Red Hat Enterprise Linux 7

Storage Administration Guide

Deploying and configuring single-node storage in Red Hat Enterprise Linux 7

Red Hat Enterprise Linux 7 Storage Administration Guide

Deploying and configuring single-node storage in Red Hat Enterprise Linux 7

Milan Navrátil
Red Hat Customer Content Services

Jacquelynn East
Red Hat Customer Content Services

Don Domingo
Red Hat Customer Content Services

Josef Bacik
Server Development Kernel File System
jwhiter@redhat.com
Disk Quotas

Kamil Dudka
Base Operating System Core Services - BRNO
kdudka@redhat.com
Access Control Lists

Hans de Goede
Base Operating System Installer
hdegoede@redhat.com
Partitions

Harald Hoyer
Engineering Software Engineering
harald@redhat.com
File Systems

Dennis Keefe
Base Operating Systems Kernel Storage
dkeefe@redhat.com
VDO

Doug Ledford
Server Development Hardware Enablement
dledford@redhat.com
RAID

Daniel Novotny
Base Operating System Core Services - BRNO
dnovotny@redhat.com
The /proc File System

Nathan Straz
Quality Engineering QE - Platform
nstraz@redhat.com

Legal Notice

Cody Watso
Base Operating Systems Kernel Storage
awalsh@redhat.com
VDO

Daniel Wysochanski
Server Development Kernel Storage
dwysocha@redhat.com
LVM/LVM2

Michael Christie
Server Development Kernel Storage
mchristi@redhat.com
Online Storage

Sachin Prabhu
Software Maintenance Engineering
sprabhu@redhat.com
NFS

Rob Evers
Server Development Kernel Storage
revers@redhat.com
Online Storage

David Howells
Server Development Kernel Storage
dhowells@redhat.com
FS-Cache

David Lehman
Base Operating System Installer
dlehman@redhat.com
Storage configuration during installation

Jeff Moyer
Server Development Kernel File System
jmoyer@redhat.com
Solid-State Disks

Eric Sandeen
Server Development Kernel File System
esandeen@redhat.com
ext3, ext4, XFS, Encrypted File Systems

Mike Snitzer
Server Development Kernel Storage
msnitzer@redhat.com
I/O Stack and Limits

Abstract

This guide provides instructions on how to effectively manage storage devices and file systems on Red Hat Enterprise Linux 7. It is intended for use by system administrators with basic to intermediate knowledge of Red Hat Enterprise Linux or Fedora.

Red Hat Subject Matter Experts

Contributors

Edited by

Marek Suchánek
Red Hat Customer Content Services
msuchane@redhat.com

Apurva Bhide
Red Hat Customer Content Services
abhide@redhat.com

Table of Contents

CHAPTER 1. OVERVIEW

The *Storage Administration Guide* contains extensive information on supported file systems and data storage features in Red Hat Enterprise Linux 7. This book is intended as a quick reference for administrators managing single-node (that is, non-clustered) storage solutions.

The Storage Administration Guide is split into the following parts: File Systems, Storage Administration, and Data Deduplication and Compression with VDO.

The File Systems part details the various file systems Red Hat Enterprise Linux 7 supports. It describes them and explains how best to utilize them.

The Storage Administration part details the various tools and storage administration tasks Red Hat Enterprise Linux 7 supports. It describes them and explains how best to utilize them.

The Data Deduplication and Compression with VDO part describes the Virtual Data Optimizer (VDO). It explains how to use VDO to reduce your storage requirements.

1.1. NEW FEATURES AND ENHANCEMENTS IN RED HAT ENTERPRISE LINUX 7

Red Hat Enterprise Linux 7 features the following file system enhancements:

eCryptfs not included

As of Red Hat Enterprise Linux 7, eCryptfs is not included. For more information on encrypting file systems, see Red Hat's Security Guide.

System Storage Manager

Red Hat Enterprise Linux 7 includes a new application called System Storage Manager which provides a command-line interface to manage various storage technologies. For more information, see Chapter 16, *System Storage Manager (SSM)*.

XFS Is the Default File System

As of Red Hat Enterprise Linux 7, XFS is the default file system. For more information about the XFS file system, see Chapter 3, *The XFS File System*.

File System Restructure

Red Hat Enterprise Linux 7 introduces a new file system structure. The directories **/bin**, **/sbin**, **/lib**, and **/lib64** are now nested under **/usr**.

Snapper

Red Hat Enterprise Linux 7 introduces a new tool called Snapper that allows for the easy creation and management of snapshots for LVM and Btrfs. For more information, see Chapter 14, *Creating and Maintaining Snapshots with Snapper*.

Btrfs (Technology Preview)

NOTE

Btrfs is available as a Technology Preview feature in Red Hat Enterprise Linux 7 but has been deprecated since the Red Hat Enterprise Linux 7.4 release. It will be removed in a future major release of Red Hat Enterprise Linux.

For more information, see Deprecated Functionality in the Red Hat Enterprise Linux 7.4 Release Notes.

Btrfs is a local file system that aims to provide better performance and scalability, including integrated LVM operations. This file system is not fully supported by Red Hat and as such is a technology preview. For more information on Btrfs, see Chapter 6, *Btrfs (Technology Preview)*.

NFSv2 No Longer Supported

As of Red Hat Enterprise Linux 7, NFSv2 is no longer supported.

PART I. FILE SYSTEMS

The File Systems section provides information on the file system structure and maintenance, the Btrfs Technology Preview, and file systems that Red Hat fully supports: ext3, ext4, GFS2, XFS, NFS, and FS-Cache.

> **NOTE**
>
> Btrfs is available as a Technology Preview feature in Red Hat Enterprise Linux 7 but has been deprecated since the Red Hat Enterprise Linux 7.4 release. It will be removed in a future major release of Red Hat Enterprise Linux.
>
> For more information, see Deprecated Functionality in the Red Hat Enterprise Linux 7.4 Release Notes.

For an overview of Red Hat Enterprise Linux file systems and storage limits, see Red Hat Enterprise Linux technology capabilities and limits at Red Hat Knowledgebase.

XFS is the default file system in Red Hat Enterprise Linux 7 and Red Hat, and Red Hat recommends you to use XFS unless you have a strong reason to use another file system. For general information on common file systems and their properties, see the following Red Hat Knowledgebase article: How to Choose your Red Hat Enterprise Linux File System.

CHAPTER 2. FILE SYSTEM STRUCTURE AND MAINTENANCE

The file system structure is the most basic level of organization in an operating system. The way an operating system interacts with its users, applications, and security model nearly always depends on how the operating system organizes files on storage devices. Providing a common file system structure ensures users and programs can access and write files.

File systems break files down into two logical categories:

Shareable and unsharable files

> *Shareable* files can be accessed locally and by remote hosts. *Unsharable* files are only available locally.

Variable and static files

> *Variable* files, such as documents, can be changed at any time. *Static* files, such as binaries, do not change without an action from the system administrator.

Categorizing files in this manner helps correlate the function of each file with the permissions assigned to the directories which hold them. How the operating system and its users interact with a file determines the directory in which it is placed, whether that directory is mounted with read-only or read and write permissions, and the level of access each user has to that file. The top level of this organization is crucial; access to the underlying directories can be restricted, otherwise security problems could arise if, from the top level down, access rules do not adhere to a rigid structure.

2.1. OVERVIEW OF FILESYSTEM HIERARCHY STANDARD (FHS)

Red Hat Enterprise Linux uses the *Filesystem Hierarchy Standard* (*FHS*) file system structure, which defines the names, locations, and permissions for many file types and directories.

The FHS document is the authoritative reference to any FHS-compliant file system, but the standard leaves many areas undefined or extensible. This section is an overview of the standard and a description of the parts of the file system not covered by the standard.

The two most important elements of FHS compliance are:

- Compatibility with other FHS-compliant systems

- The ability to mount a **/usr/** partition as read-only. This is crucial, since **/usr/** contains common executables and should not be changed by users. In addition, since **/usr/** is mounted as read-only, it should be mountable from the CD-ROM drive or from another machine via a read-only NFS mount.

2.1.1. FHS Organization

The directories and files noted here are a small subset of those specified by the FHS document. For the most complete information, see the latest FHS documentation at http://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf; the file-hierarchy(7) man page also provides an overview.

> **NOTE**
>
> The directories that are available depend on what is installed on any given system. The following lists are only an example of what may be found.

2.1.1.1. Gathering File System Information

df Command

The **df** command reports the system's disk space usage. Its output looks similar to the following:

Example 2.1. df Command Output

```
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
                      11675568   6272120   4810348  57% / /dev/sda1
                 100691      9281     86211  10% /boot
none                   322856         0    322856   0% /dev/shm
```

By default, **df** shows the partition size in 1 kilobyte blocks and the amount of used and available disk space in kilobytes. To view the information in megabytes and gigabytes, use the command **df -h**. The **-h** argument stands for "human-readable" format. The output for **df -h** looks similar to the following:

Example 2.2. df -h Command Output

```
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
                       12G  6.0G  4.6G  57% / /dev/sda1
   99M  9.1M   85M  10% /boot
none     316M     0  316M   0% /dev/shm
```

> **NOTE**
>
> In the given examples, the mounted partition **/dev/shm** represents the system's virtual memory file system.

du Command

The **du** command displays the estimated amount of space being used by files in a directory, displaying the disk usage of each subdirectory. The last line in the output of **du** shows the total disk usage of the directory. To see only the total disk usage of a directory in human-readable format, use **du -hs**. For more options, see **man du**.

Gnome System Monitor

To view the system's partitions and disk space usage in a graphical format, use the Gnome **System Monitor** by clicking on **Applications → System Tools → System Monitor** or using the command **gnome-system-monitor**. Select the **File Systems** tab to view the system's partitions. The following figure illustrates the **File Systems** tab.

Figure 2.1. File Systems Tab in GNOME System Monitor

2.1.1.2. The `/boot/` Directory

The **`/boot/`** directory contains static files required to boot the system, for example, the Linux kernel. These files are essential for the system to boot properly.

> ⚠️ **WARNING**
>
> Do not remove the **`/boot/`** directory. Doing so renders the system unbootable.

2.1.1.3. The `/dev/` Directory

The **`/dev/`** directory contains device nodes that represent the following device types:

- devices attached to the system;

- virtual devices provided by the kernel.

These device nodes are essential for the system to function properly. The **udevd** daemon creates and removes device nodes in **`/dev/`** as needed.

Devices in the **`/dev/`** directory and subdirectories are defined as either *character* (providing only a serial

stream of input and output, for example, mouse or keyboard) or *block* (accessible randomly, such as a hard drive or a floppy drive). If GNOME or KDE is installed, some storage devices are automatically detected when connected (such as with USB) or inserted (such as a CD or DVD drive), and a pop-up window displaying the contents appears.

Table 2.1. Examples of Common Files in the `/dev` Directory

File	Description
`/dev/hda`	The master device on the primary IDE channel.
`/dev/hdb`	The slave device on the primary IDE channel.
`/dev/tty0`	The first virtual console.
`/dev/tty1`	The second virtual console.
`/dev/sda`	The first device on the primary SCSI or SATA channel.
`/dev/lp0`	The first parallel port.

A valid block device can be one of two types of entries:

Mapped device

A logical volume in a volume group, for example, **`/dev/mapper/VolGroup00-LogVol02`**.

Static device

A traditional storage volume, for example, **`/dev/sdbX`**, where *sdb* is a storage device name and *X* is the partition number. **`/dev/sdbX`** can also be **`/dev/disk/by-id/WWID`**, or **`/dev/disk/by-uuid/UUID`**. For more information, see Section 25.7, "Persistent Naming".

2.1.1.4. The `/etc/` Directory

The **`/etc/`** directory is reserved for configuration files that are local to the machine. It should not contain any binaries; if there are any binaries, move them to **`/usr/bin/`** or **`/usr/sbin/`**.

For example, the **`/etc/skel/`** directory stores "skeleton" user files, which are used to populate a home directory when a user is first created. Applications also store their configuration files in this directory and may reference them when executed. The **`/etc/exports`** file controls which file systems export to remote hosts.

2.1.1.5. The `/mnt/` Directory

The **`/mnt/`** directory is reserved for temporarily mounted file systems, such as NFS file system mounts. For all removable storage media, use the **`/media/`** directory. Automatically detected removable media is mounted in the **`/media`** directory.

> **IMPORTANT**
>
> The **/mnt** directory must not be used by installation programs.

2.1.1.6. The **/opt/** Directory

The **/opt/** directory is normally reserved for software and add-on packages that are not part of the default installation. A package that installs to **/opt/** creates a directory bearing its name, for example, **/opt/*packagename/***. In most cases, such packages follow a predictable subdirectory structure; most store their binaries in **/opt/*packagename/*bin/** and their **man** pages in **/opt/*packagename/*man/**.

2.1.1.7. The **/proc/** Directory

The **/proc/** directory contains special files that either extract information from the kernel or send information to it. Examples of such information include system memory, CPU information, and hardware configuration. For more information about **/proc/**, see Section 2.3, "The /proc Virtual File System".

2.1.1.8. The **/srv/** Directory

The **/srv/** directory contains site-specific data served by a Red Hat Enterprise Linux system. This directory gives users the location of data files for a particular service, such as FTP, WWW, or CVS. Data that only pertains to a specific user should go in the **/home/** directory.

2.1.1.9. The **/sys/** Directory

The **/sys/** directory utilizes the new **sysfs** virtual file system specific to the kernel. With the increased support for hot plug hardware devices in the kernel, the **/sys/** directory contains information similar to that held by **/proc/**, but displays a hierarchical view of device information specific to hot plug devices.

2.1.1.10. The **/usr/** Directory

The **/usr/** directory is for files that can be shared across multiple machines. The **/usr/** directory is often on its own partition and is mounted read-only. At a minimum, **/usr/** should contain the following subdirectories:

/usr/bin

This directory is used for binaries.

/usr/etc

This directory is used for system-wide configuration files.

/usr/games

This directory stores games.

/usr/include

This directory is used for C header files.

/usr/kerberos

This directory is used for Kerberos-related binaries and files.

/usr/lib

This directory is used for object files and libraries that are not designed to be directly utilized by shell scripts or users.

As of Red Hat Enterprise Linux 7.0, the **/lib/** directory has been merged with **/usr/lib**. Now it also contains libraries needed to execute the binaries in **/usr/bin/** and **/usr/sbin/**. These shared library images are used to boot the system or execute commands within the root file system.

/usr/libexec

This directory contains small helper programs called by other programs.

/usr/sbin

As of Red Hat Enterprise Linux 7.0, **/sbin** has been moved to **/usr/sbin**. This means that it contains all system administration binaries, including those essential for booting, restoring, recovering, or repairing the system. The binaries in **/usr/sbin/** require root privileges to use.

/usr/share

This directory stores files that are not architecture-specific.

/usr/src

This directory stores source code.

/usr/tmp linked to /var/tmp

This directory stores temporary files.

The **/usr/** directory should also contain a **/local/** subdirectory. As per the FHS, this subdirectory is used by the system administrator when installing software locally, and should be safe from being overwritten during system updates. The **/usr/local** directory has a structure similar to **/usr/**, and contains the following subdirectories:

- **/usr/local/bin**

- **/usr/local/etc**

- **/usr/local/games**

- **/usr/local/include**

- **/usr/local/lib**

- **/usr/local/libexec**

- **/usr/local/sbin**

- **/usr/local/share**

- **/usr/local/src**

Red Hat Enterprise Linux's usage of **/usr/local/** differs slightly from the FHS. The FHS states that **/usr/local/** should be used to store software that should remain safe from system software upgrades. Since the **RPM Package Manager** can perform software upgrades safely, it is not necessary

to protect files by storing them in **/usr/local/**.

Instead, Red Hat Enterprise Linux uses **/usr/local/** for software local to the machine. For instance, if the **/usr/** directory is mounted as a read-only NFS share from a remote host, it is still possible to install a package or program under the **/usr/local/** directory.

2.1.1.11. The /var/ Directory

Since the FHS requires Linux to mount **/usr/** as read-only, any programs that write log files or need **spool/** or **lock/** directories should write them to the **/var/** directory. The FHS states **/var/** is for variable data, which includes spool directories and files, logging data, transient and temporary files.

Following are some of the directories found within the **/var/** directory:

- **/var/account/**
- **/var/arpwatch/**
- **/var/cache/**
- **/var/crash/**
- **/var/db/**
- **/var/empty/**
- **/var/ftp/**
- **/var/gdm/**
- **/var/kerberos/**
- **/var/lib/**
- **/var/local/**
- **/var/lock/**
- **/var/log/**
- **/var/mail** linked to **/var/spool/mail/**
- **/var/mailman/**
- **/var/named/**
- **/var/nis/**
- **/var/opt/**
- **/var/preserve/**
- **/var/run/**
- **/var/spool/**

- **/var/tmp/**

- **/var/tux/**

- **/var/www/**

- **/var/yp/**

> **IMPORTANT**
>
> The **/var/run/media/*user*** directory contains subdirectories used as mount points for removable media such as USB storage media, DVDs, CD-ROMs, and Zip disks. Note that previously, the **/media/** directory was used for this purpose.

System log files, such as **messages** and **lastlog**, go in the **/var/log/** directory. The **/var/lib/rpm/** directory contains RPM system databases. Lock files go in the **/var/lock/** directory, usually in directories for the program using the file. The **/var/spool/** directory has subdirectories that store data files for some programs. These subdirectories include:

- **/var/spool/at/**

- **/var/spool/clientmqueue/**

- **/var/spool/cron/**

- **/var/spool/cups/**

- **/var/spool/exim/**

- **/var/spool/lpd/**

- **/var/spool/mail/**

- **/var/spool/mailman/**

- **/var/spool/mqueue/**

- **/var/spool/news/**

- **/var/spool/postfix/**

- **/var/spool/repackage/**

- **/var/spool/rwho/**

- **/var/spool/samba/**

- **/var/spool/squid/**

- **/var/spool/squirrelmail/**

- **/var/spool/up2date/**

- **/var/spool/uucp/**

- **/var/spool/uucppublic/**

- **/var/spool/vbox/**

2.2. SPECIAL RED HAT ENTERPRISE LINUX FILE LOCATIONS

Red Hat Enterprise Linux extends the FHS structure slightly to accommodate special files.

Most files pertaining to RPM are kept in the **/var/lib/rpm/** directory. For more information on RPM, see **man rpm**.

The **/var/cache/yum/** directory contains files used by the **Package Updater**, including RPM header information for the system. This location may also be used to temporarily store RPMs downloaded while updating the system. For more information about the Red Hat Network, see https://rhn.redhat.com/.

Another location specific to Red Hat Enterprise Linux is the **/etc/sysconfig/** directory. This directory stores a variety of configuration information. Many scripts that run at boot time use the files in this directory.

2.3. THE /PROC VIRTUAL FILE SYSTEM

Unlike most file systems, **/proc** contains neither text nor binary files. Because it houses *virtual files*, the **/proc** is referred to as a virtual file system. These virtual files are typically zero bytes in size, even if they contain a large amount of information.

The **/proc** file system is not used for storage. Its main purpose is to provide a file-based interface to hardware, memory, running processes, and other system components. Real-time information can be retrieved on many system components by viewing the corresponding **/proc** file. Some of the files within **/proc** can also be manipulated (by both users and applications) to configure the kernel.

The following **/proc** files are relevant in managing and monitoring system storage:

/**proc**/**devices**

   Displays various character and block devices that are currently configured.

/**proc**/**filesystems**

   Lists all file system types currently supported by the kernel.

/**proc**/**mdstat**

   Contains current information on multiple-disk or RAID configurations on the system, if they exist.

/**proc**/**mounts**

   Lists all mounts currently used by the system.

/**proc**/**partitions**

   Contains partition block allocation information.

For more information about the **/proc** file system, see the Red Hat Enterprise Linux 7 *Deployment Guide*.

2.4. DISCARD UNUSED BLOCKS

Batch discard and online discard operations are features of mounted file systems that discard blocks not in use by the file system. They are useful for both solid-state drives and thinly-provisioned storage.

- *Batch discard operations* are run explicitly by the user with the **fstrim** command. This command discards all unused blocks in a file system that match the user's criteria.

- *Online discard operations* are specified at mount time, either with the **-o discard** option as part of a **mount** command or with the **discard** option in the **/etc/fstab** file. They run in real time without user intervention. Online discard operations only discard blocks that are transitioning from used to free.

Both operation types are supported for use with ext4 file systems as of Red Hat Enterprise Linux 6.2 and later and with XFS file systems since Red Hat Enterprise Linux 6.4. Also, the block device underlying the file system must support physical discard operations. Physical discard operations are supported if the value stored in the **/sys/block/*device*/queue/discard_max_bytes** file is not zero.

If you are executing the **fstrim** command on:

- a device that does not support discard operations, or

- a logical device (LVM or MD) comprised of multiple devices, where any one of the device does not support discard operations

the following message will be displayed:

```
fstrim -v /mnt/non_discard
fstrim: /mnt/non_discard: the discard operation is not supported
```

> **NOTE**
>
> The **mount** command allows you to mount a device that does not support discard operations with the **-o discard** option.

Red Hat recommends batch discard operations unless the system's workload is such that batch discard is not feasible, or online discard operations are necessary to maintain performance.

For more information, see the fstrim(8) and mount(8) man pages.

CHAPTER 3. THE XFS FILE SYSTEM

XFS is a highly scalable, high-performance file system which was originally designed at Silicon Graphics, Inc. XFS is the default file system for Red Hat Enterprise Linux 7.

Main Features of XFS

- XFS supports *metadata journaling*, which facilitates quicker crash recovery.

- The XFS file system can be defragmented and enlarged while mounted and active.

- In addition, Red Hat Enterprise Linux 7 supports backup and restore utilities specific to XFS.

Allocation Features

XFS features the following allocation schemes:

- Extent-based allocation

- Stripe-aware allocation policies

- Delayed allocation

- Space pre-allocation

Delayed allocation and other performance optimizations affect XFS the same way that they do ext4. Namely, a program's writes to an XFS file system are not guaranteed to be on-disk unless the program issues an **fsync()** call afterwards.

For more information on the implications of delayed allocation on a file system (ext4 and XFS), see Allocation Features in Chapter 5, *The ext4 File System*.

> **NOTE**
>
> Creating or expanding files occasionally fails with an unexpected ENOSPC write failure even though the disk space appears to be sufficient. This is due to XFS's performance-oriented design. In practice, it does not become a problem since it only occurs if remaining space is only a few blocks.

Other XFS Features

The XFS file system also supports the following:

Extended attributes (xattr)

This allows the system to associate several additional name/value pairs per file. It is enabled by default.

Quota journaling

This avoids the need for lengthy quota consistency checks after a crash.

Project/directory quotas

This allows quota restrictions over a directory tree.

Subsecond timestamps

This allows timestamps to go to the subsecond.

Default `atime` behavior is `relatime`

`Relatime` is on by default for XFS. It has almost no overhead compared to **`noatime`** while still maintaining sane **`atime`** values.

3.1. CREATING AN XFS FILE SYSTEM

Prerequisites

- Create or reuse a partition on your disk. For information on creating MBR or GPT partitions, see Chapter 13, *Partitions*.

  Alternatively, use an LVM or MD volume or a similar layer below XFS.

Procedure

Procedure 3.1. Creating an XFS File System

- To create an XFS file system, use the following command:

  ```
  # mkfs.xfs block_device
  ```

  - Replace *block_device* with the path to a partition or a logical volume. For example, **/dev/sdb1**, **/dev/disk/by-uuid/05e99ec8-def1-4a5e-8a9d-5945339ceb2a**, or **/dev/my-volgroup/my-lv**.

  - In general, the default options are optimal for common use.

  - When using **mkfs.xfs** on a block device containing an existing file system, add the **-f** option to overwrite that file system.

Example 3.1. `mkfs.xfs` Command Output

Following is a sample output of the **mkfs.xfs** command:

```
meta-data=/dev/device            isize=256    agcount=4, agsize=3277258
blks
         =                       sectsz=512   attr=2
data     =                       bsize=4096   blocks=13109032,
imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0
log      =internal log           bsize=4096   blocks=6400, version=2
         =                       sectsz=512   sunit=0 blks, lazy-
count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

> **NOTE**
>
> After an XFS file system is created, its size cannot be reduced. However, it can still be enlarged using the **xfs_growfs** command. For more information, see Section 3.4, "Increasing the Size of an XFS File System").

Striped Block Devices

For striped block devices (for example, RAID5 arrays), the stripe geometry can be specified at the time of file system creation. Using proper stripe geometry greatly enhances the performance of an XFS filesystem.

When creating filesystems on LVM or MD volumes, **mkfs.xfs** chooses an optimal geometry. This may also be true on some hardware RAIDs that export geometry information to the operating system.

If the device exports stripe geometry information, the **mkfs** utility (for ext3, ext4, and xfs) will automatically use this geometry. If stripe geometry is not detected by the **mkfs** utility and even though the storage does, in fact, have stripe geometry, it is possible to manually specify it when creating the file system using the following options:

su=*value*

Specifies a stripe unit or RAID chunk size. The *value* must be specified in bytes, with an optional **k**, **m**, or **g** suffix.

sw=*value*

Specifies the number of data disks in a RAID device, or the number of stripe units in the stripe.

The following example specifies a chunk size of 64k on a RAID device containing 4 stripe units:

```
# mkfs.xfs -d su=64k,sw=4 /dev/block_device
```

Additional Resources

For more information about creating XFS file systems, see:

- The mkfs.xfs(8) man page

- The Red Hat Enterprise Linux Performance Tuning Guide, chapter Tuning XFS

3.2. MOUNTING AN XFS FILE SYSTEM

An XFS file system can be mounted with no extra options, for example:

```
# mount /dev/device /mount/point
```

The default for Red Hat Enterprise Linux 7 is inode64.

> **NOTE**
>
> Unlike **mke2fs**, **mkfs.xfs** does not utilize a configuration file; they are all specified on the command line.

Write Barriers

By default, XFS uses write barriers to ensure file system integrity even when power is lost to a device with write caches enabled. For devices without write caches, or with battery-backed write caches, disable the barriers by using the **nobarrier** option:

```
# mount -o nobarrier /dev/device /mount/point
```

For more information about write barriers, see Chapter 22, *Write Barriers*.

Direct Access Technology Preview

Since Red Hat Enterprise Linux 7.3, **Direct Access** (DAX) is available as a Technology Preview on the ext4 and XFS file systems. It is a means for an application to directly map persistent memory into its address space. To use DAX, a system must have some form of persistent memory available, usually in the form of one or more Non-Volatile Dual Inline Memory Modules (NVDIMMs), and a file system that supports DAX must be created on the NVDIMM(s). Also, the file system must be mounted with the **dax** mount option. Then, an **mmap** of a file on the dax-mounted file system results in a direct mapping of storage into the application's address space.

3.3. XFS QUOTA MANAGEMENT

The XFS quota subsystem manages limits on disk space (blocks) and file (inode) usage. XFS quotas control or report on usage of these items on a user, group, or directory or project level. Also, note that while user, group, and directory or project quotas are enabled independently, group and project quotas are mutually exclusive.

When managing on a per-directory or per-project basis, XFS manages the disk usage of directory hierarchies associated with a specific project. In doing so, XFS recognizes cross-organizational "group" boundaries between projects. This provides a level of control that is broader than what is available when managing quotas for users or groups.

XFS quotas are enabled at mount time, with specific mount options. Each mount option can also be specified as **noenforce**; this allows usage reporting without enforcing any limits. Valid quota mount options are:

- **uquota**/**uqnoenforce**: User quotas

- **gquota**/**gqnoenforce**: Group quotas

- **pquota**/**pqnoenforce**: Project quota

Once quotas are enabled, the **xfs_quota** tool can be used to set limits and report on disk usage. By default, **xfs_quota** is run interactively, and in *basic mode*. Basic mode subcommands simply report usage, and are available to all users. Basic **xfs_quota** subcommands include:

quota *username/userID*

Show usage and limits for the given *username* or numeric *userID*

df

Shows free and used counts for blocks and inodes.

In contrast, **xfs_quota** also has an *expert mode*. The subcommands of this mode allow actual configuration of limits, and are available only to users with elevated privileges. To use expert mode subcommands interactively, use the following command:

```
# xfs_quota -x
```

Expert mode subcommands include:

report */path*

Reports quota information for a specific file system.

limit

Modify quota limits.

For a complete list of subcommands for either basic or expert mode, use the subcommand **help**.

All subcommands can also be run directly from a command line using the **-c** option, with **-x** for expert subcommands.

> **Example 3.2. Display a Sample Quota Report**
>
> For example, to display a sample quota report for **/home** (on **/dev/blockdevice**), use the command **xfs_quota -x -c 'report -h' /home**. This displays output similar to the following:
>
> ```
> User quota on /home (/dev/blockdevice)
> Blocks
> User ID      Used    Soft    Hard Warn/Grace
> ---------- --------------------------------
> root            0       0       0  00 [------]
> testuser   103.4G       0       0  00 [------]
> ...
> ```
>
> To set a soft and hard inode count limit of 500 and 700 respectively for user **john**, whose home directory is **/home/john**, use the following command:
>
> ```
> # xfs_quota -x -c 'limit isoft=500 ihard=700 john' /home/
> ```
>
> In this case, pass *mount_point* which is the mounted xfs file system.

By default, the **limit** subcommand recognizes targets as users. When configuring the limits for a group, use the **-g** option (as in the previous example). Similarly, use **-p** for projects.

Soft and hard block limits can also be configured using **bsoft** or **bhard** instead of **isoft** or **ihard**.

> **Example 3.3. Set a Soft and Hard Block Limit**
>
> For example, to set a soft and hard block limit of 1000m and 1200m, respectively, to group **accounting** on the **/target/path** file system, use the following command:
>
> ```
> # xfs_quota -x -c 'limit -g bsoft=1000m bhard=1200m accounting'
> /target/path
> ```

NOTE

The commands **bsoft** and **bhard** count by the byte.

IMPORTANT

While real-time blocks (**rtbhard**/**rtbsoft**) are described in **man xfs_quota** as valid units when setting quotas, the real-time sub-volume is not enabled in this release. As such, the **rtbhard** and **rtbsoft** options are not applicable.

Setting Project Limits

Before configuring limits for project-controlled directories, add them first to **/etc/projects**. Project names can be added to **/etc/projectid** to map project IDs to project names. Once a project is added to **/etc/projects**, initialize its project directory using the following command:

```
# xfs_quota -x -c 'project -s projectname' project_path
```

Quotas for projects with initialized directories can then be configured, with:

```
# xfs_quota -x -c 'limit -p bsoft=1000m bhard=1200m projectname'
```

Generic quota configuration tools (**quota**, **repquota**, and **edquota** for example) may also be used to manipulate XFS quotas. However, these tools cannot be used with XFS project quotas.

IMPORTANT

Red Hat recommends the use of **xfs_quota** over all other available tools.

For more information about setting XFS quotas, see **man xfs_quota**, **man projid(5)**, and **man projects(5)**.

3.4. INCREASING THE SIZE OF AN XFS FILE SYSTEM

An XFS file system may be grown while mounted using the **xfs_growfs** command:

```
# xfs_growfs /mount/point -D size
```

The **-D size** option grows the file system to the specified *size* (expressed in file system blocks). Without the **-D size** option, **xfs_growfs** will grow the file system to the maximum size supported by the device.

Before growing an XFS file system with **-D size**, ensure that the underlying block device is of an appropriate size to hold the file system later. Use the appropriate resizing methods for the affected block device.

> **NOTE**
>
> While XFS file systems can be grown while mounted, their size cannot be reduced at all.

For more information about growing a file system, see **man xfs_growfs**.

3.5. REPAIRING AN XFS FILE SYSTEM

To repair an XFS file system, use **xfs_repair**:

```
# xfs_repair /dev/device
```

The **xfs_repair** utility is highly scalable and is designed to repair even very large file systems with many inodes efficiently. Unlike other Linux file systems, **xfs_repair** does not run at boot time, even when an XFS file system was not cleanly unmounted. In the event of an unclean unmount, **xfs_repair** simply replays the log at mount time, ensuring a consistent file system.

> **WARNING**
>
> The **xfs_repair** utility cannot repair an XFS file system with a dirty log. To clear the log, mount and unmount the XFS file system. If the log is corrupt and cannot be replayed, use the **-L** option ("force log zeroing") to clear the log, that is, **xfs_repair -L /dev/device**. Be aware that this may result in further corruption or data loss.

For more information about repairing an XFS file system, see **man xfs_repair**.

3.6. SUSPENDING AN XFS FILE SYSTEM

To suspend or resume write activity to a file system, use the following command:

```
# xfs_freeze mount-point
```

Suspending write activity allows hardware-based device snapshots to be used to capture the file system in a consistent state.

> **NOTE**
>
> The **xfs_freeze** utility is provided by the **xfsprogs** package, which is only available on x86_64.

To suspend (that is, freeze) an XFS file system, use:

```
# xfs_freeze -f /mount/point
```

To unfreeze an XFS file system, use:

```
# xfs_freeze -u /mount/point
```

When taking an LVM snapshot, it is not necessary to use **xfs_freeze** to suspend the file system first. Rather, the LVM management tools will automatically suspend the XFS file system before taking the snapshot.

For more information about freezing and unfreezing an XFS file system, see **man xfs_freeze**.

3.7. BACKING UP AND RESTORING XFS FILE SYSTEMS

XFS file system backup and restoration involve these utilities:

- **xfsdump** for creating the backup

- **xfsrestore** for restoring from backup

3.7.1. Features of XFS Backup and Restoration

Backup

You can use the **xfsdump** utility to:

- Perform backups to regular file images.

  Only one backup can be written to a regular file.

- Perform backups to tape drives.

  The **xfsdump** utility also allows you to write multiple backups to the same tape. A backup can span multiple tapes.

  To back up multiple file systems to a single tape device, simply write the backup to a tape that already contains an XFS backup. This appends the new backup to the previous one. By default, **xfsdump** never overwrites existing backups.

- Create incremental backups.

  The **xfsdump** utility uses *dump levels* to determine a base backup to which other backups are relative. Numbers from **0** to **9** refer to increasing dump levels. An incremental backup only backs up files that have changed since the last dump of a lower level:

  - To perform a full backup, perform a *level 0* dump on the file system.

  - A level 1 dump is the first incremental backup after a full backup. The next incremental backup would be level 2, which only backs up files that have changed since the last level 1 dump; and so on, to a maximum of level 9.

- Exclude files from a backup using size, subtree, or inode flags to filter them.

Restoration

The **xfsrestore** utility restores file systems from backups produced by **xfsdump**. The **xfsrestore** utility has two modes:

- The *simple* mode enables users to restore an entire file system from a level 0 dump. This is the default mode.

- The *cumulative* mode enables file system restoration from an incremental backup: that is, level 1 to level 9.

A unique *session ID* or *session label* identifies each backup. Restoring a backup from a tape containing multiple backups requires its corresponding session ID or label.

To extract, add, or delete specific files from a backup, enter the **xfsrestore** interactive mode. The interactive mode provides a set of commands to manipulate the backup files.

3.7.2. Backing Up an XFS File System

This procedure describes how to back up the content of an XFS file system into a file or a tape.

Procedure 3.2. Backing Up an XFS File System

- Use the following command to back up an XFS file system:

  ```
  # xfsdump -l level [-L label] -f backup-destination path-to-xfs-
  filesystem
  ```

  - Replace *level* with the dump level of your backup. Use **0** to perform a full backup or **1** to **9** to perform consequent incremental backups.

  - Replace *backup-destination* with the path where you want to store your backup. The destination can be a regular file, a tape drive, or a remote tape device. For example, **/backup-files/Data.xfsdump** for a file or **/dev/st0** for a tape drive.

  - Replace *path-to-xfs-filesystem* with the mount point of the XFS file system you want to back up. For example, **/mnt/data/**. The file system must be mounted.

  - When backing up multiple file systems and saving them on a single tape device, add a session label to each backup using the **-L label** option so that it is easier to identify them when restoring. Replace *label* with any name for your backup: for example, **backup_data**.

Example 3.4. Backing up Multiple XFS File Systems

- To back up the content of XFS file systems mounted on the **/boot/** and **/data/** directories and save them as files in the **/backup-files/** directory:

  ```
  # xfsdump -l 0 -f /backup-files/boot.xfsdump /boot
  # xfsdump -l 0 -f /backup-files/data.xfsdump /data
  ```

- To back up multiple file systems on a single tape device, add a session label to each backup using the **-L label** option:

  ```
  # xfsdump -l 0 -L "backup_boot" -f /dev/st0 /boot
  # xfsdump -l 0 -L "backup_data" -f /dev/st0 /data
  ```

Additional Resources

- For more information about backing up XFS file systems, see the xfsdump(8) man page.

3.7.3. Restoring an XFS File System from Backup

This procedure describes how to restore the content of an XFS file system from a file or tape backup.

Prerequisites

- You need a file or tape backup of XFS file systems, as described in Section 3.7.2, "Backing Up an XFS File System".

Procedure 3.3. Restoring an XFS File System from Backup

- The command to restore the backup varies depending on whether you are restoring from a full backup or an incremental one, or are restoring multiple backups from a single tape device:

