLUME> comment="" #Once for each
volume.
```

```
$ enstore volume --recycle <VOLUME>          #Once for each
volume.
```

```
$ enstore volume --erase <VOLUME>.deleted     #Once for each
volume.
```

> *The switches--delete and --bfid could be used instead of --recycle; if
> so then the -- modify command would be skipped.*
> 2. *If all files on the new tape are **not** from volumes that have just been
>   restored.*

```
$ enstore file --erase <BFID>     #Once for each bfid.
```

If a file on a newly migrated to volume is found to be unreadable; then you
can do the following:
Verify the bfid for the bad file.

```
# migrate --status TST982L1 | grep -A 2 -B 2
D0MS113811445800000

D0MS113811429500000 D0MS122085462500000      y       y      y      y
D0MS113811438100000 D0MS122085470200000      y       y      y      y
D0MS113811445800000 D0MS122085478000000      y       y
D0MS113811454200000 D0MS122085481800000      y       y
D0MS113811462400000 D0MS122085487200000      y       y
```
Undo the metadata swap for the unreadable file.

```
# migrate --restore D0MS113811445800000

Wed Oct  1 14:45:07 2008 MIGRATION migrate --restore D0MS113811445800000
Wed Oct  1 14:45:08 2008 RESTORE D0MS113811445800000 has already been marked
undeleted ... OK
Wed Oct  1 14:45:08 2008 RESTORE set D0MS122085478000000 deleted ... OK
```
Verify that there is no longer a destination bfid paired with the source bfid.

```
# migrate --status TST982L1 | grep -A 2 -B 2
D0MS113811445800000

D0MS113811429500000 D0MS122085462500000      y       y      y      y
D0MS113811438100000 D0MS122085470200000      y       y      y      y
D0MS113811445800000
D0MS113811454200000 D0MS122085481800000      y       y
D0MS113811462400000 D0MS122085487200000      y       y
```
Remigrate the file.

--scan <VOLUME_LIST> | <BFID_LIST> | <FILENAME_LIST>
This switch closes out the migration status of the specified target files or
volumes. The volumes or bfids supplied need to be a destination target. If a
file on a tape is found to be unreadable or corrupted, it is not marked as
checked and the migration status of the volume, if applicable, will not be
updated.

If a volume was not migrated with the –with-final-scan switch enabled; then
the files on these volumes will be read to verify that everything is correct. If
–with-final-scan was enabled (and the files passed); then the files are marked
closed and the volumes have their migration status updated.

```
$ migrate --scan PSA000 PSA001

...
```

If a specific set of files are listed by their bfid, only those files are scanned.
If all the remaining bfids on a volume are supplied and they all scan
successfully the volume metadata is still left alone. This could be useful in
scanning the files from a particular source volume, while leaving the
destination volume available for more migrated files to be written to it.

--source-only
Used with --status to report the requested targets as source volumes or
BFIDs. See also --destination-only and --status.

--spool-dir <SPOOL_DIR>
Specify the spool directory where the files are read from the original tape,
before they are written to the new tape.

--status <VOLUME_LIST> | <BFID_LIST> | <FILENAME_LIST>
Report on the current status of the requested volumes, file names or bfids.
The volumes specified can be either an original volume or a new volume.

```
$ migrate --status TST397L1

(TST397L1) src_bfid SDB            dst_bfid SDB copied swapped checked closed
D0MS120685047500000 PN  D0MS121088793700000 CN      y       y       y
D0MS120688739300000 PN  D0MS121088795600000 CN      y       y       y
D0MS120692204400001 PN  D0MS121088798600000 CN      y       y       y
D0MS120730702700000 PN  D0MS121088812000000 CN      y       y       y
D0MS120730706700000 PN  D0MS121088822600000 CN      y       y       y
D0MS120730709100000 PN  D0MS121088836900000 CN      y       y       y
D0MS120730711900000 PN  D0MS121088849400000 CN      y       y       y
D0MS120730716200000 PN  D0MS121088867100000 CN      y       y       y
D0MS120744939400000 CN  D0MS121088874500000 PN      y       y       y
D0MS120744941400000 CN  D0MS121088887000000 PN      y       y       y
D0MS120744961000000 CN  D0MS121088904000000 PN      y       y       y
D0MS120753217600000 CN  D0MS121096722600000 PN      y       y       y
D0MS120753222200000 CN  D0MS121096725100000 PN      y       y       y
D0MS120753229700000 CN  D0MS121096728100000 PN      y       y       y
D0MS120753229700000 YE


DUPLICATION
```

The left BFID is the source BFID; while the right BFID is the destination
BFID. In the above example the "y"s indicate that the files have been
copied, swapped and checked.; but not closed. This would indicate that the
--**with-final-scan** switch was used and --**scan-volumes** has not
been used yet.

At the end of the output it is reported that duplication was used. The list of
valid values that can be appear here is: **DUPLICATION**, **MIGRATION**,
CLONING or left blank.

Each BFID has a three columns of additional information:

S columns:

> For migration they would be left blank.
>
> For duplication a **P** indicates it is the primary copy of the file and a **C** indicates it is a multiple copy from a duplication. They appear immediately after the BFID located on the volume specified on the command line. In the above example, the volume on the command line was the source volume and that the first 8 files are primary copies, while the rest are multiple copies. See the section on swap_original_and_copy.py for more information on primary versus multiple copy files.
>
> A value of **M** represents a multiple copy and **O** represents an original copy. These are different from P and C, since M and O were created using the multiple copy feature of **encp** outside of any duplication effort. If migration is used to make multiple destination copies, then the --status output will include multiple entries for each source bfid.

D columns: These columns report if the associated source and destination BFIDs are marked deleted (**Y**) or not marked deleted (**N**) in the database. A **U** value is possible for a file on a destination volume when the file is the result of a failed transfer.

B columns: These columns indicate if the file has been marked bad using the **enstore file --mark-bad** command; such files are reported with a **B**. An **E**, for empty, will appear here if the file record is missing information, like the pnfs id and pnfs path name, indicating that the file was a failed transfer.

Here is another use of **--status** that can be used to report if the a volume is done or not. The success return code (0) means that the volume is done; while the error return code (1) indicates that it is not completed.

```
$ migrate --status PSA003 > /dev/null; echo $?
0
$ migrate --status PSA004 > /dev/null; echo $?
1
```

See also --destination-only, --duplication-only, --migration-only, --multiple-copy-only and --source-only.

--use-disk-files

Use the files already located in the spool directory to write to the new tape. The file names must have the format <volume>:<location_cookie> and refer the correct location on the original source tape. All files need to be present in the spool directory. All unreadable files need to be marked as bad using the **enstore file --mark-bad <path> [bfid]** command.

The intended purpose is to allow for tapes that are sent to a vendor for data recovery (and are returned with files in different locations on the tape) to still be migrated. The readable files need to be read using UNIX tools, like **dd**, to dump the files.

--with-deleted
This switch tell the migration process that deleted files on the original volumes are to be migrated too. These files are migrated to a different volume then their non-deleted counterparts. This allows for tapes to be squeezed while still isolating the files that are likely not to be considered important. Volumes containing deleted migrated files have "DELETED_FILES" at the beginning of their file family.

--with-final-scan
After the file meta data has been swapped for all the files on the tape; re-read all the migrated files on the new volume or volumes. If a file was skipped because it was not yet copied or swapped; it will not be marked as checked either. If this switch is used at the time of migration, then they will not been to be re-read when **--scan-volumes** is used to close the migration of the specified volumes.

In general, it is more efficient to use this switch when only copying a single volume's worth of files or a list of individual files. If multiple volumes are being copied using this switch is not recommended.

See **--scan-volumes**.

5.2.3 Migration file_family mangling

The file family of a destination migration volume is appended with **-MIGRATION**. This is to force the migration files onto specific tapes separate than those for general users to write to. This keeps other requests from interweaving themselves between migration requests. After all the volumes are successfully scanned, the file family is set back to the file family of the original volume.

For deleted files when using **--with-deleted**, the original volumes file_family is ignored and the special file_family **DELETED_FILES** is used

instead. The -MIGRATION is still appended while the volume is being migrated.

5.3 Duplication

The process of duplication is very similar to that of migration. The differences include:

1. The volume comments indicating that a volume has been migrated from or migrated to will be "<-" or "->" instead of "<=" or "=>".
2. The swap step is not a complete swap of meta data like it is in migration. Instead it registers the new file to be a multiple copy of the original.
3. The duplication log goes to: /var/duplication/DuplicationLog@<timestamp>#<pid>
4. The file family is mangled by appending _copy_1 instead of -MIGRATION. See section 1.2.2 for more information.

5.3.1 Reasons for duplication

Duplication is usually preferred over migration when:

1. A second copy of the files on tapes is desired in a new tape library. If both tape libraries exist at the time encp is executed by the user, then using the multiple copy feature of encp is recommended. Typically, the second copy will be in a separate location from the original.
2. When a multiple copy write fails, but the original succeeded, a entry is left in the active_file_copying table of the Enstore (file & volume) database. Duplication is needed to make the second copy of these files.

5.3.2 Duplication command

Here is an example of the primary use of this command to duplicate three volumes worth of files to new media.[3]

```
# duplicate --spool-dir /data/data2/Migration_Spool VO0000
VO0001 VO0002
```

A return code of zero (0) indicates success duplicating all files (or volumes) in the argument list.

```
$ duplicate --help

Usage:
        duplicate [OPTIONS]... [bfid1 [bfid2 [bfid3 ...]]] | [vol1 [vol2 [vol3
...]]] | [file1 [file2 [file3 ...]]] | [vol1:lc1 [vol2:lc2 [vol3:lc3 ...]]]
        duplicate [OPTIONS]... [media_type [library [storage_group
```

---

3   The sample command has been split onto two lines with the standard shell continuation character (\).

```
[file_family [wrapper]]]]]
       duplicate [OPTIONS]... --restore [bfid1 [bfid2 [bfid3 ...]] | [vol1
[vol2 [vol3 ...]]] | [file1 [file2 [file3 ...]]] | [vol1:lc1 [vol2:lc2
[vol3:lc3 ...]]]
       duplicate [OPTIONS]... --scan [bfid1 [bfid2 [bfid3 ...]] | [vol1 [vol2
[vol3 ...]]] | [file1 [file2 [file3 ...]]] | [vol1:lc1 [vol2:lc2
[vol3:lc3 ...]]]
       duplicate [OPTIONS]... --migrated-from <vol1 [vol2 [vol3 ...]]>
       duplicate [OPTIONS]... --migrated-to <vol1 [vol2 [vol3 ...]]>
       duplicate [OPTIONS]... --status [bfid1 [bfid2 [bfid3 ...]]] | [vol1
[vol2 [vol3 ...]]] | [file1 [file2 [file3 ...]]] | [vol1:lc1 [vol2:lc2
[vol3:lc3 ...]]]
       duplicate [OPTIONS]... --show <media_type> [library [storage_group
[file_family [wrapper]]]]]