  ```
  # xfsrestore [-r] [-S session-id] [-L session-label] [-i]
               -f backup-location restoration-path
  ```

  - Replace *backup-location* with the location of the backup. This can be a regular file, a tape drive, or a remote tape device. For example, **/backup-files/Data.xfsdump** for a file or **/dev/st0** for a tape drive.

  - Replace *restoration-path* with the path to the directory where you want to restore the file system. For example, **/mnt/data/**.

  - To restore a file system from an incremental (level 1 to level 9) backup, add the **-r** option.

  - To restore a backup from a tape device that contains multiple backups, specify the backup using the **-S** or **-L** options.

    The **-S** lets you choose a backup by its session ID, while the **-L** lets you choose by the session label. To obtain the session ID and session labels, use the **xfsrestore -I** command.

    Replace *session-id* with the session ID of the backup. For example, **b74a3586-e52e-4a4a-8775-c3334fa8ea2c**. Replace *session-label* with the session label of the backup. For example, **my_backup_session_label**.

  - To use **xfsrestore** interactively, use the **-i** option.

    The interactive dialog begins after **xfsrestore** finishes reading the specified device. Available commands in the interactive **xfsrestore** shell include **cd**, **ls**, **add**, **delete**, and **extract**; for a complete list of commands, use the **help** command.

Example 3.5. Restoring Multiple XFS File Systems

To restore the XFS backup files and save their content into directories under **/mnt/**:

```
# xfsrestore -f /backup-files/boot.xfsdump /mnt/boot/
# xfsrestore -f /backup-files/data.xfsdump /mnt/data/
```

To restore from a tape device containing multiple backups, specify each backup by its session label or session ID:

```
# xfsrestore -f /dev/st0 -L "backup_boot" /mnt/boot/
# xfsrestore -f /dev/st0 -S "45e9af35-efd2-4244-87bc-4762e476cbab"
/mnt/data/
```

Informational Messages When Restoring a Backup from a Tape

When restoring a backup from a tape with backups from multiple file systems, the **xfsrestore** utility might issue messages. The messages inform you whether a match of the requested backup has been found when **xfsrestore** examines each backup on the tape in sequential order. For example:

```
xfsrestore: preparing drive
xfsrestore: examining media file 0
xfsrestore: inventory session uuid (8590224e-3c93-469c-a311-fc8f23029b2a)
does not match the media header's session uuid (7eda9f86-f1e9-4dfd-b1d4-
c50467912408)
xfsrestore: examining media file 1
xfsrestore: inventory session uuid (8590224e-3c93-469c-a311-fc8f23029b2a)
does not match the media header's session uuid (7eda9f86-f1e9-4dfd-b1d4-
c50467912408)
[...]
```

The informational messages keep appearing until the matching backup is found.

Additional Resources

- For more information about restoring XFS file systems, see the xfsrestore(8) man page.

3.8. CONFIGURING ERROR BEHAVIOR

When an error occurs during an I/O operation, the XFS driver responds in one of two ways:

- Continue retries until either:

    - the I/O operation succeeds, or

    - an I/O operation retry count or time limit is exceeded.

- Consider the error permanent and halt the system.

XFS currently recognizes the following error conditions for which you can configure the desired behavior specifically:

- **EIO**: Error while trying to write to the device

- **ENOSPC**: No space left on the device

- **ENODEV**: Device cannot be found

All other possible error conditions, which do not have specific handlers defined, share a single, global configuration.

You can set the conditions under which XFS deems the errors permanent, both in the maximum number of retries and the maximum time in seconds. XFS stops retrying when any one of the conditions is met.

There is also an option to immediately cancel the retries when unmounting the file system, regardless of any other configuration. This allows the unmount operation to succeed even in case of persistent errors.

3.8.1. Configuration Files for Specific and Undefined Conditions

Configuration files controlling error behavior are located in the **/sys/fs/xfs/device/error/** directory.

The **/sys/fs/xfs/*device*/error/metadata/** directory contains subdirectories for each specific error condition:

- **/sys/fs/xfs/*device*/error/metadata/EIO/** for the **EIO** error condition

- **/sys/fs/xfs/*device*/error/metadata/ENODEV/** for the **ENODEV** error condition

- **/sys/fs/xfs/*device*/error/metadata/ENOSPC/** for the **ENOSPC** error condition

Each one then contains the following configuration files:

- **/sys/fs/xfs/*device*/error/metadata/*condition*/max_retries**: controls the maximum number of times that XFS retries the operation.

- **/sys/fs/xfs/*device*/error/metadata/*condition*/retry_timeout_seconds**: the time limit in seconds after which XFS will stop retrying the operation

All other possible error conditions, apart from those described in the previous section, share a common configuration in these files:

- **/sys/fs/xfs/*device*/error/default/max_retries**: controls the maximum number of retries

- **/sys/fs/xfs/*device*/error/default/retry_timeout_seconds**: controls the time limit for retrying

3.8.2. Setting File System Behavior for Specific and Undefined Conditions

To set the maximum number of retries, write the desired number to the **max_retries** file.

- For specific conditions:

  ```
  # echo value >
  /sys/fs/xfs/device/error/metadata/condition/max_retries
  ```

- For undefined conditions:

  ```
  # echo value > /sys/fs/xfs/device/error/default/max_retries
  ```

value is a number between **-1** and the maximum possible value of **int**, the C signed integer type. This is **2147483647** on 64-bit Linux.

To set the time limit, write the desired number of seconds to the **retry_timeout_seconds** file.

- For specific conditions:

```
# echo value >
/sys/fs/xfs/device/error/metadata/condition/retry_timeout_seconds
```

- For undefined conditions:

```
# echo value >
/sys/fs/xfs/device/error/default/retry_timeout_seconds
```

value is a number between **-1** and **86400**, which is the number of seconds in a day.

In both the **max_retries** and **retry_timeout_seconds** options, **-1** means to retry forever and **0** to stop immediately.

device is the name of the device, as found in the **/dev/** directory; for example, **sda**.

> **NOTE**
>
> The default behavior for a each error condition is dependent on the error context. Some errors, like **ENODEV**, are considered to be fatal and unrecoverable, regardless of the retry count, so their default value is **0**.

3.8.3. Setting Unmount Behavior

If the **fail_at_unmount** option is set, the file system overrides all other error configurations during unmount, and immediately umnounts the file system without retrying the I/O operation. This allows the unmount operation to succeed even in case of persistent errors.

To set the unmount behavior:

```
# echo value > /sys/fs/xfs/device/error/fail_at_unmount
```

value is either **1** or **0**:

- **1** means to cancel retrying immediately if an error is found.

- **0** means to respect the **max_retries** and **retry_timeout_seconds** options.

device is the name of the device, as found in the **/dev/** directory; for example, **sda**.

> **IMPORTANT**
>
> The **fail_at_unmount** option has to be set as desired before attempting to unmount the file system. After an unmount operation has started, the configuration files and directories may be unavailable.

3.9. OTHER XFS FILE SYSTEM UTILITIES

Red Hat Enterprise Linux 7 also features other utilities for managing XFS file systems:

xfs_fsr

Used to defragment mounted XFS file systems. When invoked with no arguments, `xfs_fsr` defragments all regular files in all mounted XFS file systems. This utility also allows users to suspend a defragmentation at a specified time and resume from where it left off later.

In addition, `xfs_fsr` also allows the defragmentation of only one file, as in `xfs_fsr /path/to/file`. Red Hat advises not to periodically defrag an entire file system because XFS avoids fragmentation by default. System wide defragmentation could cause the side effect of fragmentation in free space.

xfs_bmap

Prints the map of disk blocks used by files in an XFS filesystem. This map lists each extent used by a specified file, as well as regions in the file with no corresponding blocks (that is, holes).

xfs_info

Prints XFS file system information.

xfs_admin

Changes the parameters of an XFS file system. The `xfs_admin` utility can only modify parameters of unmounted devices or file systems.

xfs_copy

Copies the contents of an entire XFS file system to one or more targets in parallel.

The following utilities are also useful in debugging and analyzing XFS file systems:

xfs_metadump

Copies XFS file system metadata to a file. Red Hat only supports using the `xfs_metadump` utility to copy unmounted file systems or read-only mounted file systems; otherwise, generated dumps could be corrupted or inconsistent.

xfs_mdrestore

Restores an XFS metadump image (generated using `xfs_metadump`) to a file system image.

xfs_db

Debugs an XFS file system.

For more information about these utilities, see their respective **man** pages.

3.10. MIGRATING FROM EXT4 TO XFS

Starting with Red Hat Enterprise Linux 7.0, XFS is the default file system instead of ext4. This section highlights the differences when using or administering an XFS file system.

The ext4 file system is still fully supported in Red Hat Enterprise Linux 7 and can be selected at installation. While it is possible to migrate from ext4 to XFS, it is not required.

3.10.1. Differences Between Ext3/4 and XFS

File system repair

Ext3/4 runs **e2fsck** in userspace at boot time to recover the journal as needed. XFS, by comparison, performs journal recovery in kernelspace at mount time. An **fsck.xfs** shell script is provided but does not perform any useful action as it is only there to satisfy initscript requirements.

When an XFS file system repair or check is requested, use the **xfs_repair** command. Use the **-n** option for a read-only check.

The **xfs_repair** command will not operate on a file system with a dirty log. To repair such a file system **mount** and **unmount** must first be performed to replay the log. If the log is corrupt and cannot be replayed, the **-L** option can be used to zero out in the log.

For more information on file system repair of XFS file systems, see Section 12.2.2, "XFS"

Metadata error behavior

The ext3/4 file system has configurable behavior when metadata errors are encountered, with the default being to simply continue. When XFS encounters a metadata error that is not recoverable it will shut down the file system and return a **EFSCORRUPTED** error. The system logs will contain details of the error encountered and will recommend running **xfs_repair** if necessary.

Quotas

XFS quotas are not a remountable option. The **-o quota** option must be specified on the initial mount for quotas to be in effect.

While the standard tools in the quota package can perform basic quota administrative tasks (tools such as **setquota** and **repquota**), the **xfs_quota** tool can be used for XFS-specific features, such as Project Quota administration.

The **quotacheck** command has no effect on an XFS file system. The first time quota accounting is turned on XFS does an automatic **quotacheck** internally. Because XFS quota metadata is a first-class, journaled metadata object, the quota system will always be consistent until quotas are manually turned off.

File system resize

The XFS file system has no utility to shrink a file system. XFS file systems can be grown online via the **xfs_growfs** command.

Inode numbers

For file systems larger than 1 TB with 256-byte inodes, or larger than 2 TB with 512-byte inodes, XFS inode numbers might exceed 2^{32}. Such large inode numbers cause 32-bit stat calls to fail with the EOVERFLOW return value. The described problem might occur when using the default Red Hat Enterprise Linux 7 configuration: non-striped with four allocation groups. A custom configuration, for example file system extension or changing XFS file system parameters, might lead to a different behavior.

Applications usually handle such larger inode numbers correctly. If needed, mount the XFS file system with the **-o inode32** parameter to enforce inode numbers below 2^{32}. Note that using **inode32** does not affect inodes that are already allocated with 64-bit numbers.

IMPORTANT

Do *not* use the **inode32** option unless it is required by a specific environment. The **inode32** option changes allocation behavior. As a consequence, the ENOSPC error might occur if no space is available to allocate inodes in the lower disk blocks.

Speculative preallocation

XFS uses *speculative preallocation* to allocate blocks past EOF as files are written. This avoids file fragmentation due to concurrent streaming write workloads on NFS servers. By default, this preallocation increases with the size of the file and will be apparent in "du" output. If a file with speculative preallocation is not dirtied for five minutes the preallocation will be discarded. If the inode is cycled out of cache before that time, then the preallocation will be discarded when the inode is reclaimed.

If premature ENOSPC problems are seen due to speculative preallocation, a fixed preallocation amount may be specified with the **-o allocsize=*amount*** mount option.

Fragmentation-related tools

Fragmentation is rarely a significant issue on XFS file systems due to heuristics and behaviors, such as delayed allocation and speculative preallocation. However, tools exist for measuring file system fragmentation as well as defragmenting file systems. Their use is not encouraged.

The **xfs_db frag** command attempts to distill all file system allocations into a single fragmentation number, expressed as a percentage. The output of the command requires significant expertise to understand its meaning. For example, a fragmentation factor of 75% means only an average of 4 extents per file. For this reason the output of xfs_db's frag is not considered useful and more careful analysis of any fragmentation problems is recommended.

WARNING

The **xfs_fsr** command may be used to defragment individual files, or all files on a file system. The later is especially not recommended as it may destroy locality of files and may fragment free space.

Commands Used with ext3 and ext4 Compared to XFS

The following table compares common commands used with ext3 and ext4 to their XFS-specific counterparts.

Table 3.1. Common Commands for ext3 and ext4 Compared to XFS

Task	ext3/4	XFS
Create a file system	**mkfs.ext4** or **mkfs.ext3**	**mkfs.xfs**
File system check	**e2fsck**	**xfs_repair**

Task	ext3/4	XFS
Resizing a file system	`resize2fs`	`xfs_growfs`
Save an image of a file system	`e2image`	`xfs_metadump` and `xfs_mdrestore`
Label or tune a file system	`tune2fs`	`xfs_admin`
Backup a file system	`dump` and `restore`	`xfsdump` and `xfsrestore`

The following table lists generic tools that function on XFS file systems as well, but the XFS versions have more specific functionality and as such are recommended.

Table 3.2. Generic Tools for ext4 and XFS

Task	ext4	XFS
Quota	`quota`	`xfs_quota`
File mapping	`filefrag`	`xfs_bmap`

More information on many the listed XFS commands is included in Chapter 3, *The XFS File System*. You can also consult the manual pages of the listed XFS administration tools for more information.

CHAPTER 4. THE EXT3 FILE SYSTEM

The ext3 file system is essentially an enhanced version of the ext2 file system. These improvements provide the following advantages:

Availability

After an unexpected power failure or system crash (also called an *unclean system shutdown*), each mounted ext2 file system on the machine must be checked for consistency by the **e2fsck** program. This is a time-consuming process that can delay system boot time significantly, especially with large volumes containing a large number of files. During this time, any data on the volumes is unreachable.

It is possible to run **fsck -n** on a live filesystem. However, it will not make any changes and may give misleading results if partially written metadata is encountered.

If LVM is used in the stack, another option is to take an LVM snapshot of the filesystem and run **fsck** on it instead.

Finally, there is the option to remount the filesystem as read only. All pending metadata updates (and writes) are then forced to the disk prior to the remount. This ensures the filesystem is in a consistent state, provided there is no previous corruption. It is now possible to run **fsck -n**.

The journaling provided by the ext3 file system means that this sort of file system check is no longer necessary after an unclean system shutdown. The only time a consistency check occurs using ext3 is in certain rare hardware failure cases, such as hard drive failures. The time to recover an ext3 file system after an unclean system shutdown does not depend on the size of the file system or the number of files; rather, it depends on the size of the *journal* used to maintain consistency. The default journal size takes about a second to recover, depending on the speed of the hardware.

> **NOTE**
>
> The only journaling mode in ext3 supported by Red Hat is **data=ordered** (default).

Data Integrity

The ext3 file system prevents loss of data integrity in the event that an unclean system shutdown occurs. The ext3 file system allows you to choose the type and level of protection that your data receives. With regard to the state of the file system, ext3 volumes are configured to keep a high level of data consistency by default.

Speed

Despite writing some data more than once, ext3 has a higher throughput in most cases than ext2 because ext3's journaling optimizes hard drive head motion. You can choose from three journaling modes to optimize speed, but doing so means trade-offs in regards to data integrity if the system was to fail.

> **NOTE**
>
> The only journaling mode in ext3 supported by Red Hat is **data=ordered** (default).

Easy Transition

It is easy to migrate from ext2 to ext3 and gain the benefits of a robust journaling file system without reformatting. For more information on performing this task, see Section 4.2, "Converting to an ext3 File System" .

> **NOTE**
>
> Red Hat Enterprise Linux 7 provides a unified extN driver. It does this by disabling the ext2 and ext3 configurations and instead uses **ext4.ko** for these on-disk formats. This means that kernel messages will always refer to ext4 regardless of the ext file system used.

4.1. CREATING AN EXT3 FILE SYSTEM

After installation, it is sometimes necessary to create a new ext3 file system. For example, if a new disk drive is added to the system, you may want to partition the drive and use the ext3 file system.

Prerequisites

- Create or reuse a partition on your disk. For information on creating MBR or GPT partitions, see Chapter 13, *Partitions*.

  Alternatively, use an LVM or MD volume or a similar layer below ext3.

Procedure

Procedure 4.1. Creating an ext3 File System

1. Format the partition or LVM volume with the ext3 file system using the **mkfs.ext3** utility:

   ```
   # mkfs.ext3 block_device
   ```

   - Replace *block_device* with the path to a partition or a logical volume. For example, **/dev/sdb1**, **/dev/disk/by-uuid/05e99ec8-def1-4a5e-8a9d-5945339ceb2a**, or **/dev/my-volgroup/my-lv**.

2. Label the file system using the **e2label** utility:

   ```
   # e2label block_device volume_label
   ```

Configuring UUID

It is also possible to set a specific UUID for a file system. To specify a UUID when creating a file system, use the **-U** option:

```
# mkfs.ext3 -U UUID device
```

- Replace *UUID* with the UUID you want to set: for example, **7cd65de3-e0be-41d9-b66d-96d749c02da7**.

- Replace *device* with the path to an ext3 file system to have the UUID added to it: for example, **/dev/sda8**.

To change the UUID of an existing file system, see Section 25.7.3.2, "Modifying Persistent Naming Attributes"

Additional Resources

- The mkfs.ext3(8) man page

- The e2label(8) man page

4.2. CONVERTING TO AN EXT3 FILE SYSTEM

The **tune2fs** command converts an **ext2** file system to **ext3**.

> **NOTE**
>
> To convert ext2 to ext3, always use the **e2fsck** utility to check your file system before and after using **tune2fs**. Before trying to convert ext2 to ext3, back up all file systems in case any errors occur.
>
> In addition, Red Hat recommends creating a new ext3 file system and migrating data to it, instead of converting from ext2 to ext3 whenever possible.

To convert an **ext2** file system to **ext3**, log in as root and type the following command in a terminal:

```
# tune2fs -j block_device
```

block_device contains the ext2 file system to be converted.

Issue the **df** command to display mounted file systems.

4.3. REVERTING TO AN EXT2 FILE SYSTEM

In order to revert to an ext2 file system, use the following procedure.

For simplicity, the sample commands in this section use the following value for the block device:

/dev/mapper/VolGroup00-LogVol02

Procedure 4.2. Revert from ext3 to ext2

1. Unmount the partition by logging in as root and typing:

   ```
   # umount /dev/mapper/VolGroup00-LogVol02
   ```

2. Change the file system type to ext2 by typing the following command:

   ```
   # tune2fs -O ^has_journal /dev/mapper/VolGroup00-LogVol02
   ```

3. Check the partition for errors by typing the following command:

   ```
   # e2fsck -y /dev/mapper/VolGroup00-LogVol02
   ```

4. Then mount the partition again as ext2 file system by typing:

```
# mount -t ext2 /dev/mapper/VolGroup00-LogVol02 /mount/point
```

Replace */mount/point* with the mount point of the partition.

NOTE

If a `.journal` file exists at the root level of the partition, delete it.

To permanently change the partition to ext2, remember to update the **/etc/fstab** file, otherwise it will revert back after booting.

CHAPTER 5. THE EXT4 FILE SYSTEM

The ext4 file system is a scalable extension of the ext3 file system. With Red Hat Enterprise Linux 7, it can support a maximum individual file size of 16 terabytes, and file systems to a maximum of 50 terabytes, unlike Red Hat Enterprise Linux 6 which only supported file systems up to 16 terabytes. It also supports an unlimited number of sub-directories (the ext3 file system only supports up to 32,000), though once the link count exceeds 65,000 it resets to 1 and is no longer increased. The bigalloc feature is not currently supported.

> **NOTE**
>
> As with ext3, an ext4 volume must be umounted in order to perform an **fsck**. For more information, see Chapter 4, *The ext3 File System*.

Main Features

The ext4 file system uses extents (as opposed to the traditional block mapping scheme used by ext2 and ext3), which improves performance when using large files and reduces metadata overhead for large files. In addition, ext4 also labels unallocated block groups and inode table sections accordingly, which allows them to be skipped during a file system check. This makes for quicker file system checks, which becomes more beneficial as the file system grows in size.

Allocation Features

The ext4 file system features the following allocation schemes:

- Persistent pre-allocation

- Delayed allocation

- Multi-block allocation

- Stripe-aware allocation

Because of delayed allocation and other performance optimizations, ext4's behavior of writing files to disk is different from ext3. In ext4, when a program writes to the file system, it is not guaranteed to be on-disk unless the program issues an **fsync()** call afterwards.

By default, ext3 automatically forces newly created files to disk almost immediately even without **fsync()**. This behavior hid bugs in programs that did not use **fsync()** to ensure that written data was on-disk. The ext4 file system, on the other hand, often waits several seconds to write out changes to disk, allowing it to combine and reorder writes for better disk performance than ext3.

> **WARNING**
>
> Unlike ext3, the ext4 file system does not force data to disk on transaction commit. As such, it takes longer for buffered writes to be flushed to disk. As with any file system, use data integrity calls such as **fsync()** to ensure that data is written to permanent storage.

Other ext4 Features

The ext4 file system also supports the following:

- *Extended attributes* (`xattr`) — This allows the system to associate several additional name and value pairs per file.

- *Quota journaling* — This avoids the need for lengthy quota consistency checks after a crash.

  > **NOTE**
  >
  > The only supported journaling mode in ext4 is `data=ordered` (default).

- *Subsecond timestamps* — This gives timestamps to the subsecond.

5.1. CREATING AN EXT4 FILE SYSTEM

Prerequisites

- Create or reuse a partition on your disk. For information on creating MBR or GPT partitions, see Chapter 13, *Partitions*.

  Alternatively, use an LVM or MD volume or a similar layer below ext4.

Procedure

Procedure 5.1. Creating an ext4 File System

- To create an ext4 file system, use the following command:

  ```
  # mkfs.ext4 block_device
  ```

  - Replace *block_device* with the path to a partition or a logical volume. For example, **/dev/sdb1**, **/dev/disk/by-uuid/05e99ec8-def1-4a5e-8a9d-5945339ceb2a**, or **/dev/my-volgroup/my-lv**.

  - In general, the default options are optimal for most usage scenarios.

Example 5.1. `mkfs.ext4` Command Output

Below is a sample output of this command, which displays the resulting file system geometry and features:

```
~]# mkfs.ext4 /dev/sdb1
mke2fs 1.41.12 (17-May-2010)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
245280 inodes, 979456 blocks
48972 blocks (5.00%) reserved for the super user
First data block=0
```

```
Maximum filesystem blocks=1006632960
30 block groups
32768 blocks per group, 32768 fragments per group
8176 inodes per group
Superblock backups stored on blocks:
 32768, 98304, 163840, 229376, 294912, 819200, 884736

Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
```

IMPORTANT

It is possible to use **tune2fs** to enable certain ext4 features on ext3 file systems. However, using **tune2fs** in this way has not been fully tested and is therefore *not* supported in Red Hat Enterprise Linux 7. As a result, Red Hat cannot guarantee consistent performance and predictable behavior for ext3 file systems converted or mounted by using **tune2fs**.

Striped Block Devices

For striped block devices (for example, RAID5 arrays), the stripe geometry can be specified at the time of file system creation. Using proper stripe geometry greatly enhances the performance of an ext4 file system.

When creating file systems on LVM or MD volumes, **mkfs.ext4** chooses an optimal geometry. This may also be true on some hardware RAIDs which export geometry information to the operating system.

To specify stripe geometry, use the **-E** option of **mkfs.ext4** (that is, extended file system options) with the following sub-options:

stride=*value*

Specifies the RAID chunk size.

stripe-width=*value*

Specifies the number of data disks in a RAID device, or the number of stripe units in the stripe.

For both sub-options, *value* must be specified in file system block units. For example, to create a file system with a 64k stride (that is, 16 x 4096) on a 4k-block file system, use the following command:

```
# mkfs.ext4 -E stride=16,stripe-width=64 /dev/block_device
```

Configuring UUID

It is also possible to set a specific UUID for a file system. To specify a UUID when creating a file system, use the **-U** option:

```
# mkfs.ext4 -U UUID device
```

- Replace *UUID* with the UUID you want to set: for example, **7cd65de3-e0be-41d9-b66d-96d749c02da7**.

- Replace *device* with the path to an ext4 file system to have the UUID added to it: for example, **/dev/sda8**.

To change the UUID of an existing file system, see Section 25.7.3.2, "Modifying Persistent Naming Attributes"

Additional Resources

For more information about creating ext4 file systems, see:

- The mkfs.ext4(8) man page

5.2. MOUNTING AN EXT4 FILE SYSTEM

An ext4 file system can be mounted with no extra options. For example:

```
# mount /dev/device /mount/point
```

The ext4 file system also supports several mount options to influence behavior. For example, the **acl** parameter enables access control lists, while the **user_xattr** parameter enables user extended attributes. To enable both options, use their respective parameters with **-o**, as in:

```
# mount -o acl,user_xattr /dev/device /mount/point
```

As with ext3, the option **data_err=abort** can be used to abort the journal if an error occurs in file data.

```
# mount -o data_err=abort /dev/device /mount/point
```

The **tune2fs** utility also allows administrators to set default mount options in the file system superblock. For more information on this, refer to **man tune2fs**.

Write Barriers

By default, ext4 uses write barriers to ensure file system integrity even when power is lost to a device with write caches enabled. For devices without write caches, or with battery-backed write caches, disable barriers using the **nobarrier** option, as in:

```
# mount -o nobarrier /dev/device /mount/point
```

For more information about write barriers, refer to Chapter 22, *Write Barriers*.

Direct Access Technology Preview

Starting with Red Hat Enterprise Linux 7.3, **Direct Access** (DAX) provides, as a Technology Preview on the ext4 and XFS file systems, a means for an application to directly map persistent memory into its address space. To use DAX, a system must have some form of persistent memory available, usually in the form of one or more Non-Volatile Dual In-line Memory Modules (NVDIMMs), and a file system that supports DAX must be created on the NVDIMM(s). Also, the file system must be mounted with the **dax** mount option. Then, an **mmap** of a file on the dax-mounted file system results in a direct mapping of storage into the application's address space.

5.3. RESIZING AN EXT4 FILE SYSTEM

Before growing an ext4 file system, ensure that the underlying block device is of an appropriate size to hold the file system later. Use the appropriate resizing methods for the affected block device.

An ext4 file system may be grown while mounted using the **resize2fs** command:

```
# resize2fs /mount/device size
```

The **resize2fs** command can also decrease the size of an *unmounted* ext4 file system:

```
# resize2fs /dev/device size
```

When resizing an ext4 file system, the **resize2fs** utility reads the size in units of file system block size, unless a suffix indicating a specific unit is used. The following suffixes indicate specific units:

- **s** — 512 byte sectors

- **K** — kilobytes

- **M** — megabytes

- **G** — gigabytes

NOTE

The size parameter is optional (and often redundant) when expanding. The **resize2fs** automatically expands to fill all available space of the container, usually a logical volume or partition.

For more information about resizing an ext4 file system, refer to **man resize2fs**.

5.4. BACKING UP EXT2, EXT3, OR EXT4 FILE SYSTEMS

This procedure describes how to back up the content of an ext4, ext3, or ext2 file system into a file.

Prerequisites

- If the system has been running for a long time, run the **e2fsck** utility on the partitions before backup:

```
# e2fsck /dev/device
```

Procedure 5.2. Backing up ext2, ext3, or ext4 File Systems

1. Back up configuration information, including the content of the **/etc/fstab** file and the output of the **fdisk -l** command. This is useful for restoring the partitions.

   To capture this information, run the **sosreport** or **sysreport** utilities. For more information about **sosreport**, see the What is a sosreport and how to create one in Red Hat Enterprise Linux 4.6 and later? Kdowledgebase article.

2. Depending on the role of the partition:

- If the partition you are backing up is an operating system partition, boot your system into the rescue mode. See the Booting to Rescue Mode section of the *System Administrator's Guide*.

- When backing up a regular, data partition, unmount it.

  Although it is possible to back up a data partition while it is mounted, the results of backing up a mounted data partition can be unpredictable.

  If you need to back up a mounted file system using the **dump** utility, do so when the file system is not under a heavy load. The more activity is happening on the file system when backing up, the higher the risk of backup corruption is.

3. Use the **dump** utility to back up the content of the partitions:

```
# dump -0uf backup-file /dev/device
```

Replace *backup-file* with a path to a file where you want the to store the backup. Replace*device* with the name of the ext4 partition you want to back up. Make sure that you are saving the backup to a directory mounted on a different partition than the partition you are backing up.

Example 5.2. Backing up Multiple ext4 Partitions

To back up the content of the **/dev/sda1**, **/dev/sda2**, and **/dev/sda3** partitions into backup files stored in the **/backup-files/** directory, use the following commands:

```
# dump -0uf /backup-files/sda1.dump /dev/sda1
# dump -0uf /backup-files/sda2.dump /dev/sda2
# dump -0uf /backup-files/sda3.dump /dev/sda3
```

To do a remote backup, use the **ssh** utility or configure a password-less **ssh** login. For more information on **ssh** and password-less login, see the Using the ssh Utility and Using Key-based Authentication sections of the *System Administrator's Guide*.

For example, when using **ssh**:

Example 5.3. Performing a Remote Backup Using ssh

```
# dump -0u -f - /dev/device | ssh root@remoteserver.example.com dd
of=backup-file
```

Note that if using standard redirection, you must pass the **-f** option separately.

Additional Resources

- For more information, see the dump(8) man page.

5.5. RESTORING EXT2, EXT3, OR EXT4 FILE SYSTEMS

This procedure describes how to restore an ext4, ext3, or ext2 file system from a file backup.

Prerequisites

- You need a backup of partitions and their metadata, as described in Section 5.4, "Backing up ext2, ext3, or ext4 File Systems".

Procedure 5.3. Restoring ext2, ext3, or ext4 File Systems

1. If you are restoring an operating system partition, boot your system into Rescue Mode. See the Booting to Rescue Mode section of the *System Administrator's Guide*.

   This step is not required for ordinary data partitions.

2. Rebuild the partitions you want to restore by using the **fdisk** or **parted** utilites.

   If the partitions no longer exist, recreate them. The new partitions must be large enough to contain the restored data. It is important to get the start and end numbers right; these are the starting and ending sector numbers of the partitions obtained from the **fdisk** utility when backing up.

   For more information on modifying partitions, see Chapter 13, *Partitions*

3. Use the **mkfs** utility to format the destination partition:

   ```
   # mkfs.ext4 /dev/device
   ```

   IMPORTANT

   Do not format the partition that stores your backup files.

4. If you created new partitions, re-label all the partitions so they match their entries in the **/etc/fstab** file:

   ```
   # e2label /dev/device label
   ```

5. Create temporary mount points and mount the partitions on them:

   ```
   # mkdir /mnt/device
   # mount -t ext4 /dev/device /mnt/device
   ```

6. Restore the data from backup on the mounted partition:

   ```
   # cd /mnt/device
   # restore -rf device-backup-file
   ```

   If you want to restore on a remote machine or restore from a backup file that is stored on a remote host, you can use the **ssh** utility. For more information on **ssh**, see the Using the ssh Utility section of the *System Administrator's Guide*.

   Note that you need to configure a password-less login for the following commands. For more information on setting up a password-less **ssh** login, see the Using Key-based Authentication section of the *System Administrator's Guide*.

   - To restore a partition on a remote machine from a backup file stored on the same machine:

   ```
   # ssh remote-address "cd /mnt/device && cat backup-file |
   ```

```
/usr/sbin/restore -r -f -"
```

- To restore a partition on a remote machine from a backup file stored on a different remote machine:

```
# ssh remote-machine-1 "cd /mnt/device && RSH=/usr/bin/ssh
/usr/sbin/restore -rf remote-machine-2:backup-file"
```

7. Reboot:

```
# systemctl reboot
```

Example 5.4. Restoring Multiple ext4 Partitions

To restore the **/dev/sda1**, **/dev/sda2**, and **/dev/sda3** partitions from Example 5.2, "Backing up Multiple ext4 Partitions":

1. Rebuild partitions you want to restore by using the **fdisk** command.

2. Format the destination partitions:

```
# mkfs.ext4 /dev/sda1
# mkfs.ext4 /dev/sda2
# mkfs.ext4 /dev/sda3
```

3. Re-label all the partitions so they match the **/etc/fstab** file:

```
# e2label /dev/sda1 Boot1
# e2label /dev/sda2 Root
# e2label /dev/sda3 Data
```

4. Prepare the working directories.

   Mount the new partitions:

```
# mkdir /mnt/sda1
# mount -t ext4 /dev/sda1 /mnt/sda1
# mkdir /mnt/sda2
# mount -t ext4 /dev/sda2 /mnt/sda2
# mkdir /mnt/sda3
# mount -t ext4 /dev/sda3 /mnt/sda3
```

   Mount the partition that contains backup files:

```
# mkdir /backup-files
# mount -t ext4 /dev/sda6 /backup-files
```

5. Restore the data from backup to the mounted partitions:

```
# cd /mnt/sda1
# restore -rf /backup-files/sda1.dump
# cd /mnt/sda2
```

```
# restore -rf /backup-files/sda2.dump
# cd /mnt/sda3
# restore -rf /backup-files/sda3.dump
```

6. Reboot:

```
# systemctl reboot
```

Additional Resources

- For more information, see the restore(8) man page.

5.6. OTHER EXT4 FILE SYSTEM UTILITIES

Red Hat Enterprise Linux 7 also features other utilities for managing ext4 file systems:

e2fsck

Used to repair an ext4 file system. This tool checks and repairs an ext4 file system more efficiently than ext3, thanks to updates in the ext4 disk structure.

e2label

Changes the label on an ext4 file system. This tool also works on ext2 and ext3 file systems.

quota

Controls and reports on disk space (blocks) and file (inode) usage by users and groups on an ext4 file system. For more information on using `quota`, refer to `man quota` and Section 17.1, "Configuring Disk Quotas".

fsfreeze

To suspend access to a file system, use the command # `fsfreeze -f *mount-point*` to freeze it and # `fsfreeze -u *mount-point*` to unfreeze it. This halts access to the file system and creates a stable image on disk.
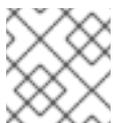
> **NOTE**
>
> It is unnecessary to use `fsfreeze` for device-mapper drives.

For more information see the `fsfreeze(8)` manpage.

As demonstrated in Section 5.2, "Mounting an ext4 File System", the **tune2fs** utility can also adjust configurable file system parameters for ext2, ext3, and ext4 file systems. In addition, the following tools are also useful in debugging and analyzing ext4 file systems:

debugfs

Debugs ext2, ext3, or ext4 file systems.

e2image

Saves critical ext2, ext3, or ext4 file system metadata to a file.

For more information about these utilities, refer to their respective **man** pages.

CHAPTER 6. BTRFS (TECHNOLOGY PREVIEW)

NOTE

Btrfs is available as a Technology Preview feature in Red Hat Enterprise Linux 7 but has been deprecated since the Red Hat Enterprise Linux 7.4 release. It will be removed in a future major release of Red Hat Enterprise Linux.

For more information, see Deprecated Functionality in the Red Hat Enterprise Linux 7.4 Release Notes.

Btrfs is a next generation Linux file system that offers advanced management, reliability, and scalability features. It is unique in offering snapshots, compression, and integrated device management.

6.1. CREATING A BTRFS FILE SYSTEM

In order to make a basic btrfs file system, use the following command:

```
# mkfs.btrfs /dev/device
```

For more information on creating btrfs file systems with added devices and specifying multi-device profiles for metadata and data, refer to Section 6.4, "Integrated Volume Management of Multiple Devices".

6.2. MOUNTING A BTRFS FILE SYSTEM

To mount any device in the btrfs file system use the following command:

```
# mount /dev/device /mount-point
```

Other useful mount options include:

device=/*dev*/*name*

Appending this option to the mount command tells btrfs to scan the named device for a btrfs volume. This is used to ensure the mount will succeed as attempting to mount devices that are not btrfs will cause the mount to fail.

NOTE

This does not mean all devices will be added to the file system, it only scans them.

max_inline=*number*

Use this option to set the maximum amount of space (in bytes) that can be used to inline data within a metadata B-tree leaf. The default is 8192 bytes. For 4k pages it is limited to 3900 bytes due to additional headers that need to fit into the leaf.

alloc_start=*number*

Use this option to set where in the disk allocations start.

thread_pool=*number*

Use this option to assign the number of worker threads allocated.

discard

Use this option to enable discard/TRIM on freed blocks.

noacl

Use this option to disable the use of ACL's.

space_cache

Use this option to store the free space data on disk to make caching a block group faster. This is a persistent change and is safe to boot into old kernels.

nospace_cache

Use this option to disable the above `space_cache`.

clear_cache

Use this option to clear all the free space caches during mount. This is a safe option but will trigger the space cache to be rebuilt. As such, leave the file system mounted in order to let the rebuild process finish. This mount option is intended to be used once and only after problems are apparent with the free space.

enospc_debug

This option is used to debug problems with "no space left".

recovery

Use this option to enable autorecovery upon mount.

6.3. RESIZING A BTRFS FILE SYSTEM

It is not possible to resize a btrfs file system but it is possible to resize each of the devices it uses. If there is only one device in use then this works the same as resizing the file system. If there are multiple devices in use then they must be manually resized to achieve the desired result.

> **NOTE**
>
> The unit size is not case specific; it accepts both **G** or **g** for GiB.
>
> The command does not accept **t** for terabytes or **p** for petabytes. It only accepts **k**, **m**, and **g**.

Enlarging a btrfs File System

To enlarge the file system on a single device, use the command:

```
# btrfs filesystem resize amount /mount-point
```

For example:

```
# btrfs filesystem resize +200M /btrfssingle
Resize '/btrfssingle' of '+200M'
```

To enlarge a multi-device file system, the device to be enlarged must be specified. First, show all devices that have a btrfs file system at a specified mount point:

```
# btrfs filesystem show /mount-point
```

For example:

```
# btrfs filesystem show /btrfstest
Label: none  uuid: 755b41b7-7a20-4a24-abb3-45fdbed1ab39
 Total devices 4 FS bytes used 192.00KiB
  devid    1 size 1.00GiB used 224.75MiB path /dev/vdc
  devid    2 size 524.00MiB used 204.75MiB path /dev/vdd
  devid    3 size 1.00GiB used 8.00MiB path /dev/vde
  devid    4 size 1.00GiB used 8.00MiB path /dev/vdf

Btrfs v3.16.2
```

Then, after identifying the **devid** of the device to be enlarged, use the following command:

```
# btrfs filesystem resize devid:amount /mount-point
```

For example:

```
# btrfs filesystem resize 2:+200M /btrfstest
Resize '/btrfstest/' of '2:+200M'
```

> **NOTE**
>
> The *amount* can also be **max** instead of a specified amount. This will use all remaining free space on the device.

Shrinking a btrfs File System

To shrink the file system on a single device, use the command:

```
# btrfs filesystem resize amount /mount-point
```

For example:

```
# btrfs filesystem resize -200M /btrfssingle
Resize '/btrfssingle' of '-200M'
```

To shrink a multi-device file system, the device to be shrunk must be specified. First, show all devices that have a btrfs file system at a specified mount point:

```
# btrfs filesystem show /mount-point
```

For example:

```
# btrfs filesystem show /btrfstest
Label: none  uuid: 755b41b7-7a20-4a24-abb3-45fdbed1ab39
 Total devices 4 FS bytes used 192.00KiB
 devid    1 size 1.00GiB used 224.75MiB path /dev/vdc
 devid    2 size 524.00MiB used 204.75MiB path /dev/vdd
 devid    3 size 1.00GiB used 8.00MiB path /dev/vde
 devid    4 size 1.00GiB used 8.00MiB path /dev/vdf

Btrfs v3.16.2
```

Then, after identifying the **devid** of the device to be shrunk, use the following command:

```
# btrfs filesystem resize devid:amount /mount-point
```

For example:

```
# btrfs filesystem resize 2:-200M /btrfstest
Resize '/btrfstest' of '2:-200M'
```

Set the File System Size

To set the file system to a specific size on a single device, use the command:

```
# btrfs filesystem resize amount /mount-point
```

For example:

```
# btrfs filesystem resize 700M /btrfssingle
Resize '/btrfssingle' of '700M'
```

To set the file system size of a multi-device file system, the device to be changed must be specified. First, show all devices that have a btrfs file system at the specified mount point:

```
# btrfs filesystem show /mount-point
```

For example:

```
# btrfs filesystem show /btrfstest
Label: none  uuid: 755b41b7-7a20-4a24-abb3-45fdbed1ab39
 Total devices 4 FS bytes used 192.00KiB
 devid    1 size 1.00GiB used 224.75MiB path /dev/vdc
 devid    2 size 724.00MiB used 204.75MiB path /dev/vdd
 devid    3 size 1.00GiB used 8.00MiB path /dev/vde
 devid    4 size 1.00GiB used 8.00MiB path /dev/vdf

Btrfs v3.16.2
```

Then, after identifying the **devid** of the device to be changed, use the following command:

```
# btrfs filesystem resize devid:amount /mount-point
```

For example:

```
# btrfs filesystem resize 2:300M /btrfstest
Resize '/btrfstest' of '2:300M'
```

6.4. INTEGRATED VOLUME MANAGEMENT OF MULTIPLE DEVICES

A btrfs file system can be created on top of many devices, and more devices can be added after the file system has been created. By default, metadata will be mirrored across two devices and data will be striped across all devices present, however if only one device is present, metadata will be duplicated on that device.

6.4.1. Creating a File System with Multiple Devices

The `mkfs.btrfs` command, as detailed in Section 6.1, "Creating a btrfs File System", accepts the options `-d` for data, and `-m` for metadata. Valid specifications are:

- `raid0`

- `raid1`

- `raid10`

- `dup`

- `single`

The `-m single` option instructs that no duplication of metadata is done. This may be desired when using hardware raid.

> **NOTE**
>
> RAID 10 requires at least four devices to run correctly.

Example 6.1. Creating a RAID 10 btrfs File System

Create a file system across four devices (metadata mirrored, data striped).