       --destination-only   Used with --status to only list output assuming
                            the volume is a destination volume.
       --file-family <FILE_FAMILY>  Specify an alternative file family to
                            override the pnfs file family tag.
  -h, --help                prints this message
       --infile <INFILE>    Read target list of bfids, volumes,
                            volume:location_cookie pairs or paths from
file.
                            Types can be intermixed.
       --library <LIBRARY>   Specify an alternative library to override the
                            pnfs library tag.
       --make-copies <MEDIA_TYPE> [LIBRARY] [STORAGE_GROUP] [FILE_FAMILY]
                  [WRAPPER]  Make copies of the supplied volume group.
       --make-failed-copies  Make duplicates where the multiple copy write
                            failed.
       --migrated-from       Report the volumes that were copied to this
                            volume.
       --migrated-to         Report the volumes that were copied from this
                            volume.
       --migration-only      Used with --status to only list output assuming
                            the target is not orhas not a multiple copy.
       --multiple-copy-only  Used with --status to only list output assuming
                            the target is orhas a multiple copy.
       --priority <PRIORITY>  Sets the initial job priority.  Only
                            knowledgeable users should set this.
       --read-to-end-of-tape  Read to end of tape before starting to write.
       --restore             Restores the original file or volume.
       --scan                Scan completed volumes or individual bfids.
       --show <MEDIA_TYPE> [LIBRARY] [STORAGE_GROUP] [FILE_FAMILY] [WRAPPER]
                            Report on the completion of volumes.
       --skip-bad            Skip bad files.
       --source-only         Used with --status to only list output assuming
                            the volume is a source volume.
       --spool-dir <SPOOL_DIR>  Specify the directory to use on disk.
       --status              Report on the completion of a volume.
                            S = State of duplication:
                                P = Primary/original copy; duplication
                                C = Muliple copy; duplication
                                O = Original/primary copy
                                M = Multiple copy
                            D = Deleted state:
                                N = Not deleted
                                Y = Yes deleted
                                U = Unknown; failed write
                            B = Bad file
                                B = Bad file
                                E = Empty metadata fields
       --usage               prints short help message
```

--use-disk-files	Skip reading files on source volume, use files already on disk.
--use-volume-assert	Use volume assert when scanning destination files.
--with-deleted	Include deleted files.
--with-final-scan	Do a final scan after all the files are recopied to tape.

--make-copies <MEDIA_TYPE> [LIBRARY] [STORAGE_GROUP] [FILE_FAMILY] [WRAPPER] This switch tell duplication to find files in the specified set of volumes, that do not have multiple copies and create multiple copies.

--make-failed-copies When **encp** succeeds in writing an original copy, but fails to write a multiple copy a table entry in the **active_file_copying** table in the file and volume database is left indicating how many multiple copies have failed to be written. When duplication is executed with **--make-failed-copies** the input file list to duplicate is chosen from the contents of the **active_file_copying** table. If this version of duplication is run on a non-admin and non-trusted PNFS filesystem, then the **/pnfs/fs/usr/Migration/** directory is not used and instead **.m.<original_name>** temporary filenames are used.

5.3.3 Duplication file_family mangling

The file family of a destination duplication volume is appended with **_copy_1**. This is to force the duplication files onto specific tapes separate than those for general users to write to. This keeps other requests from interweaving themselves between migration requests. After all the volumes are successfully scanned, the file family is set back to the file family of the original volume.

The purpose of using _copy_1, instead of the more obvious -DUPLICATION, is to mimic the multiple copies feature on encp. Encp can write multiple copies of a file to different volumes using the **--copies** switch.

See the Migration file_family section for more information on the file family when duplicating deleted files.

If --library and --file-family are used together, then some interesting things happen. First, it is possible to end up with files on tapes with file families like **_copy_2**, **_copy_3**, etc. if --library is used with a comma separated list of libraries on the original copy. If duplicating a duplicated file, one with copy1 in the file family already, then the file family would contain **_copy_1_copy_1**.

Putting these combinations together does allow for the possibility of trees of multiple copies.

5.4 Swapping metadata

There are two ways to turn migrated files into duplicated files. One way makes the 'original' file copy the multiple copy; the other makes the 'new' file copy the multiple copy. The result of the latter will be the same as if duplication had been used. In addition to this, there is also the ability to swap the files on a duplicated original tapes and the new destination tape. Using the first of the two ways to turn the files on migrated original tape into a duplicated tape followed by a swap will have the same effect as duplication had been used.

All three of these scripts require them to be run as user root and the /pnfs/fs/usr/... PNFS path be mounted.

5.4.1 make_original_as_duplicate.py

To make the files on the original tape recorded as duplicates in the Enstore database issue the following command:

```
# $ENSTORE_DIR/src/make_original_as_duplicate.py
<original_volume>
```

When this completes successfully the following will have happened:
1. All the successfully migrated files on the original volume will be recorded as duplicates in the file_copies_map table for the new copies on the destination volume(s).
2. The original volume will be marked duplicated instead of migrated or cloned.
3. In PNFS Layer 4 for the file(s) is modified to point to the new copy in the Enstore (file & volume) database.

There are no switches for this command. If the argument list is left blank the help message is printed to the terminal.

The following is a simple example:

```
# $ENSTORE_DIR/src/make_original_as_duplicate.py TEST05

making original TEST05 as copy of the migrated files ...
make_duplicate('WAMS122167084800000', 'WAMS121243218900000') ... OK
make_duplicate('WAMS122167087700000', 'WAMS121243220500000') ... OK
make_duplicate('WAMS122167090800000', 'WAMS121243222200000') ... OK
```

5.4.2 make_migrated_as_duplicate.py

To make the files on the new migrated-to tape recorded as duplicates in the Enstore database issue the following command:

```
# $ENSTORE_DIR/src/make_migrated_as_duplicate.py \
<migrated_to_volume>
```

When this completes successfully the following will have happened:
1. All of the files on the original volume will be recorded as primary copies in the file_copies_map table for the new copies on the destination volume(s).

There are no switches for this command. If the argument list is left blank the help message is printed to the terminal.

The following is a simple example:

```
# $ENSTORE_DIR/src/make_original_as_duplicate.py TEST05

making original TEST05 as copy of the migrated files ...
make_duplicate('WAMS122167084800000', 'WAMS121243218900000') ... OK
make_duplicate('WAMS122167087700000', 'WAMS121243220500000') ... OK
make_duplicate('WAMS122167090800000', 'WAMS121243222200000') ... OK
```

This version does not automatically update the system inhibit to duplicated for the original volume like make_original_as_duplicated.py does. For proper record keeping either use make_original_as_duplicated.py instead of this script or manually modify the system inhibit using the **enstore volume--modify** command.

5.4.3 swap_original_and_copy.py

For duplicated files, it is possible to swap which file is considered the original and which is the duplicate. This can be done for a list of files or for all the files on a list of volumes. The targets must refer to the current primary copy.

```
# $ENSTORE_DIR/src/swap_original_and_copy.py \
[[vol1 [vol2] ...] | [[bfid1 [bfid2] ...]
```

Only duplicated files that have successfully completed the checked phase, meaning they have been scanned, will be allowed to swap.

The following actions are taken on successful completion of this script:
1. All the files that are considered the primary copy are recorded as the the multiple copy in the file_copies_map table in the Enstore (file & volume) database.
2. All the files that are considered the multiple copy are recorded as the primary copy in the file_copies_map table in the Enstore (file &

volume) database.

3. PNFS layers 1 and 4 are modified to point to the new primary copy in the Enstore (file & volume) database.
   - The BFID in layer 1
   - The BFID in layer 4
   - The location_cookie in layer 4
   - The original drive in layer 4
   - The file_family in layer 4
4. Only when a **volume** swap is specified:
   - The file_family for the new primary file volume has the _copy_1 removed.
   - The new multiple copy file volume has _copy_# appended to the file family. The # is replaced with the multiple copy count from the file with the most other multiple copies located on other volumes. In almost all situations this will likely be 1, but greater numbers are possible.

There are no switches for this command. If the argument list is left blank the help message is printed to the terminal.

Here is a simple example:

```
# $ENSTORE_DIR/src/swap_original_and_copy.py TEST06

swapping WAMS122167084800000 ... OK
swapping WAMS122167087700000 ... OK
swapping WAMS122167090800000 ... OK
```

It is possible to swap files back and forth from being the primary copy to a multiple copy. For volumes worth of files, this is done only if **all** files for a volume can be swapped. Thus, it may be easy to swap a volume one direction, but if multiple tapes were duplicated to one destination tape, then the swap back in the reverse direction will fail until all the original volumes are swapped in the first place. If partial volumes swaps are truly needed, consider swapping on a per BFID basis.

There are three ways to able to use this script.
1. After duplication.
2. After migration followed by make_original_as_duplicate.py
3. After migration followed by make_migrated_as_duplicate.py

Note: The file family for the new volume is mangled differently between number 1 than it is for 2 or 3. It may be necessary to manually change it in order to keep everything consistent. No one wants to see an overzealous administrator recycle a volume that really did contain primary copies of files.

It is important to make the distinction between an original copy and primary

---

copy. An original copy is the copy of the file that resides on the first volume that this files was ever written onto. A primary copy is the copy of the file a user will get when they use encp to read a file from tape.

Chapter 6: Cronjobs

A number of cronjobs are considered critical to having a well running Enstore system. These critical cronjobs are delfile.py and enstore system. Most Enstore cronjobs need to run on the same machine as one of the Enstore servers. For example, the pnfs_monitor cronjob needs to run on the PNFS server node.

The name of the subsection refers to the crontab file name in the crontabs directory.

6.1 accounting_db

These cronjobs run on the same machine as the PostgreSQL accounting DB.

6.1.1 db_vacuum.py

Vacuums the accounting DB to shrink the size on disk.

Runs on Host	As User	Runs at
Postgres DB	enstore	4:30am every day

6.2 backup

These cronjobs run on the same machine as the PostgreSQL accounting and drive_stat databases, respectively. This grouping assumes they are running on the same machine.

6.2.1 db_backup.py accounting

Backups the accounting database to the backup node defined in the configuration.

Runs on Host	As User	Runs at
Where the backups go.	enstore	1:10am every day

6.2.2 db_backup.py drive_stat

Backups the drive_stat database to the backup node defined in the configuration.

Runs on Host	As User	Runs at
Where the backups go.	enstore	1:05am every day

6.3 backup2Tape

These cronjobs run on the designated "backup" node in the configuration.

6.3.1 backup2Tape

Backups the Enstore and PNFS databases to tape.

Runs on Host	As User	Runs at
Where the backups go.	enstore	7:30am every day

6.3.2 backupSystem2Tape

Obsolete.

6.4 checkdb

6.4.1 check_db.py

Loads the most recent Enstore (file & volume) database into an offline database server and verifies that all entries are correct. As a side effect, the inventory summary COMPLETE_FILE_LISTING web page is also generated.

Runs on Host	As User	Runs at
Where the backups go.	enstore	Every 2 hours on the 35th minute.

6.5 checkPNFS

These cronjob(s) run on the verify node as defined in the Enstore configuration.

6.5.1 checkPNFS

Verify that the PNFS server is still running.

Runs on Host	As User	Runs at
Where integrity checks are run.	root	Every 10 minutes on the threes.

6.6 chkcrons

These cronjob(s) run on the web server node.

6.6.1 chkcrons.py

Makes the plots of the exits statuses of the other cronjobs.

Runs on Host	As User	Runs at
Where the web server runs.	root	Every 10 minutes on the threes.

6.7 copy_ran_file

These cronjob(s) run on the verify node as defined in the Enstore configuration.

6.7.1 copy_ran_file

Select a random file(s) on a random tape and try to read them back. An exit status of nine (9) on the Cronjob Status plots page indicates that the PNFS mount point(s) are not mounted. An exit status of one (1), two (2) or three (3) means that number of encps failed to run.

copy_ran_file uses choose_ran file to perform the underlying actions. There are a number of switches that can be passed to copy_ran_file or choose_ran_file:

- -x: Enables shell verbose output. Useful for debugging.
- -V: Passes "--verbose 10" to any executed encp processes.
- -C: **choose_ran_file only:** Specifies that the files should not just be randomly chosen, but they should be copied too. copy_ran_file enables this switch when calling choose_ran_file.
- -F: Normally if the library (virtual library manager library; not physical robot library) is busy, then copy_ran_file will skip the transfer, since drives are not available. With this switch it will run encp anyway. A library is considered busy there is not at lease one idle drive. It is recommended to add admin priority for requests on this node. The exit status returned when -F is not used and the library is busy is zero.
- <count>: Number of tapes to check. Default is 1 if not specified. This value must be specified in order to specify the remaining arguments.
- <which_file>: Specifies any extra files to check on each tape beyond

the specified <count> value. Possible values are:

- o onlyran: This is the default. Only choses or copies the randomly chosen file.
- o onlyfirst: Ignore the chosen file and only use the first file on the chosen tape.
- o onlylast :Ignore the chosen file and only use the last file on the chosen tape.
- o alsofirst: In addition to the chosen file; the first file on the tape is also used.
- o alsolast: In addition to the chosen file; the last file on the tape is also used.
- o alsofirstlast: In addition to the chosen file; the first and last file on the tape is also used.

Runs on Host	As User	Runs at
Where integrity checks are run.	root	Every hour on the 27th minute.

6.8 delfile

This cronjob runs on the same node that the pnfs server runs on.

6.8.1 delfile.py

This cronjob updates the deleted status of a file in the Enstore file database. It works by looking through the files in the trash directory of pnfs for BFIDs, marking them deleted in the Enstore file database then deleting the trash file from the trash directory.

This cronjob should be run every five minutes. There will be a synchronization discrepancy between PNFS and Enstore until this cronjob runs after a file is deleted from PNFS.

Runs on Host	As User	Runs at
Where PNFS runs.	root	Every 5 minutes staring at the top of the hour.

6.9 drives_info

These cronjob(s) run on the web server node.

6.9.1 drives_info

Makes a web page reporting the serial numbers of the current set of tape

drives in use.

6.10 drivestat_db

These cronjobs run on the same machine as the PostgreSQL drivestat DB.

6.10.1 db_vacuum.py drivestat

Vacuums the drivestat DB to shrink the size on disk.

Runs on Host	As User	Runs at
Postgres DB	enstore	12:30am every day

6.11 enstore_db

These cronjobs run on the same machine as the PostgreSQL enstore DB.

6.11.1 db_vacuum.py enstoredb

Vacuums the enstore DB to shrink the size on disk.

Runs on Host	As User	Runs at
Postgres DB	enstore	2:30am every day

6.11.2 enstore backup

Backups the Enstore database to the backup node as defined in the configuration.

Runs on Host	As User	Runs at
Postgres DB	enstore	Every 10 minutes staring at the top of the hour.

6.12 enstore_html

These cronjob(s) run on the web server node.

6.12.1 enstore_system_html.py

6.12.2 make_quota_plot_page

6.12.3 make_cron_plot_page

6.12.4 make_ingest_rates_html_page.py

6.12.5 enstore system

6.12.6 enstore network

6.12.7 get_total_bytes_counter.py

6.13 enstore_plots

6.13.1 plotter.py --encp

6.13.2 plotter.py --mount

6.13.3 plotter_main.py [-m | --mount]

This cronjob script makes the plots showing the number of mounts for each media type per day.

Runs on Host	As User	Runs at
Where the web server runs.	enstore	Every day at 1:30am, 6:30am and 3:30pm.

6.13.4 plotter_main.py [-r | --rate]

This cronjob script plots the instantaneous network rates of all reads and writes.

Runs on Host	As User	Runs at
Where the web server runs.	enstore	Every hour on the 3rd and 33rd minute.

6.13.5 plotter_main.py [-u | --utilization]

This cronjob script makes the plots of the number of busy drives per media type.

Runs on Host	As User	Runs at
Where the web server runs.	enstore	Every hour on the 3rd and 33rd minute.

6.13.6 plotter_main.py [-s | --slots]

This cronjob script plots the number of free, used and disabled tape slots. For ADIC AML/2 robots this is by tower, for STK Powderhorn Silos it is by silo and for STK Streamline 8500s this is by rail.

Runs on Host	As User	Runs at
Where the web server runs.	enstore	Every hour on 10th minute.

6.13.7 make_sg_plot

6.13.8 plotter_main.py [-e | --encp-rate-multi]

This cronjob script plots each of the 5 encp rates recorded; once for read and once for writes per storage group.

Runs on Host	As User	Runs at
Where the web server runs.	enstore	Every hour on 25th minute.

6.13.9 plotter_main.py [-f | --file-family-analysis.py]

Report on the fill percentage of tapes per storage group.

Runs on Host	As User	Runs at
Where the web server runs.	enstore	Every day at 11:50pm.

6.13.10 plotter_main.py [-q | --quotas]

Plot the nuber of blank tapes available before running out or reaching the quota.

Runs on Host	As User	Runs at
Where the web server	enstore	Every 5 minutes staring

runs.		at the top of the hour.

6.13.11 plotter_main.py [-p | --pnfs-backup]

Plots the time required to backup the PNFS databases.

Runs on Host	As User	Runs at
Where the web server runs.	enstore	Every day at 1:15am and 1:15pm.

6.13.12 plotter.py --total_bytes --pts_nodes=d0ensrv2,stkensrv2,cdfensrv2 –no-plot-html

6.13.13 plotter_main.py [-i | --migration-summary]

Makes plots showing how many volumes for each media type have been migrated and/or duplicated.

Runs on Host	As User	Runs at
Where the web server runs.	enstore	Every hour at the bottom of the hour.

6.13.14 weekly_summary_report.py

6.13.15 plotter_main.py [-t | --tapes-burn-rate]

Plots the number of gigabytes used in the last 4 months by library and also library and storage_group pairs. Include information about how many blank tapes are remaining, and how many tapes were used in the last week and month.

Runs on Host	As User	Runs at
Where the web server runs.	enstore	Every hour at the top of the hour.

6.14 inventory

6.14.1 inventory.py

Makes the web pages listing all volumes and the files on each volume. Make

some of the inventory summary pages as well.

Runs on Host	As User	Runs at
Where the backups go.	enstore	Every hour on 50th minute.

6.15 inventory_web

These cronjob(s) run on the web server node.

6.15.1 cleaning_report

The CLEANING inventory page is generated.

Runs on Host	As User	Runs at
Where the web server runs.	enstore	Every day at 12:30am.

6.15.2 noaccess-tapes

The NOACCESS and VOLUMES inventory pages are generated.

Runs on Host	As User	Runs at
Where the web server runs.	enstore	Every 15 minutes staring at the top of the hour.

6.15.3 Vols

The files known to the robots are reported in *-VOLUMES.html. The asterisk is represents the short name of each media changer converted to uppercase.

Runs on Host	As User	Runs at
Where the web server runs.	enstore	Every day at 1:30am.

6.15.4 quota_alert

Makes the QUOTA_ALERT web page. Lists statistics reflecting the projected number of days each media_type, library and storage_group triplet will last with the current quota.

Runs on Host	As User	Runs at
Where the web server runs.	enstore	Every day at 8:30am and 4:30pm.

6.16 log_html

These cronjob(s) run on the web server node.

6.16.1 getnodeinfo

The nodeinfo.html file is created. This report lists all nodes in the enstore farmlet the kernel version, CPU speed, memory and other OS related value on one page.

Runs on Host	As User	Runs at
Where the log server runs.	root	Every hour at the 11th and 41st minute.

6.16.2 log_trans_fail

The failed transfer report is generated. This script greps through the recent log files and sorts the errors by volume and mover. This information can be useful when investigating any possible patterns involving a specific tape or drive.

Runs on Host	As User	Runs at
Where the log server runs.	enstore	Every hour on the 45th minute.

6.16.3 STKlog

Retrieves the last 1000 lines from the each acls_host found in every configured STK media changer. Names the output file *-log.html where * represents the short name of the configured media changer.

Runs on Host	As User	Runs at
Where the log server runs.	enstore	Every hour on the 2nd, 17th, 32nd and 47th minutes.

6.17 log_server

These cronjob(s) run on the log_server node.

6.17.1 log-stash

Zips and moves old logs files to the history directories.

Runs on Host	As User	Runs at

Where the log server runs.	enstore	On the 15th day of the month at 4:30am.

6.17.2 check_for_traceback

This script greps the log files for any Tracebacks. Any tracebacks are e-mailed to the addresses listed in the crons | developer_email section of the configuration.

Runs on Host	As User	Runs at
Where the log server runs.	enstore	Every hour on the 1st minute.

6.17.3 rdist-log

Copies the recent log files (that are not already copied) to the backup node for safe keeping. After log-stash moves the old log files, the matching old log files on the backup node are removed.

Runs on Host	As User	Runs at
Where the log server runs.	enstore	Every hour on the 5th, 20th , 35th and 50th minutes.

6.18 pnfs_misc

These cronjobs run on the same node that the pnfs server runs on.

6.18.1 PnfsExports

This cronjob makes the /enstore/pnfsExports.html web page. It is typically linked from the Log Files web page. It lists the PNFS mount points that are allowed to mount pnfs.

Runs on Host	As User	Runs at
Where the PNFS server runs.	root	Every hour on the 12th minute.

6.18.2 pnfs_monitor

A report is made of files listed in all PNFS databases (except admin) that do not have complete Enstore meta data. Exceptions are given to files that are dCache volatile files, zero length dCache files and temporary NFS files beginning with .nfs.

Runs on Host	As User	Runs at
Where the PNFS server runs.	root	Every day at 5:12am.

Chapter 7: ecron

This section describes the ecron script. For Enstore it is run by crond. Then ecron in turn starts the intended script or program to be run as a cronjob. This allows ecron to decide if e-mail should be sent; as opposed to crond sending an e-mail for every invocation of a cronjob. Histogram information is also reported by ecron.

7.1 Switches

- -x: debugging; must be first switch if present
- -q: quite output; must follow -x, go before others
- -p <name>: use specified name; default is the name of the script.
- -setup <products>: setup specified ups product before running cronjob
- -need-enstore: Verify that Enstore is running by asking the inquisitor; Default. -no-enstore skips this verification.
- -D <key>=<value>: Specify environmental variable to pass to the Enstore crontab script to be executed.
- -d <directory>: Set the directory that will be CWD for the cronjob script.
- -c <count>: The number of failures that need to occur before e-mail is sent. Default is 1.
- -l: Enable logging; this is default. -no-log disables logging; use for debugging only. Logging refers to ~/CRON/* files and accounting DB update.

7.2 Files

The files are located in the home area of the user executing them. Typically, the users should only be enstore or root.

- ~/CRON/<cronjob_name>:
- ~/CRON/<cronjob_name>HISTORGRAM: Contains all the starts and exit status of the cronjob. <cronjob_name> is the name of the script or the value specified with -p. These values are duplicated in the accounting DB.
- ~/CRON/<cronjob_name>ACTIVE: If present, this script is currently running. This is a lock file.
- ~/CRON/<cronjob_name>-output: The stdout and stderr contents from the last time the cronjob was run.
- ~/CRON/<cronjob_name>-output.<timestamp>: The last 5 error runs are kept.

7.3 Samples

Here are two sample command lines for various cronjobs. They are color coded to match the descriptions below.

- 27 * * * * root source /usr/local/etc/setups.sh; setup enstore; $ENSTORE_DIR/sbin/ecron -p copy_ran_file $ENSTORE_DIR/sbin/copy_ran_file 1 alsofirstlast > /dev/null 2> /dev/null
- 5,20,35,50 * * * * enstore source /usr/local/etc/setups.sh; setup enstore; $ENSTORE_DIR/sbin/ecron -p rdist-log rdist-log > /dev/null 2> /dev/null

The follow are the descriptions of the different parts of the crontab file line when using ecron.

- Time when crond should start the scripts.
- User to run the cronjob as.
- Command line invocated by crond.
- Ecron command line options.
- Script to be run by ecron.
- Arguements passed to the script run by ecron.
- Redirect the output from ecron to /dev/null. ecron decides to send e-mail or not. crond will always send e-mail if the output is not redirected.

7.4 Troubleshooting

Problems	Solutions
Cronjob is active for to long and is verified to still be running.	Determine what is the problem. Likely causes are bad disk, networking or a server is down.
Cronjob is active for to long and is verified to not be running.	Remove the ~/CRON/<cronjob_name>ACTIVE file.
Cronjob is failing consistently.	Check the output in the e-mail or ~/CRON/<cronjob_name>-output file.
ecron has determined Enstore to be down.	Verify if Enstore is down or not. If Enstore is up, check if the inquisitor is responding to commands; nothing to do if inquisitor is not responding.
Cronjob is not getting started.	Check if the line in the crontab file is commented out.

Chapter 8: PNFS Maintenance

This chapter documents various PNFS related operations that need to be done from time to time.

8.1 Adding a new PNFS Database

When a new experiment requests access to Enstore they will need to have their own PNFS database assigned. Names for their database and database mount point need to be chosen; it is recommended that the same name be used for both, but this is not a requirement.

Key information that will be needed before starting this task:
1. Will this database be a new top level PNFS database area or reside underneath an already existing PNFS database area.
2. The name of the PNFS database that will be created.
3. The name of the directory that users will see as the top of the new PNFS database. For simplicity, it is recommended that this be the same as #1 above.
4. The user id (UID) and group id (GID) of the group of users the new PNFS area will be assigned to.
5. The name of the storage group that this database will be associated with. This should match the name in #1 and #2 above, but is not a hard requirement.
6. The name of the file family that the files will be written with. For a new top level database this can be set to the same value as the storage group. If not, then there is generally a reason that the owner(s) wants to segregate there data; which likely means this should be set to something unique.

These instructions assume that mdb is located in /opt/pnfs/tools; substitute accordingly for your installation. Lets assume that the database name will be called <u>edata</u>. The following commands are all run on the node running the PNFS server.

0. # source /usr/etc/pnfsSetup

1. *For PNFS servers using GDBM only (**PostgreSQL backed PNFS servers do not do this step**); stop the PNFS daemons.*
   # $pnfs/tools/pnfs.server stop

2. *Create the new database.*
   # $pnfs/tools/mdb create <u>edata</u> $database/<u>edata</u>

3. *Start the new dbserver for the new database.*
   a. *For **PostgreSQL** backed PNFS databases to start the newly*

---

created PNFS dbserver:

```
# $pnfs/tools/mdb update
```

    b. *For **GDBM** backed PNFS databases to restart PNFS.*

```
# $pnfs/tools/pnfs.server start
```

4. *Find the database number for the <u>edata</u> database.*

```
# $pnfs/tools/mdb show
 ID   Name         Type    Status        Path
 ------------------------------------------------
 0    admin        r       enabled (r)   /srv/pnfs/db/admin
 1    mist         r       enabled (r)   /srv/pnfs/db/mist
 3    mist2        r       enabled (r)   /srv/pnfs/db/mist2
 4    flake        r       enabled (r)   /srv/pnfs/db/flake
 5    Migration    r       enabled (r)   /srv/pnfs/db/Migration
 6    edata        r       enabled (r)   /srv/pnfs/db/edata
```

5. *From the mdb show command, we can see that edata was assigned database number 6. We need to create a starting point to this database. The key command in the example(s) below is the mkdir. The 6 (or 7) in the examples would be the assigned database number mdb assigned. And the edata (or edata2) here refers to the name of the directory users will see. The directory created will not be named '. (#)(name)', but this is how this special information is passed to PNFS using the NFSv2 protocol.*

    o *If the database is intended to be a **top level** database in PNFS.*

```
# cd /pnfs/fs/usr/
# mkdir '.(6)(edata)'
# chmod 777 edata
```

    o If the *database is destined for use **underneath an already existing PNFS database** area then the cd would have been done to another directory followed by the mkdir and the chmod. The following example shows how this would be done to add an edata2 PNFS database area underneath an already existing edata PNFS database area.*

```
# cd /pnfs/fs/usr/edata
# mkdir '.(7)(edata2)
# chmod 777 edata2
```

    The rest of this example will go back to explaining how to create the top level edata PNFS database area.

6. *We need to create the PNFS tags in this directory. The value for the library should match that of a library in the current configuration. In most cases the storage_group should be set to match that of the newly created database name, edata, in this example.[4]*

```
# cd edata
# enstore pnfs --library LTO4
```

---

4 For PNFS areas to be used for volatile or resilient dCache pool metadata, different tags are used. Consult dCache documentation for instructions.