```
# mkfs.btrfs /dev/device1 /dev/device2 /dev/device3 /dev/device4
```

Stripe the metadata without mirroring.

```
# mkfs.btrfs -m raid0 /dev/device1 /dev/device2
```

Use raid10 for both data and metadata.

```
# mkfs.btrfs -m raid10 -d raid10 /dev/device1 /dev/device2 /dev/device3
/dev/device4
```

Do not duplicate metadata on a single drive.

```
# mkfs.btrfs -m single /dev/device
```

Use the **single** option to use the full capacity of each drive when the drives are different sizes.

```
# mkfs.btrfs -d single /dev/device1 /dev/device2 /dev/device3
```

To add a new device to an already created multi-device file system, use the following command:

```
# btrfs device add /dev/device1 /mount-point
```

After rebooting or reloading the btrfs module, use the **btrfs device scan** command to discover all multi-device file systems. See Section 6.4.2, "Scanning for btrfs Devices" for more information.

6.4.2. Scanning for btrfs Devices

Use **btrfs device scan** to scan all block devices under **/dev** and probe for btrfs volumes. This must be performed after loading the btrfs module if running with more than one device in a file system.

To scan all devices, use the following command:

```
# btrfs device scan
```

To scan a single device, use the following command:

```
# btrfs device scan /dev/device
```

6.4.3. Adding New Devices to a btrfs File System

Use the **btrfs filesystem show** command to list all the btrfs file systems and which devices they include.

The **btrfs device add** command is used to add new devices to a mounted file system.

The **btrfs filesystem balance** command balances (restripes) the allocated extents across all existing devices.

An example of all these commands together to add a new device is as follows:

Example 6.2. Add a New Device to a btrfs File System

First, create and mount a btrfs file system. Refer to Section 6.1, "Creating a btrfs File System" for more information on how to create a btrfs file system, and to Section 6.2, "Mounting a btrfs file system" for more information on how to mount a btrfs file system.

```
# mkfs.btrfs /dev/device1
# mount /dev/device1
```

Next, add a second device to the mounted btrfs file system.

```
# btrfs device add /dev/device2 /mount-point
```

The metadata and data on these devices are still stored only on **/dev/device1**. It must now be balanced to spread across all devices.

```
# btrfs filesystem balance /mount-point
```

Balancing a file system will take some time as it reads all of the file system's data and metadata and rewrites it across the new device.

6.4.4. Converting a btrfs File System

To convert a non-raid file system to a raid, add a device and run a balance filter that changes the chunk allocation profile.

Example 6.3. Converting a btrfs File System

To convert an existing single device system, **/dev/sdb1** in this case, into a two device, raid1 system in order to protect against a single disk failure, use the following commands:

```
# mount /dev/sdb1 /mnt
# btrfs device add /dev/sdc1 /mnt
# btrfs balance start -dconvert=raid1 -mconvert=raid1 /mnt
```
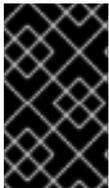
IMPORTANT

If the metadata is not converted from the single-device default, it remains as DUP. This does not guarantee that copies of the block are on separate devices. If data is not converted it does not have any redundant copies at all.

6.4.5. Removing btrfs Devices

Use the **btrfs device delete** command to remove an online device. It redistributes any extents in use to other devices in the file system in order to be safely removed.

Example 6.4. Removing a Device on a btrfs File System

First create and mount a few btrfs file systems.

```
# mkfs.btrfs /dev/sdb /dev/sdc /dev/sdd /dev/sde
# mount /dev/sdb /mnt
```

Add some data to the file system.

Finally, remove the required device.

```
# btrfs device delete /dev/sdc /mnt
```

6.4.6. Replacing Failed Devices on a btrfs File System

Section 6.4.5, "Removing btrfs Devices" can be used to remove a failed device provided the super block can still be read. However, if a device is missing or the super block corrupted, the file system will need to be mounted in a degraded mode:

```
# mkfs.btrfs -m raid1 /dev/sdb /dev/sdc /dev/sdd /dev/sde


   ssd is destroyed or removed, use -o degraded to force the mount
   to ignore missing devices


# mount -o degraded /dev/sdb /mnt


   'missing' is a special device name


# btrfs device delete missing /mnt
```

The command **btrfs device delete missing** removes the first device that is described by the file system metadata but not present when the file system was mounted.

IMPORTANT

It is impossible to go below the minimum number of devices required for the specific raid layout, even including the missing one. It may be required to add a new device in order to remove the failed one.

For example, for a raid1 layout with two devices, if a device fails it is required to:

1. mount in degraded mode,

2. add a new device,

3. and, remove the missing device.

6.4.7. Registering a btrfs File System in `/etc/fstab`

If you do not have an **initrd** or it does not perform a btrfs device scan, it is possible to mount a multi-volume **btrfs** file system by passing all the devices in the file system explicitly to the **mount** command.

Example 6.5. Example `/etc/fstab` Entry

An example of a suitable **/etc/fstab** entry would be:

```
/dev/sdb     /mnt     btrfs
device=/dev/sdb,device=/dev/sdc,device=/dev/sdd,device=/dev/sde     0
```

Note that using universally unique identifiers (UUIDs) also works and is more stable than using device paths.

6.5. SSD OPTIMIZATION

Using the btrfs file system can optimize SSD. There are two ways this can be done.

The first way is **mkfs.btrfs** turns off metadata duplication on a single device when

/sys/block/*device*/queue/rotational is zero for the single specified device. This is equivalent to specifying **-m single** on the command line. It can be overridden and duplicate metadata forced by providing the **-m dup** option. Duplication is not required due to SSD firmware potentially losing both copies. This wastes space and is a performance cost.

The second way is through a group of SSD mount options: **ssd**, **nossd**, and **ssd_spread**.

The **ssd** option does several things:

- It allows larger metadata cluster allocation.

- It allocates data more sequentially where possible.

- It disables btree leaf rewriting to match key and block order.

- It commits log fragments without batching multiple processes.

> **NOTE**
>
> The **ssd** mount option only enables the ssd option. Use the **nossd** option to disable it.

Some SSDs perform best when reusing block numbers often, while others perform much better when clustering strictly allocates big chunks of unused space. By default, **mount -o ssd** will find groupings of blocks where there are several free blocks that might have allocated blocks mixed in. The command **mount -o ssd_spread** ensures there are no allocated blocks mixed in. This improves performance on lower end SSDs.

> **NOTE**
>
> The **ssd_spread** option enables both the **ssd** and the **ssd_spread** options. Use the **nossd** to disable both these options.
>
> The **ssd_spread** option is never automatically set if none of the ssd options are provided and any of the devices are non-rotational.

These options will all need to be tested with your specific build to see if their use improves or reduces performance, as each combination of SSD firmware and application loads are different.

6.6. BTRFS REFERENCES

The man page **btrfs(8)** covers all important management commands. In particular this includes:

- All the subvolume commands for managing snapshots.

- The **device** commands for managing devices.

- The **scrub**, **balance**, and **defragment** commands.

The man page **mkfs.btrfs(8)** contains information on creating a btrfs file system including all the options regarding it.

The man page **btrfsck(8)** for information regarding **fsck** on btrfs systems.

CHAPTER 7. GLOBAL FILE SYSTEM 2

The Red Hat Global File System 2 (GFS2) is a native file system that interfaces directly with the Linux kernel file system interface (VFS layer). When implemented as a cluster file system, GFS2 employs distributed metadata and multiple journals.

GFS2 is based on 64-bit architecture, which can theoretically accommodate an 8 exabyte file system. However, the current supported maximum size of a GFS2 file system is 100 TB. If a system requires GFS2 file systems larger than 100 TB, contact your Red Hat service representative.

When determining the size of a file system, consider its recovery needs. Running the `fsck` command on a very large file system can take a long time and consume a large amount of memory. Additionally, in the event of a disk or disk-subsystem failure, recovery time is limited by the speed of backup media.

When configured in a Red Hat Cluster Suite, Red Hat GFS2 nodes can be configured and managed with Red Hat Cluster Suite configuration and management tools. Red Hat GFS2 then provides data sharing among GFS2 nodes in a Red Hat cluster, with a single, consistent view of the file system namespace across the GFS2 nodes. This allows processes on different nodes to share GFS2 files in the same way that processes on the same node can share files on a local file system, with no discernible difference. For information about the Red Hat Cluster Suite, see Red Hat's *Cluster Administration* guide.

A GFS2 must be built on a logical volume (created with LVM) that is a linear or mirrored volume. Logical volumes created with LVM in a Red Hat Cluster suite are managed with CLVM (a cluster-wide implementation of LVM), enabled by the CLVM daemon `clvmd`, and running in a Red Hat Cluster Suite cluster. The daemon makes it possible to use LVM2 to manage logical volumes across a cluster, allowing all nodes in the cluster to share the logical volumes. For information on the Logical Volume Manager, see Red Hat's Logical Volume Manager Administration guide.

The `gfs2.ko` kernel module implements the GFS2 file system and is loaded on GFS2 cluster nodes.

For comprehensive information on the creation and configuration of GFS2 file systems in clustered and non-clustered storage, see Red Hat's Global File System 2 guide.

CHAPTER 8. NETWORK FILE SYSTEM (NFS)

A *Network File System* (*NFS*) allows remote hosts to mount file systems over a network and interact with those file systems as though they are mounted locally. This enables system administrators to consolidate resources onto centralized servers on the network.

This chapter focuses on fundamental NFS concepts and supplemental information.

8.1. INTRODUCTION TO NFS

Currently, there are two major versions of NFS included in Red Hat Enterprise Linux:

- NFS version 3 (NFSv3) supports safe asynchronous writes and is more robust at error handling than the previous NFSv2; it also supports 64-bit file sizes and offsets, allowing clients to access more than 2 GB of file data.

- NFS version 4 (NFSv4) works through firewalls and on the Internet, no longer requires an **rpcbind** service, supports ACLs, and utilizes stateful operations.

Red Hat Enterprise Linux fully supports NFS version 4.2 (NFSv4.2) since the Red Hat Enterprise Linux 7.4 release.

Following are the features of NFSv4.2 in Red Hat Enterprise Linux 7.5 :

- Server-Side Copy: NFSv4.2 supports **copy_file_range()** system call, which allows the NFS client to efficiently copy data without wasting network resources.

- Sparse Files: It verifies space efficiency of a file and allows placeholder to improve storage efficiency. It is a file having one or more holes; holes are unallocated or uninitialized data blocks consisting only of zeroes. **lseek()** operation in NFSv4.2, supports **seek_hole()** and **seek_data()**, which allows application to map out the location of holes in the sparse file.

- Space Reservation: It permits storage servers to reserve free space, which prohibits servers to run out of space. NFSv4.2 supports **allocate()** operation to reserve space, **deallocate()** operation to unreserve space, and **fallocate()** operation to preallocate or deallocate space in a file.

- Labeled NFS: It enforces data access rights and enables SELinux labels between a client and a server for individual files on an NFS file system.

- Layout Enhancements: NFSv4.2 provides new operation, **layoutstats()**, which the client can use to notify the metadata server about its communication with the layout.

Versions of Red Hat Enterprise Linux earlier than 7.4 support NFS up to version 4.1.

Following are the features of NFSv4.1:

- Enhances performance and security of network, and also includes client-side support for Parallel NFS (pNFS).

- No longer requires a separate TCP connection for callbacks, which allows an NFS server to grant delegations even when it cannot contact the client. For example, when NAT or a firewall interferes.

- It provides exactly once semantics (except for reboot operations), preventing a previous issue whereby certain operations could return an inaccurate result if a reply was lost and the operation was sent twice.

NFS clients attempt to mount using NFSv4.1 by default, and fall back to NFSv4.0 when the server does not support NFSv4.1. The mount later fall back to NFSv3 when server does not support NFSv4.0.

> **NOTE**
>
> NFS version 2 (NFSv2) is no longer supported by Red Hat.

All versions of NFS can use *Transmission Control Protocol* (*TCP*) running over an IP network, with NFSv4 requiring it. NFSv3 can use the *User Datagram Protocol* (UDP) running over an IP network to provide a stateless network connection between the client and server.

When using NFSv3 with UDP, the stateless UDP connection (under normal conditions) has less protocol overhead than TCP. This can translate into better performance on very clean, non-congested networks. However, because UDP is stateless, if the server goes down unexpectedly, UDP clients continue to saturate the network with requests for the server. In addition, when a frame is lost with UDP, the entire RPC request must be retransmitted; with TCP, only the lost frame needs to be resent. For these reasons, TCP is the preferred protocol when connecting to an NFS server.

The mounting and locking protocols have been incorporated into the NFSv4 protocol. The server also listens on the well-known TCP port 2049. As such, NFSv4 does not need to interact with `rpcbind` [1], `lockd`, and `rpc.statd` daemons. The `rpc.mountd` daemon is still required on the NFS server to set up the exports, but is not involved in any over-the-wire operations.

> **NOTE**
>
> TCP is the default transport protocol for NFS version 3 under Red Hat Enterprise Linux. UDP can be used for compatibility purposes as needed, but is not recommended for wide usage. NFSv4 requires TCP.
>
> All the RPC/NFS daemons have a `'-p'` command line option that can set the port, making firewall configuration easier.

After TCP wrappers grant access to the client, the NFS server refers to the `/etc/exports` configuration file to determine whether the client is allowed to access any exported file systems. Once verified, all file and directory operations are available to the user.

> **IMPORTANT**
>
> In order for NFS to work with a default installation of Red Hat Enterprise Linux with a firewall enabled, configure IPTables with the default TCP port 2049. Without proper IPTables configuration, NFS will not function properly.
>
> The NFS initialization script and `rpc.nfsd` process now allow binding to any specified port during system start up. However, this can be error-prone if the port is unavailable, or if it conflicts with another daemon.

8.1.1. Required Services

Red Hat Enterprise Linux uses a combination of kernel-level support and daemon processes to provide

NFS file sharing. All NFS versions rely on *Remote Procedure Calls* (*RPC*) between clients and servers. RPC services under Red Hat Enterprise Linux 7 are controlled by the **rpcbind** service. To share or mount NFS file systems, the following services work together depending on which version of NFS is implemented:

> **NOTE**
>
> The **portmap** service was used to map RPC program numbers to IP address port number combinations in earlier versions of Red Hat Enterprise Linux. This service is now replaced by **rpcbind** in Red Hat Enterprise Linux 7 to enable IPv6 support.

nfs

systemctl start nfs starts the NFS server and the appropriate RPC processes to service requests for shared NFS file systems.

nfslock

systemctl start nfs-lock activates a mandatory service that starts the appropriate RPC processes allowing NFS clients to lock files on the server.

rpcbind

rpcbind accepts port reservations from local RPC services. These ports are then made available (or advertised) so the corresponding remote RPC services can access them. **rpcbind** responds to requests for RPC services and sets up connections to the requested RPC service. This is not used with NFSv4.

The following RPC processes facilitate NFS services:

rpc.mountd

This process is used by an NFS server to process **MOUNT** requests from NFSv3 clients. It checks that the requested NFS share is currently exported by the NFS server, and that the client is allowed to access it. If the mount request is allowed, the rpc.mountd server replies with a **Success** status and provides the **File-Handle** for this NFS share back to the NFS client.

rpc.nfsd

rpc.nfsd allows explicit NFS versions and protocols the server advertises to be defined. It works with the Linux kernel to meet the dynamic demands of NFS clients, such as providing server threads each time an NFS client connects. This process corresponds to the **nfs** service.

lockd

lockd is a kernel thread which runs on both clients and servers. It implements the *Network Lock Manager* (NLM) protocol, which allows NFSv3 clients to lock files on the server. It is started automatically whenever the NFS server is run and whenever an NFS file system is mounted.

rpc.statd

This process implements the *Network Status Monitor* (NSM) RPC protocol, which notifies NFS clients when an NFS server is restarted without being gracefully brought down. **rpc.statd** is started automatically by the **nfslock** service, and does not require user configuration. This is not used with NFSv4.

rpc.rquotad

This process provides user quota information for remote users. **rpc.rquotad** is started automatically by the **nfs** service and does not require user configuration.

rpc.idmapd

rpc.idmapd provides NFSv4 client and server upcalls, which map between on-the-wire NFSv4 names (strings in the form of *user@domain*) and local UIDs and GIDs. For **idmapd** to function with NFSv4, the **/etc/idmapd.conf** file must be configured. At a minimum, the "Domain" parameter should be specified, which defines the NFSv4 mapping domain. If the NFSv4 mapping domain is the same as the DNS domain name, this parameter can be skipped. The client and server must agree on the NFSv4 mapping domain for ID mapping to function properly.

> **NOTE**
>
> In Red Hat Enterprise Linux 7, only the NFSv4 server uses **rpc.idmapd**. The NFSv4 client uses the keyring-based idmapper **nfsidmap**. **nfsidmap** is a stand-alone program that is called by the kernel on-demand to perform ID mapping; it is not a daemon. If there is a problem with **nfsidmap** does the client fall back to using **rpc.idmapd**. More information regarding **nfsidmap** can be found on the nfsidmap man page.

8.2. PNFS

Support for Parallel NFS (pNFS) as part of the NFS v4.1 standard is available as of Red Hat Enterprise Linux 6.4. The pNFS architecture improves the scalability of NFS, with possible improvements to performance. That is, when a server implements pNFS as well, a client is able to access data through multiple servers concurrently. It supports three storage protocols or layouts: files, objects, and blocks.

> **NOTE**
>
> The protocol allows for three possible pNFS layout types: files, objects, and blocks. While the Red Hat Enterprise Linux 6.4 client only supported the files layout type, Red Hat Enterprise Linux 7 supports the files layout type, with objects and blocks layout types being included as a technology preview.

pNFS Flex Files

Flexible Files is a new layout for pNFS that enables the aggregation of standalone NFSv3 and NFSv4 servers into a scale out name space. The Flex Files feature is part of the NFSv4.2 standard as described in the RFC 7862 specification.

Red Hat Enterprise Linux can mount NFS shares from Flex Files servers since Red Hat Enterprise Linux 7.4.

Mounting pNFS Shares

- To enable pNFS functionality, mount shares from a pNFS-enabled server with NFS version 4.1 or later:

  ```
  # mount -t nfs -o v4.1 server:/remote-export /local-directory
  ```

After the server is pNFS-enabled, the **nfs_layout_nfsv41_files** kernel is automatically loaded on the first mount. The mount entry in the output should contain **minorversion=1**. Use the following command to verify the module was loaded:

```
$ lsmod | grep nfs_layout_nfsv41_files
```

- To mount an NFS share with the Flex Files feature from a server that supports Flex Files, use NFS version 4.2 or later:

```
# mount -t nfs -o v4.2 server:/remote-export /local-directory
```

Verify that the **nfs_layout_flexfiles** module has been loaded:

```
$ lsmod | grep nfs_layout_flexfiles
```

Additional Resources

For more information on pNFS, refer to: http://www.pnfs.com.

8.3. CONFIGURING NFS CLIENT

The **mount** command mounts NFS shares on the client side. Its format is as follows:

```
# mount -t nfs -o options server:/remote/export /local/directory
```

This command uses the following variables:

options

A comma-delimited list of mount options; for more information on valid NFS mount options, see Section 8.5, "Common NFS Mount Options".

server

The hostname, IP address, or fully qualified domain name of the server exporting the file system you wish to mount

/remote/export

The file system or directory being exported from the *server*, that is, the directory you wish to mount

/local/directory

The client location where */remote/export* is mounted

The NFS protocol version used in Red Hat Enterprise Linux 7 is identified by the **mount** options **nfsvers** or **vers**. By default, **mount** uses NFSv4 with **mount -t nfs**. If the server does not support NFSv4, the client automatically steps down to a version supported by the server. If the **nfsvers**/**vers** option is used to pass a particular version not supported by the server, the mount fails. The file system type nfs4 is also available for legacy reasons; this is equivalent to running **mount -t nfs -o nfsvers=4 host:/remote/export /local/directory**.

For more information, see **man mount**.

If an NFS share was mounted manually, the share will not be automatically mounted upon reboot. Red

Hat Enterprise Linux offers two methods for mounting remote file systems automatically at boot time: the **/etc/fstab** file and the **autofs** service. For more information, see Section 8.3.1, "Mounting NFS File Systems Using **/etc/fstab**" and Section 8.4, "**autofs**".

8.3.1. Mounting NFS File Systems Using `/etc/fstab`

An alternate way to mount an NFS share from another machine is to add a line to the **/etc/fstab** file. The line must state the hostname of the NFS server, the directory on the server being exported, and the directory on the local machine where the NFS share is to be mounted. You must be root to modify the **/etc/fstab** file.

> **Example 8.1. Syntax Example**
>
> The general syntax for the line in **/etc/fstab** is as follows:
>
> ```
> server:/usr/local/pub    /pub    nfs    defaults 0 0
> ```

The mount point **/pub** must exist on the client machine before this command can be executed. After adding this line to **/etc/fstab** on the client system, use the command **mount /pub**, and the mount point **/pub** is mounted from the server.

A valid **/etc/fstab** entry to mount an NFS export should contain the following information:

```
server:/remote/export /local/directory nfs options 0 0
```

The variables *server*, */remote/export*, */local/directory*, and *options* are the same ones used when manually mounting an NFS share. For more information, see Section 8.3, "Configuring NFS Client".

> **NOTE**
>
> The mount point */local/directory* must exist on the client before **/etc/fstab** is read. Otherwise, the mount fails.

After editing **/etc/fstab**, regenerate mount units so that your system registers the new configuration:

```
# systemctl daemon-reload
```

Additional Resources

- For more information about **/etc/fstab**, refer to **man fstab**.

8.4. AUTOFS

One drawback of using **/etc/fstab** is that, regardless of how infrequently a user accesses the NFS mounted file system, the system must dedicate resources to keep the mounted file system in place. This is not a problem with one or two mounts, but when the system is maintaining mounts to many systems at one time, overall system performance can be affected. An alternative to **/etc/fstab** is to use the kernel-based **automount** utility. An automounter consists of two components:

- a kernel module that implements a file system, and

- a user-space daemon that performs all of the other functions.

The **automount** utility can mount and unmount NFS file systems automatically (on-demand mounting), therefore saving system resources. It can be used to mount other file systems including AFS, SMBFS, CIFS, and local file systems.

> **IMPORTANT**
>
> The nfs-utils package is now a part of both the 'NFS file server' and the 'Network File System Client' groups. As such, it is no longer installed by default with the Base group. Ensure that nfs-utils is installed on the system first before attempting to automount an NFS share.
>
> autofs is also part of the 'Network File System Client' group.

autofs uses **/etc/auto.master** (master map) as its default primary configuration file. This can be changed to use another supported network source and name using the **autofs** configuration (in **/etc/sysconfig/autofs**) in conjunction with the Name Service Switch (NSS) mechanism. An instance of the **autofs** version 4 daemon was run for each mount point configured in the master map and so it could be run manually from the command line for any given mount point. This is not possible with **autofs** version 5, because it uses a single daemon to manage all configured mount points; as such, all automounts must be configured in the master map. This is in line with the usual requirements of other industry standard automounters. Mount point, hostname, exported directory, and options can all be specified in a set of files (or other supported network sources) rather than configuring them manually for each host.

8.4.1. Improvements in autofs Version 5 over Version 4

autofs version 5 features the following enhancements over version 4:

Direct map support

Direct maps in **autofs** provide a mechanism to automatically mount file systems at arbitrary points in the file system hierarchy. A direct map is denoted by a mount point of **/-** in the master map. Entries in a direct map contain an absolute path name as a key (instead of the relative path names used in indirect maps).

Lazy mount and unmount support

Multi-mount map entries describe a hierarchy of mount points under a single key. A good example of this is the **-hosts** map, commonly used for automounting all exports from a host under **/net/***host* as a multi-mount map entry. When using the **-hosts** map, an **ls** of **/net/***host* will mount *autofs* trigger mounts for each export from *host*. These will then mount and expire them as they are accessed. This can greatly reduce the number of active mounts needed when accessing a server with a large number of exports.

Enhanced LDAP support

The **autofs** configuration file (**/etc/sysconfig/autofs**) provides a mechanism to specify the **autofs** schema that a site implements, thus precluding the need to determine this via trial and error in the application itself. In addition, authenticated binds to the LDAP server are now supported, using most mechanisms supported by the common LDAP server implementations. A new configuration file has been added for this support: **/etc/autofs_ldap_auth.conf**. The default configuration file is self-documenting, and uses an XML format.

Proper use of the Name Service Switch (`nsswitch`) configuration.

The Name Service Switch configuration file exists to provide a means of determining from where specific configuration data comes. The reason for this configuration is to allow administrators the flexibility of using the back-end database of choice, while maintaining a uniform software interface to access the data. While the version 4 automounter is becoming increasingly better at handling the NSS configuration, it is still not complete. Autofs version 5, on the other hand, is a complete implementation.

For more information on the supported syntax of this file, see `man nsswitch.conf`. Not all NSS databases are valid map sources and the parser will reject ones that are invalid. Valid sources are files, `yp`, `nis`, `nisplus`, `ldap`, and `hesiod`.

Multiple master map entries per autofs mount point

One thing that is frequently used but not yet mentioned is the handling of multiple master map entries for the direct mount point `/-`. The map keys for each entry are merged and behave as one map.

> **Example 8.2. Multiple Master Map Entries per autofs Mount Point**
>
> Following is an example in the connectathon test maps for the direct mounts:
>
> ```
> /- /tmp/auto_dcthon
> /- /tmp/auto_test3_direct
> /- /tmp/auto_test4_direct
> ```

8.4.2. Configuring autofs

The primary configuration file for the automounter is `/etc/auto.master`, also referred to as the master map which may be changed as described in the Section 8.4.1, "Improvements in autofs Version 5 over Version 4". The master map lists `autofs`-controlled mount points on the system, and their corresponding configuration files or network sources known as automount maps. The format of the master map is as follows:

```
mount-point map-name options
```

The variables used in this format are:

mount-point

The `autofs` mount point, `/home`, for example.

map-name

The name of a map source which contains a list of mount points, and the file system location from which those mount points should be mounted.

options

If supplied, these applies to all entries in the given map provided they do not themselves have options specified. This behavior is different from `autofs` version 4 where options were cumulative. This has been changed to implement mixed environment compatibility.

Example 8.3. `/etc/auto.master` File

The following is a sample line from **`/etc/auto.master`** file (displayed with **`cat /etc/auto.master`**):

```
/home /etc/auto.misc
```

The general format of maps is similar to the master map, however the "options" appear between the mount point and the location instead of at the end of the entry as in the master map:

```
mount-point   [options]   location
```

The variables used in this format are:

mount-point

This refers to the **`autofs`** mount point. This can be a single directory name for an indirect mount or the full path of the mount point for direct mounts. Each direct and indirect map entry key (***mount-point***) may be followed by a space separated list of offset directories (subdirectory names each beginning with a **`/`**) making them what is known as a multi-mount entry.

options

Whenever supplied, these are the mount options for the map entries that do not specify their own options.

location

This refers to the file system location such as a local file system path (preceded with the Sun map format escape character ":" for map names beginning with **`/`**), an NFS file system or other valid file system location.

The following is a sample of contents from a map file (for example, **`/etc/auto.misc`**):

```
payroll -fstype=nfs personnel:/dev/hda3
sales -fstype=ext3 :/dev/hda4
```

The first column in a map file indicates the **`autofs`** mount point (**`sales`** and **`payroll`** from the server called **`personnel`**). The second column indicates the options for the **`autofs`** mount while the third column indicates the source of the mount. Following the given configuration, the autofs mount points will be **`/home/payroll`** and **`/home/sales`**. The **`-fstype=`** option is often omitted and is generally not needed for correct operation.

The automounter create the directories if they do not exist. If the directories exist before the automounter was started, the automounter will not remove them when it exits.

To start the automount daemon, use the following command:

```
# systemctl start autofs
```

To restart the automount daemon, use the following command:

```
# systemctl restart autofs
```

▪

Using the given configuration, if a process requires access to an **autofs** unmounted directory such as **/home/payroll/2006/July.sxc**, the automount daemon automatically mounts the directory. If a timeout is specified, the directory is automatically unmounted if the directory is not accessed for the timeout period.

To view the status of the automount daemon, use the following command:

```
# systemctl status autofs
```

8.4.3. Overriding or Augmenting Site Configuration Files

It can be useful to override site defaults for a specific mount point on a client system. For example, consider the following conditions:

- Automounter maps are stored in NIS and the **/etc/nsswitch.conf** file has the following directive:

  ```
  automount:     files nis
  ```

- The **auto.master** file contains:

  ```
  +auto.master
  ```

- The NIS **auto.master** map file contains:

  ```
  /home auto.home
  ```

- The NIS **auto.home** map contains:

  ```
  beth        fileserver.example.com:/export/home/beth
  joe         fileserver.example.com:/export/home/joe
  *         fileserver.example.com:/export/home/&
  ```

- The file map **/etc/auto.home** does not exist.

Given these conditions, let's assume that the client system needs to override the NIS map **auto.home** and mount home directories from a different server. In this case, the client needs to use the following **/etc/auto.master** map:

```
/home /etc/auto.home
+auto.master
```

The **/etc/auto.home** map contains the entry:

```
*     labserver.example.com:/export/home/&
```

Because the automounter only processes the first occurrence of a mount point, **/home** contain the contents of **/etc/auto.home** instead of the NIS **auto.home** map.

Alternatively, to augment the site-wide **auto.home** map with just a few entries, create an **/etc/auto.home** file map, and in it put the new entries. At the end, include the NIS**auto.home** map. Then the **/etc/auto.home** file map looks similar to:

```
mydir someserver:/export/mydir
+auto.home
```

With these NIS **auto.home** map conditions, the **ls /home** command outputs:

```
beth joe mydir
```

This last example works as expected because **autofs** does not include the contents of a file map of the same name as the one it is reading. As such, **autofs** moves on to the next map source in the **nsswitch** configuration.

8.4.4. Using LDAP to Store Automounter Maps

LDAP client libraries must be installed on all systems configured to retrieve automounter maps from LDAP. On Red Hat Enterprise Linux, the **openldap** package should be installed automatically as a dependency of the **automounter**. To configure LDAP access, modify **/etc/openldap/ldap.conf**. Ensure that BASE, URI, and schema are set appropriately for your site.

The most recently established schema for storing automount maps in LDAP is described by **rfc2307bis**. To use this schema it is necessary to set it in the **autofs** configuration (**/etc/sysconfig/autofs**) by removing the comment characters from the schema definition. For example:

Example 8.4. Setting autofs Configuration

```
DEFAULT_MAP_OBJECT_CLASS="automountMap"
DEFAULT_ENTRY_OBJECT_CLASS="automount"
DEFAULT_MAP_ATTRIBUTE="automountMapName"
DEFAULT_ENTRY_ATTRIBUTE="automountKey"
DEFAULT_VALUE_ATTRIBUTE="automountInformation"
```

Ensure that these are the only schema entries not commented in the configuration. The **automountKey** replaces the **cn** attribute in the **rfc2307bis** schema. Following is an example of an LDAP Data Interchange Format (**LDIF**) configuration:

Example 8.5. LDF Configuration

```
# extended LDIF
#
# LDAPv3
# base <> with scope subtree
# filter: (&(objectclass=automountMap)(automountMapName=auto.master))
# requesting: ALL
#

# auto.master, example.com
dn: automountMapName=auto.master,dc=example,dc=com
objectClass: top
```

```
objectClass: automountMap
automountMapName: auto.master

# extended LDIF
#
# LDAPv3
# base <automountMapName=auto.master,dc=example,dc=com> with scope
subtree
# filter: (objectclass=automount)
# requesting: ALL
#

# /home, auto.master, example.com
dn: automountMapName=auto.master,dc=example,dc=com
objectClass: automount
cn: /home

automountKey: /home
automountInformation: auto.home

# extended LDIF
#
# LDAPv3
# base <> with scope subtree
# filter: (&(objectclass=automountMap)(automountMapName=auto.home))
# requesting: ALL
#

# auto.home, example.com
dn: automountMapName=auto.home,dc=example,dc=com
objectClass: automountMap
automountMapName: auto.home

# extended LDIF
#
# LDAPv3
# base <automountMapName=auto.home,dc=example,dc=com> with scope subtree
# filter: (objectclass=automount)
# requesting: ALL
#

# foo, auto.home, example.com
dn: automountKey=foo,automountMapName=auto.home,dc=example,dc=com
objectClass: automount
automountKey: foo
automountInformation: filer.example.com:/export/foo