```
# enstore pnfs --file-family edata
# enstore pnfs --file-family-wrapper cpio_odc
# enstore pnfs --file-family-width 1
# enstore pnfs --storage_group edata
```

7. *Set the group to own these tags. Also, set the permission.*
   Tags file_family and file_family_width should be owned by the group
   with permissions 664. The other three should be owned by root with
   persmissions 644.

```
# enstore pnfs --tagchmod 644 file_family
# enstore pnfs --tagchmod 644 file_family_width
# enstore pnfs --tagchown <owner>.<group>
file_family
# enstore pnfs --tagchown <owner>.<group>
file_family_width
```

8. *They can be checked with:*
   # enstore pnfs --tags

```
.(tag)(library) = LTO4
.(tag)(storage_group) = edata
.(tag)(file_family) = edata
.(tag)(file_family_width) = 1
.(tag)(file_family_wrapper) = cpio_odc
-rw-rw-r--  1 root root 4 Oct 31  2001 /pnfs/fs/usr/edata/.(tag)
(library)
-rw-rw-r--  1 root root 3 Oct 31  2001 /pnfs/fs/usr/edata/.(tag)
(storage_group)
-rw-rw-r--  1 edata edata 3 Jan 23  2003 /pnfs/fs/usr/edata/.(tag)
(file_family)
-rw-rw-r--  1 edata edata 1 Mar  1  2006 /pnfs/fs/usr/edata/.(tag)
(file_family_width)
-rw-rw-r--  1 root root 4 Oct 31  2001 /pnfs/fs/usr/edata/.(tag)
(file_family_wrapper)
```

At this point, if the database is being added as a sub directory underneath an
already existing PNFS database area everything is done. If the PNFS database
is a new top level PNFS database for a new group using Enstore keep
following steps 9 through 13.

9. *The next step is to enable wormhole access. The conventional value*
   for $shmkey is 1122. The 6 is the database number from the mdb
   show command.

```
# pnfsCID=`cat "/pnfs/fs/admin/etc/.(id)(config)"`
# pnfs/tools/sclient getroot $shmkey 6 $pnfsCID
```

10. *Next enable the Enstore servers to mount PNFS. Substitute "ensrv1"*
    in the example with the names of the Enstore server nodes. Do the
    same for any user's nodes that need to mount PNFS. Be sure to verify
    that the user's nodes have static IP addresses. If they have dynamic
    address send email stating why this can not be completed.

```
# $pnfs/tools/pmount add ensrv1 /edata
```

11. *Now create the mount directory on each Enstore server. Substitute the correct rgang farmlet for ensrv in the following example.*
    ```
    # rgang ensrv 'mkdir /pnfs/edata'
    ```
12. *Next modify the /etc/fstab file for each Enstore server to contain the line:*
    ```
    www-ensrv1:/edata   /pnfs/edata   nfs
    sync,rsize=4096,wsize=4096,user,intr,bg,hard,rw,noac   0   0
    ```
    The example is really one long line, but is broken into two for readability. If the contents of /etc/fstab are governed by a configuration management tool (like cfengine) then follow the procedure to update the system configuration. Otherwise, an rgang command similar to that in the preceding step will suffice.
13. Mount the new PNFS area on at least the PNFS node and the node configured to run the copy_ran_file cronjob.
    # mount /pnfs/edata
14. Send an e-mail to the new group with the **/etc/fstab** line they will need to add to their node's **/etc/fstab** file.

8.2 Giving Systems Access to PNFS

8.2.1 Using pmount (1st method)

First look to determine if the hostname or the IP address is already assigned. If it is then corrective actions will need to take place.

```
# $PNFS_DIR/tools/pmount show hosts | egrep "<hostname>[|<IP address>]"
```

Second, the mount point needs to be obtained. In the following output, the mount point is from the first column: /admin, /mist, etc.

```
# $PNFS_DIR/tools/pmount show host <hostname>

 ----------<hostname> ------------------
 /admin       0 /0/root/fs/admin/etc
 /mist       30 /0/root/fs/usr/mist
 /mist-volmap 30 /0/root/fs/usr/mist-volmap
 /fs          0 /0/root/fs
 /flake      30 /0/root/fs/usr/flake
```

Next, the new hostname will be assigned. The <hostname> value is the hostname without any domain name included. The mount point

```
$ pmount add <hostname> <mountpoint>
```

8.2.2 Using UNIX tools (2nd method)

We assume that **/pnfs/fs** is mounted on the PNFS node for these steps. These commands should be run on the PNFS server node.

First look to determine if the hostname or the IP address is already assigned. If it is then corrective actions will need to take place.

```
# cd /pnfs/fs/admin/etc/exports

# ls -l | egrep "<hostname>[|<IP address>]"
```

Copy an existing file in this directory to a temporary file on local disk. It is recommended that it already have the mount point you are giving access to. Modify the now local copy of the file to just contain the mount points that the new node will be allowed to mount. When done just copy the temporary file to the filename of the IP address of the node gaining access to this PNFS mount point. In the example, the 131.225.333.444 exports file is used to create the exports file for 131.225.333.555.

You can not use **vi** on files in this PNFS directory . Its use of temporary files results in the file being corrupted.

```
# cp 131.225.333.444 /tmp/131.225.333.555

# vi /tmp/131.225.333.555

# cp /tmp/131.225.333.555 131.225.333.555
```

Now create the hostname symbolic link to the new file. The hostname must not contain the domain name.

```
# ln -s 131.225.333.555 <hostname>
```

8.2.3 Trusted PNFS Nodes

Trusted PNFS nodes are allowed to have user root manipulate all filesystem related metadata in **/pnfs/fs/usr/** style paths. All requests for trusted access should be confirmed before granting.

To Configure a machine to be trusted, an access file with the same name as the mounting nodes IP address needs to be created in the trusted directory.

```
# cd /pnfs/fs/admin/etc/exports/trusted

# echo 15 > 131.225.222.333
```

8.3 Removing Invalid Directory Entries

These files are those with a directory entry that points to an invalid/nonexistent inode. They can be found by doing (in bash):

1. An **ls** of the specific file and getting "No such file or directory".
2. An "**ls | grep** <basename>" and getting a file listing.

Reminder: You need to log onto the pnfs server, use the /pnfs/fs path
and be user root for these commands to work.

Notes: Taking the file /pnfs/fs/usr/mydir/myfile; <basename> refers to
myfile in the following examples and <directory> refers to
/pnfs/fs/usr/mydir.

On occasion the NFS client will continue to cache that a file was there
after it has been successfully removed from the database. Waiting a while
for the cache to clear out is the simplest option. For the impatient
you can force an unmount ("umount -f -l ..." on linux; the second switch is ell
not one) and then remount the file to clear up the problem.

8.3.1 To remove them (1st method):

If pnfs was installed using UPS
```
# setup pnfs
```
or if installed from RPM
```
# source /usr/etc/pnfsSetup

# cd <directory>
# $pnfs/tools/scandir.sh | grep <basname>
```

Paste the results from scandir.sh as arguments to Sclient.

```
# Sclient <line from scandir.sh output>
```

Answer yes to **Sclient** to execute the listed **sclient rmdirentrypos**
command.

8.3.2 To remove them (2nd method):

If pnfs was installed using UPS
```
# setup pnfs
```
or if installed from RPM
```
# source /usr/etc/pnfsSetup

# cd <directory>
# $pnfs/tools/scandir.sh | grep <basname>
```

First output from scandir is the directory id, followed by the file id, then other output then the position.

```
# $pnfs/tools/sclient rmdirentrypos $shmkey \
<dirID>   <rmID> <position>
```

8.3.3 To remove them (3rd method):

Sometimes the positional way(s) does not work. Try this next where <basename> is the name of the 'broken' file.

If pnfs was installed using UPS
```
# setup pnfs
```
or if installed from RPM
```
# source /usr/etc/pnfsSetup
```

```
# cd <directory>
# $pnfs/tools/sclient rmdirentry $shmkey <dirID> \
<basename>
```

8.4 Restoring Tag Inheritance

By default a directory's tags point to the same tag in its parent directory. If directories are moved around and removed; it is entirely possible that a directory's tags point to the tags of a nonexistent directory. To fix this requires finding out the PNFSID of the matching current parent directory's tag. First a short example of how this can happen.

```
$ cd /pnfs/test/
$ mkdir xyz          #make the first directory
$ cd xyz
$ mkdir abc          #make the second under the first
$ mv abc ..          #move the second up a level
$ cd ..
$ rmdir xyz          #remove the first directory
$ cd abc
$ enstore pnfs --tags
.(tag)(library) : [Errno 2] No such file or directory: '/pnfs/test/abc/.(tag)
(library)'
.(tag)(storage_group) : [Errno 2] No such file or directory: '/pnfs/test/abc/.
(tag)(storage_group)'
.(tag)(file_family) : [Errno 2] No such file or directory: '/pnfs/test/abc/.
(tag)(file_family)'
.(tag)(file_family_width) : [Errno 2] No such file or directory:
'/pnfs/test/abc/.(tag)(file_family_width)'
.(tag)(file_family_wrapper) : [Errno 2] No such file or directory:
```

```
                '/pnfs/test/abc/.(tag)(file_family_wrapper)'
                ls: /pnfs/test/abc/.(tag)(library): No such file or directory
                ls: /pnfs/test/abc/.(tag)(storage_group): No such file or directory
                ls: /pnfs/test/abc/.(tag)(file_family): No such file or directory
                ls: /pnfs/test/abc/.(tag)(file_family_width): No such file or directory
                ls: /pnfs/test/abc/.(tag)(file_family_wrapper): No such file or directory
```

To restore the storage group tag inheritance we need to first find the PNFS ID of the /pnfs/test storage_group tag. To do this we need to obtain the PNFS ID of the current parent directory[5] then find its pointer to the PNFS ID of the first tag.

$ enstore pnfs --id /pnfs/fs/usr/test
000100000000000000001060
$ enstore pnfs --showid 000100000000000000001060
```
        ----------------------------------------------------
        ID            : 000100000000000000001060
        Type          : --I--d-----
        next ID       : 000000000000000000000000
        base ID       : 000000000000000000000000
        parent ID     : 000000000000000000001080
        creation time : Wed Oct 31 14:54:05 2001
        modif. time   : Wed Mar 26 09:56:05 2008
        Type          : Directory ( Inode )
        mst_dev       : 1
        mst_ino       : 16781408
        mst_mode      : 40755
        mst_nlink     : 1
        mst_uid       : 9276
        mst_gid       : 1530
        mst_rdev      : 100
        mst_size      : 512
        mst_atime     : Wed Mar 26 09:56:05 2008
        mst_mtime     : Wed Mar 26 09:56:05 2008
        mst_ctime     : Wed Oct 31 14:54:05 2001
        mst_blksize   : 512
        mst_blocks    : 1
        Tag              : 000100000000000000001080      The first tag pnfsid.
        Group            : 0
        Entries          : 31
        Hash Function    : 0
        Hash Size        : 128
        Hash EntriesPerRow : 77
        Hash Rows        : 2
        000100000000000000001068 000100000000000000001070
```
Next we need to look at the contents of this first tag PNFS ID. The "next ID" field contains the PNFS ID of the next tag in the list of tags for the directory. "base ID" gives the PNFS ID of the directory that this tag belongs to. The "Name" field gives the name of the Tag. Some output clipped for compactness.

$ enstore pnfs --showid 000100000000000000001080
```
        ----------------------------------------------------
        ID            : 000100000000000000001080
        Type          : --I------t-
        next ID       : 000100000000000000001088
```

---

5   Any directory's tags could be used, but using the current parent directory makes the most sense.

```
base ID      : 000100000000000000001060
parent ID    : 000000000000000000000000
...
 Type        : Tag ( Inode )
 Name    : library
...
```

The first tag in the list is the library tag. We are looking for the storage group tag, so we need to take the "next ID" value and do the **enstore pnfs --showid** command again.

```
$ enstore pnfs --showid 000100000000000000001088
------------------------------------------------
ID           : 000100000000000000001088
Type         : --I------t-
next ID      : 000100000000000000001090
base ID      : 000100000000000000001060
parent ID    : 000000000000000000000000
...
 Type        : Tag ( Inode )
 Name    : storage_group
...
```

Next repeat this steps to find the PNFS ID of the broken tag looking through the showid output from the broken directory and the "next ID" fields in the tags list until the correct one (storage_group for this example) is found.

```
$ enstore pnfs --id /pnfs/test/abc
000100000000000000852C0
$ enstore pnfs --showid 0001000000000000000852C0
...
 Tag                  : 0001000000000000000852D8
...
$ enstore pnfs --showid 0001000000000000000852D8
...
 next ID      : 0001000000000000000852E0
...
 Name    : library
$ enstore pnfs --showid 0001000000000000000852E0
...
 Name    : storage_group
...
```

Now the storage group tag has been found. We need to take these PNFS IDs and pass it to **sclient** to fix the tags. Running sclient without any options prints the help message.

```
# /opt/pnfs/tools/sclient
...
 USAGE : /fnal/ups/prd/pnfs/v3_1_10-f2/tools/sclient chparent <shmkey>
<objectId> <parentId>
...
```

Now we can run this command. The <shmkey> should be 1122; look in the /usr/etc/pnfsSetup file for the shmkey line to be sure. Then we just need to pass the PNFS ID of the broken tag we want to fix followed by the tag we want it to inherit from.

```
# cd /pnfs/fs/usr/test/abc
# /opt/pnfs/tools/sclient chparent 1122 \
0001000000000000000852E0 000100000000000000001088
```

We can see the effect with the **enstore pnfs --tags** command.

```
# enstore pnfs --tags
.(tag)(library) : [Errno 2] No such file or directory:
'/pnfs/fs/usr/test/abc/.(tag)(library)'
.(tag)(storage_group) = zee
.(tag)(file_family) : [Errno 2] No such file or directory:
'/pnfs/fs/usr/test/abc/.(tag)(file_family)'
.(tag)(file_family_width) : [Errno 2] No such file or directory:
'/pnfs/fs/usr/test/abc/.(tag)(file_family_width)'
.(tag)(file_family_wrapper) : [Errno 2] No such file or directory:
'/pnfs/fs/usr/test/abc/.(tag)(file_family_wrapper)'
ls: /pnfs/fs/usr/test/abc/.(tag)(library): No such file or directory
-rw-rw-r--  11 enstore enstore 3 Oct 31  2001
/pnfs/fs/usr/test/abc/.(tag)(storage_group)
ls: /pnfs/fs/usr/test/abc/.(tag)(file_family): No such file or directory
ls: /pnfs/fs/usr/test/abc/.(tag)(file_family_width): No such file or directory
ls: /pnfs/fs/usr/test/abc/.(tag)(file_family_wrapper): No such file or
directory
```

Repeat this procedure with the remaining broken tags.

8.5 Fixing Broken Tags

A tag becomes invalid when there is no 'local' tag value and the parent tag it references does not exist. Sometimes it is necessary to forcibly rewrite the local portion of the tag to clear the 'invalid' status. The "0 0 10" are 'magic values' from the PNFS developers[6] that are the level, offset and size, respectively.

```
# source /usr/ec/pnfsSetup
# $pnfs/tools/sclient writedata $shmkey \
  <tag pnfs id> 0 0 10
```

---

6   Patrick Fuhrmann

Chapter 9: Configuration File

9.1 Configuration Descripton

The Enstore configuration file is a native python file. This allows for a lot of flexability in setting up the configuration.

9.2 Useful Shortcuts and Variables

9.3 Non-server Entries

9.3.1 blocksizes

This section defines the block sizes used when reading and writing different media types. The media types listed are the valid types available for the **enstore volume −add** command in section 1.24.

```
configdict['blocksizes'] = { '8MM'     : 131072,
                 'DECDLT'   : 131072,
                 '9840'    : 131072,
                 '9940'    : 131072,
                 '9940B'    : 131072,
                 '3480'    : 131072,
                 'LTO'     : 131072,
                 'LTO2'     : 131072,
                 'LTO3'     : 131072,
                 'LTO4'     : 131072,
                 'null'     : 131072,
                 'diskfile'  : 512,
                 }
```

9.3.2 crons

These values are used by the cronjob scripts to know the locations of various files or directories, hostnames and e-mail addresses. Every site needs to define there own values for html_dir, log_dir, backup_dir and monitoring and web.

```
configdict['crons'] = {
   #'www_alias'      : www_alias, #optional
   #'conf_alias'     : conf_alias, #optional
   #'pnfs_alias'     : pnfs_alias, #optional
   'web_node'        : monitoring_and_web,
   'html_dir'        : html_dir,
   'log_dir'         : log_dir,
   'tmp_dir'         : "/tmp",
   'email'           : "enstore-auto@fnal.gov",
   'developer_email' : "enstore-devel@fnal.gov"
   'backup_node'     : enstore_backup,
   'backup_dir'      : backup_dir,
   'monitored_nodes' : [
                 "ensrv0",
                 "ensrv1",
                 "ensrv2",
                 "ensrv3",
                 "ensrv4",
                 "ensrv5",
                 ],
   'farmlets_dir'    : "/usr/local/etc/farmlets",
   'url_dir'         : "http://www-en.fnal.gov/enstore/",
   #'test_library_list' : ["test-9940B"],
   }
```

- www_alias, conf_alias and pnfs_alias are optional DNS names to identify the web server host, configuration server host and PNFS server host. These are not required for Enstore to work, however they can be used to more easily move services from one node to another.
- web_node is the host name of the web server.
- html_dir is the directory that the inquisitor, alarm server, monitor server and many cronjobs write html, jpg, gif and other web content too.
- log_dir is the directory that the log server writes the log files into.
- tmp_dir can be used to use a different temporary directory for some cronjobs.
- email specifies the comma separated email addresses operational important errors should be sent to.
- developer_email will at most sites be the same as email.
- backup_node defines the host that the Enstore DB and PNFS DB backups are copied to.
- backup_dir is the directory on the backup_node that the Enstore DB and PNFS DB backups are copied to.
- monitored_nodes specifies the list of host names to limit the

generation of cron job status plots. In all likelihood this should contain the current names of all the Enstore server host names; not the mover host names.

- farmlets_dir specifies the location of the farmlets files.
- url_dir indicates the base URL value for the Enstore systems web area. It is used by some cron jobs that create pages in sub directories to point to images or other web pages in this top level URL directory or predetermined sub directories.
- enstore_name should contain a unique string that defines what this instance is used for. Likely considerations include naming production and test stands differently. For sites with multiple production Enstore systems, this allows them to be given human differentiable names.
- test_library_list is used by **choose_ran_file** and **copy_ran_file** to exclude the supplied list of libraries.

9.3.3 crontabs

This section defines which nodes should run which cronjobs. If the primary key matches the name of an Enstore server then those crontab files are copied to /etc/cron.d by install_crons.py. Primary keys examples are **log_server** and **accounting_server**. The files that install_crons.py copies to /etc/cron.d are listed in each **cronfiles** subsection. For the log_server, there are **log_server** and **log_html**. If the primary key is not an Enstore server, **verifying** and **pnfs**, are two such examples, then a host subfield is required.

```
configdict['crontabs'] = {
   'log_server'      : {'cronfiles' : ["log_server",
                          "log_html",
                          ]},
   'web_server'      : {'cronfiles' : ["enstore_plots",
                          "enstore_html",
                          "inventory",
                          "chkcrons",
                          #"tab_flipping",
                          ],
              'host'    : monitoring_and_web},
   'verifying'       : {'cronfiles' : ["checkPNFS",
                          "copy_ran_file",
                          ],
              'host'    : verify_host},
   'pnfs'            : {'cronfiles' : ["delfile",
                          "pnfs_misc",
                          ],
              'host'    : enstore_pnfs},
   'accounting_server' : {'cronfiles' : ["accounting_db"]},
   'drivestat_server'  : {'cronfiles' : ["drivestat_db"]},
   'file_clerk'      : {'cronfiles' : ["enstore_db"]},
   'databases'       : {'cronfiles' : ["backup",
                          #"backup.operation",
                          #"backup2Tape",
                          ],
              'host'    : enstore_backup},
}
```

9.3.4 database

The volume clerk, file clerk and info_server share a single PostGreSQL database. The shared information for these servers to locate the database is placed here.

```
configdict['database']    = {
   'dbname': 'enstoredb',
   'dbhost': DB_host,
   'dbport': 8888,
   'dbuser': 'enstore',
   'dbserverowner': 'products',
   'dbarea':'%s/enstore-db' % (db_basedir,),
   }
```

9.3.5 discipline

Discipline is used to limit the number of simultaneous transfers for a node. Nodes containing the string saen will be limited to 2 simultaneous transfers for the null library (2nd argument in the 'args' list. For mover with already mounted tape it is allowed 1 more simultaneous transfer for read request (3rd argument) and 1 more simultaneous transfer for write request (4th argument).

```
configdict['discipline'] = {
   'null.library_manager':{1:{'host':"saen",
                    'function':'restrict_host_access',
                    'args':['saen',2, 1, 1],
                    'action':'ignore'},
                 },
              },
   }
```

9.3.6 encp

The CRC seed for new sites should set this to 1. For older sites, consult knowledgeable experts. If the this value is missing from the configuration file then the crc_seed will default to 1.

```
configdict['encp'] = {'crc_seed' : crc_seed,
                 }
```

9.3.7 inventory

```
configdict['inventory'] = {
   'inventory_dir'          : inventory_dir,
   'inventory_tmp_dir'       : "%s/tmp" % (inventory_dir,),
   'inventory_cache_dir'      : "%s_cache" % (inventory_dir,),
   'inventory_extract_dir'    : "%s/extract" % (inventory_dir,),
   'inventory_rcp_dir'       : '%s:%s/tape_inventory' % (monitoring_and_web,
html_dir,),
   'wpa_states'           : ["full", "readonly"],
   'wpa_excluded_libraries'   : ["samnull", "testlto", "testlto2", "TEST-9940B
", "D0-LTO4G1T"],
   }
```

- inventory_dir:
- inventory_tmp_dir:
- inventory_cache_dir: Directory where a local on-disk copy of the inventory is kept to skip processing volume metadata that has not been modified since the previous time the inventory cronjob was run.
- inventory_rcp_dir: Remote host and directory to copy the output files to once they are done being generated.
- wpa_states:
- wpa_excluded_libraries:

9.3.8 priority

The most common use of the priority section is to bestow admin_priority to
encp transfers. In the following example the **null** library manager will assign
adminpri priority to encp transfers running from the **verify_host** as user
enstore.

```
configdict['priority'] = {
    'null.library_manager':{'adminpri':{2:
                        {'host': verify_host,
                         'uname':'enstore',
                         #'storage_group':'null_group1',
                         #'work' : 'write_to_hsm',
                         #'work' : 'read_from_hsm',
                         },
                         },
                },
    }
```

- host: if defined limits the admin priority to encp transfers from this
  host.
- uname: The user name of the owner of the encp transfer.
- storage_group: Limit these privileges to certain experiments or
  groups.
- work: may be **write_to_hsm** or **read_from_hsm**. These limit the
  priority setting to just writes or reads, respectively.

9.3.9 wrappersizes

There are only three defined file wrappers for Enstore. These wrappers are
used to "wrap" metadata onto the media with the data file. These values are
what needs to be inserted into the PNFS file_family_wrapper tag, see
section ????????????.

```
configdict['wrappersizes'] = { 'null' : (100L*GB) - 1,
                      'cern' : long("9"*20),
                      'cpio_odc' : (8L*GB) - 1,
                       }
```

9.3.10 web_server

These are the values used by the post installation steps of the
enstore_html.rpm.

```
configdict['web_server'] = {
    'ServerHost'    : web_server_host,
    'User'         : 'enstore',
    'Group'         : 'enstore',
    'port'         : 80,
```

```
    'ServerName'    : '%s.fnal.gov' % (www_alias,),
    'DocumentRoot'  : "%s/html" % (httpd_dir2,),
    'ScoreBoardFile' : "%s/apache_status" % (httpd_dir,),
    'PidFile'       : "%s/adm/httpd.pid" % (httpd_dir,),
    'ErrorLog'      : "%s/adm/error.log" % (httpd_dir,),
    'CustomLog'     : {
       'combined' : "%s/adm/access.log" % (httpd_dir,),
       'referer'  : "%s/adm/referer.log" % (httpd_dir,),
       'agent'    : "%s/adm/agent.log" % (httpd_dir,),
       },
    'ServerRoot'  : '/etc/httpd/' ,
    'ScriptAlias' : {
       'fake' : '/cgi-bin/',
       'real' : "%s/cgi-bin/" % (httpd_dir2,),
    }
}
```

9.4 Server Entries

Common Entries: The host, port andlogname fields are mandatory. The rest
are optional.

1. host – The hostname of the machine the Enstore server will run on.
2. hostip – Performs the same function of host. The additional
   functionality is to specify a single IP address for Enstore use on
   multi-homed machines.
3. port – The port number the Enstore server should listen on.
4. logname – The name of the server that appears in the logfile. Should
   be between 2 and 8 characters; but this is not a hard limit.
5. inq_ignore – If this is defined, then the inquisitor will not monitor the
   server. This is recommended for test library manager, test media
   changes and test movers.
6. norestart – This configuration item lets the inquisitor to know not to
   restart the server if it is found to be down.
7. noupdown – The "enstore system" process will not look at the
   corresponding server when generating the Status At-A-Glance web
   page.

9.4.1 alarm_server

The only unique configuration information for the alarm server is
alarm_actions. The only currently defined actions are for sending some
e-mailable alarms straight to users instead of putting it on the alarms page. If

the storage group is not found here then the default is to place the alarm on the alarms web page.