# /, auto.home, example.com
dn: automountKey=/,automountMapName=auto.home,dc=example,dc=com
objectClass: automount
automountKey: /
automountInformation: filer.example.com:/export/&
```

8.5. COMMON NFS MOUNT OPTIONS

Beyond mounting a file system with NFS on a remote host, it is also possible to specify other options at mount time to make the mounted share easier to use. These options can be used with manual **mount** commands, **/etc/fstab** settings, and **autofs**.

The following are options commonly used for NFS mounts:

intr

Allows NFS requests to be interrupted if the server goes down or cannot be reached.

lookupcache=*mode*

Specifies how the kernel should manage its cache of directory entries for a given mount point. Valid arguments for *mode* are **all**, **none**, or **pos**/**positive**.

nfsvers=*version*

Specifies which version of the NFS protocol to use, where *version* is 3 or 4. This is useful for hosts that run multiple NFS servers. If no version is specified, NFS uses the highest version supported by the kernel and **mount** command.

The option **vers** is identical to **nfsvers**, and is included in this release for compatibility reasons.

noacl

Turns off all ACL processing. This may be needed when interfacing with older versions of Red Hat Enterprise Linux, Red Hat Linux, or Solaris, since the most recent ACL technology is not compatible with older systems.

nolock

Disables file locking. This setting is sometimes required when connecting to very old NFS servers.

noexec

Prevents execution of binaries on mounted file systems. This is useful if the system is mounting a non-Linux file system containing incompatible binaries.

nosuid

Disables **set-user-identifier** or **set-group-identifier** bits. This prevents remote users from gaining higher privileges by running a **setuid** program.

port=*num*

Specifies the numeric value of the NFS server port. If *num* is **0** (the default value), then **mount** queries the remote host's **rpcbind** service for the port number to use. If the remote host's NFS daemon is not registered with its **rpcbind** service, the standard NFS port number of TCP 2049 is used instead.

rsize=*num* **and wsize=***num*

These options set the maximum number of bytes to be transfered in a single NFS read or write operation.

There is no fixed default value for **rsize** and **wsize**. By default, NFS uses the largest possible value that both the server and the client support. In Red Hat Enterprise Linux 7, the client and server maximum is 1,048,576 bytes. For more details, see the What are the default and maximum values for rsize and wsize with NFS mounts? KBase article.

sec=*mode*

Its default setting is **sec=sys**, which uses local UNIX UIDs and GIDs. These use **AUTH_SYS** to authenticate NFS operations.

sec=krb5 uses Kerberos V5 instead of local UNIX UIDs and GIDs to authenticate users.

sec=krb5i uses Kerberos V5 for user authentication and performs integrity checking of NFS operations using secure checksums to prevent data tampering.

sec=krb5p uses Kerberos V5 for user authentication, integrity checking, and encrypts NFS traffic to prevent traffic sniffing. This is the most secure setting, but it also involves the most performance overhead.

tcp

Instructs the NFS mount to use the TCP protocol.

udp

Instructs the NFS mount to use the UDP protocol.

For more information, see **man mount** and **man nfs**.

8.6. STARTING AND STOPPING THE NFS SERVER

Prerequisites

- For servers that support NFSv2 or NFSv3 connections, the **rpcbind**[1] service must be running. To verify that **rpcbind** is active, use the following command:

  ```
  $ systemctl status rpcbind
  ```

  To configure an NFSv4-only server, which does not require **rpcbind**, see Section 8.7.7, "Configuring an NFSv4-only Server".

- On Red Hat Enterprise Linux 7.0, if your NFS server exports NFSv3 and is enabled to start at boot, you need to manually start and enable the **nfs-lock** service:

  ```
  # systemctl start nfs-lock
  # systemctl enable nfs-lock
  ```

  On Red Hat Enterprise Linux 7.1 and later, **nfs-lock** starts automatically if needed, and an attempt to enable it manually fails.

Procedures

- To start an NFS server, use the following command:

  ```
  # systemctl start nfs
  ```

- To enable NFS to start at boot, use the following command:

  ```
  # systemctl enable nfs
  ```

- To stop the server, use:

  ```
  # systemctl stop nfs
  ```

- The **restart** option is a shorthand way of stopping and then starting NFS. This is the most efficient way to make configuration changes take effect after editing the configuration file for NFS. To restart the server type:

  ```
  # systemctl restart nfs
  ```

- After you edit the **/etc/sysconfig/nfs** file, restart the nfs-config service by running the following command for the new values to take effect:

  ```
  # systemctl restart nfs-config
  ```

- The **try-restart** command only starts **nfs** if it is currently running. This command is the equivalent of **condrestart** (*conditional restart*) in Red Hat init scripts and is useful because it does not start the daemon if NFS is not running.

  To conditionally restart the server, type:

  ```
  # systemctl try-restart nfs
  ```

- To reload the NFS server configuration file without restarting the service type:

  ```
  # systemctl reload nfs
  ```

8.7. CONFIGURING THE NFS SERVER

There are two ways to configure exports on an NFS server:

- Manually editing the NFS configuration file, that is, **/etc/exports**, and

- Through the command line, that is, by using the command **exportfs**

8.7.1. The **/etc/exports** Configuration File

The **/etc/exports** file controls which file systems are exported to remote hosts and specifies options. It follows the following syntax rules:

- Blank lines are ignored.

- To add a comment, start a line with the hash mark (#).

- You can wrap long lines with a backslash (\).

- Each exported file system should be on its own individual line.

- Any lists of authorized hosts placed after an exported file system must be separated by space characters.

- Options for each of the hosts must be placed in parentheses directly after the host identifier, without any spaces separating the host and the first parenthesis.

Each entry for an exported file system has the following structure:

```
export host(options)
```

The aforementioned structure uses the following variables:

export

The directory being exported

host

The host or network to which the export is being shared

options

The options to be used for *host*

It is possible to specify multiple hosts, along with specific options for each host. To do so, list them on the same line as a space-delimited list, with each hostname followed by its respective options (in parentheses), as in:

```
export host1(options1) host2(options2) host3(options3)
```

For information on different methods for specifying hostnames, see Section 8.7.5, "Hostname Formats".

In its simplest form, the **/etc/exports** file only specifies the exported directory and the hosts permitted to access it, as in the following example:

Example 8.6. The /etc/exports File

```
/exported/directory bob.example.com
```

Here, **bob.example.com** can mount **/exported/directory/** from the NFS server. Because no options are specified in this example, NFS uses *default* settings.

The default settings are:

ro

The exported file system is read-only. Remote hosts cannot change the data shared on the file system. To allow hosts to make changes to the file system (that is, read and write), specify the **rw** option.

sync

The NFS server will not reply to requests before changes made by previous requests are written to disk. To enable asynchronous writes instead, specify the option **async**.

wdelay

The NFS server will delay writing to the disk if it suspects another write request is imminent. This can

improve performance as it reduces the number of times the disk must be accessed by separate write commands, thereby reducing write overhead. To disable this, specify the **no_wdelay**. **no_wdelay** is only available if the default **sync** option is also specified.

root_squash

This prevents root users connected *remotely* (as opposed to locally) from having root privileges; instead, the NFS server assigns them the user ID **nfsnobody**. This effectively "squashes" the power of the remote root user to the lowest local user, preventing possible unauthorized writes on the remote server. To disable root squashing, specify **no_root_squash**.

To squash every remote user (including root), use **all_squash**. To specify the user and group IDs that the NFS server should assign to remote users from a particular host, use the **anonuid** and **anongid** options, respectively, as in:

```
export host(anonuid=uid,anongid=gid)
```

Here, *uid* and *gid* are user ID number and group ID number, respectively. The **anonuid** and **anongid** options allow you to create a special user and group account for remote NFS users to share.

By default, *access control lists* (*ACLs*) are supported by NFS under Red Hat Enterprise Linux. To disable this feature, specify the **no_acl** option when exporting the file system.

Each default for every exported file system must be explicitly overridden. For example, if the **rw** option is not specified, then the exported file system is shared as read-only. The following is a sample line from **/etc/exports** which overrides two default options:

```
/another/exported/directory 192.168.0.3(rw,async)
```

In this example **192.168.0.3** can mount **/another/exported/directory/** read and write and all writes to disk are asynchronous. For more information on exporting options, see **man exportfs**.

Other options are available where no default value is specified. These include the ability to disable sub-tree checking, allow access from insecure ports, and allow insecure file locks (necessary for certain early NFS client implementations). For more information on these less-used options, see **man exports**.

> **IMPORTANT**
>
> The format of the **/etc/exports** file is very precise, particularly in regards to use of the space character. Remember to always separate exported file systems from hosts and hosts from one another with a space character. However, there should be no other space characters in the file except on comment lines.
>
> For example, the following two lines do not mean the same thing:
>
> ```
> /home bob.example.com(rw)
> /home bob.example.com (rw)
> ```
>
> The first line allows only users from **bob.example.com** read and write access to the **/home** directory. The second line allows users from **bob.example.com** to mount the directory as read-only (the default), while the rest of the world can mount it read/write.

8.7.2. The `exportfs` Command

Every file system being exported to remote users with NFS, as well as the access level for those file systems, are listed in the **/etc/exports** file. When the **nfs** service starts, the **/usr/sbin/exportfs** command launches and reads this file, passes control to **rpc.mountd** (if NFSv3) for the actual mounting process, then to **rpc.nfsd** where the file systems are then available to remote users.

When issued manually, the **/usr/sbin/exportfs** command allows the root user to selectively export or unexport directories without restarting the NFS service. When given the proper options, the **/usr/sbin/exportfs** command writes the exported file systems to **/var/lib/nfs/xtab**. Since **rpc.mountd** refers to the **xtab** file when deciding access privileges to a file system, changes to the list of exported file systems take effect immediately.

The following is a list of commonly-used options available for **/usr/sbin/exportfs**:

-r

Causes all directories listed in **/etc/exports** to be exported by constructing a new export list in **/etc/lib/nfs/xtab**. This option effectively refreshes the export list with any changes made to **/etc/exports**.

-a

Causes all directories to be exported or unexported, depending on what other options are passed to **/usr/sbin/exportfs**. If no other options are specified, **/usr/sbin/exportfs** exports all file systems specified in **/etc/exports**.

-o *file-systems*

Specifies directories to be exported that are not listed in **/etc/exports**. Replace *file-systems* with additional file systems to be exported. These file systems must be formatted in the same way they are specified in **/etc/exports**. This option is often used to test an exported file system before adding it permanently to the list of file systems to be exported. For more information on **/etc/exports** syntax, see Section 8.7.1, "The **/etc/exports** Configuration File".

-i

Ignores **/etc/exports**; only options given from the command line are used to define exported file systems.

-u

Unexports all shared directories. The command **/usr/sbin/exportfs -ua** suspends NFS file sharing while keeping all NFS daemons up. To re-enable NFS sharing, use **exportfs -r**.

-v

Verbose operation, where the file systems being exported or unexported are displayed in greater detail when the **exportfs** command is executed.

If no options are passed to the **exportfs** command, it displays a list of currently exported file systems. For more information about the **exportfs** command, see **man exportfs**.

8.7.2.1. Using **exportfs** with NFSv4

In Red Hat Enterprise Linux 7, no extra steps are required to configure NFSv4 exports as any filesystems mentioned are automatically available to NFSv3 and NFSv4 clients using the same path. This was not the case in previous versions.

To prevent clients from using NFSv4, turn it off by setting **RPCNFSDARGS= -N 4** in **/etc/sysconfig/nfs**.

8.7.3. Running NFS Behind a Firewall

NFS requires **rpcbind**, which dynamically assigns ports for RPC services and can cause issues for configuring firewall rules. To allow clients to access NFS shares behind a firewall, edit the **/etc/sysconfig/nfs** file to set which ports the RPC services run on.

The **/etc/sysconfig/nfs** file does not exist by default on all systems. If **/etc/sysconfig/nfs** does not exist, create it and specify the following:

RPCMOUNTDOPTS="-p *port*"

This adds "-p *port*" to the rpc.mount command line: **rpc.mount -p *port***.

To specify the ports to be used by the **nlockmgr** service, set the port number for the **_nlm_tcpport_** and **_nlm_udpport_** options in the **/etc/modprobe.d/lockd.conf** file.

If NFS fails to start, check **/var/log/messages**. Commonly, NFS fails to start if you specify a port number that is already in use. After editing **/etc/sysconfig/nfs**, you need to restart the **nfs-config** service for the new values to take effect in Red Hat Enterprise Linux 7.2 and prior by running:

```
#  systemctl restart nfs-config
```

Then, restart the NFS server:

```
#  systemctl restart nfs-server
```

Run **rpcinfo -p** to confirm the changes have taken effect.

> **NOTE**
>
> To allow NFSv4.0 callbacks to pass through firewalls set **/proc/sys/fs/nfs/nfs_callback_tcpport** and allow the server to connect to that port on the client.
>
> This process is not needed for NFSv4.1 or higher, and the other ports for **mountd**, **statd**, and **lockd** are not required in a pure NFSv4 environment.

8.7.3.1. Discovering NFS exports

There are two ways to discover which file systems an NFS server exports.

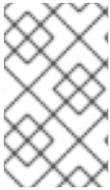- On any server that supports NFSv3, use the **showmount** command:

  ```
  $ showmount -e myserver
  Export list for mysever
  /exports/foo
  /exports/bar
  ```

- On any server that supports NFSv4, **mount** the root directory and look around.

```
# mount myserver:/ /mnt/
# cd /mnt/
exports
# ls exports
foo
bar
```

On servers that support both NFSv4 and NFSv3, both methods work and give the same results.

> **NOTE**
>
> Before Red Hat Enterprise Linux 6 on older NFS servers, depending on how they are configured, it is possible to export filesystems to NFSv4 clients at different paths. Because these servers do not enable NFSv4 by default, this should not be a problem.

8.7.4. Accessing RPC Quota through a Firewall

If you export a file system that uses disk quotas, you can use the quota Remote Procedure Call (RPC) service to provide disk quota data to NFS clients.

Procedure 8.1. Making RPC Quota Accessible Behind a Firewall

1. To enable the **rpc-rquotad** service, use the following command:

   ```
   # systemctl enable rpc-rquotad
   ```

2. To start the **rpc-rquotad** service, use the following command:

   ```
   # systemctl start rpc-rquotad
   ```

   Note that **rpc-rquotad** is, if enabled, started automatically after starting the **nfs-server** service.

3. To make the quota RPC service accessible behind a firewall, UDP or TCP port **875** need to be open. The default port number is defined in the **/etc/services** file.

   You can override the default port number by appending **-p** *port-number* to the **RPCRQUOTADOPTS** variable in the **/etc/sysconfig/rpc-rquotad** file.

4. Restart **rpc-rquotad** for changes in the **/etc/sysconfig/rpc-rquotad** file to take effect:

   ```
   # systemctl restart rpc-rquotad
   ```

Setting Quotas from Remote Hosts

By default, quotas can only be read by remote hosts. To allow setting quotas, append the **-S** option to the **RPCRQUOTADOPTS** variable in the **/etc/sysconfig/rpc-rquotad** file.

Restart **rpc-rquotad** for changes in the **/etc/sysconfig/rpc-rquotad** file to take effect:

```
# systemctl restart rpc-rquotad
```

8.7.5. Hostname Formats

The host(s) can be in the following forms:

Single machine

A fully-qualified domain name (that can be resolved by the server), hostname (that can be resolved by the server), or an IP address.

Series of machines specified with wildcards

Use the ***** or **?** character to specify a string match. Wildcards are not to be used with IP addresses; however, they may accidentally work if reverse DNS lookups fail. When specifying wildcards in fully qualified domain names, dots (**.**) are not included in the wildcard. For example, ***.example.com** includes **one.example.com** but does not **include one.two.example.com**.

IP networks

Use *a.b.c.d/z*, where *a.b.c.d* is the network and *z* is the number of bits in the netmask (for example 192.168.0.0/24). Another acceptable format is *a.b.c.d/netmask*, where *a.b.c.d* is the network and *netmask* is the netmask (for example, 192.168.100.8/255.255.255.0).

Netgroups

Use the format *@group-name*, where *group-name* is the NIS netgroup name.

8.7.6. Enabling NFS over RDMA (NFSoRDMA)

The remote direct memory access (RDMA) service works automatically in Red Hat Enterprise Linux 7 if there is RDMA-capable hardware present.

To enable NFS over RDMA:

1. Install the rdma and rdma-core packages.

   The **/etc/rdma/rdma.conf** file contains a line that sets **XPRTRDMA_LOAD=yes** by default, which requests the **rdma** service to load the NFSoRDMA *client* module.

2. To enable automatic loading of NFSoRDMA *server* modules, add **SVCRDMA_LOAD=yes** on a new line in **/etc/rdma/rdma.conf**.

   RPCNFSDARGS="--rdma=20049" in the **/etc/sysconfig/nfs** file specifies the port number on which the NFSoRDMA service listens for clients. *RFC 5667* specifies that servers must listen on port **20049** when providing NFSv4 services over RDMA.

3. Restart the **nfs** service after editing the **/etc/rdma/rdma.conf** file:

   ```
   # systemctl restart nfs
   ```

   Note that with earlier kernel versions, a system reboot is needed after editing **/etc/rdma/rdma.conf** for the changes to take effect.

8.7.7. Configuring an NFSv4-only Server

By default, the NFS server supports NFSv2, NFSv3, and NFSv4 connections in Red Hat Enterprise Linux 7. However, you can also configure NFS to support only NFS version 4.0 and later. This minimizes the number of open ports and running services on the system, because NFSv4 does not require the **rpcbind** service to listen on the network.

When your NFS server is configured as NFSv4-only, clients attempting to mount shares using NFSv2 or NFSv3 fail with an error like the following:

```
Requested NFS version or transport protocol is not supported.
```

Procedure 8.2. Configuring an NFSv4-only Server

To configure your NFS server to support only NFS version 4.0 and later:

1. Disable NFSv2, NFSv3, and UDP by adding the following line to the **/etc/sysconfig/nfs** configuration file:

   ```
   RPCNFSDARGS="-N 2 -N 3 -U"
   ```

2. Optionally, disable listening for the **RPCBIND**, **MOUNT**, and **NSM** protocol calls, which are not necessary in the NFSv4-only case.

   The effects of disabling these options are:

   - Clients that attempt to mount shares from your server using NFSv2 or NFSv3 become unresponsive.

   - The NFS server itself is unable to mount NFSv2 and NFSv3 file systems.

   To disable these options:

   - Add the following to the **/etc/sysconfig/nfs** file:

     ```
     RPCMOUNTDOPTS="-N 2 -N 3"
     ```

   - Disable related services:

     ```
     # systemctl mask --now rpc-statd.service rpcbind.service
     rpcbind.socket
     ```

3. Restart the NFS server:

   ```
   # systemctl restart nfs
   ```

   The changes take effect as soon as you start or restart the NFS server.

Verifying the NFSv4-only Configuration

You can verify that your NFS server is configured in the NFSv4-only mode by using the **netstat** utility.

- The following is an example **netstat** output on an NFSv4-only server; listening for **RPCBIND**, **MOUNT**, and **NSM** is also disabled. Here, **nfs** is the only listening NFS service:

  ```
  # netstat -ltu

  Active Internet connections (only servers)
  Proto Recv-Q Send-Q Local Address           Foreign Address
  State
  ```

```
tcp        0        0 0.0.0.0:nfs            0.0.0.0:*
LISTEN
tcp        0        0 0.0.0.0:ssh            0.0.0.0:*
LISTEN
tcp        0        0 localhost:smtp         0.0.0.0:*
LISTEN
tcp6       0        0 [::]:nfs               [::]:*
LISTEN
tcp6       0        0 [::]:12432             [::]:*
LISTEN
tcp6       0        0 [::]:12434             [::]:*
LISTEN
tcp6       0        0 localhost:7092         [::]:*
LISTEN
tcp6       0        0 [::]:ssh               [::]:*
LISTEN
udp        0        0 localhost:323          0.0.0.0:*
udp        0        0 0.0.0.0:bootpc         0.0.0.0:*
udp6       0        0 localhost:323          [::]:*
```

- In comparison, the **netstat** output before configuring an NFSv4-only server includes the **sunrpc** and **mountd** services:

```
# netstat -ltu

Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address          Foreign Address
State
tcp        0        0 0.0.0.0:nfs            0.0.0.0:*
LISTEN
tcp        0        0 0.0.0.0:36069          0.0.0.0:*
LISTEN
tcp        0        0 0.0.0.0:52364          0.0.0.0:*
LISTEN
tcp        0        0 0.0.0.0:sunrpc         0.0.0.0:*
LISTEN
tcp        0        0 0.0.0.0:mountd         0.0.0.0:*
LISTEN
tcp        0        0 0.0.0.0:ssh            0.0.0.0:*
LISTEN
tcp        0        0 localhost:smtp         0.0.0.0:*
LISTEN
tcp6       0        0 [::]:34941             [::]:*
LISTEN
tcp6       0        0 [::]:nfs               [::]:*
LISTEN
tcp6       0        0 [::]:sunrpc            [::]:*
LISTEN
tcp6       0        0 [::]:mountd            [::]:*
LISTEN
tcp6       0        0 [::]:12432             [::]:*
LISTEN
tcp6       0        0 [::]:56881             [::]:*
LISTEN
tcp6       0        0 [::]:12434             [::]:*
LISTEN
```

```
tcp6        0        0 localhost:7092          [::]:*
LISTEN
tcp6        0        0 [::]:ssh                [::]:*
LISTEN
udp         0        0 localhost:323           0.0.0.0:*
udp         0        0 0.0.0.0:37190           0.0.0.0:*
udp         0        0 0.0.0.0:876             0.0.0.0:*
udp         0        0 localhost:877           0.0.0.0:*
udp         0        0 0.0.0.0:mountd          0.0.0.0:*
udp         0        0 0.0.0.0:38588           0.0.0.0:*
udp         0        0 0.0.0.0:nfs             0.0.0.0:*
udp         0        0 0.0.0.0:bootpc          0.0.0.0:*
udp         0        0 0.0.0.0:sunrpc          0.0.0.0:*
udp6        0        0 localhost:323           [::]:*
udp6        0        0 [::]:57683              [::]:*
udp6        0        0 [::]:876                [::]:*
udp6        0        0 [::]:mountd             [::]:*
udp6        0        0 [::]:40874              [::]:*
udp6        0        0 [::]:nfs                [::]:*
udp6        0        0 [::]:sunrpc             [::]:*
```

8.8. SECURING NFS

NFS is suitable for transparent sharing of entire file systems with a large number of known hosts. However, with ease-of-use comes a variety of potential security problems. To minimize NFS security risks and protect data on the server, consider the following sections when exporting NFS file systems on a server or mounting them on a client.

8.8.1. NFS Security with AUTH_SYS and Export Controls

Traditionally, NFS has given two options in order to control access to exported files.

First, the server restricts which hosts are allowed to mount which file systems either by IP address or by host name.

Second, the server enforces file system permissions for users on NFS clients in the same way it does local users. Traditionally it does this using **AUTH_SYS** (also called **AUTH_UNIX**) which relies on the client to state the UID and GID's of the user. Be aware that this means a malicious or misconfigured client can easily get this wrong and allow a user access to files that it should not.

To limit the potential risks, administrators often allow read-only access or squash user permissions to a common user and group ID. Unfortunately, these solutions prevent the NFS share from being used in the way it was originally intended.

Additionally, if an attacker gains control of the DNS server used by the system exporting the NFS file system, the system associated with a particular hostname or fully qualified domain name can be pointed to an unauthorized machine. At this point, the unauthorized machine *is* the system permitted to mount the NFS share, since no username or password information is exchanged to provide additional security for the NFS mount.

Wildcards should be used sparingly when exporting directories through NFS, as it is possible for the scope of the wildcard to encompass more systems than intended.

It is also possible to restrict access to the **rpcbind**[1] service with TCP wrappers. Creating rules with **iptables** can also limit access to ports used by **rpcbind**, **rpc.mountd**, and **rpc.nfsd**.

For more information on securing NFS and `rpcbind`, refer to `man iptables`.

8.8.2. NFS Security with AUTH_GSS

NFSv4 revolutionized NFS security by mandating the implementation of RPCSEC_GSS and the Kerberos version 5 GSS-API mechanism. However, RPCSEC_GSS and the Kerberos mechanism are also available for all versions of NFS. In FIPS mode, only FIPS-approved algorithms can be used.

Unlike AUTH_SYS, with the RPCSEC_GSS Kerberos mechanism, the server does not depend on the client to correctly represent which user is accessing the file. Instead, cryptography is used to authenticate users to the server, which prevents a malicious client from impersonating a user without having that user's Kerberos credentials. Using the RPCSEC_GSS Kerberos mechanism is the most straightforward way to secure mounts because after configuring Kerberos, no additional setup is needed.

Configuring Kerberos

Before configuring an NFSv4 Kerberos-aware server, you need to install and configure a Kerberos Key Distribution Centre (KDC). Kerberos is a network authentication system that allows clients and servers to authenticate to each other by using symmetric encryption and a trusted third party, the KDC. Red Hat recommends using Identity Management (IdM) for setting up Kerberos.

Procedure 8.3. Configuring an NFS Server and Client for IdM to Use RPCSEC_GSS

1. ○ Create the **nfs/hostname.*domain*@*REALM*** principal on the NFS server side.

   ○ Create the **host/hostname.*domain*@*REALM*** principal on both the server and the client side.

   ○ Add the corresponding keys to keytabs for the client and server.

   For instructions, see the Adding and Editing Service Entries and Keytabs and Setting up a Kerberos-aware NFS Server sections in the Red Hat Enterprise Linux 7 Linux Domain Identity, Authentication, and Policy Guide.

2. On the server side, use the **sec=** option to enable the wanted security flavors. To enable all security flavors as well as non-cryptographic mounts:

   ```
   /export *(sec=sys:krb5:krb5i:krb5p)
   ```

   Valid security flavors to use with the **sec=** option are:

   ○ **sys**: no cryptographic protection, the default

   ○ **krb5**: authentication only

   ○ **krb5i**: integrity protection

   ○ **krb5p**: privacy protection

3. On the client side, add **sec=krb5** (or **sec=krb5i**, or **sec=krb5p**, depending on the setup) to the mount options:

   ```
   # mount -o sec=krb5 server:/export /mnt
   ```

For information on how to configure a NFS client, see the Setting up a Kerberos-aware NFS Client section in the Red Hat Enterprise Linux 7 Linux Domain Identity, Authentication, and Policy Guide.

Although Red Hat recommends using IdM, Active Directory (AD) Kerberos servers are also supported. For details, see the following Red Hat Knowledgebase article: How to set up NFS using Kerberos authentication on RHEL 7 using SSSD and Active Directory.

For more information, see the exports(5) and nfs(5) manual pages, and Section 8.5, "Common NFS Mount Options".

For further information on the **RPCSEC_GSS** framework, including how **gssproxy** and **rpc.gssd** inter-operate, see the GSSD flow description.

8.8.2.1. NFS Security with NFSv4

NFSv4 includes ACL support based on the Microsoft Windows NT model, not the POSIX model, because of the Microsoft Windows NT model's features and wide deployment.

Another important security feature of NFSv4 is the removal of the use of the **MOUNT** protocol for mounting file systems. The **MOUNT** protocol presented a security risk because of the way the protocol processed file handles.

8.8.3. File Permissions

Once the NFS file system is mounted as either read or read and write by a remote host, the only protection each shared file has is its permissions. If two users that share the same user ID value mount the same NFS file system, they can modify each others' files. Additionally, anyone logged in as root on the client system can use the **su -** command to access any files with the NFS share.

By default, access control lists (ACLs) are supported by NFS under Red Hat Enterprise Linux. Red Hat recommends that this feature is kept enabled.

By default, NFS uses *root squashing* when exporting a file system. This sets the user ID of anyone accessing the NFS share as the root user on their local machine to **nobody**. Root squashing is controlled by the default option **root_squash**; for more information about this option, refer to Section 8.7.1, "The **/etc/exports** Configuration File". If possible, never disable root squashing.

When exporting an NFS share as read-only, consider using the **all_squash** option. This option makes every user accessing the exported file system take the user ID of the **nfsnobody** user.

8.9. NFS AND RPCBIND

NOTE

The following section only applies to NFSv3 implementations that require the **rpcbind** service for backward compatibility.

For information on how to configure an NFSv4-only server, which does not need **rpcbind**, see Section 8.7.7, "Configuring an NFSv4-only Server".

The **rpcbind**[1] utility maps RPC services to the ports on which they listen. RPC processes notify **rpcbind** when they start, registering the ports they are listening on and the RPC program numbers they

expect to serve. The client system then contacts **rpcbind** on the server with a particular RPC program number. The **rpcbind** service redirects the client to the proper port number so it can communicate with the requested service.

Because RPC-based services rely on **rpcbind** to make all connections with incoming client requests, **rpcbind** must be available before any of these services start.

The **rpcbind** service uses TCP wrappers for access control, and access control rules for **rpcbind** affect *all* RPC-based services. Alternatively, it is possible to specify access control rules for each of the NFS RPC daemons. The **man** pages for **rpc.mountd** and **rpc.statd** contain information regarding the precise syntax for these rules.

8.9.1. Troubleshooting NFS and `rpcbind`

Because **rpcbind**[1] provides coordination between RPC services and the port numbers used to communicate with them, it is useful to view the status of current RPC services using **rpcbind** when troubleshooting. The **rpcinfo** command shows each RPC-based service with port numbers, an RPC program number, a version number, and an IP protocol type (TCP or UDP).

To make sure the proper NFS RPC-based services are enabled for **rpcbind**, use the following command:

```
# rpcinfo -p
```

Example 8.7. `rpcinfo -p` command output

The following is sample output from this command:

```
program vers proto   port service
     100021    1   udp  32774  nlockmgr
     100021    3   udp  32774  nlockmgr
     100021    4   udp  32774  nlockmgr
     100021    1   tcp  34437  nlockmgr
     100021    3   tcp  34437  nlockmgr
     100021    4   tcp  34437  nlockmgr
     100011    1   udp    819  rquotad
     100011    2   udp    819  rquotad
     100011    1   tcp    822  rquotad
     100011    2   tcp    822  rquotad
     100003    2   udp   2049  nfs
     100003    3   udp   2049  nfs
     100003    2   tcp   2049  nfs
     100003    3   tcp   2049  nfs
     100005    1   udp    836  mountd
     100005    1   tcp    839  mountd
     100005    2   udp    836  mountd
     100005    2   tcp    839  mountd
     100005    3   udp    836  mountd
     100005    3   tcp    839  mountd
```

If one of the NFS services does not start up correctly, `rpcbind` will be unable to map RPC requests from clients for that service to the correct port. In many cases, if NFS is not present in `rpcinfo` output, restarting NFS causes the service to correctly register with `rpcbind` and begin working.

For more information and a list of options on `rpcinfo`, see its `man` page.

8.10. NFS REFERENCES

Administering an NFS server can be a challenge. Many options, including quite a few not mentioned in this chapter, are available for exporting or mounting NFS shares. For more information, see the following sources:

Installed Documentation

- `man mount` — Contains a comprehensive look at mount options for both NFS server and client configurations.

- `man fstab` — Provides detail for the format of the `/etc/fstab` file used to mount file systems at boot-time.

- `man nfs` — Provides details on NFS-specific file system export and mount options.

- `man exports` — Shows common options used in the `/etc/exports` file when exporting NFS file systems.

Useful Websites

- http://linux-nfs.org — The current site for developers where project status updates can be viewed.

- http://nfs.sourceforge.net/ — The old home for developers which still contains a lot of useful information.

- http://www.citi.umich.edu/projects/nfsv4/linux/ — An NFSv4 for Linux 2.6 kernel resource.

- http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.111.4086 — An excellent whitepaper on the features and enhancements of the NFS Version 4 protocol.

Related Books

- *Managing NFS and NIS* by Hal Stern, Mike Eisler, and Ricardo Labiaga; O'Reilly & Associates — Makes an excellent reference guide for the many different NFS export and mount options available.

- *NFS Illustrated* by Brent Callaghan; Addison-Wesley Publishing Company — Provides comparisons of NFS to other network file systems and shows, in detail, how NFS communication occurs.

---

[1] The `rpcbind` service replaces `portmap`, which was used in previous versions of Red Hat Enterprise Linux to map RPC program numbers to IP address port number combinations. For more information, refer to Section 8.1.1, "Required Services".

CHAPTER 9. SERVER MESSAGE BLOCK (SMB)

The Server Message Block (SMB) protocol implements an application-layer network protocol used to access resources on a server, such as file shares and shared printers. On Microsoft Windows, SMB is implemented by default. If you run Red Hat Enterprise Linux, use Samba to provide SMB shares and the `cifs-utils` utility to mount SMB shares from a remote server.

> **NOTE**
>
> In the context of SMB, you sometimes read about the Common Internet File System (CIFS) protocol, which is a dialect of SMB. Both the SMB and CIFS protocol are supported and the kernel module and utilities involved in mounting SMB and CIFS shares both use the name `cifs`.

9.1. PROVIDING SMB SHARES

See the *Samba* section in the *Red Hat System Administrator's Guide*.

9.2. MOUNTING AN SMB SHARE

On Red Hat Enterprise Linux, the `cifs.ko` file system module of the kernel provides support for the SMB protocol. However, to mount and work with SMB shares, you must also install the cifs-utils package:

```
# yum install cifs-utils
```

The cifs-utils package provides utilities to:

- Mount SMB and CIFS shares

- Manage NT Lan Manager (NTLM) credentials in the kernel's keyring

- Set and display Access Control Lists (ACL) in a security descriptor on SMB and CIFS shares

9.2.1. Supported SMB Protocol Versions

The `cifs.ko` kernel module supports the following SMB protocol versions:

- SMB 1

- SMB 2.0

- SMB 2.1

- SMB 3.0

> **NOTE**
>
> Depending on the protocol version, not all SMB features are implemented.

9.2.1.1. UNIX Extensions Support

Samba uses the **CAP_UNIX** capability bit in the SMB protocol to provide the UNIX extensions feature. These extensions are also supported by the **cifs.ko** kernel module. However, both Samba and the kernel module support UNIX extensions only in the SMB 1 protocol.

To use UNIX extensions:

1. Set the **server min protocol** option in the **[global]** section in the **/etc/samba/smb.conf** file to **NT1**. This is the default on Samba servers.

2. Mount the share using the SMB 1 protocol by providing the **-o vers=1.0** option to the **mount** command. For example:

   ```
   mount -t cifs -o vers=1.0,username=user_name
   //server_name/share_name /mnt/
   ```

   By default, the kernel module uses SMB 2 or the highest later protocol version supported by the server. Passing the **-o vers=1.0** option to the **mount** command forces that the kernel module uses the SMB 1 protocol that is required for using UNIX extensions.

To verify if UNIX extensions are enabled, display the options of the mounted share:

```
# mount
...
//server/share on /mnt type cifs (...,unix,...)
```

If the **unix** entry is displayed in the list of mount options, UNIX extensions are enabled.

9.2.2. Manually Mounting an SMB Share

To manually mount an SMB share, use the **mount** utility with the *-t cifs* parameter:

```
# mount -t cifs -o username=user_name //server_name/share_name /mnt/
Password for user_name@//server_name/share_name:  ********
```

In the *-o options* parameter, you can specify options that will be used to mount the share. For details, see Section 9.2.6, "Frequently Used Mount Options" and the *OPTIONS* section in the mount.cifs(8) man page.

> **Example 9.1. Mounting a Share Using an Encrypted SMB 3.0 Connection**
>
> To mount the **\\server\example** share as the *DOMAIN\Administrator* user over an encrypted SMB 3.0 connection into the **/mnt/** directory:
>
> ```
> # mount -t cifs -o username=DOMAIN\Administrator,seal,vers=3.0
> //server/example /mnt/
> Password for user_name@//server_name/share_name:  ********
> ```

9.2.3. Mounting an SMB Share Automatically When the System Boots

To mount an SMB share automatically when the system boots, add an entry for the share to the **/etc/fstab** file. For example:

```
//server_name/share_name  /mnt  cifs  credentials=/root/smb.cred  0 0
```

> **IMPORTANT**
>
> To enable the system to mount a share automatically, you must store the user name, password, and domain name in a credentials file. For details, see Section 9.2.4, "Authenticating To an SMB Share Using a Credentials File".

In the fourth field of the **/etc/fstab** file, specify mount options, such as the path to the credentials file. For details, see Section 9.2.6, "Frequently Used Mount Options" and the *OPTIONS* section in the mount.cifs(8) man page.

To verify that the share mounts successfully, enter:

```
# mount /mnt/
```

9.2.4. Authenticating To an SMB Share Using a Credentials File

In certain situations, administrators want to mount a share without entering the user name and password. To implement this, create a credentials file. For example:

Procedure 9.1. Creating a Credentials File

1. Create a file, such as **~/smb.cred**, and specify the user name, password, and domain name that file:

   ```
   username=user_name
   password=password
   domain=domain_name
   ```

2. Set the permissions to only allow the owner to access the file:

   ```
   # chown user_name ~/smb.cred
   # chmod 600 ~/smb.cred
   ```

You can now pass the **credentials=*file_name*** mount option to the **mount** utility or use it in the **/etc/fstab** file to mount the share without being prompted for the user name and password.

9.2.5. Performing a Multi-user SMB Mount

The credentials you provide to mount a share determine the access permissions on the mount point by default. For example, if you use the **DOMAIN\example** user when you mount a share, all operations on the share will be executed as this user, regardless which local user performs the operation.

However, in certain situations, the administrator wants to mount a share automatically when the system boots, but users should perform actions on the share's content using their own credentials. The **multiuser** mount options lets you configure this scenario.
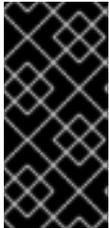
> **IMPORTANT**
>
> To use **multiuser**, you must additionally set the **sec=*security_type*** mount option to a security type which supports providing credentials in a non-interactive way, such as **krb5** or the **ntlmssp** option with a credentials file. See the section called "Accessing a Share as a User".

The **root** user mounts the share using the **multiuser** option and an account that has minimal access to the contents of the share. Regular users can then provide their user name and password to the current session's kernel keyring using the **cifscreds** utility. If the user accesses the content of the mounted share, the kernel uses the credentials from the kernel keyring instead of the one initially used to mount the share.

Mounting a Share with the **multiuser** Option

To mount a share automatically with the **multiuser** option when the system boots:

Procedure 9.2. Creating an /etc/fstab File Entry with the multiuser Option

1. Create the entry for the share in the **/etc/fstab** file. For example:

   ```
   //server_name/share_name  /mnt  cifs
   multiuser,sec=ntlmssp,credentials=/root/smb.cred  0 0
   ```

2. Mount the share:

   ```
   # mount /mnt/
   ```

If you do not want to mount the share automatically when the system boots, mount it manually by passing **-o multiuser,sec=*security_type*** to the **mount** command. For details about mounting an SMB share manually, see Section 9.2.2, "Manually Mounting an SMB Share".

Verifying if an SMB Share is Mounted with the **multiuser** Option

To verify if a share is mounted with the **multiuser** option:

```
# mount
...
//server_name/share_name on /mnt type cifs (sec=ntlmssp,multiuser,...)
```

Accessing a Share as a User

If an SMB share is mounted with the **multiuser** option, users can provide their credentials for the server to the kernel's keyring:

```
# cifscreds add -u SMB_user_name server_name
Password: ********
```

Now, when the user performs operations in the directory that contains the mounted SMB share, the server applies the file system permissions for this user, instead of the one initially used when the share was mounted.

NOTE

Multiple users can perform operations using their own credentials on the mounted share at the same time.

9.2.6. Frequently Used Mount Options

When you mount an SMB share, the mount options determine:

- How the connection will be established with the server. For example, which SMB protocol version is used when connecting to the server.

- How the share will be mounted into the local file system. For example, if the system overrides the remote file and directory permissions to enable multiple local users to access the content on the server.

To set multiple options in the fourth field of the **/etc/fstab** file or in the **-o** parameter of a **mount** command, separate them with commas. For example, see Procedure 9.2, "Creating an **/etc/fstab** File Entry with the **multiuser** Option".

The following list gives an overview of frequently used mount options:

Table 9.1. Frequently Used Mount Options

Option	Description
credentials=*file_name*	Sets the path to the credentials file. See Section 9.2.4, "Authenticating To an SMB Share Using a Credentials File".
dir_mode=*mode*	Sets the directory mode if the server does not support CIFS UNIX extensions.
file_mode=*mode*	Sets the file mode if the server does not support CIFS UNIX extensions.
password=*password*	Sets the password used to authenticate to the SMB server. Alternatively, specify a credentials file using the **credentials** option.
seal	Enables encryption support for connections using SMB 3.0 or a later protocol version. Therefore, use **seal** together with the **vers** mount option set to **3.0** or later. See Example 9.1, "Mounting a Share Using an Encrypted SMB 3.0 Connection".
sec=*security_mode*	Sets the security mode, such as **ntlmsspi**, to enable NTLMv2 password hashing and enabled packet signing. For a list of supported values, see the option's description in the mount.cifs(8) man page. If the server does not support the **ntlmv2** security mode, use **sec=ntlmssp**, which is the default. For security reasons, do not use the insecure **ntlm** security mode.
username=*user_name*	Sets the user name used to authenticate to the SMB server. Alternatively, specify a credentials file using the **credentials** option.

Option	Description
vers=*SMB_protocol_version*	Sets the SMB protocol version used for the communication with the server.

For a complete list, see the *OPTIONS* section in the mount.cifs(8) man page.

CHAPTER 10. FS-CACHE

FS-Cache is a persistent local cache that can be used by file systems to take data retrieved from over the network and cache it on local disk. This helps minimize network traffic for users accessing data from a file system mounted over the network (for example, NFS).

The following diagram is a high-level illustration of how FS-Cache works:
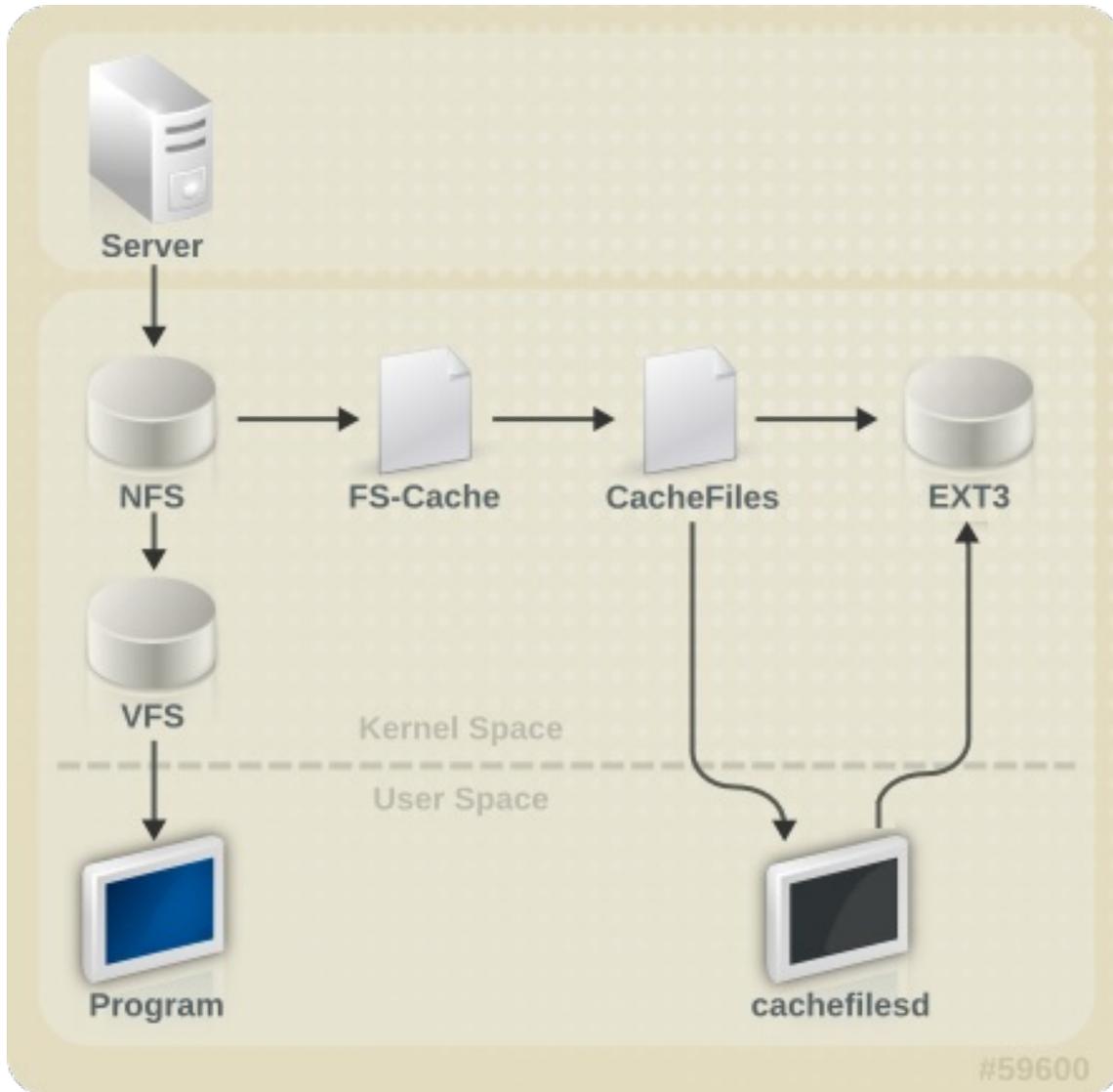
Figure 10.1. FS-Cache Overview

FS-Cache is designed to be as transparent as possible to the users and administrators of a system. Unlike **cachefs** on Solaris, FS-Cache allows a file system on a server to interact directly with a client's local cache without creating an overmounted file system. With NFS, a mount option instructs the client to mount the NFS share with FS-cache enabled.

FS-Cache does not alter the basic operation of a file system that works over the network - it merely provides that file system with a persistent place in which it can cache data. For instance, a client can still mount an NFS share whether or not FS-Cache is enabled. In addition, cached NFS can handle files that won't fit into the cache (whether individually or collectively) as files can be partially cached and do not have to be read completely up front. FS-Cache also hides all I/O errors that occur in the cache from the client file system driver.

To provide caching services, FS-Cache needs a *cache back end*. A cache back end is a storage driver configured to provide caching services (i.e. **cachefiles**). In this case, FS-Cache requires a mounted block-based file system that supports **bmap** and extended attributes (e.g. ext3) as its cache back end.

FS-Cache cannot arbitrarily cache any file system, whether through the network or otherwise: the shared file system's driver must be altered to allow interaction with FS-Cache, data storage/retrieval, and metadata setup and validation. FS-Cache needs *indexing keys* and *coherency data* from the cached file system to support persistence: indexing keys to match file system objects to cache objects, and coherency data to determine whether the cache objects are still valid.

> **NOTE**
>
> In Red Hat Enterprise Linux 7, the cachefilesd package is not installed by default and needs to be installed manually.

10.1. PERFORMANCE GUARANTEE

FS-Cache does *not* guarantee increased performance, however it ensures consistent performance by avoiding network congestion. Using a cache back end incurs a performance penalty: for example, cached NFS shares add disk accesses to cross-network lookups. While FS-Cache tries to be as asynchronous as possible, there are synchronous paths (e.g. reads) where this isn't possible.

For example, using FS-Cache to cache an NFS share between two computers over an otherwise unladen GigE network will not demonstrate any performance improvements on file access. Rather, NFS requests would be satisfied faster from server memory rather than from local disk.

The use of FS-Cache, therefore, is a *compromise* between various factors. If FS-Cache is being used to cache NFS traffic, for instance, it may slow the client down a little, but massively reduce the network and server loading by satisfying read requests locally without consuming network bandwidth.

10.2. SETTING UP A CACHE

Currently, Red Hat Enterprise Linux 7 only provides the **cachefiles** caching back end. The **cachefilesd** daemon initiates and manages **cachefiles**. The **/etc/cachefilesd.conf** file controls how **cachefiles** provides caching services.

The first setting to configure in a cache back end is which directory to use as a cache. To configure this, use the following parameter:

```
$ dir /path/to/cache
```

Typically, the cache back end directory is set in **/etc/cachefilesd.conf** as **/var/cache/fscache**, as in:

```
$ dir /var/cache/fscache
```

If you want to change the cache back end directory, the selinux context must be same as **/var/cache/fscache**:

```
# semanage fcontext -a -e /var/cache/fscache /path/to/cache
# restorecon -Rv /path/to/cache
```

Replace */path/to/cache* with the directory name while setting up cache.

NOTE

If the given commands for setting selinux context did not work, use the following commands:

```
# semanage permissive -a cachefilesd_t
# semanage permissive -a cachefiles_kernel_t
```

FS-Cache will store the cache in the file system that hosts */path/to/cache*. On a laptop, it is advisable to use the root file system (**/**) as the host file system, but for a desktop machine it would be more prudent to mount a disk partition specifically for the cache.

File systems that support functionalities required by FS-Cache cache back end include the Red Hat Enterprise Linux 7 implementations of the following file systems:

- ext3 (with extended attributes enabled)

- ext4

- Btrfs

- XFS

The host file system must support user-defined extended attributes; FS-Cache uses these attributes to store coherency maintenance information. To enable user-defined extended attributes for ext3 file systems (i.e. *device*), use:

```
# tune2fs -o user_xattr /dev/device
```

Alternatively, extended attributes for a file system can be enabled at mount time, as in:

```
# mount /dev/device /path/to/cache -o user_xattr
```

The cache back end works by maintaining a certain amount of free space on the partition hosting the cache. It grows and shrinks the cache in response to other elements of the system using up free space, making it safe to use on the root file system (for example, on a laptop). FS-Cache sets defaults on this behavior, which can be configured via *cache cull limits*. For more information about configuring cache cull limits, refer to Section 10.4, "Setting Cache Cull Limits".

Once the configuration file is in place, start up the **cachefilesd** service:

```
# systemctl start cachefilesd
```

To configure **cachefilesd** to start at boot time, execute the following command as root:

```
# systemctl enable cachefilesd
```

10.3. USING THE CACHE WITH NFS

NFS will not use the cache unless explicitly instructed. To configure an NFS mount to use FS-Cache, include the **-o fsc** option to the **mount** command:

```
# mount nfs-share:/ /mount/point -o fsc
```

All access to files under ***/mount/point*** will go through the cache, unless the file is opened for direct I/O or writing. For more information, see Section 10.3.2, "Cache Limitations with NFS". NFS indexes cache contents using NFS file handle, *not* the file name, which means hard-linked files share the cache correctly.

Caching is supported in version 2, 3, and 4 of NFS. However, each version uses different branches for caching.

10.3.1. Cache Sharing

There are several potential issues to do with NFS cache sharing. Because the cache is persistent, blocks of data in the cache are indexed on a sequence of four keys:

- Level 1: Server details

- Level 2: Some mount options; security type; FSID; uniquifier

- Level 3: File Handle

- Level 4: Page number in file

To avoid coherency management problems between superblocks, all NFS superblocks that wish to cache data have unique Level 2 keys. Normally, two NFS mounts with same source volume and options share a superblock, and thus share the caching, even if they mount different directories within that volume.

Example 10.1. Cache Sharing

Take the following two **mount** commands:

mount home0:/disk0/fred /home/fred -o fsc

mount home0:/disk0/jim /home/jim -o fsc

Here, **/home/fred** and **/home/jim** likely share the superblock as they have the same options, especially if they come from the same volume/partition on the NFS server (**home0**). Now, consider the next two subsequent mount commands:

mount home0:/disk0/fred /home/fred -o fsc,rsize=230

mount home0:/disk0/jim /home/jim -o fsc,rsize=231

In this case, **/home/fred** and **/home/jim** will not share the superblock as they have different network access parameters, which are part of the Level 2 key. The same goes for the following mount sequence:

mount home0:/disk0/fred /home/fred1 -o fsc,rsize=230

mount home0:/disk0/fred /home/fred2 -o fsc,rsize=231

Here, the contents of the two subtrees (**/home/fred1** and **/home/fred2**) will be cached *twice*.

Another way to avoid superblock sharing is to suppress it explicitly with the **nosharecache** parameter. Using the same example:

mount home0:/disk0/fred /home/fred -o nosharecache,fsc

mount home0:/disk0/jim /home/jim -o nosharecache,fsc

However, in this case only one of the superblocks is permitted to use cache since there is nothing to distinguish the Level 2 keys of **home0:/disk0/fred** and **home0:/disk0/jim**. To address this, add a *unique identifier* on at least one of the mounts, i.e. **fsc=*unique-identifier***. For example:

mount home0:/disk0/fred /home/fred -o nosharecache,fsc

mount home0:/disk0/jim /home/jim -o nosharecache,fsc=jim

Here, the unique identifier **jim** is added to the Level 2 key used in the cache for **/home/jim**.

10.3.2. Cache Limitations with NFS

- Opening a file from a shared file system for direct I/O automatically bypasses the cache. This is because this type of access must be direct to the server.

- Opening a file from a shared file system for writing will not work on NFS version 2 and 3. The protocols of these versions do not provide sufficient coherency management information for the client to detect a concurrent write to the same file from another client.

- Opening a file from a shared file system for either direct I/O or writing flushes the cached copy of the file. FS-Cache will not cache the file again until it is no longer opened for direct I/O or writing.

- Furthermore, this release of FS-Cache only caches regular NFS files. FS-Cache will *not* cache directories, symlinks, device files, FIFOs and sockets.

10.4. SETTING CACHE CULL LIMITS

The **cachefilesd** daemon works by caching remote data from shared file systems to free space on the disk. This could potentially consume all available free space, which could be bad if the disk also housed the root partition. To control this, **cachefilesd** tries to maintain a certain amount of free space by discarding old objects (i.e. accessed less recently) from the cache. This behavior is known as *cache culling*.

Cache culling is done on the basis of the percentage of blocks and the percentage of files available in the underlying file system. There are six limits controlled by settings in **/etc/cachefilesd.conf**:

brun *N*% (percentage of blocks) , frun *N*% (percentage of files)

  If the amount of free space and the number of available files in the cache rises above both these limits, then culling is turned off.

bcull *N*% (percentage of blocks), fcull *N*% (percentage of files)

  If the amount of available space or the number of files in the cache falls below either of these limits, then culling is started.

bstop *N*% (percentage of blocks), fstop *N*% (percentage of files)

If the amount of available space or the number of available files in the cache falls below either of these limits, then no further allocation of disk space or files is permitted until culling has raised things above these limits again.

The default value of **N** for each setting is as follows:

- **brun**/**frun** - 10%

- **bcull**/**fcull** - 7%

- **bstop**/**fstop** - 3%

When configuring these settings, the following must hold true:

- 0 ≤ **bstop** < **bcull** < **brun** < 100

- 0 ≤ **fstop** < **fcull** < **frun** < 100

These are the percentages of available space and available files and do not appear as 100 minus the percentage displayed by the **df** program.

IMPORTANT

Culling depends on both b*xxx* and f*xxx* pairs simultaneously; they can not be treated separately.

10.5. STATISTICAL INFORMATION

FS-Cache also keeps track of general statistical information. To view this information, use:

```
# cat /proc/fs/fscache/stats
```

FS-Cache statistics includes information on decision points and object counters. For more information, see the following kernel document:

/usr/share/doc/kernel-doc-*version*/Documentation/filesystems/caching/fscache.txt

10.6. FS-CACHE REFERENCES

For more information on **cachefilesd** and how to configure it, see **man cachefilesd** and **man cachefilesd.conf**. The following kernel documents also provide additional information:

- **/usr/share/doc/cachefilesd-*version-number*/README**

- **/usr/share/man/man5/cachefilesd.conf.5.gz**

- **/usr/share/man/man8/cachefilesd.8.gz**

For general information about FS-Cache, including details on its design constraints, available statistics, and capabilities, see the following kernel document: **/usr/share/doc/kernel-doc-*version*/Documentation/filesystems/caching/fscache.txt**

PART II. STORAGE ADMINISTRATION

The Storage Administration section starts with storage considerations for Red Hat Enterprise Linux 7. Instructions regarding partitions, logical volume management, and swap partitions follow this. Disk Quotas, RAID systems are next, followed by the functions of mount command, volume_key, and acls. SSD tuning, write barriers, I/O limits and diskless systems follow this. The large chapter of Online Storage is next, and finally device mapper multipathing and virtual storage to finish.

CHAPTER 11. STORAGE CONSIDERATIONS DURING INSTALLATION

Many storage device and file system settings can only be configured at install time. Other settings, such as file system type, can only be modified up to a certain point without requiring a reformat. As such, it is prudent that you plan your storage configuration accordingly before installing Red Hat Enterprise Linux 7.

This chapter discusses several considerations when planning a storage configuration for your system. For installation instructions (including storage configuration during installation), see the Installation Guide provided by Red Hat.

For information on what Red Hat officially supports with regards to size and storage limits, see the article http://www.redhat.com/resourcelibrary/articles/articles-red-hat-enterprise-linux-6-technology-capabilities-and-limits.

11.1. SPECIAL CONSIDERATIONS

This section enumerates several issues and factors to consider for specific storage configurations.

Separate Partitions for /home, /opt, /usr/local

If it is likely that you will upgrade your system in the future, place **/home**, **/opt**, and **/usr/local** on a separate device. This allows you to reformat the devices or file systems containing the operating system while preserving your user and application data.

DASD and zFCP Devices on IBM System Z

On the IBM System Z platform, DASD and zFCP devices are configured via the *Channel Command Word* (CCW) mechanism. CCW paths must be explicitly added to the system and then brought online. For DASD devices, this means listing the device numbers (or device number ranges) as the **DASD=** parameter at the boot command line or in a CMS configuration file.

For zFCP devices, you must list the device number, *logical unit number* (LUN), and *world wide port name* (WWPN). Once the zFCP device is initialized, it is mapped to a CCW path. The **FCP_x=** lines on the boot command line (or in a CMS configuration file) allow you to specify this information for the installer.

Encrypting Block Devices Using LUKS

Formatting a block device for encryption using LUKS/**dm-crypt** destroys any existing formatting on that device. As such, you should decide which devices to encrypt (if any) before the new system's storage configuration is activated as part of the installation process.

Stale BIOS RAID Metadata

Moving a disk from a system configured for firmware RAID *without* removing the RAID metadata from the disk can prevent **Anaconda** from correctly detecting the disk.

> **WARNING**
>
> Removing/deleting RAID metadata from disk could potentially destroy any stored data. Red Hat recommends that you back up your data before proceeding.

To delete RAID metadata from the disk, use the following command:

```
dmraid -r -E /device/
```

For more information about managing RAID devices, see **man dmraid** and Chapter 18, *Redundant Array of Independent Disks (RAID)*.

iSCSI Detection and Configuration

For plug and play detection of iSCSI drives, configure them in the firmware of an iBFT boot-capable *network interface card* (NIC). CHAP authentication of iSCSI targets is supported during installation. However, iSNS discovery is not supported during installation.

FCoE Detection and Configuration

For plug and play detection of *Fibre Channel over Ethernet* (FCoE) drives, configure them in the firmware of an EDD boot-capable NIC.

DASD

Direct-access storage devices (DASD) cannot be added or configured during installation. Such devices are specified in the CMS configuration file.

Block Devices with DIF/DIX Enabled

DIF/DIX is a hardware checksum feature provided by certain SCSI host bus adapters and block devices. When DIF/DIX is enabled, error occurs if the block device is used as a general-purpose block device. Buffered I/O or **mmap(2)**-based I/O will not work reliably, as there are no interlocks in the buffered write path to prevent buffered data from being overwritten after the DIF/DIX checksum has been calculated.

This causes the I/O to later fail with a checksum error. This problem is common to all block device (or file system-based) buffered I/O or **mmap(2)** I/O, so it is not possible to work around these errors caused by overwrites.

As such, block devices with DIF/DIX enabled should only be used with applications that use **O_DIRECT**. Such applications should use the raw block device. Alternatively, it is also safe to use the XFS file system on a DIF/DIX enabled block device, as long as only **O_DIRECT** I/O is issued through the file system. XFS is the only file system that does not fall back to buffered I/O when doing certain allocation operations.

The responsibility for ensuring that the I/O data does not change after the DIF/DIX checksum has been computed always lies with the application, so only applications designed for use with **O_DIRECT** I/O and DIF/DIX hardware should use DIF/DIX.

CHAPTER 12. FILE SYSTEM CHECK

File systems may be checked for consistency, and optionally repaired, with file system-specific userspace tools. These tools are often referred to as **fsck** tools, where **fsck** is a shortened version of *file system check*.

> **NOTE**
>
> These file system checkers only guarantee metadata consistency across the file system; they have no awareness of the actual data contained within the file system and are not data recovery tools.

File system inconsistencies can occur for various reasons, including but not limited to hardware errors, storage administration errors, and software bugs.

Before modern metadata-journaling file systems became common, a file system check was required any time a system crashed or lost power. This was because a file system update could have been interrupted, leading to an inconsistent state. As a result, a file system check is traditionally run on each file system listed in **/etc/fstab** at boot-time. For journaling file systems, this is usually a very short operation, because the file system's metadata journaling ensures consistency even after a crash.

However, there are times when a file system inconsistency or corruption may occur, even for journaling file systems. When this happens, the file system checker must be used to repair the file system. The following provides best practices and other useful information when performing this procedure.

> **IMPORTANT**
>
> Red Hat does not recommended this unless the machine does not boot, the file system is extremely large, or the file system is on remote storage. It is possible to disable file system check at boot by setting the sixth field in /etc/fstab to 0.

12.1. BEST PRACTICES FOR FSCK

Generally, running the file system check and repair tool can be expected to automatically repair at least some of the inconsistencies it finds. In some cases, severely damaged inodes or directories may be discarded if they cannot be repaired. Significant changes to the file system may occur. To ensure that unexpected or undesirable changes are not permanently made, perform the following precautionary steps:

Dry run

Most file system checkers have a mode of operation which checks but does not repair the file system. In this mode, the checker prints any errors that it finds and actions that it would have taken, without actually modifying the file system.

> **NOTE**
>
> Later phases of consistency checking may print extra errors as it discovers inconsistencies which would have been fixed in early phases if it were running in repair mode.

Operate first on a file system image

Most file systems support the creation of a *metadata image*, a sparse copy of the file system which

contains only metadata. Because file system checkers operate only on metadata, such an image can be used to perform a dry run of an actual file system repair, to evaluate what changes would actually be made. If the changes are acceptable, the repair can then be performed on the file system itself.
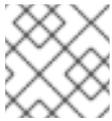
> **NOTE**
>
> Severely damaged file systems may cause problems with metadata image creation.

Save a file system image for support investigations

A pre-repair file system metadata image can often be useful for support investigations if there is a possibility that the corruption was due to a software bug. Patterns of corruption present in the pre-repair image may aid in root-cause analysis.

Operate only on unmounted file systems

A file system repair must be run only on unmounted file systems. The tool must have sole access to the file system or further damage may result. Most file system tools enforce this requirement in repair mode, although some only support check-only mode on a mounted file system. If check-only mode is run on a mounted file system, it may find spurious errors that would not be found when run on an unmounted file system.

Disk errors

File system check tools cannot repair hardware problems. A file system must be fully readable and writable if repair is to operate successfully. If a file system was corrupted due to a hardware error, the file system must first be moved to a good disk, for example with the **dd(8)** utility.

12.2. FILE SYSTEM-SPECIFIC INFORMATION FOR FSCK

12.2.1. ext2, ext3, and ext4

All of these file sytems use the **e2fsck** binary to perform file system checks and repairs. The file names **fsck.ext2**, **fsck.ext3**, and **fsck.ext4** are hardlinks to this same binary. These binaries are run automatically at boot time and their behavior differs based on the file system being checked and the state of the file system.

A full file system check and repair is invoked for ext2, which is not a metadata journaling file system, and for ext4 file systems without a journal.

For ext3 and ext4 file systems with metadata journaling, the journal is replayed in userspace and the binary exits. This is the default action as journal replay ensures a consistent file system after a crash.

If these file systems encounter metadata inconsistencies while mounted, they record this fact in the file system superblock. If **e2fsck** finds that a file system is marked with such an error, **e2fsck** performs a full check after replaying the journal (if present).

e2fsck may ask for user input during the run if the **-p** option is not specified. The **-p** option tells **e2fsck** to automatically do all repairs that may be done safely. If user intervention is required, **e2fsck** indicates the unfixed problem in its output and reflect this status in the exit code.

Commonly used **e2fsck** run-time options include:

-n

No-modify mode. Check-only operation.

-b superblock

Specify block number of an alternate suprerblock if the primary one is damaged.

-f

Force full check even if the superblock has no recorded errors.

-j journal-dev

Specify the external journal device, if any.

-p

Automatically repair or "preen" the file system with no user input.

-y

Assume an answer of "yes" to all questions.

All options for **e2fsck** are specified in the **e2fsck(8)** manual page.

The following five basic phases are performed by **e2fsck** while running:

1. Inode, block, and size checks.

2. Directory structure checks.

3. Directory connectivity checks.

4. Reference count checks.

5. Group summary info checks.

The **e2image(8)** utility can be used to create a metadata image prior to repair for diagnostic or testing purposes. The **-r** option should be used for testing purposes in order to create a sparse file of the same size as the file system itself. **e2fsck** can then operate directly on the resulting file. The **-Q** option should be specified if the image is to be archived or provided for diagnostic. This creates a more compact file format suitable for transfer.

12.2.2. XFS

No repair is performed automatically at boot time. To initiate a file system check or repair, use the **xfs_repair** tool.

> **NOTE**
>
> Although an **fsck.xfs** binary is present in the xfsprogs package, this is present only to satisfy initscripts that look for an **fsck.*file system*** binary at boot time. **fsck.xfs** immediately exits with an exit code of 0.
>
> Older xfsprogs packages contain an **xfs_check** tool. This tool is very slow and does not scale well for large file systems. As such, it has been deprecated in favor of **xfs_repair -n**.

A clean log on a file system is required for **xfs_repair** to operate. If the file system was not cleanly unmounted, it should be mounted and unmounted prior to using **xfs_repair**. If the log is corrupt and cannot be replayed, the **-L** option may be used to zero the log.

> **IMPORTANT**
>
> The **-L** option must only be used if the log cannot be replayed. The option discards all metadata updates in the log and results in further inconsistencies.

It is possible to run **xfs_repair** in a dry run, check-only mode by using the **-n** option. No changes will be made to the file system when this option is specified.

xfs_repair takes very few options. Commonly used options include:

-n

No modify mode. Check-only operation.

-L

Zero metadata log. Use only if log cannot be replayed with mount.

-m maxmem

Limit memory used during run to maxmem MB. 0 can be specified to obtain a rough estimate of the minimum memory required.

-l logdev

Specify the external log device, if present.

All options for **xfs_repair** are specified in the **xfs_repair(8)** manual page.

The following eight basic phases are performed by **xfs_repair** while running:

1. Inode and inode blockmap (addressing) checks.

2. Inode allocation map checks.

3. Inode size checks.

4. Directory checks.

5. Pathname checks.

6. Link count checks.

7. Freemap checks.

8. Super block checks.

For more information, see the **xfs_repair(8)** manual page.

xfs_repair is not interactive. All operations are performed automatically with no input from the user.

If it is desired to create a metadata image prior to repair for diagnostic or testing purposes, the **xfs_metadump(8)** and **xfs_mdrestore(8)** utilities may be used.

12.2.3. Btrfs

> **NOTE**
>
> Btrfs is available as a Technology Preview feature in Red Hat Enterprise Linux 7 but has been deprecated since the Red Hat Enterprise Linux 7.4 release. It will be removed in a future major release of Red Hat Enterprise Linux.
>
> For more information, see Deprecated Functionality in the Red Hat Enterprise Linux 7.4 Release Notes.

The **btrfsck** tool is used to check and repair btrfs file systems. This tool is still in early development and may not detect or repair all types of file system corruption.

By default, **btrfsck** does not make changes to the file system; that is, it runs check-only mode by default. If repairs are desired the **--repair** option must be specified.

The following three basic phases are performed by **btrfsck** while running:

1. Extent checks.

2. File system root checks.

3. Root reference count checks.

The **btrfs-image(8)** utility can be used to create a metadata image prior to repair for diagnostic or testing purposes.

CHAPTER 13. PARTITIONS

With the **parted** utility, you can:

- View the existing partition table.

- Change the size of existing partitions.

- Add partitions from free space or additional hard drives.

The parted package is installed by default on Red Hat Enterprise Linux 7. To start **parted**, log in as root and enter the following command:

```
# parted /dev/sda
```

Replace */dev/sda* with the device name for the drive to configure.

Manipulating Partitions on Devices in Use

For a device to not be in use, none of the partitions on the device can be mounted, and no swap space on the device can be enabled.

If you want to remove or resize a partition, the device on which that partition resides must not be in use.

It is possible to create a new partition on a device that is in use, but this is not recommended.

Modifying the Partition Table

Modifying the partition table while another partition on the same disk is in use is generally not recommended because the kernel is not able to reread the partition table. As a consequence, changes are not applied to a running system. In the described situation, reboot the system, or use the following command to make the system register new or modified partitions:

```
# partx --update --nr partition-number disk
```

The easiest way to modify disks that are currently in use is:

1. Boot the system in rescue mode if the partitions on the disk are impossible to unmount, for example in the case of a system disk.

2. When prompted to mount the file system, select **Skip**.

If the drive does not contain any partitions in use, that is there are no system processes that use or lock the file system from being unmounted, you can unmount the partitions with the **umount** command and turn off all the swap space on the hard drive with the **swapoff** command.

To see commonly used **parted** commands, see Table 13.1, "**parted** Commands".
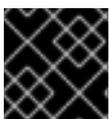
> **IMPORTANT**
>
> Do not use the **parted** utility to create file systems. Use the **mkfs** tool instead.

Table 13.1. parted Commands

Command	Description
help	Display list of available commands
mklabel *label*	Create a disk label for the partition table
mkpart *part-type [fs-type] start-mb end-mb*	Make a partition without creating a new file system
name *minor-num name*	Name the partition for Mac and PC98 disklabels only
print	Display the partition table
quit	Quit **parted**
rescue *start-mb end-mb*	Rescue a lost partition from *start-mb* to *end-mb*
rm *minor-num*	Remove the partition
select *device*	Select a different device to configure
set *minor-num flag state*	Set the flag on a partition; *state* is either on or off
toggle [*NUMBER* [*FLAG*]	Toggle the state of *FLAG* on partition *NUMBER*
unit *UNIT*	Set the default unit to *UNIT*

13.1. VIEWING THE PARTITION TABLE

To view the partition table:

1. Start **parted**.

2. Use the following command to view the partition table:

   ```
   (parted) print
   ```

A table similar to the following one appears:

Example 13.1. Partition Table

```
Model: ATA ST3160812AS (scsi)
Disk /dev/sda: 160GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End    Size   Type     File system  Flags
 1      32.3kB  107MB  107MB  primary  ext3         boot
 2      107MB   105GB  105GB  primary  ext3
```

```
 3       105GB   107GB   2147MB   primary    linux-swap
 4       107GB   160GB   52.9GB   extended         root
 5       107GB   133GB   26.2GB   logical    ext3
 6       133GB   133GB   107MB    logical    ext3
 7       133GB   160GB   26.6GB   logical                  lvm
```

Following is the description of the partition table:

- *Model: ATA ST3160812AS (scsi)*: explains the disk type, manufacturer, model number, and interface.

- *Disk /dev/sda: 160GB*: displays the disk label type.

- In the partition table, **Number** is the partition number. For example, the partition with minor number 1 corresponds to **/dev/sda1**. The **Start** and **End** values are in megabytes. Valid **Types** are metadata, free, primary, extended, or logical. The **File system** is the file system type. The **Flags** column lists the flags set for the partition. Available flags are boot, root, swap, hidden, raid, lvm, or lba.

The **File system** in the partition table can be any of the following:

- ext2

- ext3

- fat16

- fat32

- hfs

- jfs

- linux-swap

- ntfs

- reiserfs

- hp-ufs

- sun-ufs

- xfs

If a **File system** of a device shows no value, this means that its file system type is unknown.

NOTE

To select a different device without having to restart **parted**, use the following command and replace */dev/sda* with the device you want to select:

```
(parted) select /dev/sda
```

It allows you to view or configure the partition table of a device.

13.2. CREATING A PARTITION

WARNING

Do not attempt to create a partition on a device that is in use.

Procedure 13.1. Creating a Partition

1. Before creating a partition, boot into rescue mode, or unmount any partitions on the device and turn off any swap space on the device.

2. Start **parted**:

   ```
   # parted /dev/sda
   ```

   Replace */dev/sda* with the device name on which you want to create the partition.

3. View the current partition table to determine if there is enough free space:

   ```
   (parted) print
   ```

   If there is not enough free space, you can resize an existing partition. For more information, see Section 13.5, "Resizing a Partition with fdisk".

   From the partition table, determine the start and end points of the new partition and what partition type it should be. You can only have four primary partitions, with no extended partition, on a device. If you need more than four partitions, you can have three primary partitions, one extended partition, and multiple logical partitions within the extended. For an overview of disk partitions, see the appendix An Introduction to Disk Partitions in the Red Hat Enterprise Linux 7 Installation Guide.

4. To create partition:

   ```
   (parted) mkpart part-type name fs-type start end
   ```

   Replace *part-type* with with primary, logical, or extended as per your requirement.

   Replace *name* with partition-name; name is required for GPT partition tables.

Replace *fs-type* with any one of btrfs, ext2, ext3, ext4, fat16, fat32, hfs, hfs+, linux-swap, ntfs, reiserfs, or xfs; fs-type is optional.

Replace *start end* with the size in megabytes as per your requirement.

For example, to create a primary partition with an ext3 file system from 1024 megabytes until 2048 megabytes on a hard drive, type the following command:

```
(parted) mkpart primary 1024 2048
```

> **NOTE**
>
> If you use the **mkpartfs** command instead, the file system is created after the partition is created. However, **parted** does not support creating an ext3 file system. Thus, if you wish to create an ext3 file system, use **mkpart** and create the file system with the **mkfs** command as described later.

The changes start taking place as soon as you press **Enter**, so review the command before executing to it.

5. View the partition table to confirm that the created partition is in the partition table with the correct partition type, file system type, and size using the following command:

```
(parted) print
```

Also remember the minor number of the new partition so that you can label any file systems on it.

6. Exit the parted shell:

```
(parted) quit
```

7. Use the following command after parted is closed to make sure the kernel recognizes the new partition:

```
# cat /proc/partitions
```

The maximum number of partitions parted can create is 128. While the *GUID Partition Table* (GPT) specification allows for more partitions by growing the area reserved for the partition table, common practice used by parted is to limit it to enough area for 128 partitions.

13.2.1. Formatting and Labeling the Partition

To format and label the partition use the following procedure:

Procedure 13.2. Format and Label the Partition

1. The partition does not have a file system. To create the **ext4** file system, use:

```
# mkfs.ext4 /dev/sda6
```

> **WARNING**
>
> Formatting the partition permanently destroys any data that currently exists on the partition.

2. Label the file system on the partition. For example, if the file system on the new partition is **/dev/sda6** and you want to label it **Work**, use:

   ```
   # e2label /dev/sda6 "Work"
   ```

   By default, the installation program uses the mount point of the partition as the label to make sure the label is unique. You can use any label you want.

3. Create a mount point (e.g. **/work**) as root.

13.2.2. Add the Partition to `/etc/fstab`

1. As root, edit the **/etc/fstab** file to include the new partition using the partition's UUID.

   Use the command **blkid -o list** for a complete list of the partition's UUID, or **blkid device** for individual device details.

   In **/etc/fstab**:

   - The first column should contain **UUID=** followed by the file system's UUID.

   - The second column should contain the mount point for the new partition.

   - The third column should be the file system type: for example, **ext4** or **swap**.

   - The fourth column lists mount options for the file system. The word **defaults** here means that the partition is mounted at boot time with default options.

   - The fifth and sixth field specify backup and check options. Example values for a non-root partition are **0 2**.

2. Regenerate mount units so that your system registers the new configuration:

   ```
   # systemctl daemon-reload
   ```

3. Try mounting the file system to verify that the configuration works:

   ```
   # mount /work
   ```

Additional Information

- If you need more information about the format of **/etc/fstab**, see the fstab(5) man page.

13.3. REMOVING A PARTITION

> **WARNING**
>
> Do not attempt to remove a partition on a device that is in use.

Procedure 13.3. Remove a Partition

1. Before removing a partition, do one of the following:

   - Boot into rescue mode, or

   - Unmount any partitions on the device and turn off any swap space on the device.

2. Start the **parted** utility:

   ```
   # parted device
   ```

   Replace *device* with the device on which to remove the partition: for example, **/dev/sda**.

3. View the current partition table to determine the minor number of the partition to remove:

   ```
   (parted) print
   ```

4. Remove the partition with the command **rm**. For example, to remove the partition with minor number 3:

   ```
   (parted) rm 3
   ```

   The changes start taking place as soon as you press **Enter**, so review the command before committing to it.

5. After removing the partition, use the **print** command to confirm that it is removed from the partition table:

   ```
   (parted) print
   ```

6. Exit from the **parted** shell:

   ```
   (parted) quit
   ```

7. Examine the content of the **/proc/partitions** file to make sure the kernel knows the partition is removed:

   ```
   # cat /proc/partitions
   ```

8. Remove the partition from the **/etc/fstab** file. Find the line that declares the removed partition, and remove it from the file.

9. Regenerate mount units so that your system registers the new **/etc/fstab** configuration:

```
# systemctl daemon-reload
```

13.4. SETTING A PARTITION TYPE

The partition type, not to be confused with the file system type, is used by a running system only rarely. However, the partition type matters to on-the-fly generators, such as **systemd-gpt-auto-generator**, which use the partition type to, for example, automatically identify and mount devices.

You can start the **fdisk** utility and use the **t** command to set the partition type. The following example shows how to change the partition type of the first partition to 0x83, default on Linux:

```
# fdisk /dev/sdc
Command (m for help): t
Selected partition 1
Partition type (type L to list all types): 83
Changed type of partition 'Linux LVM' to 'Linux'.
```

The **parted** utility provides some control of partition types by trying to map the partition type to 'flags', which is not convenient for end users. The **parted** utility can handle only certain partition types, for example LVM or RAID. To remove, for example, the lvm flag from the first partition with **parted**, use:

```
# parted /dev/sdc 'set 1 lvm off'
```

For a list of commonly used partition types and hexadecimal numbers used to represent them, see the Partition Types table in the Partitions: Turning One Drive Into Many appendix of the *Red Hat Enterprise Linux 7 Installation Guide*.

13.5. RESIZING A PARTITION WITH FDISK

The **fdisk** utility allows you to create and manipulate GPT, MBR, Sun, SGI, and BSD partition tables. On disks with a GUID Partition Table (GPT), using the **parted** utility is recommended, as **fdisk** GPT support is in an experimental phase.

Before resizing a partition, *back up* the data stored on the file system and test the procedure, as the only way to change a partition size using **fdisk** is by deleting and recreating the partition.

> **IMPORTANT**
>
> The partition you are resizing must be the last partition on a particular disk.
>
> Red Hat only supports extending and resizing LVM partitions.

Procedure 13.4. Resizing a Partition

The following procedure is provided only for reference. To resize a partition using **fdisk**:

1. Unmount the device:

```
# umount /dev/vda
```

2. Run **fdisk** *disk_name*. For example:

```
# fdisk /dev/vda
Welcome to fdisk (util-linux 2.23.2).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Command (m for help):
```

3. Use the **p** option to determine the line number of the partition to be deleted.