```
configdict['alarm_server'] = {
    'host':alarm_server_host,
    'port':7503,
    'logname':'ALMSRV',
    'norestart':'INQ',
    'alarm_actions' : {'C' : [['send_mail', '*',
                        {
                        'sdss' : 'sdssdp@fnal.gov',
                        }
                        ],
                        ['send_mail', '1',
                        {'cms' : 'cms-t1@fnal.gov',
                        }
                        ],
                        ]}
}
```

- alarm_actions: The alarm type value, C in the preceding example, must be one of the following list:
  - A for a regular alarm
  - E for an error alarm.
  - U for a User error/alarm.
  - W for a Warning alarm.
  - I for Information only alarm.
  - C for an E-mailable alarm.

There is little functional difference among these types of alarms. What is different is the single letter log message designation when the alarm is logged with the log server.

When an alarm type of C is received by the alarm server it looks though list of possible actions. Currently only send_mail is defined. It has limit value. Either ***** or **1** for send an e-mail for every occurrence of the same alarm or only send it once via e-mail then put it on the alarms page for every occurrence after that, respectively. The third element in these lists is a dictionary paring the storage group to an e-mail address or a list of comma separated e-mail addresses.

9.4.2 event_relay

The event_relay does not contain any non-common server configuration items.

9.4.3 log_server

```
configdict['log_server'] = {
   'host':log_server_host,
   'port':7504,
   'norestart':'INQ',
   'msg_type_logs': {'MSG_TYPE=MC_LOAD_REQ'  : 'MOUNTS-',
                     'MSG_TYPE=MC_LOAD_DONE' : 'MOUNTS-',
                     'MSG_TYPE=ENCP_XFER'    : 'ENCPS-',
                     'MSG_TYPE=EVENT_RELAY'  : 'EVRLY-',
                     'MSG_TYPE=ADD_TO_LMQ'   : 'LMQADDS-'},
   'log_file_path':log_dir,
   }
```

- log_file_path refers to the directory that the log files are to be written into. In the example, the log_dir variable is used to hold the actual directory name.
- msg_type_logs

9.4.4 file_clerk

```
configdict['file_clerk'] = {
   'host':file_clerk_host,
   'port':7501,
   'logname':'FILSRV',
   'norestart':'INQ',
   'brand':'D0MS',
}
```

- brand: Brand is an optional field. If set it names characters that appear at the be beginning of every bit file id (BFID). If this is not set, then a suitable brand string is determined based on the hostname. There is no limit on the length, though 4 alpha numeric characters is recommended. The last brand character must be alphabetical only. This field is used to differentiate different Enstore systems running at a single site; in which case it is **strongly** recommended to be used.

9.4.5 volume_clerk

```
configdict['volume_clerk'] = {
   'host':volume_clerk_host,
   'port':7502,
   'logname':'VOLSRV',
   'norestart':'INQ',
   'max_noaccess_cnt' : vol_max_noaccess_cnt,
}
```

- max_noaccess_cnt:

9.4.6 info_server

```
configdict['info_server'] = {
   'host': info_server_host,
   'port': 7777,
   'logname': 'INFSRV',
   'norestart':'INQ',
   }
```

9.4.7 pnfs_agent

Reserved for future use.

9.4.8 monitor_server

```
configdict['monitor_server'] = {
   'html_dir'         : html_dir,
   'html_gen_host'    : web_server_host,
   'refresh'          : 3600,
   'veto_nodes'       : {'watertaxi': 'not in system my reason'},
   'block_size'       : 65536,
   'block_count'      : 160,
   'default_timeout'  : 3,
   }
```

- html_dir: Must point to the same location as the html_dir field in the **crons** section of the configuration file.
- html_gen_host: This refers to the host running the monitor server that also is running the web server. It is important for the monitor server specified here to be running all the time.
- refresh:
- veto_nodes: This is a list of hosts to skip during the **enstore network** check. A reason for each host is also allowed.
- block_size and block_count: These two values multiplied together represent the amount of data sent during each **enstore network** rate test performed. The values listed in the example are known to

work well for Fast Ethernet speeds.

- default_timeout: Amount of time to wait for a response from a monitor server running on each node. After this timeout **enstore network** skips to the next node in the list.

9.4.9 ratekeeper

```
configdict['ratekeeper'] = {
    'host'          : ratekeeper_host,
    'port'          : 55511,
    'norestart'     : 'INQ',
    'noupdown'      : 'efb',
    'logname'       : 'RATSRV',
    'dir'           : ratekeeper_dir,
    'tmp'           : "%s/tmp/" % (ratekeeper_dir,),
    'ps'            : '%s/*rates.ps' % (html_dir,),
    'jpg'           : '%s/*rates.jpg' % (html_dir,),
    'stamp'         : '%s/*rates_stamp.jpg' % (html_dir,),
    }
```

- dir: Deprecated field that names the directory where the ratekeeper rate files are written into. This information be default is automatically inserted into the accounting database. If this field is set then the rate information is written to the text files and the database.
- tmp, ps, jpg and stamp: These are deprecated fields used by makeplot. The enstore plot command has superseded makeplot.

9.4.10 library_manager

```
configdict['9940B.library_manager'] ={
   'host':library_manager_host,
   'port':7522,
   'encp_port':7523
   'mover_port':7524
   'logname':'9940BLM',
   'norestart':'INQ',
#   'blank_error_increment':'5',
#   'max_encp_retries':3,
#   'max_suspect_movers':'3',
   'max_file_size':(200L*GB) - 1,
   'min_file_size':100*MB,
   'CleanTapeVolumeFamily': 'CLEAN.CleanFileFamily.noWrapper',
   'suspect_volume_expiration_time':3600*24,
   'legal_encp_version':legal_encp_version,
   'storage_group_limits':{'cms':10,
                  'cdf':10,
                  'D0':10,
                  },
   'max_requests':10000,
   'lock':"nowrite",
}
```

All the library manager specific configuration options are optional.

- encp_port: An alternate port that only **encp** v3_8 and later uses for submitting file requests. This is an optional value and the default for encps is to use regular port number if this is not specified.
- mover_port: An alternate port that only movers use for communicating with the library manager. This is an optional value and the default for movers is to use regular port number if this is not specified.
- blank_error_increment: In case of FTT_EBLANK errors, do not set the volume NOACCESS until the error count is greater than (max_suspect_movers + blank_error_increment). This is an optional value and the default is 5.
- max_encp_retries: This allows an administrator to up the number of attempts encp can try to this library before it gives up because of two many errors. This is an optional value and the default is 3.
- max_suspect_movers – This value is the number suspect movers that a single volume read or write can fail on before the volume is set NOACCESS. This is an optional field and the default is 3.
- max_file_size: This represents the maximum filesize that can be

written to this library. This should be set to the maximum size of the associated media. If not present the default is 2GB - 2kb.

- min_file_size: Minimum size that is allowed to write to tape. This would allow an administrator to prevent a user from writing small files into permanent media storage. This is an optional field and the default is 0.

- CleanTapeVolumeFamily: This lists the volume family for the cleaning tapes belonging to this library manager.

- suspect_volume_expiration_time: Duration a volume will remain blacklisted from a particular mover. See also—get-suspect-vols. The default time is 24 hours.

- legal_encp_version: This is the oldest version of encp allowed to access the system. An example value would be **v3_7**. As of March 21st 2008 this value is **v3_6c**. This is an optional field.

- lock: This is the default state the library manager will go into when started or the **enstore library –stop-draining** command is issued. The possible values are: pause, lock, noread and nowrite. See the section 1.8 for more information.

- storage_group_limits – Sets the fair share limits for the specified storage groups to access tape drives. This is optional and by default there are no fair share limits.

- max_requests – The maximum number of requests the library manager will place into the queue. An admin priority request will always be added to the queue, even if it means exceeding this limit. When the limit is reached and a new (non-admin) request arrives the library manager will reply to the the encp with a successfully queued request message, but will not add it to the queue. Encps resend pending requests every 15 minutes[7], if the queue is not full then, the request will be added to the queue. This is optional and the default .is 2000.

---

7 Encps resend pending requests every 15 minutes by default. It is possible to modify that time using the encp –resubmit-timeout switch.

9.4.11 mover

```
configdict['null.mover'] = {
   'host':'enmvr1a',
   'port':7530,
   'logname':'NULMV',
   'norestart':'INQ',
   'max_buffer':50*MB,
   'library':'null.library_manager',
#   'library':['test.library_manager', 'null.library_manager'],
   'device':'/dev/null',
   'driver':'NullDriver',
   'mc_device':'-1',
   'media_changer':'null.media_changer',
}
```

```
configdict['D31DLTO.mover'] = {
   'host': 'enmvr17a',
   'port': 7545,
   'logname': 'D31DMV',
   'noupdown':'dmb',
   'inq_ignore':'dmb',
#   'data_ip':'enmvr17a',
#   'do_eject':'yes',
   'statistics_path': '/tmp/enstore/enstore/D31DLTO.stat',
   'max_consecutive_failures': mvr_max_consecutive_failures,
   'max_failures': mvr_max_failures,
#   'failure_interval' : 3600,
   'compression':0,
   'check_written_file': lto_mvr_check_f,
   'max_buffer' : 1*GB,
#   'min_buffer' : 8 * MB,
   'max_rate':lto_rate,
   'mount_delay':30,
   'max_dismount_delay':max_dismount_delay,
   'dismount_delay':dismount_delay,
   'update_interval':5,
   'library': 'testlto.library_manager',
   'device' : '/dev/rmt/tps0d4n',
   'driver' : 'FTTDriver',
   'mc_device':'D31D',
   'media_changer':'aml2.media_changer',
   'syslog_entry':low_level_diag_pattern,
   'do_cleaning':'No',
   'norestart':'INQ',
#   'send_stats':1,
#   'connect_timeout' : 5,
#   'connect_retries' : 4,
   }
```