```
Command (m for help): p
Disk /dev/vda: 16.1 GB, 16106127360 bytes, 31457280 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x0006d09a

Device     Boot      Start         End      Blocks   Id  System
/dev/vda1   *         2048     1026047      512000   83  Linux
/dev/vda2          1026048    31457279    15215616   8e  Linux LVM
```

4. Use the **d** option to delete a partition. If there is more than one partition available, **fdisk** prompts you to provide a number of the partition to delete:

```
Command (m for help): d
Partition number (1,2, default 2): 2
Partition 2 is deleted
```

5. Use the **n** option to create a partition and follow the prompts. Allow enough space for any future resizing. The **fdisk** default behavior (press **Enter**) is to use all space on the device. You can specify the end of the partition by sectors, or specify a human-readable size by using *+<size><suffix>*, for example +500M, or +10G.

   Red Hat recommends using the human-readable size specification if you do not want to use all free space, as **fdisk** aligns the end of the partition with the physical sectors. If you specify the size by providing an exact number (in sectors), **fdisk** does not align the end of the partition.

```
Command (m for help): n
Partition type:
   p   primary (1 primary, 0 extended, 3 free)
   e   extended
Select (default p): *Enter*
Using default response p
Partition number (2-4, default 2): *Enter*
First sector (1026048-31457279, default 1026048): *Enter*
Using default value 1026048
```

```
Last sector, +sectors or +size{K,M,G} (1026048-31457279, default
31457279): +500M
Partition 2 of type Linux and of size 500 MiB is set
```

6. Set the partition type to LVM:

```
Command (m for help): t
Partition number (1,2, default 2): *Enter*
Hex code (type L to list all codes): 8e
Changed type of partition 'Linux' to 'Linux LVM'
```

7. Write the changes with the **w** option when you are sure the changes are correct, as errors can cause instability with the selected partition.

8. Run **e2fsck** on the device to check for consistency:

```
# e2fsck /dev/vda
e2fsck 1.41.12 (17-May-2010)
Pass 1:Checking inodes, blocks, and sizes
Pass 2:Checking directory structure
Pass 3:Checking directory connectivity
Pass 4:Checking reference counts
Pass 5:Checking group summary information
ext4-1:11/131072 files (0.0% non-contiguous),27050/524128 blocks
```

9. Mount the device:

```
# mount /dev/vda
```

For more information, see the fdisk(8) manual page.

CHAPTER 14. CREATING AND MAINTAINING SNAPSHOTS WITH SNAPPER

A snapshot volume is a point in time copy of a target volume that provides a way to revert a file system back to an earlier state. Snapper is a command-line tool to create and maintain snapshots for Btrfs and thinly-provisioned LVM file systems.

14.1. CREATING INITIAL SNAPPER CONFIGURATION

Snapper requires discrete configuration files for each volume it operates on. You must set up the configuration files manually. By default, only the root user is allowed to perform snapper commands.

The file system recommended by Red Hat with Snapper depends on your Red Hat Enterprise Linux version:

- In Red Hat Enterprise Linux 7.4 or earlier versions of Red Hat Enterprise Linux 7, use ext4 with Snapper. Use the XFS file system on lvm-thin volumes only if you are monitoring the amount of free space in the pool to prevent out-of-space problems that can lead to a failure.

- In Red Hat Enterprise Linux 7.5 or later versions, use XFS with Snapper.

Note that the Btrfs tools and file system are provided as a Technology Preview, which make them unsuitable for production systems.

Although it is possible to allow a user or group other than root to use certain Snapper commands, Red Hat recommends that you do *not* add elevated permissions to otherwise unprivileged users or groups. Such a configuration bypasses SELinux and could pose a security risk. Red Hat recommends that you review these capabilities with your Security Team and consider using the **sudo** infrastructure instead.

> **NOTE**
>
> Btrfs is available as a Technology Preview feature in Red Hat Enterprise Linux 7 but has been deprecated since the Red Hat Enterprise Linux 7.4 release. It will be removed in a future major release of Red Hat Enterprise Linux.
>
> For more information, see Deprecated Functionality in the Red Hat Enterprise Linux 7.4 Release Notes.

Procedure 14.1. Creating a Snapper Configuration File

1. Create or choose either:

   - A thinly-provisioned logical volume with a Red Hat supported file system on top of it, or

   - A Btrfs subvolume.

2. Mount the file system.

3. Create the configuration file that defines this volume.

   For LVM2:

```
# snapper -c config_name create-config -f "lvm(fs_type)" /mount-
point
```

For example, to create a configuration file called lvm_config on an LVM2 subvolume with an ext4 file system, mounted at /lvm_mount, use:

```
# snapper -c lvm_config create-config -f "lvm(ext4)" /lvm_mount
```

For Btrfs:

```
# snapper -c config_name create-config -f btrfs /mount-point
```

- The **-c config_name** option specifies the name of the configuration file.

- The **create-config** tells snapper to create a configuration file.

- The **-f file_system** tells snapper what file system to use; if this is omitted snapper will attempt to detect the file system.

- The **/mount-point** is where the subvolume or thinly-provisioned LVM2 file system is mounted.

Alternatively, to create a configuration file called **btrfs_config**, on a Btrfs subvolume that is mounted at **/btrfs_mount**, use:

```
# snapper -c btrfs_config create-config -f btrfs /btrfs_mount
```

The configuration files are stored in the **/etc/snapper/configs/** directory.

14.2. CREATING A SNAPPER SNAPSHOT

Snapper can create the following kinds of snapshots:

Pre Snapshot

A pre snapshot serves as a point of origin for a post snapshot. The two are closely tied and designed to track file system modification between the two points. The pre snapshot must be created before the post snapshot.

Post Snapshot

A post snapshot serves as the end point to the pre snapshot. The coupled pre and post snapshots define a range for comparison. By default, every new snapper volume is configured to create a background comparison after a related post snapshot is created successfully.

Single Snapshot

A single snapshot is a standalone snapshot created at a specific moment. These can be used to track a timeline of modifications and have a general point to return to later.

14.2.1. Creating a Pre and Post Snapshot Pair

14.2.1.1. Creating a Pre Snapshot with Snapper

To create a pre snapshot, use:

```
# snapper -c config_name create -t pre
```

The **-c config_name** option creates a snapshot according to the specifications in the named configuration file. If the configuration file does not yet exist, see Section 14.1, "Creating Initial Snapper Configuration".

The **create -t** option specifies what type of snapshot to create. Accepted entries are **pre**, **post**, or **single**.

For example, to create a pre snapshot using the **lvm_config** configuration file, as created in Section 14.1, "Creating Initial Snapper Configuration", use:

```
# snapper -c SnapperExample create -t pre -p
1
```

The **-p** option prints the number of the created snapshot and is optional.

14.2.1.2. Creating a Post Snapshot with Snapper

A post snapshot is the end point of the snapshot and should be created after the parent pre snapshot by following the instructions in Section 14.2.1.1, "Creating a Pre Snapshot with Snapper".

Procedure 14.2. Creating a Post Snapshot

1. Determine the number of the pre snapshot:

   ```
   # snapper -c config_name list
   ```

   For example, to display the list of snapshots created using the configuration file **lvm_config**, use the following:

   ```
   # snapper -c lvm_config list
   Type   | # | Pre # | Date               | User | Cleanup   |
   Description | Userdata
   -------+---+-------+------------------+------+---------+-------
   ------+---------
   single | 0 |       |                    | root |           | current
   |
   pre    | 1 |       | Mon 06<...>        | root |           |
   |
   ```

   This output shows that the pre snapshot is number 1.

2. Create a post snapshot that is linked to a previously created pre snapshot:

   ```
   # snapper -c config_file create -t post --pre-num
   pre_snapshot_number
   ```

   - The **-t post** option specifies the creation of the post snapshot type.

   - The **--pre-num** option specifies the corresponding pre snapshot.

For example, to create a post snapshot using the **lvm_config** configuration file and is linked to pre snapshot number 1, use:

```
# snapper -c lvm_config create -t post --pre-num 1 -p
2
```

The **-p** option prints the number of the created snapshot and is optional.

3. The pre and post snapshots 1 and 2 are now created and paired. Verify this with the **list** command:

```
# snapper -c lvm_config list
Type   | # | Pre # | Date              | User | Cleanup  |
Description | Userdata
-------+---+-------+------------------+------+---------+-------
------+---------
single | 0 |       |                  | root |          | current
|
pre    | 1 |       | Mon 06<...>      | root |          |
|
post   | 2 | 1     | Mon 06<...>      | root |          |
|
```

14.2.1.3. Wrapping a Command in Pre and Post Snapshots

You can also wrap a command within a pre and post snapshot, which can be useful when testing. See Procedure 14.3, "Wrapping a Command in Pre and Post Snapshots", which is a shortcut for the following steps:

1. Running the **snapper create pre snapshot** command.

2. Running a command or a list of commands to perform actions with a possible impact on the file system content.

3. Running the **snapper create post snapshot** command.

Procedure 14.3. Wrapping a Command in Pre and Post Snapshots

1. To wrap a command in pre and post snapshots:

```
# snapper -c lvm_config create --command "command_to_be_tracked"
```

For example, to track the creation of the **/lvm_mount/hello_file** file:

```
# snapper -c lvm_config create --command "echo Hello >
/lvm_mount/hello_file"
```

2. To verify this, use the **status** command:

```
# snapper -c config_file status
first_snapshot_number..second_snapshot_number
```

For example, to track the changes made in the first step:

```
# snapper -c lvm_config status 3..4
+..... /lvm_mount/hello_file
```

Use the **list** command to verify the number of the snapshot if needed.

For more information on the **status** command, see Section 14.3, "Tracking Changes Between Snapper Snapshots".

Note that there is no guarantee that the command in the given example is the only thing the snapshots capture. Snapper also records anything that is modified by the system, not just what a user modifies.

14.2.2. Creating a Single Snapper Snapshot

Creating a single snapper snapshot is similar to creating a pre or post snapshot, only the create **-t** option specifies single. The single snapshot is used to create a single snapshot in time without having it relate to any others. However, if you are interested in a straightforward way to create snapshots of LVM2 thin volumes without the need to automatically generate comparisons or list additional information, Red Hat recommends using the System Storage Manager instead of Snapper for this purpose, as described in Section 16.2.6, "Snapshot".

To create a single snapshot, use:

```
# snapper -c config_name create -t single
```

For example, the following command creates a single snapshot using the **lvm_config** configuration file.

```
# snapper -c lvm_config create -t single
```

Although single snapshots are not specifically designed to track changes, you can use the **snapper diff**, **xadiff**, and **status** commands to compare any two snapshots. For more information on these commands, see Section 14.3, "Tracking Changes Between Snapper Snapshots".

14.2.3. Configuring Snapper to Take Automated Snapshots

Taking automated snapshots is one of key features of Snapper. By default, when you configure Snapper for a volume, Snapper starts taking a snapshot of the volume every hour.

Under the default configuration, Snapper keeps:

- 10 hourly snapshots, and the final hourly snapshot is saved as a "daily" snapshot.

- 10 daily snapshots, and the final daily snapshot for a month is saved as a "monthly" snapshot.

- 10 monthly snapshots, and the final monthly snapshot is saved as a "yearly" snapshot.

- 10 yearly snapshots.

Note that Snapper keeps by default no more that 50 snapshots in total. However, Snapper keeps by default all snapshots created less than 1,800 seconds ago.

The default configuration is specified in the **/etc/snapper/config-templates/default** file. When you use the **snapper create-config** command to create a configuration, any unspecified values are set based on the default configuration. You can edit the configuration for any defined volume in the **/etc/snapper/configs/config_name** file.

14.3. TRACKING CHANGES BETWEEN SNAPPER SNAPSHOTS

Use the `status`, `diff`, and `xadiff` commands to track the changes made to a subvolume between snapshots:

status

The `status` command shows a list of files and directories that have been created, modified, or deleted between two snapshots, that is a comprehensive list of changes between two snapshots. You can use this command to get an overview of the changes without excessive details.

For more information, see Section 14.3.1, "Comparing Changes with the `status` Command".

diff

The `diff` command shows a diff of modified files and directories between two snapshots as received from the `status` command if there is at least one modification detected.

For more information, see Section 14.3.2, "Comparing Changes with the `diff` Command".

xadiff

The `xadiff` command compares how the extended attributes of a file or directory have changed between two snapshots.

For more information, see Section 14.3.3, "Comparing Changes with the `xadiff` Command".

14.3.1. Comparing Changes with the `status` Command

The `status` command shows a list of files and directories that have been created, modified, or deleted between two snapshots.

To display the status of files between two snapshots, use:

```
# snapper -c config_file status
first_snapshot_number..second_snapshot_number
```

Use the `list` command to determine snapshot numbers if needed.

For example, the following command displays the changes made between snapshot 1 and 2, using the configuration file **lvm_config**.

```
#snapper -c lvm_config status 1..2
tp.... /lvm_mount/dir1
-..... /lvm_mount/dir1/file_a
c.ug.. /lvm_mount/file2
+..... /lvm_mount/file3
....x. /lvm_mount/file4
cp..xa /lvm_mount/file5
```

Read letters and dots in the first part of the output as columns:

```
+..... /lvm_mount/file3
||||||
123456
```

■

Column 1 indicates any modification of the file (directory entry) type. Possible values are:

Column 1

Output	Meaning
.	Nothing has changed.
+	File created.
-	File deleted.
c	Content changed.
t	The type of directory entry has changed. For example, a former symbolic link has changed to a regular file with the same file name.

Column 2 indicates any changes in the file permissions. Possible values are:

Column 2

Output	Meaning
.	No permissions changed.
p	Permissions changed.

Column 3 indicates any changes in the user ownership. Possible values are:

Column 3

Output	Meaning
.	No user ownership changed.
u	User ownership has changed.

Column 4 indicates any changes in the group ownership. Possible values are:

Column 4

Output	Meaning
.	No group ownership changed.
g	Group ownership has changed.

Column 5 indicates any changes in the extended attributes. Possible values are:

Column 5

Output	Meaning
.	No extended attributes changed.
x	Extended attributes changed.

Column 6 indicates any changes in the access control lists (ACLs). Possible values are:

Column 6

Output	Meaning
.	No ACLs changed.
a	ACLs modified.

14.3.2. Comparing Changes with the `diff` Command

The **diff** command shows the changes of modified files and directories between two snapshots.

```
# snapper -c config_name diff
first_snapshot_number..second_snapshot_number
```

Use the **list** command to determine the number of the snapshot if needed.

For example, to compare the changes made in files between snapshot 1 and snapshot 2 that were made using the **lvm_config** configuration file, use:

```
# snapper -c lvm_config diff 1..2
--- /lvm_mount/.snapshots/13/snapshot/file4 19<...>
+++ /lvm_mount/.snapshots/14/snapshot/file4 20<...>
@@ -0,0 +1 @@
+words
```

This output shows that **file4** had been modified to add "words" into the file.

14.3.3. Comparing Changes with the `xadiff` Command

The **xadiff** command compares how the extended attributes of a file or directory have changed between two snapshots:

```
# snapper -c config_name xadiff
first_snapshot_number..second_snapshot_number
```

Use the **list** command to determine the number of the snapshot if needed.

For example, to show the xadiff output between snapshot number 1 and snapshot number 2 that were made using the **lvm_config** configuration file, use:

```
# snapper -c lvm_config xadiff 1..2
```

14.4. REVERSING CHANGES IN BETWEEN SNAPSHOTS

To reverse changes made between two existing Snapper snapshots, use the **undochange** command in the following format, where **1** is the first snapshot and **2** is the second snapshot:

```
snapper -c config_name undochange 1..2
```

> **IMPORTANT**
>
> Using the **undochange** command does not revert the Snapper volume back to its original state and does not provide data consistency. Any file modification that occurs outside of the specified range, for example after snapshot 2, will remain unchanged after reverting back, for example to the state of snapshot 1. For example, if **undochange** is run to undo the creation of a user, any files owned by that user can still remain.
>
> There is also no mechanism to ensure file consistency as a snapshot is made, so any inconsistencies that already exist can be transferred back to the snapshot when the **undochange** command is used.
>
> Do not use the Snapper **undochange** command with the root file system, as doing so is likely to lead to a failure.

The following diagram demonstrates how the **undochange** command works:
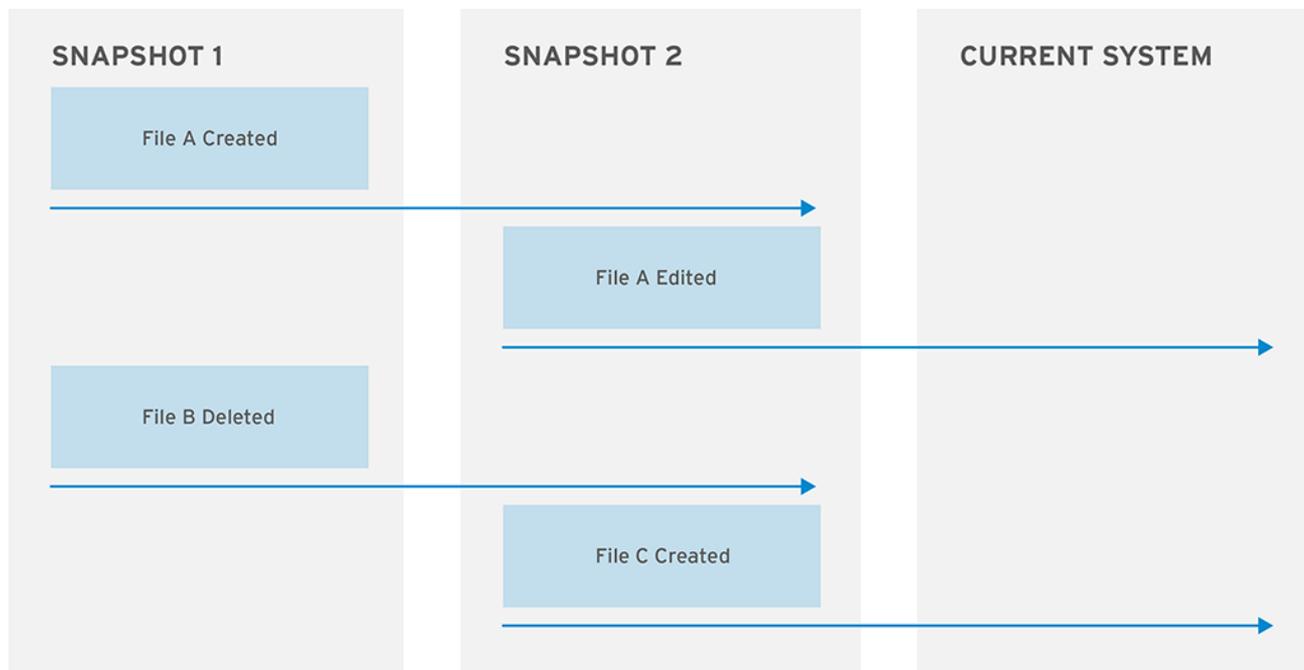
RHEL_387527_0125

Figure 14.1. Snapper Status over Time

The diagram shows the point in time in which **snapshot_1** is created, **file_a** is created, then **file_b** deleted. **Snapshot_2** is then created, after which **file_a** is edited and **file_c** is created. This is now the current state of the system. The current system has an edited version of **file_a**, no **file_b**, and a newly created **file_c**.

When the **undochange** command is called, Snapper generates a list of modified files between the first listed snapshot and the second. In the diagram, if you use the **snapper -c SnapperExample undochange 1..2** command, Snapper creates a list of modified files (that is, **file_a** is created; **file_b** is deleted) and applies them to the current system. Therefore:

- the current system will not have **file_a**, as it has yet to be created when **snapshot_1** was created.

- **file_b** will exist, copied from **snapshot_1** into the current system.

- **file_c** will exist, as its creation was outside the specified time.

Be aware that if **file_b** and **file_c** conflict, the system can become corrupted.

You can also use the **snapper -c SnapperExample undochange 2..1** command. In this case, the current system replaces the edited version of **file_a** with one copied from **snapshot_1**, which undoes edits of that file made after **snapshot_2** was created.

Using the mount and unmount Commands to Reverse Changes

The **undochange** command is not always the best way to revert modifications. With the **status** and **diff** command, you can make a qualified decision, and use the **mount** and **unmount** commands instead of Snapper. The **mount** and **unmount** commands are only useful if you want to mount snapshots and browse their content independently of Snapper workflow.

If needed, the **mount** command activates respective LVM Snapper snapshot before mounting. Use the **mount** and **unmount** commands if you are, for example, interested in mounting snapshots and extracting older version of several files manually. To revert files manually, copy them from a mounted

snapshot to the current file system. The current file system, snapshot 0, is the live file system created in Procedure 14.1, "Creating a Snapper Configuration File". Copy the files to the subtree of the original /mount-point.

Use the **mount** and **unmount** commands for explicit client-side requests. The **/etc/snapper/configs/*config_name*** file contains the ALLOW_USERS= and ALLOW_GROUPS= variables where you can add users and groups. Then, **snapperd** allows you to perform mount operations for the added users and groups.

14.5. DELETING A SNAPPER SNAPSHOT

To delete a snapshot:

```
# snapper -c config_name delete snapshot_number
```

You can use the **list** command to verify that the snapshot was successfully deleted.

CHAPTER 15. SWAP SPACE

Swap space in Linux is used when the amount of physical memory (RAM) is full. If the system needs more memory resources and the RAM is full, inactive pages in memory are moved to the swap space. While swap space can help machines with a small amount of RAM, it should not be considered a replacement for more RAM. Swap space is located on hard drives, which have a slower access time than physical memory. Swap space can be a dedicated swap partition (recommended), a swap file, or a combination of swap partitions and swap files. Note that *Btrfs* does *not* support swap space.

In years past, the recommended amount of swap space increased linearly with the amount of RAM in the system. However, modern systems often include hundreds of gigabytes of RAM. As a consequence, recommended swap space is considered a function of system memory workload, not system memory.

Table 15.1, "Recommended System Swap Space" illustrates the recommended size of a swap partition depending on the amount of RAM in your system and whether you want sufficient memory for your system to hibernate. The recommended swap partition size is established automatically during installation. To allow for hibernation, however, you need to edit the swap space in the custom partitioning stage.

Recommendations in Table 15.1, "Recommended System Swap Space" are especially important on systems with low memory (1 GB and less). Failure to allocate sufficient swap space on these systems can cause issues such as instability or even render the installed system unbootable.

Table 15.1. Recommended System Swap Space

Amount of RAM in the system	Recommended swap space	Recommended swap space if allowing for hibernation
⩽ 2 GB	2 times the amount of RAM	3 times the amount of RAM
> 2 GB – 8 GB	Equal to the amount of RAM	2 times the amount of RAM
> 8 GB – 64 GB	At least 4 GB	1.5 times the amount of RAM
> 64 GB	At least 4 GB	Hibernation not recommended

At the border between each range listed in Table 15.1, "Recommended System Swap Space", for example a system with 2 GB, 8 GB, or 64 GB of system RAM, discretion can be exercised with regard to chosen swap space and hibernation support. If your system resources allow for it, increasing the swap space may lead to better performance. A swap space of at least 100 GB is recommended for systems with over 140 logical processors or over 3 TB of RAM.

Note that distributing swap space over multiple storage devices also improves swap space performance, particularly on systems with fast drives, controllers, and interfaces.

IMPORTANT

File systems and LVM2 volumes assigned as swap space *should not* be in use when being modified. Any attempts to modify swap fail if a system process or the kernel is using swap space. Use the `free` and `cat /proc/swaps` commands to verify how much and where swap is in use.

You should modify swap space while the system is booted in `rescue` mode, see Booting Your Computer in Rescue Mode in the *Red Hat Enterprise Linux 7 Installation Guide*. When prompted to mount the file system, select `Skip`.

15.1. ADDING SWAP SPACE

Sometimes it is necessary to add more swap space after installation. For example, you may upgrade the amount of RAM in your system from 1 GB to 2 GB, but there is only 2 GB of swap space. It might be advantageous to increase the amount of swap space to 4 GB if you perform memory-intense operations or run applications that require a large amount of memory.

You have three options: create a new swap partition, create a new swap file, or extend swap on an existing LVM2 logical volume. It is recommended that you extend an existing logical volume.

15.1.1. Extending Swap on an LVM2 Logical Volume

By default, Red Hat Enterprise Linux 7 uses all available space during installation. If this is the case with your system, then you must first add a new physical volume to the volume group used by the swap space.

After adding additional storage to the swap space's volume group, it is now possible to extend it. To do so, perform the following procedure (assuming `/dev/VolGroup00/LogVol01` is the volume you want to extend by 2 GB):

Procedure 15.1. Extending Swap on an LVM2 Logical Volume

1. Disable swapping for the associated logical volume:

   ```
   # swapoff -v /dev/VolGroup00/LogVol01
   ```

2. Resize the LVM2 logical volume by 2 GB:

   ```
   # lvresize /dev/VolGroup00/LogVol01 -L +2G
   ```

3. Format the new swap space:

   ```
   # mkswap /dev/VolGroup00/LogVol01
   ```

4. Enable the extended logical volume:

   ```
   # swapon -v /dev/VolGroup00/LogVol01
   ```

5. To test if the swap logical volume was successfully extended and activated, inspect active swap space:

```
$ cat /proc/swaps
$ free -h
```

15.1.2. Creating an LVM2 Logical Volume for Swap

To add a swap volume group 2 GB in size, assuming **/dev/VolGroup00/LogVol02** is the swap volume you want to add:

1. Create the LVM2 logical volume of size 2 GB:

   ```
   # lvcreate VolGroup00 -n LogVol02 -L 2G
   ```

2. Format the new swap space:

   ```
   # mkswap /dev/VolGroup00/LogVol02
   ```

3. Add the following entry to the **/etc/fstab** file:

   ```
   /dev/VolGroup00/LogVol02    swap      swap     defaults     0 0
   ```

4. Regenerate mount units so that your system registers the new configuration:

   ```
   # systemctl daemon-reload
   ```

5. Activate swap on the logical volume:

   ```
   # swapon -v /dev/VolGroup00/LogVol02
   ```

6. To test if the swap logical volume was successfully created and activated, inspect active swap space:

   ```
   $ cat /proc/swaps
   $ free -h
   ```

15.1.3. Creating a Swap File

To add a swap file:

Procedure 15.2. Add a Swap File

1. Determine the size of the new swap file in megabytes and multiply by 1024 to determine the number of blocks. For example, the block size of a 64 MB swap file is 65536.

2. Create an empty file:

   ```
   # dd if=/dev/zero of=/swapfile bs=1024 count=65536
   ```

   Replace *count* with the value equal to the desired block size.

3. Set up the swap file with the command:

```
# mkswap /swapfile
```

4. Change the security of the swap file so it is not world readable.

```
# chmod 0600 /swapfile
```

5. To enable the swap file at boot time, edit **/etc/fstab** as root to include the following entry:

```
/swapfile          swap          swap    defaults        0 0
```

The next time the system boots, it activates the new swap file.

6. Regenerate mount units so that your system registers the new **/etc/fstab** configuration:

```
# systemctl daemon-reload
```

7. To activate the swap file immediately:

```
# swapon /swapfile
```

8. To test if the new swap file was successfully created and activated, inspect active swap space:

```
$ cat /proc/swaps
$ free -h
```

15.2. REMOVING SWAP SPACE

Sometimes it can be prudent to reduce swap space after installation. For example, you have downgraded the amount of RAM in your system from 1 GB to 512 MB, but there is 2 GB of swap space still assigned. It might be advantageous to reduce the amount of swap space to 1 GB, since the larger 2 GB could be wasting disk space.

You have three options: remove an entire LVM2 logical volume used for swap, remove a swap file, or reduce swap space on an existing LVM2 logical volume.

15.2.1. Reducing Swap on an LVM2 Logical Volume

To reduce an LVM2 swap logical volume (assuming **/dev/VolGroup00/LogVol01** is the volume you want to reduce):

Procedure 15.3. Reducing an LVM2 Swap Logical Volume

1. Disable swapping for the associated logical volume:

```
# swapoff -v /dev/VolGroup00/LogVol01
```

2. Reduce the LVM2 logical volume by 512 MB:

```
# lvreduce /dev/VolGroup00/LogVol01 -L -512M
```

3. Format the new swap space:

```
# mkswap /dev/VolGroup00/LogVol01
```

4. Activate swap on the logical volume:

```
# swapon -v /dev/VolGroup00/LogVol01
```

5. To test if the swap logical volume was successfully reduced, inspect active swap space:

```
$ cat /proc/swaps
$ free -h
```

15.2.2. Removing an LVM2 Logical Volume for Swap

To remove a swap volume group (assuming **/dev/VolGroup00/LogVol02** is the swap volume you want to remove):

Procedure 15.4. Remove a Swap Volume Group

1. Disable swapping for the associated logical volume:

```
# swapoff -v /dev/VolGroup00/LogVol02
```

2. Remove the LVM2 logical volume:

```
# lvremove /dev/VolGroup00/LogVol02
```

3. Remove the following associated entry from the **/etc/fstab** file:

```
/dev/VolGroup00/LogVol02   swap      swap     defaults     0 0
```

4. Regenerate mount units so that your system registers the new configuration:

```
# systemctl daemon-reload
```

5. To test if the logical volume was successfully removed, inspect active swap space:

```
$ cat /proc/swaps
$ free -h
```

15.2.3. Removing a Swap File

To remove a swap file:

Procedure 15.5. Remove a Swap File

1. At a shell prompt, execute the following command to disable the swap file (where **/swapfile** is the swap file):

```
# swapoff -v /swapfile
```

2. Remove its entry from the **/etc/fstab** file.

3. Regenerate mount units so that your system registers the new configuration:

   ```
   # systemctl daemon-reload
   ```

4. Remove the actual file:

   ```
   # rm /swapfile
   ```

15.3. MOVING SWAP SPACE

To move swap space from one location to another:

1. Removing swap space Section 15.2, "Removing Swap Space".

2. Adding swap space Section 15.1, "Adding Swap Space".

CHAPTER 16. SYSTEM STORAGE MANAGER (SSM)

System Storage Manager (SSM) provides a command line interface to manage storage in various technologies. Storage systems are becoming increasingly complicated through the use of Device Mappers (DM), Logical Volume Managers (LVM), and Multiple Devices (MD). This creates a system that is not user friendly and makes it easier for errors and problems to arise. SSM alleviates this by creating a unified user interface. This interface allows users to run complicated systems in a simple manner. For example, to create and mount a new file system without SSM, there are five commands that must be used. With SSM only one is needed.

This chapter explains how SSM interacts with various back ends and some common use cases.

16.1. SSM BACK ENDS

SSM uses a core abstraction layer in **ssmlib/main.py** which complies with the device, pool, and volume abstraction, ignoring the specifics of the underlying technology. Back ends can be registered in **ssmlib/main.py** to handle specific storage technology methods, such as **create**, **snapshot**, or to **remove** volumes and pools.

There are already several back ends registered in SSM. The following sections provide basic information on them as well as definitions on how they handle pools, volumes, snapshots, and devices.

16.1.1. Btrfs Back End

> **NOTE**
>
> Btrfs is available as a Technology Preview feature in Red Hat Enterprise Linux 7 but has been deprecated since the Red Hat Enterprise Linux 7.4 release. It will be removed in a future major release of Red Hat Enterprise Linux.
>
> For more information, see Deprecated Functionality in the Red Hat Enterprise Linux 7.4 Release Notes.

Btrfs, a file system with many advanced features, is used as a volume management back end in SSM. Pools, volumes, and snapshots can be created with the Btrfs back end.

16.1.1.1. Btrfs Pool

The Btrfs file system itself is the pool. It can be extended by adding more devices or shrunk by removing devices. SSM creates a Btrfs file system when a Btrfs pool is created. This means that every new Btrfs pool has one volume of the same name as the pool which cannot be removed without removing the entire pool. The default Btrfs pool name is **btrfs_pool**.

The name of the pool is used as the file system label. If there is already an existing Btrfs file system in the system without a label, the Btrfs pool will generate a name for internal use in the format of **btrfs_*device_base_name***.

16.1.1.2. Btrfs Volume

Volumes created after the first volume in a pool are the same as sub-volumes. SSM temporarily mounts the Btrfs file system if it is unmounted in order to create a sub-volume.

The name of a volume is used as the subvolume path in the Btrfs file system. For example, a subvolume displays as **/dev/lvm_pool/lvol001**. Every object in this path must exist in order for the volume to be created. Volumes can also be referenced with its mount point.

16.1.1.3. Btrfs Snapshot

Snapshots can be taken of any Btrfs volume in the system with SSM. Be aware that Btrfs does not distinguish between subvolumes and snapshots. While this means that SSM cannot recognize the Btrfs snapshot destination, it will try to recognize special name formats. If the name specified when creating the snapshot does the specific pattern, the snapshot is not be recognized and instead be listed as a regular Btrfs volume.

16.1.1.4. Btrfs Device

Btrfs does not require any special device to be created on.

16.1.2. LVM Back End

Pools, volumes, and snapshots can be created with LVM. The following definitions are from an LVM point of view.

16.1.2.1. LVM Pool

LVM pool is the same as an LVM volume group. This means that grouping devices and new logical volumes can be created out of the LVM pool. The default LVM pool name is **lvm_pool**.

16.1.2.2. LVM Volume

An LVM volume is the same as an ordinary logical volume.

16.1.2.3. LVM Snapshot

When a snapshot is created from the LVM volume a new **snapshot** volume is created which can then be handled just like any other LVM volume. Unlike Btrfs, LVM is able to distinguish snapshots from regular volumes so there is no need for a snapshot name to match a particular pattern.

16.1.2.4. LVM Device

SSM makes the need for an LVM back end to be created on a physical device transparent for the user.

16.1.3. Crypt Back End

The crypt back end in SSM uses **cryptsetup** and **dm-crypt target** to manage encrypted volumes. Crypt back ends can be used as a regular back end for creating encrypted volumes on top of regular block devices (or on other volumes such as LVM or MD volumes), or to create encrypted LVM volumes in a single steps.

Only volumes can be created with a crypt back end; pooling is not supported and it does not require special devices.

The following sections define volumes and snapshots from the crypt point of view.

16.1.3.1. Crypt Volume

Crypt volumes are created by **dm-crypt** and represent the data on the original encrypted device in an unencrypted form. It does not support RAID or any device concatenation.

Two modes, or extensions, are supported: luks and plain. Luks is used by default. For more information on the extensions, see **man cryptsetup**.

16.1.3.2. Crypt Snapshot

While the crypt back end does not support snapshotting, if the encrypted volume is created on top of an LVM volume, the volume itself can be snapshotted. The snapshot can then be opened by using **cryptsetup**.

16.1.4. Multiple Devices (MD) Back End

MD back end is currently limited to only gathering the information about MD volumes in the system.

16.2. COMMON SSM TASKS

The following sections describe common SSM tasks.

16.2.1. Installing SSM

To install SSM use the following command:

```
# yum install system-storage-manager
```

There are several back ends that are enabled only if the supporting packages are installed:

- The LVM back end requires the **lvm2** package.

- The Btrfs back end requires the **btrfs-progs** package.

- The Crypt back end requires the **device-mapper** and **cryptsetup** packages.

16.2.2. Displaying Information about All Detected Devices

Displaying information about all detected devices, pools, volumes, and snapshots is done with the **list** command. The **ssm list** command with no options display the following output:

```
# ssm list
----------------------------------------------------------
Device          Free      Used       Total  Pool  Mount point
----------------------------------------------------------
/dev/sda                             2.00 GB       PARTITIONED
/dev/sda1                           47.83 MB       /test
/dev/vda                            15.00 GB       PARTITIONED
/dev/vda1                          500.00 MB       /boot
/dev/vda2  0.00 KB  14.51 GB   14.51 GB  rhel
----------------------------------------------------------
-------------------------------------------------
Pool  Type  Devices     Free      Used      Total
-------------------------------------------------
rhel  lvm   1        0.00 KB  14.51 GB  14.51 GB
```

```
---------------------------------------------
------------------------------------------------------------------------
-----------
Volume           Pool  Volume size  FS     FS size       Free  Type
Mount point
------------------------------------------------------------------------
-----------
/dev/rhel/root  rhel      13.53 GB  xfs    13.52 GB    9.64 GB  linear  /
/dev/rhel/swap  rhel   1000.00 MB                               linear
/dev/sda1                 47.83 MB  xfs    44.50 MB   44.41 MB  part
/test
/dev/vda1                500.00 MB  xfs   496.67 MB  403.56 MB  part
/boot
------------------------------------------------------------------------
-----------
```

This display can be further narrowed down by using arguments to specify what should be displayed. The list of available options can be found with the **ssm list --help** command.

> **NOTE**
>
> Depending on the argument given, SSM may not display everything.
>
> - Running the **devices** or **dev** argument omits some devices. CDRoms and DM/MD devices, for example, are intentionally hidden as they are listed as volumes.
>
> - Some back ends do not support snapshots and cannot distinguish between a snapshot and a regular volume. Running the **snapshot** argument on one of these back ends cause SSM to attempt to recognize the volume name in order to identify a snapshot. If the SSM regular expression does not match the snapshot pattern then the snapshot is not be recognized.
>
> - With the exception of the main Btrfs volume (the file system itself), any unmounted Btrfs volumes are not shown.

16.2.3. Creating a New Pool, Logical Volume, and File System

In this section, a new pool is be created with a default name which have the devices **/dev/vdb** and **/dev/vdc**, a logical volume of 1G, and an XFS file system.

The command to create this scenario is **ssm create --fs xfs -s 1G /dev/vdb /dev/vdc**. The following options are used:

- The **--fs** option specifies the required file system type. Current supported file system types are:

  - ext3

  - ext4

  - xfs

  - btrfs

- The **-s** specifies the size of the logical volume. The following suffixes are supported to define units:

    - **K** or **k** for kilobytes

    - **M** or **m** for megabytes

    - **G** or **g** for gigabytes

    - **T** or **t** for terabytes

    - **P** or **p** for petabytes

    - **E** or **e** for exabytes

- The two listed devices, **/dev/vdb** and **/dev/vdc**, are the two devices you wish to create.

```
# ssm create --fs xfs -s 1G /dev/vdb /dev/vdc
  Physical volume "/dev/vdb" successfully created
  Physical volume "/dev/vdc" successfully created
  Volume group "lvm_pool" successfully created
  Logical volume "lvol001" created
```

There are two other options for the **ssm command** that may be useful. The first is the **-p** *pool* command. This specifies the pool the volume is to be created on. If it does not yet exist, then SSM creates it. This was omitted in the given example which caused SSM to use the default name **lvm_pool**. However, to use a specific name to fit in with any existing naming conventions, the **-p** option should be used.

The second useful option is the **-n** *name* command. This names the newly created logical volume. As with the **-p**, this is needed in order to use a specific name to fit in with any existing naming conventions.

An example of these two options being used follows:

```
# ssm create --fs xfs -p new_pool -n XFS_Volume /dev/vdd
  Volume group "new_pool" successfully created
  Logical volume "XFS_Volume" created
```

SSM has now created two physical volumes, a pool, and a logical volume with the ease of only one command.

16.2.4. Checking a File System's Consistency

The **ssm check** command checks the file system consistency on the volume. It is possible to specify multiple volumes to check. If there is no file system on the volume, then the volume is skipped.

To check all devices in the volume **lvol001**, run the command **ssm check /dev/lvm_pool/lvol001**.

```
# ssm check /dev/lvm_pool/lvol001
Checking xfs file system on '/dev/mapper/lvm_pool-lvol001'.
Phase 1 - find and verify superblock...
Phase 2 - using internal log
        - scan filesystem freespace and inode maps...
```

```
              - found root inode chunk
Phase 3 - for each AG...
              - scan (but don't clear) agi unlinked lists...
              - process known inodes and perform inode discovery...
              - agno = 0
              - agno = 1
              - agno = 2
              - agno = 3
              - agno = 4
              - agno = 5
              - agno = 6
              - process newly discovered inodes...
Phase 4 - check for duplicate blocks...
              - setting up duplicate extent list...
              - check for inodes claiming duplicate blocks...
              - agno = 0
              - agno = 1
              - agno = 2
              - agno = 3
              - agno = 4
              - agno = 5
              - agno = 6
No modify flag set, skipping phase 5
Phase 6 - check inode connectivity...
              - traversing filesystem ...
              - traversal finished ...
              - moving disconnected inodes to lost+found ...
Phase 7 - verify link counts...
No modify flag set, skipping filesystem flush and exiting.
```

16.2.5. Increasing a Volume's Size

The **ssm resize** command changes the size of the specified volume and file system. If there is no file
system then only the volume itself will be resized.

For this example, we currently have one logical volume on **/dev/vdb** that is 900MB called **lvol001**.

```
# ssm list
-----------------------------------------------------------------
Device          Free        Used        Total  Pool       Mount point
-----------------------------------------------------------------
/dev/vda                                 15.00 GB            PARTITIONED
/dev/vda1                               500.00 MB            /boot
/dev/vda2     0.00 KB    14.51 GB     14.51 GB  rhel
/dev/vdb   120.00 MB   900.00 MB      1.00 GB  lvm_pool
/dev/vdc                                 1.00 GB
-----------------------------------------------------------------
-----------------------------------------------------------------
Pool        Type  Devices      Free        Used       Total
-----------------------------------------------------------------
lvm_pool  lvm   1        120.00 MB   900.00 MB   1020.00 MB
rhel        lvm   1          0.00 KB    14.51 GB     14.51 GB
-----------------------------------------------------------------
-----------------------------------------------------------------
--------------------
```

```
Volume                  Pool         Volume size  FS     FS size       Free
Type    Mount point
----------------------------------------------------------------------
---------------------
/dev/rhel/root          rhel            13.53 GB  xfs   13.52 GB    9.64 GB
linear  /
/dev/rhel/swap          rhel         1000.00 MB
linear
/dev/lvm_pool/lvol001   lvm_pool      900.00 MB  xfs  896.67 MB  896.54 MB
linear
/dev/vda1                             500.00 MB  xfs  496.67 MB  403.56 MB
part    /boot
----------------------------------------------------------------------
---------------------
```

The logical volume needs to be increased by another 500MB. To do so we will need to add an extra device to the pool:

```
~]# ssm resize -s +500M /dev/lvm_pool/lvol001 /dev/vdc
  Physical volume "/dev/vdc" successfully created
  Volume group "lvm_pool" successfully extended
Phase 1 - find and verify superblock...
Phase 2 - using internal log
        - scan filesystem freespace and inode maps...
        - found root inode chunk
Phase 3 - for each AG...
        - scan (but don't clear) agi unlinked lists...
        - process known inodes and perform inode discovery...
        - agno = 0
        - agno = 1
        - agno = 2
        - agno = 3
        - process newly discovered inodes...
Phase 4 - check for duplicate blocks...
        - setting up duplicate extent list...
        - check for inodes claiming duplicate blocks...
        - agno = 0
        - agno = 1
        - agno = 2
        - agno = 3
No modify flag set, skipping phase 5
Phase 6 - check inode connectivity...
        - traversing filesystem ...
        - traversal finished ...
        - moving disconnected inodes to lost+found ...
Phase 7 - verify link counts...
No modify flag set, skipping filesystem flush and exiting.
  Extending logical volume lvol001 to 1.37 GiB
  Logical volume lvol001 successfully resized
meta-data=/dev/mapper/lvm_pool-lvol001 isize=256    agcount=4,
agsize=57600 blks
        =                              sectsz=512   attr=2, projid32bit=1
        =                              crc=0
data    =                              bsize=4096   blocks=230400, imaxpct=25
        =                              sunit=0      swidth=0 blks
naming  =version 2                     bsize=4096   ascii-ci=0 ftype=0
```

```
log      =internal                   bsize=4096   blocks=853, version=2
         =                           sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                       extsz=4096   blocks=0, rtextents=0
data blocks changed from 230400 to 358400
```

SSM runs a check on the device and then extends the volume by the specified amount. This can be verified with the **ssm list** command.

```
# ssm list
----------------------------------------------------------------------
Device              Free          Used       Total  Pool       Mount point
----------------------------------------------------------------------
/dev/vda                                    15.00 GB            PARTITIONED
/dev/vda1                                  500.00 MB            /boot
/dev/vda2    0.00 KB     14.51 GB   14.51 GB  rhel
/dev/vdb     0.00 KB   1020.00 MB    1.00 GB  lvm_pool
/dev/vdc   640.00 MB    380.00 MB    1.00 GB  lvm_pool
----------------------------------------------------------------------
-------------------------------------------------------
Pool      Type  Devices      Free       Used      Total
-------------------------------------------------------
lvm_pool  lvm   2        640.00 MB   1.37 GB    1.99 GB
rhel      lvm   1          0.00 KB  14.51 GB   14.51 GB
-------------------------------------------------------
----------------------------------------------------------------------
-----------------------
Volume                  Pool      Volume size  FS      FS size
Free  Type    Mount point
----------------------------------------------------------------------
-----------------------
/dev/rhel/root          rhel         13.53 GB  xfs     13.52 GB     9.64 GB
linear  /
/dev/rhel/swap          rhel       1000.00 MB
linear
/dev/lvm_pool/lvol001  lvm_pool      1.37 GB  xfs      1.36 GB     1.36 GB
linear
/dev/vda1                           500.00 MB  xfs    496.67 MB   403.56 MB
part    /boot
----------------------------------------------------------------------
-----------------------
```

NOTE

It is only possible to decrease an LVM volume's size; it is not supported with other volume types. This is done by using a **-** instead of a **+**. For example, to decrease the size of an LVM volume by 50M the command would be:

```
# ssm resize -s-50M /dev/lvm_pool/lvol002
  Rounding size to boundary between physical extents: 972.00 MiB
  WARNING: Reducing active logical volume to 972.00 MiB
  THIS MAY DESTROY YOUR DATA (filesystem etc.)
Do you really want to reduce lvol002? [y/n]: y
  Reducing logical volume lvol002 to 972.00 MiB
  Logical volume lvol002 successfully resized
```

Without either the **+** or **-**, the value is taken as absolute.

16.2.6. Snapshot

To take a snapshot of an existing volume, use the **ssm snapshot** command.

NOTE

This operation fails if the back end that the volume belongs to does not support snapshotting.

To create a snapshot of the **lvol001**, use the following command:

```
# ssm snapshot /dev/lvm_pool/lvol001
  Logical volume "snap20150519T130900" created
```

To verify this, use the **ssm list**, and note the extra snapshot section.

```
# ssm list
-----------------------------------------------------------------
Device          Free          Used       Total  Pool       Mount point
-----------------------------------------------------------------
/dev/vda                                 15.00 GB            PARTITIONED
/dev/vda1                                500.00 MB           /boot
/dev/vda2  0.00 KB    14.51 GB    14.51 GB  rhel
/dev/vdb   0.00 KB  1020.00 MB     1.00 GB  lvm_pool
/dev/vdc                                  1.00 GB
-----------------------------------------------------------------
-----------------------------------------------------------
Pool       Type  Devices     Free          Used        Total
-----------------------------------------------------------
lvm_pool  lvm   1         0.00 KB  1020.00 MB  1020.00 MB
rhel      lvm   1         0.00 KB    14.51 GB    14.51 GB
-----------------------------------------------------------
-----------------------------------------------------------------------
-----------------------
Volume                   Pool       Volume size  FS       FS size
Free  Type    Mount point
-----------------------------------------------------------------------
```

```
-----------------------
/dev/rhel/root          rhel            13.53 GB  xfs     13.52 GB     9.64 GB
linear   /
/dev/rhel/swap          rhel         1000.00 MB
linear
/dev/lvm_pool/lvol001   lvm_pool      900.00 MB  xfs     896.67 MB   896.54 MB
linear
/dev/vda1                             500.00 MB  xfs     496.67 MB   403.56 MB
part    /boot
------------------------------------------------------------------------
----------------------
------------------------------------------------------------------------
-----------
Snapshot                              Origin    Pool      Volume size
Size   Type
------------------------------------------------------------------------
-----------
/dev/lvm_pool/snap20150519T130900  lvol001   lvm_pool    120.00 MB  0.00 KB
linear
------------------------------------------------------------------------
-----------
```

16.2.7. Removing an Item

The **ssm remove** is used to remove an item, either a device, pool, or volume.

> **NOTE**
>
> If a device is being used by a pool when removed, it will fail. This can be forced using the **-f** argument.
>
> If the volume is mounted when removed, it will fail. Unlike the device, it cannot be forced with the **-f** argument.

To remove the **lvm_pool** and everything within it use the following command:

```
# ssm remove lvm_pool
Do you really want to remove volume group "lvm_pool" containing 2 logical
volumes? [y/n]: y
Do you really want to remove active logical volume snap20150519T130900?
[y/n]: y
  Logical volume "snap20150519T130900" successfully removed
Do you really want to remove active logical volume lvol001? [y/n]: y
  Logical volume "lvol001" successfully removed
  Volume group "lvm_pool" successfully removed
```

16.3. SSM RESOURCES

For more information on SSM, see the following resources:

- The **man ssm** page provides good descriptions and examples, as well as details on all of the commands and options too specific to be documented here.

- Local documentation for SSM is stored in the **doc/** directory.

- The SSM wiki can be accessed at http://storagemanager.sourceforge.net/index.html.

- The mailing list can be subscribed from https://lists.sourceforge.net/lists/listinfo/storagemanager-devel and mailing list archives from http://sourceforge.net/mailarchive/forum.php?forum_name=storagemanager-devel. The mailing list is where developers communicate. There is currently no user mailing list so feel free to post questions there as well.

CHAPTER 17. DISK QUOTAS

Disk space can be restricted by implementing disk quotas which alert a system administrator before a user consumes too much disk space or a partition becomes full.

Disk quotas can be configured for individual users as well as user groups. This makes it possible to manage the space allocated for user-specific files (such as email) separately from the space allocated to the projects a user works on (assuming the projects are given their own groups).

In addition, quotas can be set not just to control the number of disk blocks consumed but to control the number of inodes (data structures that contain information about files in UNIX file systems). Because inodes are used to contain file-related information, this allows control over the number of files that can be created.

The **quota** RPM must be installed to implement disk quotas.

> **NOTE**
>
> This chapter is for all file systems, however some file systems have their own quota management tools. See the corresponding description for the applicable file systems.
>
> For XFS file systems, see Section 3.3, "XFS Quota Management".
>
> Btrfs does not have disk quotas so is not covered.

17.1. CONFIGURING DISK QUOTAS

To implement disk quotas, use the following steps:

1. Enable quotas per file system by modifying the **/etc/fstab** file.

2. Remount the file system(s).

3. Create the quota database files and generate the disk usage table.

4. Assign quota policies.

Each of these steps is discussed in detail in the following sections.

17.1.1. Enabling Quotas

Procedure 17.1. Enabling Quotas

1. Log in as root.

2. Edit the **/etc/fstab** file.

3. Add either the **usrquota** or **grpquota** or both options to the file systems that require quotas.

> **Example 17.1. Edit /etc/fstab**
>
> For example, to use the text editor **vim** type the following:
>
> ```
> # vim /etc/fstab
> ```

Example 17.2. Add Quotas

```
/dev/VolGroup00/LogVol00 /           ext3    defaults          1 1
LABEL=/boot               /boot      ext3    defaults          1 2
none                      /dev/pts   devpts  gid=5,mode=620  0 0
none                      /dev/shm   tmpfs   defaults          0 0
none                      /proc      proc    defaults          0 0
none                      /sys       sysfs   defaults          0 0
/dev/VolGroup00/LogVol02 /home       ext3    defaults,usrquota,grpquota
1 2
/dev/VolGroup00/LogVol01 swap        swap    defaults          0 0 . . .
```

In this example, the **/home** file system has both user and group quotas enabled.

NOTE

The following examples assume that a separate **/home** partition was created during the installation of Red Hat Enterprise Linux. The root (**/**) partition can be used for setting quota policies in the **/etc/fstab** file.

17.1.2. Remounting the File Systems

After adding either the **usrquota** or **grpquota** or both options, remount each file system whose **fstab** entry has been modified. If the file system is not in use by any process, use one of the following methods:

- Run the **umount** command followed by the **mount** command to remount the file system. See the **man** page for both **umount** and **mount** for the specific syntax for mounting and unmounting various file system types.

- Run the **mount -o remount** *file-system* command (where *file-system* is the name of the file system) to remount the file system. For example, to remount the **/home** file system, run the **mount -o remount /home** command.

If the file system is currently in use, the easiest method for remounting the file system is to reboot the system.

17.1.3. Creating the Quota Database Files

After each quota-enabled file system is remounted run the **quotacheck** command.

The **quotacheck** command examines quota-enabled file systems and builds a table of the current disk usage per file system. The table is then used to update the operating system's copy of disk usage. In addition, the file system's disk quota files are updated.

NOTE

The **quotacheck** command has no effect on XFS as the table of disk usage is completed automatically at mount time. See the man page **xfs_quota(8)** for more information.

Procedure 17.2. Creating the Quota Database Files

1. Create the quota files on the file system using the following command:

```
# quotacheck -cug /file system
```

2. Generate the table of current disk usage per file system using the following command:

```
# quotacheck -avug
```

Following are the options used to create quota files:

c

Specifies that the quota files should be created for each file system with quotas enable.

u

Checks for user quotas.

g

Checks for group quotas. If only **-g** is specified, only the group quota file is created.

If neither the **-u** or **-g** options are specified, only the user quota file is created.

The following options are used to generate the table of current disk usage:

a

Check all quota-enabled, locally-mounted file systems

v

Display verbose status information as the quota check proceeds

u

Check user disk quota information

g

Check group disk quota information

After **quotacheck** has finished running, the quota files corresponding to the enabled quotas (either user or group or both) are populated with data for each quota-enabled locally-mounted file system such as **/home**.

17.1.4. Assigning Quotas per User

The last step is assigning the disk quotas with the **edquota** command.

Prerequisite

- User must exist prior to setting the user quota.

Procedure 17.3. Assigning Quotas per User

Procedure 17.3. Assigning Quotas per User

1. To assign the quota for a user, use the following command:

   ```
   # edquota username
   ```

   Replace *username* with the user to which you want to assign the quotas.

2. To verify that the quota for the user has been set, use the following command:

   ```
   # quota username
   ```

Example 17.3. Assigning Quotas to a user

For example, if a quota is enabled in **/etc/fstab** for the **/home** partition (**/dev/VolGroup00/LogVol02** in the following example) and the command **edquota testuser** is executed, the following is shown in the editor configured as the default for the system:

```
Disk quotas for user testuser (uid 501):
  Filesystem                   blocks      soft      hard     inodes     soft
hard
  /dev/VolGroup00/LogVol02   440436           0         0      37418        0
0
```

> **NOTE**
>
> The text editor defined by the **EDITOR** environment variable is used by **edquota**. To change the editor, set the **EDITOR** environment variable in your **~/.bash_profile** file to the full path of the editor of your choice.

The first column is the name of the file system that has a quota enabled for it. The second column shows how many blocks the user is currently using. The next two columns are used to set soft and hard block limits for the user on the file system. The **inodes** column shows how many inodes the user is currently using. The last two columns are used to set the soft and hard inode limits for the user on the file system.

The hard block limit is the absolute maximum amount of disk space that a user or group can use. Once this limit is reached, no further disk space can be used.

The soft block limit defines the maximum amount of disk space that can be used. However, unlike the hard limit, the soft limit can be exceeded for a certain amount of time. That time is known as the *grace period*. The grace period can be expressed in seconds, minutes, hours, days, weeks, or months.

If any of the values are set to 0, that limit is not set. In the text editor, change the desired limits.

Example 17.4. Change Desired Limits

For example:

```
Disk quotas for user testuser (uid 501):
Filesystem                     blocks      soft      hard     inodes     soft
hard
```

```
/dev/VolGroup00/LogVol02  440436    500000    550000    37418        0
0
```

To verify that the quota for the user has been set, use the command:

```
# quota testuser
Disk quotas for user username (uid 501):
   Filesystem  blocks   quota   limit   grace   files   quota   limit
grace
      /dev/sdb   1000*   1000    1000                0       0       0
```

17.1.5. Assigning Quotas per Group

Quotas can also be assigned on a per-group basis.

Prerequisite

- Group must exist prior to setting the group quota.

Procedure 17.4. Assigning Quotas per Group

1. To set a group quota, use the following command:

   ```
   # edquota -g groupname
   ```

2. To verify that the group quota is set, use the following command:

   ```
   # quota -g groupname
   ```

Example 17.5. Assigning quotas to group

For example, to set a group quota for the **devel** group, use the command:

```
# edquota -g devel
```

This command displays the existing quota for the group in the text editor:

```
Disk quotas for group devel (gid 505):
Filesystem                blocks    soft     hard    inodes    soft
hard
/dev/VolGroup00/LogVol02  440400       0        0     37418       0
0
```

Modify the limits, then save the file.

To verify that the group quota has been set, use the command:

```
# quota -g devel
```

17.1.6. Setting the Grace Period for Soft Limits

If a given quota has soft limits, you can edit the grace period (i.e. the amount of time a soft limit can be exceeded) with the following command:

```
# edquota -t
```

This command works on quotas for inodes or blocks, for either users or groups.

> **IMPORTANT**
>
> While other **edquota** commands operate on quotas for a particular user or group, the **-t** option operates on every file system with quotas enabled.

17.2. MANAGING DISK QUOTAS

If quotas are implemented, they need some maintenance mostly in the form of watching to see if the quotas are exceeded and making sure the quotas are accurate.

If users repeatedly exceed their quotas or consistently reach their soft limits, a system administrator has a few choices to make depending on what type of users they are and how much disk space impacts their work. The administrator can either help the user determine how to use less disk space or increase the user's disk quota.

17.2.1. Enabling and Disabling

It is possible to disable quotas without setting them to 0. To turn all user and group quotas off, use the following command:

```
# quotaoff -vaug
```

If neither the **-u** or **-g** options are specified, only the user quotas are disabled. If only **-g** is specified, only group quotas are disabled. The **-v** switch causes verbose status information to display as the command executes.

To enable user and group quotas again, use the following command:

```
# quotaon
```

To enable user and group quotas for all file systems, use the following command:

```
# quotaon -vaug
```

If neither the **-u** or **-g** options are specified, only the user quotas are enabled. If only **-g** is specified, only group quotas are enabled.

To enable quotas for a specific file system, such as **/home**, use the following command:

```
# quotaon -vug /home
```

NOTE

The `quotaon` command is not always needed for XFS because it is performed automatically at mount time. Refer to the man page `quotaon(8)` for more information.

17.2.2. Reporting on Disk Quotas

Creating a disk usage report entails running the `repquota` utility.

Example 17.6. Output of the `repquota` Command

For example, the command `repquota /home` produces this output:

```
*** Report for user quotas on device /dev/mapper/VolGroup00-LogVol02
Block grace time: 7days; Inode grace time: 7days
   Block limits   File limits
User  used soft hard grace used soft hard grace
----------------------------------------------------------------------
--
root       --      36      0      0               4      0      0
kristin    --     540      0      0             125      0      0
testuser   --  440400  500000  550000          37418      0      0
```

To view the disk usage report for all (option `-a`) quota-enabled file systems, use the command:

```
# repquota -a
```

While the report is easy to read, a few points should be explained. The `--` displayed after each user is a quick way to determine whether the block or inode limits have been exceeded. If either soft limit is exceeded, a `+` appears in place of the corresponding `-`; the first `-` represents the block limit, and the second represents the inode limit.

The `grace` columns are normally blank. If a soft limit has been exceeded, the column contains a time specification equal to the amount of time remaining on the grace period. If the grace period has expired, **none** appears in its place.

17.2.3. Keeping Quotas Accurate

When a file system fails to unmount cleanly, for example due to a system crash, it is necessary to run the following command:

```
# quotacheck
```

However, `quotacheck` can be run on a regular basis, even if the system has not crashed. Safe methods for periodically running `quotacheck` include:

Ensuring quotacheck runs on next reboot

NOTE

This method works best for (busy) multiuser systems which are periodically rebooted.

Save a shell script into the **/etc/cron.daily/** or **/etc/cron.weekly/** directory or schedule one using the following command:

```
# crontab -e
```

The **crontab -e** command contains the **touch /forcequotacheck** command. This creates an empty **forcequotacheck** file in the root directory, which the system init script looks for at boot time. If it is found, the init script runs **quotacheck**. Afterward, the init script removes the **/forcequotacheck** file; thus, scheduling this file to be created periodically with **cron** ensures that **quotacheck** is run during the next reboot.

For more information about **cron**, see **man cron**.

Running quotacheck in single user mode

An alternative way to safely run **quotacheck** is to boot the system into single-user mode to prevent the possibility of data corruption in quota files and run the following commands:

```
# quotaoff -vug /file_system
```

```
# quotacheck -vug /file_system
```

```
# quotaon -vug /file_system
```

Running quotacheck on a running system

If necessary, it is possible to run **quotacheck** on a machine during a time when no users are logged in, and thus have no open files on the file system being checked. Run the command **quotacheck -vug file_system**; this command will fail if **quotacheck** cannot remount the given *file_system* as read-only. Note that, following the check, the file system will be remounted read-write.

> **⚠ WARNING**
>
> Running **quotacheck** on a live file system mounted read-write is not recommended due to the possibility of quota file corruption.

See **man cron** for more information about configuring **cron**.

17.3. DISK QUOTA REFERENCES

For more information on disk quotas, refer to the **man** pages of the following commands:

- **quotacheck**

- **edquota**

- **repquota**

- **quota**

- **quotaon**

- **quotaoff**

CHAPTER 18. REDUNDANT ARRAY OF INDEPENDENT DISKS (RAID)

The basic idea behind RAID is to combine multiple small, inexpensive disk drives into an array to accomplish performance or redundancy goals not attainable with one large and expensive drive. This array of drives appears to the computer as a single logical storage unit or drive.

RAID allows information to be spread across several disks. RAID uses techniques such as *disk striping* (RAID Level 0), *disk mirroring* (RAID Level 1), and *disk striping with parity* (RAID Level 5) to achieve redundancy, lower latency, increased bandwidth, and maximized ability to recover from hard disk crashes.

RAID distributes data across each drive in the array by breaking it down into consistently-sized chunks (commonly 256K or 512k, although other values are acceptable). Each chunk is then written to a hard drive in the RAID array according to the RAID level employed. When the data is read, the process is reversed, giving the illusion that the multiple drives in the array are actually one large drive.

System Administrators and others who manage large amounts of data would benefit from using RAID technology. Primary reasons to deploy RAID include:

- Enhances speed

- Increases storage capacity using a single virtual disk

- Minimizes data loss from disk failure

18.1. RAID TYPES

There are three possible RAID approaches: Firmware RAID, Hardware RAID, and Software RAID.

Firmware RAID

Firmware RAID, also known as ATARAID, is a type of software RAID where the RAID sets can be configured using a firmware-based menu. The firmware used by this type of RAID also hooks into the BIOS, allowing you to boot from its RAID sets. Different vendors use different on-disk metadata formats to mark the RAID set members. The Intel Matrix RAID is a good example of a firmware RAID system.

Hardware RAID

The hardware-based array manages the RAID subsystem independently from the host. It presents a single disk per RAID array to the host.

A Hardware RAID device may be internal or external to the system, with internal devices commonly consisting of a specialized controller card that handles the RAID tasks transparently to the operating system and with external devices commonly connecting to the system via SCSI, Fibre Channel, iSCSI, InfiniBand, or other high speed network interconnect and presenting logical volumes to the system.

RAID controller cards function like a SCSI controller to the operating system, and handle all the actual drive communications. The user plugs the drives into the RAID controller (just like a normal SCSI controller) and then adds them to the RAID controllers configuration. The operating system will not be able to tell the difference.

Software RAID

Software RAID implements the various RAID levels in the kernel disk (block device) code. It offers the cheapest possible solution, as expensive disk controller cards or hot-swap chassis [2] are not required. Software RAID also works with cheaper IDE disks as well as SCSI disks. With today's faster CPUs, Software RAID also generally outperforms Hardware RAID.

The Linux kernel contains a *multi-disk* (MD) driver that allows the RAID solution to be completely hardware independent. The performance of a software-based array depends on the server CPU performance and load.

Key features of the Linux software RAID stack:

- Multithreaded design

- Portability of arrays between Linux machines without reconstruction

- Backgrounded array reconstruction using idle system resources

- Hot-swappable drive support

- Automatic CPU detection to take advantage of certain CPU features such as streaming SIMD support

- Automatic correction of bad sectors on disks in an array

- Regular consistency checks of RAID data to ensure the health of the array

- Proactive monitoring of arrays with email alerts sent to a designated email address on important events

- Write-intent bitmaps which drastically increase the speed of resync events by allowing the kernel to know precisely which portions of a disk need to be resynced instead of having to resync the entire array

- Resync checkpointing so that if you reboot your computer during a resync, at startup the resync will pick up where it left off and not start all over again

- The ability to change parameters of the array after installation. For example, you can grow a 4-disk RAID5 array to a 5-disk RAID5 array when you have a new disk to add. This grow operation is done live and does not require you to reinstall on the new array.

18.2. RAID LEVELS AND LINEAR SUPPORT

RAID supports various configurations, including levels 0, 1, 4, 5, 6, 10, and linear. These RAID types are defined as follows:

Level 0

RAID level 0, often called "striping," is a performance-oriented striped data mapping technique. This means the data being written to the array is broken down into strips and written across the member disks of the array, allowing high I/O performance at low inherent cost but provides no redundancy.

Many RAID level 0 implementations will only stripe the data across the member devices up to the size of the smallest device in the array. This means that if you have multiple devices with slightly different sizes, each device will get treated as though it is the same size as the smallest drive. Therefore, the common storage capacity of a level 0 array is equal to the capacity of the smallest member disk in a Hardware RAID or the capacity of smallest member partition in a Software RAID multiplied by the number of disks or partitions in the array.

Level 1

RAID level 1, or "mirroring," has been used longer than any other form of RAID. Level 1 provides redundancy by writing identical data to each member disk of the array, leaving a "mirrored" copy on each disk. Mirroring remains popular due to its simplicity and high level of data availability. Level 1 operates with two or more disks, and provides very good data reliability and improves performance for read-intensive applications but at a relatively high cost. [3]

The storage capacity of the level 1 array is equal to the capacity of the smallest mirrored hard disk in a Hardware RAID or the smallest mirrored partition in a Software RAID. Level 1 redundancy is the highest possible among all RAID types, with the array being able to operate with only a single disk present.

Level 4

Level 4 uses parity [4] concentrated on a single disk drive to protect data. Because the dedicated parity disk represents an inherent bottleneck on all write transactions to the RAID array, level 4 is seldom used without accompanying technologies such as write-back caching, or in specific circumstances where the system administrator is intentionally designing the software RAID device with this bottleneck in mind (such as an array that will have little to no write transactions once the array is populated with data). RAID level 4 is so rarely used that it is not available as an option in Anaconda. However, it could be created manually by the user if truly needed.

The storage capacity of Hardware RAID level 4 is equal to the capacity of the smallest member partition multiplied by the number of partitions *minus one*. Performance of a RAID level 4 array will always be asymmetrical, meaning reads will outperform writes. This is because writes consume extra CPU and main memory bandwidth when generating parity, and then also consume extra bus bandwidth when writing the actual data to disks because you are writing not only the data, but also the parity. Reads need only read the data and not the parity unless the array is in a degraded state. As a result, reads generate less traffic to the drives and across the busses of the computer for the same amount of data transfer under normal operating conditions.

Level 5

This is the most common type of RAID. By distributing parity across all of an array's member disk drives, RAID level 5 eliminates the write bottleneck inherent in level 4. The only performance bottleneck is the parity calculation process itself. With modern CPUs and Software RAID, that is usually not a bottleneck at all since modern CPUs can generate parity very fast. However, if you have a sufficiently large number of member devices in a software RAID5 array such that the combined aggregate data transfer speed across all devices is high enough, then this bottleneck can start to come into play.

As with level 4, level 5 has asymmetrical performance, with reads substantially outperforming writes. The storage capacity of RAID level 5 is calculated the same way as with level 4.

Level 6

This is a common level of RAID when data redundancy and preservation, and not performance, are the paramount concerns, but where the space inefficiency of level 1 is not acceptable. Level 6 uses a complex parity scheme to be able to recover from the loss of any two drives in the array. This complex parity scheme creates a significantly higher CPU burden on software RAID devices and also imposes an increased burden during write transactions. As such, level 6 is considerably more asymmetrical in performance than levels 4 and 5.

The total capacity of a RAID level 6 array is calculated similarly to RAID level 5 and 4, except that you must subtract 2 devices (instead of 1) from the device count for the extra parity storage space.

Level 10

This RAID level attempts to combine the performance advantages of level 0 with the redundancy of level 1. It also helps to alleviate some of the space wasted in level 1 arrays with more than 2 devices. With level 10, it is possible to create a 3-drive array configured to store only 2 copies of each piece of data, which then allows the overall array size to be 1.5 times the size of the smallest devices instead of only equal to the smallest device (like it would be with a 3-device, level 1 array).

The number of options available when creating level 10 arrays as well as the complexity of selecting the right options for a specific use case make it impractical to create during installation. It is possible to create one manually using the command line `mdadm` tool. For more information on the options and their respective performance trade-offs, see `man md`.

Linear RAID

Linear RAID is a grouping of drives to create a larger virtual drive. In linear RAID, the chunks are allocated sequentially from one member drive, going to the next drive only when the first is completely filled. This grouping provides no performance benefit, as it is unlikely that any I/O operations split between member drives. Linear RAID also offers no redundancy and decreases reliability; if any one member drive fails, the entire array cannot be used. The capacity is the total of all member disks.

18.3. LINUX RAID SUBSYSTEMS

RAID in Linux is composed of the following subsystems:

Linux Hardware RAID Controller Drivers

Hardware RAID controllers have no specific RAID subsystem in Linux. Because they use special RAID chipsets, hardware RAID controllers come with their own drivers; these drivers allow the system to detect the RAID sets as regular disks.

mdraid

The `mdraid` subsystem was designed as a software RAID solution for Linux; it is also the preferred solution for software RAID under Linux. This subsystem uses its own metadata format, generally referred to as native `mdraid` metadata.

`mdraid` also supports other metadata formats, known as external metadata. Red Hat Enterprise Linux 7 uses `mdraid` with external metadata to access ISW / IMSM (Intel firmware RAID) sets. `mdraid` sets are configured and controlled through the `mdadm` utility.

dmraid

Device-mapper RAID or `dmraid` refers to device-mapper kernel code that offers the mechanism to piece disks together into a RAID set. This same kernel code does not provide any RAID configuration mechanism.

`dmraid` is configured entirely in user-space, making it easy to support various on-disk metadata formats. As such, `dmraid` is used on a wide variety of firmware RAID implementations. `dmraid` also supports Intel firmware RAID, although Red Hat Enterprise Linux 7 uses `mdraid` to access Intel firmware RAID sets.

18.4. RAID SUPPORT IN THE ANACONDA INSTALLER

The **Anaconda** installer automatically detects any hardware and firmware RAID sets on a system, making them available for installation. **Anaconda** also supports software RAID using `mdraid`, and can recognize existing `mdraid` sets.

Anaconda provides utilities for creating RAID sets during installation; however, these utilities only allow partitions (as opposed to entire disks) to be members of new sets. To use an entire disk for a set, create a partition on it spanning the entire disk, and use that partition as the RAID set member.

When the root file system uses a RAID set, **Anaconda** adds special kernel command-line options to the bootloader configuration telling the `initrd` which RAID set(s) to activate before searching for the root file system.

For instructions on configuring RAID during installation, see the Red Hat Enterprise Linux 7 Installation Guide.

18.5. CONVERTING ROOT DISK TO RAID1 AFTER INSTALLATION

If you need to convert a non-raided root disk to a RAID1 mirror after installing Red Hat Enterprise Linux 7, see the instructions in the following Red Hat Knowledgebase article: How do I convert my root disk to RAID1 after installation of Red Hat Enterprise Linux 7?

On the PowerPC (PPC) architecture, take the following additional steps:

1. Copy the contents of the PowerPC Reference Platform (PReP) boot partition from **/dev/sda1** to **/dev/sdb1**:

   ```
   # dd if=/dev/sda1 of=/dev/sdb1
   ```

2. Update the Prep and boot flag on the first partition on both disks:

   ```
   $ parted /dev/sda set 1 prep on
   $ parted /dev/sda set 1 boot on