```
configdict['disk.mover'] = {
    'device':'/data',
    'type':'DiskMover',
    'host':'rain',
    'hostip':'131.225.84.108',
    'ip_map':'rain', #disk mover only
    'port':7531,
    'library':['disk.library_manager',] ,
    'driver':'DiskDriver',
    'mc_device':'1',
    'logname':'DISKMV',
    'update_interval':5,
    'connect_timeout':10,
    'connect_retries':15,
    }
```

- connect_timeout: The number of seconds to wait for a connection to the encp process to be established. Default is 15 seconds. This timeout is waited for **connect_retries** number of times.
- connect_retries: The number of times the mover tries to connect back to the requesting encp process. Default is 4 which yields 4 total attempts. The length of each connection attempt is defined by **connect_timeout**.
- dismount_delay: Number of seconds a mounted volume should stay mounted in a drive without a new request for the mounted volume. If the counter expires the volume is dismounted. This is useful in use cases where a new request might take longer than the default 60 seconds to arrive at the library manager; where it higher than 60 is hoped to prevent frequent mounts and dismounts of the same volume over and over again. Setting this value to high for general use may impact other requests waiting for a mover in an otherwise idle Enstore system. Setting this to a negative value will prevent the volume from being automatically dismounted. See the **encp --dismount-delay** switch for specifying this value for a single encp's amount of files.
- failure_interval: If the **max_failures** number of error occurs within this number of seconds the mover will go offline. Default is 3600 seconds (1 hour).
- get_remaining_from_stats:
- library: The name of the library manager to contact for work. This value can be a single string or a list containing the libraries to contact. The order the library managers are contacted is the order listed in the configuration.
- log_state:
- max_buffer: Maximum amount of memory the mover should use when buffing data on and off of tape. This value is optional and defaults to 8 MB in bytes.
- max_dismount_delay: Maximum number of seconds that can be set in

for the dismount delay. In a proper configuration this will always be grater than or equal to the dismount delay. This value is more useful as a way to limit the values specified using the **encp --dismount-delay** switch.

- max_consecutive_failures: This number of consecutive errors will set the mover offline. Default is 2.
- max_failures: This number of errors in the **failure_interval** will set the mover offline. Default is 3.
- max_in_state_cnt:
- max_rate: Highly recommended optional value to specify the maximum write rate of the drive. Defaults to 11.2 MB per second.
- max_time_in_state:
- media_changer: The name of the media changer to contact for mounting and dismounting volumes. This value is not specified for disk movers.
- min_buffer: Minimum amount of memory the mover allocates for buffering the data transfer. This value is optional and defaults to 64 MB in bytes.
- mount_delay: Number of seconds to wait after media changer mount completes before opening the tape device (i.e. /dev/rmt/tps0d0n or /dev/null). This is an optional value with the default for tape movers 15 seconds and for null movers 0 seconds.
- restart_on_error: The mover restarts itself automatically if it does into the error state. Default does not restart. Set to 1 to enable this feature. Most modern drives, like 9940A/B and LTO, are reliable enough that when the movers go into error state something is likely wrong.
- send_stats: Send the drive statistics to the drivestat server to be inserted into the drive statistics database table; which are used to populate the tape drive statistics web page. Optional boolean value; default is 1 for enabled.
- statistics_path:
- update_interval: Interval in seconds between requests of the mover to the library_manager requesting the next item in the queue. Optional value, default is 15 seconds. Smaller values are better when the use case is to submit one request at a time, instead of submitting multiple requests from the beginning.

The following are mover type specific values.

- device:
  - Tape movers: This represents the path to the tape drives device file. An example on Linux would be: /dev/rmt/tps0d1n.[8]
  - Disk movers: This represents the path to use for writing

---

[8] The /dev/rmt/tps*d*n pattern is not the native Linux tape device. See the FTT mkscsidev.Linux utility for more information into creating Enstore compatible tape devices.

incomming files. If multiple disk movers are running on a single host, they should have unique paths.
- o Null movers: /dev/null
- driver:
  - o Tape movers: FTTDriver
  - o Disk movers: DiskMover
  - o Null movers: NullDriver
- mc_device:
  - o Tape movers: This field tells the mover the name the robot calls the attached tape drive. For STK robots an example value is the comma separated value: 1,1,10,8. For ADIC robots an example value is: D41D.
  - o Disk movers: 1
  - o Null movers: -1

The following are tape mover specific values.
- blank_error_increment:
- check_first_written_file: This enables a re-read of the first written file after a tape is mounted to recalculate and verify the CRC of the file on tape. Default is 0, which disables this check.
- check_written_file: This enables a periodic check after a file is written to tape. It looks for silent corruptions in the data while the file was being written. If this value is greater than 0, it randomly re-reads newly written files to recalculate the CRC as often as the value specifies. A value of 1 will run this check after every file. Default is 0, which disables this check.
- Compression: This is considered an optional configuration value. However, in almost every situation it should be set to 0. Anything else (including None) will enable drive compression of the data. Enabling drive compression is not recommended because of the reduced drive rates and compressing already compressed data (from compress or gzip) increases the size of the data on tape.
- do_cleaning: Enable the mover to clean the drive when the cleaning bit is set on the drive.
- do_eject: Eject the tape on dismount. This is enabled by default. Possible values are **yes** and **no**.
- media_type:
- single_filemark: Default value is 0. Recommended value is 1 for most modern drives. Some older drives (Mammoth 1 for example) designated End-of-Data with two consecutive file marks. Thus, it is important after every write for two file marks to be written. This has the downside that when writing many files sequentially, the tape first must be rewound to between the two file marks before the next file can be written. Stopping, rewinding and writing after each file takes a

longer time and puts more ware and tear on the tapes and the drives than does writing one file mark then starting to write the next file.

- syslog_entry: Expression to match in the Linux syslog for the low level drive failure diagnostics. This value is optional. A typical value is **sense|st[0-9]** when it is set.

The following are disk mover specific values.

- type: Technically, this is not specific to disk movers, but is only required to be set by them, since the default is for Tape/Null movers. The default is the empty string (""), but the other possible values are: **Mover** and **DiskMover** for Tape/Null and Disk movers respectively.

9.4.12 media_changer

```
configdict['SL8500.media_changer'] = {
   'host':media_changer_host,
   'port':7508,
   'logname':'SL8500MC',
   'type':'STK_MediaLoader',
   'norestart':'INQ',
   'acls_host':'fntt-gcc',
   'acls_uname':'acsss',
   'DriveCleanTime':{'LTO3':[60,1],
               'LTO4':[60,1],
               },
   'tape_library':"GCC StreamLine 8500",
   }

configdict['aml2.media_changer'] = {
   'host':media_changer_host,
   'port':7525,
   'logname':'AML2MC',
   'type':'AML2_MediaLoader',
   'norestart':'INQ',
   'RobotArm':'Both',
   'IdleTimeHome':30000000,
   'DriveCleanTime':{'DE':[60,1],
               'DC':[60,1],
               'DM':[60,1],
               'D3':[120,2]
               },
   'IOBoxMedia':{'ACI_8MM':['E01','E08'],
           'ACI_LTO':['E03','E05','E06'],
           'ACI_DECDLT':['E02','E04','E07']},
   'tape_library':"D0 AML/2",
   }
```

- type: This field declares the type of robot that this media changer will be interfacing with. Valid values for this field are:
  - STK_MediaLoader for StorageTek (now Sun Microsystems) 9310 Powderhorn Silos and StreamLine 8500s.
  - AML2_MediaLoader for ADIC (now Quantum) AML/2 Quadratower or the AML/J.
  - RDD_MediaLoader for manually loading a tape.
  - MTX_MediaLoader for Overland 8000 stackers.
  - IBM_MediaLoader for IBM robots using SMC.

It is important to note that there is not a media changer for disk volumes.

- DriveCleanTime:
- tape_library: This is the string identifying the associated robot. This is an optional value, but is very useful to differentiate robots for a site that has multiple robots of the same type (i.e. "STK Silo room1" and "STK Silo room2"). One ramification of not specifying this value will be that the Slot Usage and Drive Utilization plots will not be created.

For configuration items specific to the STK_MediaLoader:

- acls_host: The name of the host running the STK robot(s).
- acls_uname: The name of the user used to rsh as into the acls_host.

For configuration items specific to the AML2_MediaLoader:

- RobotArm: Specifies which side of the AML/2 the media changer controls. R1, R2, or Both are the valid values. For AML/J use R1.
- IdleTimeHome:
- IOBoxMedia:

For configuration items specific to Manual_MediaLoader:

- test: If this is any value that evaluates to boolean true; then the dialog box asking for confirmation that the tape has been inserted into the drive includes additional button choices. The additional button choices name specific errors to simulate for testing purposes.

Chapter 10: Restoring Enstore and PNFS databases

10.1 Rebuilding a PNFS database from an Enstore Database

However unlikely and unfortunately you may need to rebuild a set of PNFS database from the Enstore file and volume database. Some issues will arise from this. First, is that the database number in the PNFS IDs will almost certainly not match the new database number that will be assigned to the same database name. This will not pose a problem for the active files being restored, but maybe confusing for someone investigating what happened to a deleted file.

First, extract the tag information for each directory. This will dump the results into a text file, named /tmp/tag_dir_dump in the example, so that the directory structure complete with tags can be recreated. Most likely the default port (8888) and user/role (enstore) will be used; if on the same host as the database the "-h <hostname>" may be left off. The example also assumes that the enstore database name is enstoredb.

```
$ psql –h <hostname> –p 8888 enstoredb –U enstore –c "
select library, storage_group, file_family, wrapper,

rtrim(pnfs_path,'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTU
VWXYZ._-+=') as dirname
from file, volume
where file.volume = volume.id
  and file.deleted = 'n';
" > /tmp/tag_dir_dump
```

Next, we find just the unique combinations of directories and tags. The first sed removes any empty lines. The second sed removes the first two lines and the last line. Then the list is sorted and the duplicate values are consolidated together.

```
$ sed '/^$/d' /tmp/tag_dir_dump | sed –e '1,2d' –e '$d' \
| sort | uniq –c > /tmp/tag_dir_dump_uniq
```

The new databases need to be (re)created. See the *PNFS Maintenance* chapter, section "Adding a new PNFS Database" for more information.

Use the information from the previous step to decide what are the appropriate tag values for the top directory for each PNFS database mount point. The directions for setting these top level tags are included in that text and not repeated here.

Now, the directory structure can be recreated. We simply loop over the unique list of directories using mkdir -p. This is best done on a machine that has both the /pnfs/xyz and /pnfs/fs/usr/xyz style mount points mounted. Also, the node this command needs to be run as root on a trusted PNFS machine.

```
# awk '{print $10}' /tmp/tag_dir_dump_uniq | while read line; do
   echo mkdir -p $line;
   mkdir -p $line;
done
```

At this point, root owns all the newly created directories. They need to be set to each storage groups group UID and GID. This information is not in the Enstore DB, which requires some level of intuition to pick an appropriate value. And it is likely that some of them will end up incorrect. However, each user/group should be able to fix any remaining incorrect ownership issues on their own.

```
# chown -R <UID1>.<GID1> /pnfs/<mount_point_1>

# chown -R <UID2>.<GID2> /pnfs/<mount_point_2>
```

The tags also need to be corrected to reflect where new files should be written. The following command will report on differences between tags. Note: directories with multiple libraries specified will not be caught by this check; those need to be handled by the administrator doing the rebuild.

```
# cat /tmp/tag_dir_dump_uniq | while read line; do
   #Get the tags from the database dump.
   library=`echo $line | awk '{print $2}'`;
   storage_group=`echo $line | awk '{print $4}'`;
   file_family=`echo $line | awk '{print $6}'`;
   wrapper=`echo $line | awk '{print $8}'`;

   #Get the directory we are currently looking at.
   dname=`echo $line | awk '{print $10}'`;
   if [ -d $dname ]; then
      #Get the tags currently in pnfs.
      p_library=`enstore pnfs --tag library $dname`;
      p_storage_group=`enstore pnfs --tag storage_group $dname`;
      p_file_family=`enstore pnfs --tag file_family $dname`;
      p_wrapper=`enstore pnfs --tag file_family_wrapper $dname`;

      #Report which tags differ from what PNFS currently has.
      if [ "$library" != "$p_library" ]; then
         echo "library ($p_library, $library) differs for $dname";
      fi;
      if [ "$storage_group" != "$p_storage_group" ]; then
         echo "storage_group ($p_storage_group, $storage_group) differs
for $dname";
      fi;
      if [ "$file_family" != "$p_file_family" ]; then
         echo "file_family ($p_file_family, $file_family) differs for
```

```
$dname";
      fi;
      if [ "$wrapper" != "$p_wrapper" ]; then
          echo "wrapper ($p_wrapper, $wrapper) differs for $dname";
      fi;
    else echo "$dname does not exist";
    fi;
done
```

Dump the non-deleted bfids from the database. Migration originals and duplication/multiple copies need to be handled special.

```
$ psql -h <hostname> -p 8888 enstoredb -U enstore -c "
--Find normal file bfids, skip migration originals, skip multiple copies.
select file.bfid
from file
where deleted = 'n'
  and bfid not in (select alt_bfid
                     from file_copies_map
                     where alt_bfid = file.bfid)
  and bfid not in (select src_bfid
                     from migration
                     where src_bfid = file.bfid)
union
--Find duplication primary bfids.  This includes multiple copies.
select file.bfid
from file,file_copies_map
where deleted = 'n'
  and file.bfid = file_copies_map.bfid
" > /tmp/bfids_to_restore
```

Then loop over the bfids recreating the files.

```
# cat /tmp/bfids_to_restore | while read line; do
    echo $line;
    enstore file --restore $line --force;
done > /tmp/file_rebuild_log 2>&1
```

10.2 Rebuilding an Enstore DB from a PNFS database

This direction of rebuild is more painful than the other way around. Deleted files will not be able to be recovered. This will leave confusing gaps in tape lists.

Warning: This documentation is untested, but should provide a basis for performing this operation.

We need to obtain a list of all files in PNFS. This list may be able to be obtained from the COMPLETE_FILE_LIST page from the Inventory Summary.

```
$ wget
```
http://www-stken.fnal.gov/enstore/tape_inventory/COMPLETE_FILE_LISTING

or

```
$ find /pnfs/fs/usr -type f > COMPLETE_FILE_LISTING
```

Note: The column position of the file name will be in different depending on which method was used. For the former, it is 8 and 1 one for latter.

```
$ for item in `cut -f [1 | 8]-d " " /tmp/COMPLETE_FILE_SYSTEM`; do
   layer4=`enstore pnfs --layer $item 4`;
   volume=`echo "$layer4" | sed -n 1p`;
   lc=`echo "$layer4" | sed -n 2p`;
   size=`echo "$layer4" | sed -n 3p`;
   fname=`echo "$layer4" | sed -n 5p`;
   pnfsid=`echo "$layer4" | sed -n 7p`;
   bfid=`echo "$layer4" | sed -n 9p`;
   drive=`echo "$layer4" | sed -n 10p`;
   crc=`echo "$layer4" | sed -n 11p`;
   uid=`ls -ln $item | awk '{print $3}'`;
   gid=`ls -ln $item | awk '{print $4'}`;

   sg=`echo $fname | sed -e 's:/pnfs/fs/usr/::' -e 's:/pnfs/::' | tr '/' '
' | awk '{print $1}'`
   ff=`echo "$layer4" | sed -n 4p`;

   #Make sure that we have valid information from layer 4.
   if [ -z "$volume" ]; then continue; fi;

   #Determine if volume has been added already.
   grep $volume /tmp/vol_list
   if [ $? -eq 1 ]; then
      #Add volume to DB.
      enstore volume --add $volume UNKNOWN $sg $ff UNKNOWN UNKNOWN
UNKNOWN;
   fi

   #Add the file to the DB.
   enstore file --add $bfid pnfs_name0=$fname complete_crc=$crc size=$size
pnsfid=$pnfsid deleted=n drive=$drive external_label=$volume
location_cookie=$lc uid=$uid gid=$gid;
done
```

The values for the UNKNOWN volume fields above need to be corrected.
- For NULL volumes (which should be easy to identify from the volume name):
  - library will be the null library from the configuration.
  - wrapper will be 'null'.

- o media_type will be 'null'
  - o capacity will be some large made up value.
- For tape volumes:
  - o library will be matching library for the media type. The few "test" volumes belonging to test libraries can be fixed later.
  - o The file locations will need to be evaluated to identify any cern wrapper tapes. cpio_odc wrapped tapes write files to all locations, while cern wrapper wrapped tapes skip locations where the file headers and footers are written. File sizes larger than 8GB can only be written to cern wrapper volumes; this may provide a clue for some volumes.
  - o For the media_type, a dump of volume information from the robots should provide enough of a clue to set these values.
  - o Once the media_type is known, the capacity can be calculated.
- For disk volumes (which should be easy to identify from the volume name):
  - o The library will be the disk library from the configuraiton.
  - o wrapper will be 'null'.
  - o media_type will be 'disk'
  - o capacity will need to be some made up value. The most correct thing to do would be to base it on the size of the file system that the file is stored on.

Chapter 11: Metadata Scanning

A metadata scan audits the contents of the Chimera and Enstore databases. The audit report is written to a natural-language text file and a separate machine-parsable text file.

11.1 Definitions

A scan is done in one of two directions indicated below. Running a scan in both directions is necessary in order to find all inconsistencies.

1. **Forward scan**: Walk through the Chimera namespace, check it for errors, and compare the metadata with that in the file and volume tables in the Enstore database.
2. **Reverse scan**: Walk through the Enstore database, check it for errors, and compare it with the contents in the Chimera namespace.

A scan generates **notices** organized by file path. These can be informational, warnings, errors, or critical errors.

11.2 Requirements

1. The scan is required be run with UID and GID 0, i.e. as root. Failing this, an error will be reported and the scan will exit. This is necessary to allow Chimera metadata files with restricted access to be read.
2. At most, only one instance of the scan can be run at a time. Failing this, an error will be reported and the scan will exit. This is necessary to prevent file-write conflicts and also a reduction in the scan's speed.

11.3 Recommendations

1. A full scan should not be run on a production system. Scanning a very limited subset of the data should, however, be fine on a production system. A full scan should be done on a non-production copy of the databases from their recent backups.
2. Using backups implies a difference in the time at which each of the two backups were created. This difference manifests itself as discrepancies among the two databases. The discrepancies may lead to some scan notices being falsely reported. It is therefore recommended that this time difference be minimized.
3. A scan may print status updates to the `stdout` and `stderror` streams. For a full scan, is recommended that these be saved to a

temporary file.

4. Provisioning the databases, especially the Chimera database, on a SSD is expected to boost scan speed relative to provisioning them on a HDD.

11.4 Usage

The "`enstore scan`" command, with the appropriate command-line options, starts a scan. The only required option is `--type`. The following is the usage information for each available option, expanded upon from what is available at the command-line.

`-h, --help`
Show a help message, and exit.

`-t SCAN_TYPE, --type=SCAN_TYPE`
Start a scan of the specified type. This is a required option with no default value. The specified value of `SCAN_TYPE` may be `forward` or `reverse`, with the latter pending implementation.

`-d FS_ROOT, --directory=FS_ROOT`
For a forward scan, with `FS_ROOT` as an absolute directory path, scan the directory recursively. The default is `/pnfs/fs/usr`. It is <u>not</u> recommended to specify this option especially with large nested directories, as the scan speed may then be lower than normal. It is not possible to specify multiple unrelated directories or individual files to be scanned with a single command, but this feature may be implemented in a future release.

`-o OUTPUT_FILE, --output_file=OUTPUT_FILE`
Absolute path to the natural-language output file for notices. The default is dynamic and is based on the current date and time, e.g. `/home/enstore/scan_20121203T164717.log`). The file is appended if it exists. If the file exists and is to be overwritten instead, it must be deleted before starting the scan. The machine-parsable output file is always dynamic; it has the "`.dict`" extension appended, e.g. `/home/enstore/scan_20121203T164717.log.dict`.

`-p PRINT, --print=PRINT`
For the specified scan type, print either a list of all checks and their overviews, or a list of all notice -templates, and exit . Accordingly, `PRINT` may be `checks` or `notices`. A scan is not run.

`-r, --resume`

For the specified existing output file (as specified by `--output-file`), resume the scan approximately where it was previously aborted. If this option is not specified, if the output file exists, the scan is not resumed, but is restarted instead. It is recommended that a scan be resumed with the same set of databases only. The corresponding output files must not have been moved, failing which its checkpoint will be deleted and the scan cannot then be resumed. Checkpoints older than 60 days are also deleted.

```
-s STATUS_INTERVAL,
--status_interval=STATUS_INTERVAL
```
Maximum interval in seconds between successive status output updates to `stdout`. The default is increased gradually from 1 up to 600 and is maintained at this number. A final status update is also printed at the completion of the scan.

11.4.1 Usage examples

```
enstore scan -t forward
```
This command starts the full forward scan for `/pnfs/fs/usr`. The notices output file path, along with periodic status updates are printed to `stdout`.

```
enstore scan -t forward -p notices
```
This command prints a list of all possible notice types that may generated by a forward scan, and exits. A scan is not run.

```
enstore scan -t forward -o scan_20121203T164717.log
-r
```
This command resumes a previously aborted forward scan which used the noted output file. Assuming the output file and its checkpoint exist, the scan will be resumed approximately at where it was previously aborted.

```
enstore scan -t forward -d /pnfs/fs/usr/foo/bar
```
This command performs a forward scan limited to the specified directory and its sub-directories recursively.

11.5 Status output

The periodic status output, as printed to `stdout`, may resemble:

```
[root@stkenscan1 ~]# enstore scan -t forward
The scan notices output files are
"/root/scan_20121221T134118.log" and
"/root/scan_20121221T134118.log.dict". These files
```

```
                  do not exist and will be created.

                  2012-12-21 13:41:22
                  Active workers              : 12
                  Time elapsed                : 0:00:00
                  Items scanned (cumulative)  : 0
                  Speed (cumulative) (items/s): 0
                  Speed (current) (items/s)   : 0
                  Items remaining             : 15,926,335
                  Time to next update         : 0:00:01

                  2012-12-21 13:41:23
                  Active workers              : 12
                  Time elapsed                : 0:00:01
                  Items scanned (cumulative)  : 48
                  Speed (cumulative) (items/s): 45
                  Speed (current) (items/s)   : 47
                  Items remaining             : 15,926,287
                  Time remaining              : 4 days, 0:49:31
                  Time total                  : 4 days, 0:49:32
                  Time to next update         : 0:00:03
```

The estimate of the time remaining is computed by dividing the number of items remaining by the cumulative speed of the items scanned. For a full scan, this estimate takes a few iterations to stabilize.

The number of active workers is a multiple of the number of CPU cores. If this number of active workers falls to below its initial peak, an error may have occurred and printed to `stderr`. While no such error is known to exist, if one does occur, it is to be investigated. If the number of active workers drops to 0, the scan should be aborted as it will never complete.

11.6 Notices output

Logged notices are organized by filesystem object (files, directories, etc.) path. Filesystem objects without any notices are not logged. In other words, each logged filesystem object has one or more associated notices.

Each notice has a level, which can be one of `INFO`, `WARNING`, `ERROR`, or `CRITICAL`. While an `INFO` notice may not be a concern in itself, it may help in the understanding of any co-occurring non-`INFO` notices.

Each notice also has a code. Examples of these are `CRCNone`, `MarkedDel`,

and `PathMismatch`. Notice codes and descriptions are specific to the current scan type. For forward scan, a list is available with the "`-t forward -p notices`" options. Depending upon the notice code, its associated natural language description may be generated dynamically.

Some notices may not be detected and logged until the existing notices for the file are mitigated and a scan is repeated.

11.6.1 Natural language notices output

The natural language log is meant to be human readable. As such, it is not meant for use by any automated mitigation method.

Structure

Entries have the format:
```
<filesystem object path>
<notice 1 line>
<notice 2 line> (as applicable)
...
<empty line>
```

Each notice line has the format:
```
<level> (<code>): <description>
```

Examples

Examples of successively logged entries are:

```
/pnfs/fs/usr/beamstool/DocDB/BeamDocDB-2003-12-06
WARNING (CRCNone): CRC is missing in both layer 4
and Enstore file info.
ERROR (PathMismatch): File path mismatch. Normalized
file paths  for file with layer 1 BFID
"CDMS144999398900000" provided by Enstore and layer
4 (/<pnfs>/beamstool/DocDB/BeamDocDB-2004-01-05);
and filesystem
(/<pnfs>/beamstool/DocDB/BeamDocDB-2003-12-06) don't
all match. File may have been moved.

/pnfs/fs/usr/beamstool/SBDAWS/V_1_36_filling
ERROR (Size0FileInfoOk): File is empty. Its lstat
size is 0 and its layer 2 size is not present. Its
info in Enstore is ok.
ERROR (FileInfoDelNone): The "deleted" field is
```

```
missing in Enstore file info.
```

11.6.2 Machine parsable notices output

Entries in the machine parsable log are organized in the same manner as the natural language log, but are formatted as Unicode string representations of Python `dict` objects, with one on each line.

It is not possible to write the entire log meaningfully with a standard set of comma-separated values (CSV) because the needed fields vary greatly in each line. In particular, the notices logged in each line can vary, and the notice arguments vary by notice code as well.

Structure

The `dict` in each line has the items:
```
path: <path>
notices: <notices dict>
```

The `notices dict` in each line has the items:
```
<notice code 1>: <notice dict>
<notice code 2>: <notice dict> (as applicable)
...
```

Each `notice dict` has the items:
```
level: <level>
args: <args dict>
```

Each `args dict` has the items:
```
<arg 1>: <value> (as applicable)
<arg 2>: <value> (as applicable)
...
```

A notice code can be expected to always have the same set of `args` keys, with differing values. For example, the notice code `PathMismatch` can be expected to have the `args` keys `path_filesystem`, `path_layer4`, `path_enstore`, and `bfid_layer1`.

Examples

Examples of successively logged entries are:

```
{u'notices': {u'CRCNone': {u'args': {}, u'level':
u'WARNING'}, u'PathMismatch': {u'args':
```

```
{u'path_enstore':
u'/<pnfs>/beamstool/DocDB/BeamDocDB-2004-01-05',
u'bfid_layer1': 'CDMS144999398900000',
u'path_layer4':
u'/<pnfs>/beamstool/DocDB/BeamDocDB-2004-01-05',
u'path_filesystem':
u'/<pnfs>/beamstool/DocDB/BeamDocDB-2003-12-06'},
u'level': u'ERROR'}}, u'path':
u'/pnfs/fs/usr/beamstool/DocDB/BeamDocDB-2003-12-06'
}
{u'notices': {u'FileInfoDelNone': {u'args': {},
u'level': u'ERROR'}, u'Size0FileInfoOk': {u'args':
{u'layer2_size': u'not present'}, u'level':
u'ERROR'}}, u'path':
u'/pnfs/fs/usr/beamstool/SBDAWS/V_1_36_filling'}
```

The prefix "u" at the start of each string indicates Unicode.

Usage example

Python can be used to process the log file and provide the required output. For example, the layer 1 BFID, filesystem path, and Enstore path of all `PathMismatch` notices can be printed to `stdout` in a tab-separated format. The following sample Python code can be used:

```
#!/usr/bin/env python
import ast

file_in = 'scan_20121218T140215.log.dict'
notice_reqd = 'PathMismatch'
```

```
        args_reqd = ('bfid_layer1',

                     'path_filesystem',

                     'path_enstore',)


        for line in open(file_in):

            record = ast.literal_eval(line)

            notices = record['notices']

            if notice_reqd in notices:

                args = notices['PathMismatch']['args']

                args = [args[arg] for arg in args_reqd]

                output = '\t'.join(args)

                print(output)
```

11.7 Mitigation (link)

To mitigate the logged notices, refer to the pertinent section in the resource linked below:

http://www-ccf.fnal.gov/PNFS_Scan.pdf

Alphabetical Index