   $ parted /dev/sdb set 1 prep on
   $ parted /dev/sdb set 1 boot on
   ```

> **NOTE**
>
> Running the `grub2-install /dev/sda` command does not work on a PowerPC machine and returns an error, but the system boots as expected.

18.6. CONFIGURING RAID SETS

Most RAID sets are configured during creation, typically through the firmware menu or from the installer. In some cases, you may need to create or modify RAID sets after installing the system, preferably without having to reboot the machine and enter the firmware menu to do so.

Some hardware RAID controllers allow you to configure RAID sets on-the-fly or even define completely new sets after adding extra disks. This requires the use of driver-specific utilities, as there is no standard API for this. For more information, see your hardware RAID controller's driver documentation for information on this.

mdadm

The **mdadm** command-line tool is used to manage software RAID in Linux, i.e. **mdraid**. For information on the different **mdadm** modes and options, see **man mdadm**. The **man** page also contains useful examples for common operations like creating, monitoring, and assembling software RAID arrays.

dmraid

As the name suggests, **dmraid** is used to manage device-mapper RAID sets. The **dmraid** tool finds ATARAID devices using multiple metadata format handlers, each supporting various formats. For a complete list of supported formats, run **dmraid -l**.

As mentioned earlier in Section 18.3, "Linux RAID Subsystems", the **dmraid** tool cannot configure RAID sets after creation. For more information about using **dmraid**, see **man dmraid**.

18.7. CREATING ADVANCED RAID DEVICES

In some cases, you may wish to install the operating system on an array that can't be created after the installation completes. Usually, this means setting up the **/boot** or root file system arrays on a complex RAID device; in such cases, you may need to use array options that are not supported by **Anaconda**. To work around this, perform the following procedure:

Procedure 18.1. Creating Advanced RAID Devices

1. Insert the install disk.

2. During the initial boot up, select **Rescue Mode** instead of **Install** or **Upgrade**. When the system fully boots into *Rescue mode*, the user will be presented with a command line terminal.

3. From this terminal, use **parted** to create RAID partitions on the target hard drives. Then, use **mdadm** to manually create raid arrays from those partitions using any and all settings and options available. For more information on how to do these, see Chapter 13, *Partitions*, **man parted**, and **man mdadm**.

4. Once the arrays are created, you can optionally create file systems on the arrays as well.

5. Reboot the computer and this time select **Install** or **Upgrade** to install as normal. As **Anaconda** searches the disks in the system, it will find the pre-existing RAID devices.

6. When asked about how to use the disks in the system, select **Custom Layout** and click **Next**. In the device listing, the pre-existing MD RAID devices will be listed.

7. Select a RAID device, click **Edit** and configure its mount point and (optionally) the type of file system it should use (if you did not create one earlier) then click **Done**. **Anaconda** will perform the install to this pre-existing RAID device, preserving the custom options you selected when you created it in *Rescue Mode*.
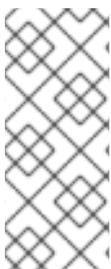
> **NOTE**
>
> The limited *Rescue Mode* of the installer does not include **man** pages. Both the **man mdadm** and **man md** contain useful information for creating custom RAID arrays, and may be needed throughout the workaround. As such, it can be helpful to either have access to a machine with these **man** pages present, or to print them out prior to booting into *Rescue Mode* and creating your custom arrays.

[2] A hot-swap chassis allows you to remove a hard drive without having to power-down your system.

[3] RAID level 1 comes at a high cost because you write the same information to all of the disks in the array, provides data reliability, but in a much less space-efficient manner than parity based RAID levels such as level 5. However, this space inefficiency comes with a performance benefit: parity-based RAID levels consume considerably more CPU power in order to generate the parity while RAID level 1 simply writes the same data more than once to the multiple RAID members with very little CPU overhead. As such, RAID level 1 can outperform the parity-based RAID levels on machines where software RAID is employed and CPU resources on the machine are consistently taxed with operations other than RAID activities.

[4] Parity information is calculated based on the contents of the rest of the member disks in the array. This information can then be used to reconstruct data when one disk in the array fails. The reconstructed data can then be used to satisfy I/O requests to the failed disk before it is replaced and to repopulate the failed disk after it has been replaced.

CHAPTER 19. USING THE MOUNT COMMAND

On Linux, UNIX, and similar operating systems, file systems on different partitions and removable devices (CDs, DVDs, or USB flash drives for example) can be attached to a certain point (the *mount point*) in the directory tree, and then detached again. To attach or detach a file system, use the **mount** or **umount** command respectively. This chapter describes the basic use of these commands, as well as some advanced topics, such as moving a mount point or creating shared subtrees.

19.1. LISTING CURRENTLY MOUNTED FILE SYSTEMS

To display all currently attached file systems, use the following command with no additional arguments:

```
$ mount
```

This command displays the list of known mount points. Each line provides important information about the device name, the file system type, the directory in which it is mounted, and relevant mount options in the following form:

> *device* on *directory* type *type* (*options*)

The **findmnt** utility, which allows users to list mounted file systems in a tree-like form, is also available from Red Hat Enterprise Linux 6.1. To display all currently attached file systems, run the **findmnt** command with no additional arguments:

```
$ findmnt
```

19.1.1. Specifying the File System Type

By default, the output of the **mount** command includes various virtual file systems such as **sysfs** and **tmpfs**. To display only the devices with a certain file system type, provide the **-t** option:

```
$ mount -t type
```

Similarly, to display only the devices with a certain file system using the **findmnt** command:

```
$ findmnt -t type
```

For a list of common file system types, see Table 19.1, "Common File System Types". For an example usage, see Example 19.1, "Listing Currently Mounted **ext4** File Systems".

> **Example 19.1. Listing Currently Mounted ext4 File Systems**
>
> Usually, both **/** and **/boot** partitions are formatted to use **ext4**. To display only the mount points that use this file system, use the following command:
>
> ```
> $ mount -t ext4
> /dev/sda2 on / type ext4 (rw)
> /dev/sda1 on /boot type ext4 (rw)
> ```
>
> To list such mount points using the **findmnt** command, type:

```
$ findmnt -t ext4
TARGET SOURCE     FSTYPE OPTIONS
/       /dev/sda2 ext4   rw,realtime,seclabel,barrier=1,data=ordered
/boot   /dev/sda1 ext4   rw,realtime,seclabel,barrier=1,data=ordered
```

19.2. MOUNTING A FILE SYSTEM

To attach a certain file system, use the **mount** command in the following form:

```
$ mount [option…] device directory
```

The *device* can be identified by:

- a full path to a *block device*: for example, **/dev/sda3**

- a *universally unique identifier* (UUID): for example, **UUID=34795a28-ca6d-4fd8-a347-73671d0c19cb**

- a *volume label*: for example, **LABEL=home**

Note that while a file system is mounted, the original content of the *directory* is not accessible.

IMPORTANT

Linux does not prevent a user from mounting a file system to a directory with a file system already attached to it. To determine whether a particular directory serves as a mount point, run the **findmnt** utility with the directory as its argument and verify the exit code:

```
findmnt directory; echo $?
```

If no file system is attached to the directory, the given command returns **1**.

When you run the **mount** command without all required information, that is without the device name, the target directory, or the file system type, the **mount** reads the contents of the **/etc/fstab** file to check if the given file system is listed. The **/etc/fstab** file contains a list of device names and the directories in which the selected file systems are set to be mounted as well as the file system type and mount options. Therefore, when mounting a file system that is specified in **/etc/fstab**, you can choose one of the following options:

```
mount [option…] directory
mount [option…] device
```

Note that permissions are required to mount the file systems unless the command is run as **root** (see Section 19.2.2, "Specifying the Mount Options").

> **NOTE**
>
> To determine the UUID and—if the device uses it—the label of a particular device, use the **blkid** command in the following form:
>
> > **blkid** *device*
>
> For example, to display information about **/dev/sda3**:
>
> > ```
> > # blkid /dev/sda3
> > /dev/sda3: LABEL="home" UUID="34795a28-ca6d-4fd8-a347-
> > 73671d0c19cb" TYPE="ext3"
> > ```

19.2.1. Specifying the File System Type

In most cases, **mount** detects the file system automatically. However, there are certain file systems, such as **NFS** (Network File System) or **CIFS** (Common Internet File System), that are not recognized, and need to be specified manually. To specify the file system type, use the **mount** command in the following form:

> ```
> $ mount -t type device directory
> ```

Table 19.1, "Common File System Types" provides a list of common file system types that can be used with the **mount** command. For a complete list of all available file system types, see the section called "Manual Page Documentation".

Table 19.1. Common File System Types

Type	Description
ext2	The **ext2** file system.
ext3	The **ext3** file system.
ext4	The **ext4** file system.
btrfs	The **btrfs** file system.
xfs	The **xfs** file system.
iso9660	The **ISO 9660** file system. It is commonly used by optical media, typically CDs.
jfs	The **JFS** file system created by IBM.
nfs	The **NFS** file system. It is commonly used to access files over the network.
nfs4	The **NFSv4** file system. It is commonly used to access files over the network.

Type	Description
`ntfs`	The **NTFS** file system. It is commonly used on machines that are running the Windows operating system.
`udf`	The **UDF** file system. It is commonly used by optical media, typically DVDs.
`vfat`	The **FAT** file system. It is commonly used on machines that are running the Windows operating system, and on certain digital media such as USB flash drives or floppy disks.

See Example 19.2, "Mounting a USB Flash Drive" for an example usage.

> **Example 19.2. Mounting a USB Flash Drive**
>
> Older USB flash drives often use the FAT file system. Assuming that such drive uses the **/dev/sdc1** device and that the **/media/flashdisk/** directory exists, mount it to this directory by typing the following at a shell prompt as **root**:
>
> ```
> ~]# mount -t vfat /dev/sdc1 /media/flashdisk
> ```

19.2.2. Specifying the Mount Options

To specify additional mount options, use the command in the following form:

```
mount -o options device directory
```

When supplying multiple options, do not insert a space after a comma, or **mount** interprets incorrectly the values following spaces as additional parameters.

Table 19.2, "Common Mount Options" provides a list of common mount options. For a complete list of all available options, consult the relevant manual page as referred to in the section called "Manual Page Documentation".

Table 19.2. Common Mount Options

Option	Description
`async`	Allows the asynchronous input/output operations on the file system.
`auto`	Allows the file system to be mounted automatically using the **mount -a** command.
`defaults`	Provides an alias for **async,auto,dev,exec,nouser,rw,suid**.
`exec`	Allows the execution of binary files on the particular file system.
`loop`	Mounts an image as a loop device.

Option	Description
noauto	Default behavior disallows the automatic mount of the file system using the **mount -a** command.
noexec	Disallows the execution of binary files on the particular file system.
nouser	Disallows an ordinary user (that is, other than **root**) to mount and unmount the file system.
remount	Remounts the file system in case it is already mounted.
ro	Mounts the file system for reading only.
rw	Mounts the file system for both reading and writing.
user	Allows an ordinary user (that is, other than **root**) to mount and unmount the file system.

See Example 19.3, "Mounting an ISO Image" for an example usage.

Example 19.3. Mounting an ISO Image

An ISO image (or a disk image in general) can be mounted by using the loop device. Assuming that the ISO image of the Fedora 14 installation disc is present in the current working directory and that the **/media/cdrom/** directory exists, mount the image to this directory by running the following command:

```
# mount -o ro,loop Fedora-14-x86_64-Live-Desktop.iso /media/cdrom
```

Note that ISO 9660 is by design a read-only file system.

19.2.3. Sharing Mounts

Occasionally, certain system administration tasks require access to the same file system from more than one place in the directory tree (for example, when preparing a chroot environment). This is possible, and Linux allows you to mount the same file system to as many directories as necessary. Additionally, the **mount** command implements the **--bind** option that provides a means for duplicating certain mounts. Its usage is as follows:

```
$ mount --bind old_directory new_directory
```

Although this command allows a user to access the file system from both places, it does not apply on the file systems that are mounted within the original directory. To include these mounts as well, use the following command:

```
$ mount --rbind old_directory new_directory
```

Additionally, to provide as much flexibility as possible, Red Hat Enterprise Linux 7 implements the functionality known as *shared subtrees*. This feature allows the use of the following four mount types:

Shared Mount

A shared mount allows the creation of an exact replica of a given mount point. When a mount point is marked as a shared mount, any mount within the original mount point is reflected in it, and vice versa. To change the type of a mount point to a shared mount, type the following at a shell prompt:

```
$ mount --make-shared mount_point
```

Alternatively, to change the mount type for the selected mount point and all mount points under it:

```
$ mount --make-rshared mount_point
```

See for an example usage.

Example 19.4. Creating a Shared Mount Point

There are two places where other file systems are commonly mounted: the **/media/** directory for removable media, and the **/mnt/** directory for temporarily mounted file systems. By using a shared mount, you can make these two directories share the same content. To do so, as **root**, mark the **/media/** directory as shared:

```
# mount --bind /media /media
# mount --make-shared /media
```

Create its duplicate in **/mnt/** by using the following command:

```
# mount --bind /media /mnt
```

It is now possible to verify that a mount within **/media/** also appears in **/mnt/**. For example, if the CD-ROM drive contains non-empty media and the **/media/cdrom/** directory exists, run the following commands:

```
# mount /dev/cdrom /media/cdrom
# ls /media/cdrom
EFI  GPL  isolinux  LiveOS
# ls /mnt/cdrom
EFI  GPL  isolinux  LiveOS
```

Similarly, it is possible to verify that any file system mounted in the **/mnt/** directory is reflected in **/media/**. For instance, if a non-empty USB flash drive that uses the **/dev/sdc1** device is plugged in and the **/mnt/flashdisk/** directory is present, type:

```
# # mount /dev/sdc1 /mnt/flashdisk
# ls /media/flashdisk
en-US  publican.cfg
# ls /mnt/flashdisk
en-US  publican.cfg
```

Slave Mount

A slave mount allows the creation of a limited duplicate of a given mount point. When a mount point is marked as a slave mount, any mount within the original mount point is reflected in it, but no mount within a slave mount is reflected in its original. To change the type of a mount point to a slave mount, type the following at a shell prompt:

```
mount --make-slave mount_point
```

Alternatively, it is possible to change the mount type for the selected mount point and all mount points under it by typing:

```
mount --make-rslave mount_point
```

See Example 19.5, "Creating a Slave Mount Point" for an example usage.

Example 19.5. Creating a Slave Mount Point

This example shows how to get the content of the **/media/** directory to appear in **/mnt/** as well, but without any mounts in the **/mnt/** directory to be reflected in **/media/**. As **root**, first mark the **/media/** directory as shared:

```
~]# mount --bind /media /media
~]# mount --make-shared /media
```

Then create its duplicate in **/mnt/**, but mark it as "slave":

```
~]# mount --bind /media /mnt
~]# mount --make-slave /mnt
```

Now verify that a mount within **/media/** also appears in **/mnt/**. For example, if the CD-ROM drive contains non-empty media and the **/media/cdrom/** directory exists, run the following commands:

```
~]# mount /dev/cdrom /media/cdrom
~]# ls /media/cdrom
EFI  GPL  isolinux  LiveOS
~]# ls /mnt/cdrom
EFI  GPL  isolinux  LiveOS
```

Also verify that file systems mounted in the **/mnt/** directory are not reflected in **/media/**. For instance, if a non-empty USB flash drive that uses the **/dev/sdc1** device is plugged in and the **/mnt/flashdisk/** directory is present, type:

```
~]# mount /dev/sdc1 /mnt/flashdisk
~]# ls /media/flashdisk
~]# ls /mnt/flashdisk
en-US  publican.cfg
```

Private Mount

A private mount is the default type of mount, and unlike a shared or slave mount, it does not receive or forward any propagation events. To explicitly mark a mount point as a private mount, type the following at a shell prompt:

```
mount --make-private mount_point
```

Alternatively, it is possible to change the mount type for the selected mount point and all mount points under it:

```
mount --make-rprivate mount_point
```

See Example 19.6, "Creating a Private Mount Point" for an example usage.

Example 19.6. Creating a Private Mount Point

Taking into account the scenario in Example 19.4, "Creating a Shared Mount Point", assume that a shared mount point has been previously created by using the following commands as **root**:

```
~]# mount --bind /media /media
~]# mount --make-shared /media
~]# mount --bind /media /mnt
```

To mark the **/mnt/** directory as private, type:

```
~]# mount --make-private /mnt
```

It is now possible to verify that none of the mounts within **/media/** appears in **/mnt/**. For example, if the CD-ROM drives contains non-empty media and the **/media/cdrom/** directory exists, run the following commands:

```
~]# mount /dev/cdrom /media/cdrom
~]# ls /media/cdrom
EFI  GPL  isolinux  LiveOS
~]# ls /mnt/cdrom
~]#
```

It is also possible to verify that file systems mounted in the **/mnt/** directory are not reflected in **/media/**. For instance, if a non-empty USB flash drive that uses the **/dev/sdc1** device is plugged in and the **/mnt/flashdisk/** directory is present, type:

```
~]# mount /dev/sdc1 /mnt/flashdisk
~]# ls /media/flashdisk
~]# ls /mnt/flashdisk
en-US  publican.cfg
```

Unbindable Mount

In order to prevent a given mount point from being duplicated whatsoever, an unbindable mount is used. To change the type of a mount point to an unbindable mount, type the following at a shell prompt:

```
mount --make-unbindable mount_point
```

Alternatively, it is possible to change the mount type for the selected mount point and all mount points under it:

```
mount --make-runbindable mount_point
```

See Example 19.7, "Creating an Unbindable Mount Point" for an example usage.

Example 19.7. Creating an Unbindable Mount Point

To prevent the **/media/** directory from being shared, as **root**:

```
# mount --bind /media /media
# mount --make-unbindable /media
```

This way, any subsequent attempt to make a duplicate of this mount fails with an error:

```
# mount --bind /media /mnt
mount: wrong fs type, bad option, bad superblock on /media,
missing codepage or helper program, or other error
In some cases useful info is found in syslog - try
dmesg | tail  or so
```

19.2.4. Moving a Mount Point

To change the directory in which a file system is mounted, use the following command:

```
# mount --move old_directory new_directory
```

See Example 19.8, "Moving an Existing NFS Mount Point" for an example usage.

Example 19.8. Moving an Existing NFS Mount Point

An NFS storage contains user directories and is already mounted in **/mnt/userdirs/**. As **root**, move this mount point to **/home** by using the following command:

```
# mount --move /mnt/userdirs /home
```

To verify the mount point has been moved, list the content of both directories:

```
# ls /mnt/userdirs
# ls /home
jill  joe
```

19.2.5. Setting Read-only Permissions for **root**

Sometimes, you need to mount the root file system with read-only permissions. Example use cases include enhancing security or ensuring data integrity after an unexpected system power-off.

19.2.5.1. Configuring **root** to Mount with Read-only Permissions on Boot

1. In the **/etc/sysconfig/readonly-root** file, change **READONLY** to **yes**:

```
# Set to 'yes' to mount the file systems as read-only.
READONLY=yes
[output truncated]
```

2. Change **defaults** to **ro** in the root entry (**/**) in the **/etc/fstab** file:

```
/dev/mapper/luks-c376919e... / ext4 ro,x-systemd.device-timeout=0 1
1
```

3. Add **ro** to the **GRUB_CMDLINE_LINUX** directive in the **/etc/default/grub** file and ensure that it does not contain **rw**:

```
GRUB_CMDLINE_LINUX="crashkernel=auto rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap rhgb quiet ro"
```

4. Recreate the GRUB2 configuration file:

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

5. If you need to add files and directories to be mounted with write permissions in the **tmpfs** file system, create a text file in the **/etc/rwtab.d/** directory and put the configuration there. For example, to mount **/etc/example/file** with write permissions, add this line to the **/etc/rwtab.d/***example* file:

```
files /etc/example/file
```

> **IMPORTANT**
>
> Changes made to files and directories in **tmpfs** do not persist across boots.

See Section 19.2.5.3, "Files and Directories That Retain Write Permissions" for more information on this step.

6. Reboot the system.

19.2.5.2. Remounting **root** Instantly

If root (**/**) was mounted with read-only permissions on system boot, you can remount it with write permissions:

```
# mount -o remount,rw /
```

This can be particularly useful when **/** is incorrectly mounted with read-only permissions.

To remount **/** with read-only permissions again, run:

```
# mount -o remount,ro /
```

NOTE

This command mounts the *whole /* with read-only permissions. A better approach is to retain write permissions for certain files and directories by copying them into RAM, as described in Section 19.2.5.1, "Configuring **root** to Mount with Read-only Permissions on Boot".

19.2.5.3. Files and Directories That Retain Write Permissions

For the system to function properly, some files and directories need to retain write permissions. With root in read-only mode, they are mounted in RAM in the **tmpfs** temporary file system. The default set of such files and directories is read from the **/etc/rwtab** file, which contains:

```
dirs    /var/cache/man
dirs    /var/gdm
[output truncated]
empty   /tmp
empty   /var/cache/foomatic
[output truncated]
files   /etc/adjtime
files   /etc/ntp.conf
[output truncated]
```

Entries in the **/etc/rwtab** file follow this format:

```
how the file or directory is copied to tmpfs        path to the file or
directory
```

A file or directory can be copied to **tmpfs** in the following three ways:

- **empty** *path*: An empty path is copied to **tmpfs**. Example: **empty /tmp**

- **dirs** *path*: A directory tree is copied to **tmpfs**, empty. Example: **dirs /var/run**

- **files** *path*: A file or a directory tree is copied to **tmpfs** intact. Example: **files /etc/resolv.conf**

The same format applies when adding custom paths to **/etc/rwtab.d/**.

19.3. UNMOUNTING A FILE SYSTEM

To detach a previously mounted file system, use either of the following variants of the **umount** command:

```
$ umount directory
$ umount device
```

Note that unless this is performed while logged in as **root**, the correct permissions must be available to unmount the file system. For more information, see Section 19.2.2, "Specifying the Mount Options". See Example 19.9, "Unmounting a CD" for an example usage.

> **IMPORTANT**
>
> When a file system is in use (for example, when a process is reading a file on this file system, or when it is used by the kernel), running the **umount** command fails with an error. To determine which processes are accessing the file system, use the **fuser** command in the following form:
>
> ```
> $ fuser -m directory
> ```
>
> For example, to list the processes that are accessing a file system mounted to the **/media/cdrom/** directory:
>
> ```
> $ fuser -m /media/cdrom
> /media/cdrom:            1793  2013  2022  2435 10532c 10672c
> ```

Example 19.9. Unmounting a CD

To unmount a CD that was previously mounted to the **/media/cdrom/** directory, use the following command:

```
$ umount /media/cdrom
```

19.4. MOUNT COMMAND REFERENCES

The following resources provide an in-depth documentation on the subject.

Manual Page Documentation

- **man 8 mount**: The manual page for the **mount** command that provides a full documentation on its usage.

- **man 8 umount**: The manual page for the **umount** command that provides a full documentation on its usage.

- **man 8 findmnt**: The manual page for the **findmnt** command that provides a full documentation on its usage.

- **man 5 fstab**: The manual page providing a thorough description of the **/etc/fstab** file format.

Useful Websites

- *Shared subtrees* — An LWN article covering the concept of shared subtrees.

CHAPTER 20. THE VOLUME_KEY FUNCTION

The volume_key function provides two tools, libvolume_key and **volume_key**. libvolume_key is a library for manipulating storage volume encryption keys and storing them separately from volumes. **volume_key** is an associated command line tool used to extract keys and passphrases in order to restore access to an encrypted hard drive.

This is useful for when the primary user forgets their keys and passwords, after an employee leaves abruptly, or in order to extract data after a hardware or software failure corrupts the header of the encrypted volume. In a corporate setting, the IT help desk can use **volume_key** to back up the encryption keys before handing over the computer to the end user.

Currently, **volume_key** only supports the LUKS volume encryption format.

> **NOTE**
>
> **volume_key** is not included in a standard install of Red Hat Enterprise Linux 7 server. For information on installing it, refer to http://fedoraproject.org/wiki/Disk_encryption_key_escrow_use_cases.

20.1. VOLUME_KEY COMMANDS

The format for **volume_key** is:

```
volume_key [OPTION]... OPERAND
```

The operands and mode of operation for **volume_key** are determined by specifying one of the following options:

--save

This command expects the operand *volume* [*packet*]. If a *packet* is provided then **volume_key** will extract the keys and passphrases from it. If *packet* is not provided, then **volume_key** will extract the keys and passphrases from the *volume*, prompting the user where necessary. These keys and passphrases will then be stored in one or more output packets.

--restore

This command expects the operands *volume packet*. It then opens the *volume* and uses the keys and passphrases in the *packet* to make the *volume* accessible again, prompting the user where necessary, such as allowing the user to enter a new passphrase, for example.

--setup-volume

This command expects the operands *volume packet name*. It then opens the *volume* and uses the keys and passphrases in the *packet* to set up the *volume* for use of the decrypted data as *name*.

Name is the name of a dm-crypt volume. This operation makes the decrypted volume available as **/dev/mapper/*name***.

This operation does not permanently alter the *volume* by adding a new passphrase, for example. The user can access and modify the decrypted volume, modifying *volume* in the process.

--reencrypt, --secrets, and **--dump**

These three commands perform similar functions with varying output methods. They each require the operand *packet*, and each opens the *packet*, decrypting it where necessary. **--reencrypt** then stores the information in one or more new output packets. **--secrets** outputs the keys and passphrases contained in the *packet*. **--dump** outputs the content of the *packet*, though the keys and passphrases are not output by default. This can be changed by appending **--with-secrets** to the command. It is also possible to only dump the unencrypted parts of the packet, if any, by using the **--unencrypted** command. This does not require any passphrase or private key access.

Each of these can be appended with the following options:

-o, --output *packet*

> This command writes the default key or passphrase to the *packet*. The default key or passphrase depends on the volume format. Ensure it is one that is unlikely to expire, and will allow **--restore** to restore access to the volume.

--output-format *format*

> This command uses the specified *format* for all output packets. Currently, *format* can be one of the following:
>
> - **asymmetric**: uses CMS to encrypt the whole packet, and requires a certificate
>
> - **asymmetric_wrap_secret_only**: wraps only the secret, or keys and passphrases, and requires a certificate
>
> - **passphrase**: uses GPG to encrypt the whole packet, and requires a passphrase

--create-random-passphrase *packet*

> This command generates a random alphanumeric passphrase, adds it to the *volume* (without affecting other passphrases), and then stores this random passphrase into the *packet*.

20.2. USING VOLUME_KEY AS AN INDIVIDUAL USER

As an individual user, **volume_key** can be used to save encryption keys by using the following procedure.

> **NOTE**
>
> For all examples in this file, */path/to/volume* is a LUKS device, not the plaintext device contained within. **blkid -s** *type /path/to/volume* should report *type*="crypto_LUKS".

Procedure 20.1. Using volume_key Stand-alone

1. Run:

   ```
   volume_key --save /path/to/volume -o escrow-packet
   ```

   A prompt will then appear requiring an escrow packet passphrase to protect the key.

2. Save the generated **escrow-packet** file, ensuring that the passphrase is not forgotten.

If the volume passphrase is forgotten, use the saved escrow packet to restore access to the data.

Procedure 20.2. Restore Access to Data with Escrow Packet

1. Boot the system in an environment where **volume_key** can be run and the escrow packet is available (a rescue mode, for example).

2. Run:

   ```
   volume_key --restore /path/to/volume escrow-packet
   ```

   A prompt will appear for the escrow packet passphrase that was used when creating the escrow packet, and for the new passphrase for the volume.

3. Mount the volume using the chosen passphrase.

To free up the passphrase slot in the LUKS header of the encrypted volume, remove the old, forgotten passphrase by using the command **cryptsetup luksKillSlot**.

20.3. USING VOLUME_KEY IN A LARGER ORGANIZATION

In a larger organization, using a single password known by every system administrator and keeping track of a separate password for each system is impractical and a security risk. To counter this, **volume_key** can use asymmetric cryptography to minimize the number of people who know the password required to access encrypted data on any computer.

This section will cover the procedures required for preparation before saving encryption keys, how to save encryption keys, restoring access to a volume, and setting up emergency passphrases.

20.3.1. Preparation for Saving Encryption Keys

In order to begin saving encryption keys, some preparation is required.

Procedure 20.3. Preparation

1. Create an X509 certificate/private pair.

2. Designate trusted users who are trusted not to compromise the private key. These users will be able to decrypt the escrow packets.

3. Choose which systems will be used to decrypt the escrow packets. On these systems, set up an NSS database that contains the private key.

   If the private key was not created in an NSS database, follow these steps:

   - Store the certificate and private key in an **PKCS#12** file.

   - Run:

     ```
     certutil -d /the/nss/directory -N
     ```

     At this point it is possible to choose an NSS database password. Each NSS database can have a different password so the designated users do not need to share a single password if a separate NSS database is used by each user.

- Run:

  ```
  pk12util -d /the/nss/directory -i the-pkcs12-file
  ```

4. Distribute the certificate to anyone installing systems or saving keys on existing systems.

5. For saved private keys, prepare storage that allows them to be looked up by machine and volume. For example, this can be a simple directory with one subdirectory per machine, or a database used for other system management tasks as well.

20.3.2. Saving Encryption Keys

After completing the required preparation (see Section 20.3.1, "Preparation for Saving Encryption Keys") it is now possible to save the encryption keys using the following procedure.

> **NOTE**
>
> For all examples in this file, **/path/to/volume** is a LUKS device, not the plaintext device contained within; **blkid -s type /path/to/volume** should report **type="crypto_LUKS"**.

Procedure 20.4. Saving Encryption Keys

1. Run:

   ```
   volume_key --save /path/to/volume -c /path/to/cert escrow-packet
   ```

2. Save the generated **escrow-packet** file in the prepared storage, associating it with the system and the volume.

These steps can be performed manually, or scripted as part of system installation.

20.3.3. Restoring Access to a Volume

After the encryption keys have been saved (see Section 20.3.1, "Preparation for Saving Encryption Keys" and Section 20.3.2, "Saving Encryption Keys"), access can be restored to a driver where needed.

Procedure 20.5. Restoring Access to a Volume

1. Get the escrow packet for the volume from the packet storage and send it to one of the designated users for decryption.

2. The designated user runs:

   ```
   volume_key --reencrypt -d /the/nss/directory escrow-packet-in -o
   escrow-packet-out
   ```

   After providing the NSS database password, the designated user chooses a passphrase for encrypting **escrow-packet-out**. This passphrase can be different every time and only protects the encryption keys while they are moved from the designated user to the target system.

3. Obtain the **escrow-packet-out** file and the passphrase from the designated user.

4. Boot the target system in an environment that can run **volume_key** and have the **escrow-packet-out** file available, such as in a rescue mode.

5. Run:

```
volume_key --restore /path/to/volume escrow-packet-out
```

A prompt will appear for the packet passphrase chosen by the designated user, and for a new passphrase for the volume.

6. Mount the volume using the chosen volume passphrase.

It is possible to remove the old passphrase that was forgotten by using **cryptsetup luksKillSlot**, for example, to free up the passphrase slot in the LUKS header of the encrypted volume. This is done with the command **cryptsetup luksKillSlot *device key-slot***. For more information and examples see **cryptsetup --help**.

20.3.4. Setting up Emergency Passphrases

In some circumstances (such as traveling for business) it is impractical for system administrators to work directly with the affected systems, but users still need access to their data. In this case, **volume_key** can work with passphrases as well as encryption keys.

During the system installation, run:

```
volume_key --save /path/to/volume -c /path/to/ert --create-random-passphrase passphrase-packet
```

This generates a random passphrase, adds it to the specified volume, and stores it to *passphrase-packet*. It is also possible to combine the **--create-random-passphrase** and **-o** options to generate both packets at the same time.

If a user forgets the password, the designated user runs:

```
volume_key --secrets -d /your/nss/directory passphrase-packet
```

This shows the random passphrase. Give this passphrase to the end user.

20.4. VOLUME_KEY REFERENCES

More information on **volume_key** can be found:

- in the readme file located at **/usr/share/doc/volume_key-*/README**

- on **volume_key**'s manpage using **man volume_key**

- online at http://fedoraproject.org/wiki/Disk_encryption_key_escrow_use_cases

CHAPTER 21. SOLID-STATE DISK DEPLOYMENT GUIDELINES

Solid-state disks (SSD) are storage devices that use NAND flash chips to persistently store data. This sets them apart from previous generations of disks, which store data in rotating, magnetic platters. In an SSD, the access time for data across the full Logical Block Address (LBA) range is constant; whereas with older disks that use rotating media, access patterns that span large address ranges incur seek costs. As such, SSD devices have better latency and throughput.

Performance degrades as the number of used blocks approaches the disk capacity. The degree of performance impact varies greatly by vendor. However, all devices experience some degradation.

To address the degradation issue, the host system (for example, the Linux kernel) may use discard requests to inform the storage that a given range of blocks is no longer in use. An SSD can use this information to free up space internally, using the free blocks for wear-leveling. Discards will only be issued if the storage advertises support in terms of its storage protocol (be it ATA or SCSI). Discard requests are issued to the storage using the negotiated discard command specific to the storage protocol (**TRIM** command for ATA, and **WRITE SAME** with **UNMAP** set, or **UNMAP** command for SCSI).

Enabling **discard** support is most useful when the following points are true:

- Free space is still available on the file system.

- Most logical blocks on the underlying storage device have already been written to.

For more information about **TRIM**, see [Data Set Management T13 Specifications](#).

For more information about **UNMAP**, see the section 4.7.3.4 of the [SCSI Block Commands 3 T10 Specification](#).

> **NOTE**
>
> Not all solid-state devices in the market have **discard** support. To determine if your solid-state device has **discard** support, check for **/sys/block/sda/queue/discard_granularity**, which is the size of internal allocation unit of device.

Deployment Considerations

Because of the internal layout and operation of SSDs, it is best to partition devices on an internal *erase block boundary*. Partitioning utilities in Red Hat Enterprise Linux 7 chooses sane defaults if the SSD exports topology information. However, if the device does *not* export topology information, Red Hat recommends that the first partition should be created at a 1MB boundary.

SSD has various types of TRIM mechanism depending on the vendors choice. The early versions of disks improved the performance by compromising possible data leakage after the read command.

Following are the types of **TRIM** mechanism:

- Non-deterministic **TRIM**

- Deterministic **TRIM** (DRAT)

- Deterministic Read Zero after **TRIM** (RZAT)

The first two types of **TRIM** mechanism can cause data leakage as the **read** command to the LBA after a **TRIM** returns different or same data. RZAT returns zero after the **read** command and Red Hat

recommends this **TRIM** mechanism to avoid data leakage. It is affected only in SSD. Choose the disk which supports RZAT mechanism.

Type of **TRIM** mechanism used depends on hardware implementation. To find the type of **TRIM** mechanism on ATA, use the **hdparm** command. See the following example to find the type of **TRIM** mechanism:

```
# hdparm -I /dev/sda | grep TRIM
Data Set Management TRIM supported (limit 8 block)
Deterministic read data after TRIM
```

For more information, see **man hdparm**.

The Logical Volume Manager (LVM), the device-mapper (DM) targets, and MD (software raid) targets that LVM uses support discards. The only DM targets that do not support discards are dm-snapshot, dm-crypt, and dm-raid45. Discard support for the dm-mirror was added in Red Hat Enterprise Linux 6.1 and as of 7.0 MD supports discards.

Using RAID level 5 over SSD results in low performance if SSDs do not handle **discard** correctly. You can set discard in the **raid456.conf** file, or in the GRUB2 configuration. For instructions, see the following procedures.

Procedure 21.1. Setting discard in raid456.conf

The *devices_handle_discard_safely* module parameter is set in the **raid456** module. To enable discard in the **raid456.conf** file:

1. Verify that your hardware supports discards:

   ```
   # cat /sys/block/disk-name/queue/discard_zeroes_data
   ```

   If the returned value is **1**, discards are supported. If the command returns **0**, the RAID code has to zero the disk out, which takes more time.

2. Create the **/etc/modprobe.d/raid456.conf** file, and include the following line:

   ```
   options raid456 devices_handle_discard_safely=Y
   ```

3. Use the **dracut -f** command to rebuild the initial ramdisk (**initrd**).

4. Reboot the system for the changes to take effect.

Procedure 21.2. Setting discard in the GRUB2 Configuration

The *devices_handle_discard_safely* module parameter is set in the **raid456** module. To enable discard in the GRUB2 configuration:

1. Verify that your hardware supports discards:

   ```
   # cat /sys/block/disk-name/queue/discard_zeroes_data
   ```

   If the returned value is **1**, discards are supported. If the command returns **0**, the RAID code has to zero the disk out, which takes more time.

2. Add the following line to the **/etc/default/grub** file:

```
raid456.devices_handle_discard_safely=Y
```

3. The location of the GRUB2 configuration file is different on systems with the BIOS firmware and on systems with UEFI. Use one of the following commands to recreate the GRUB2 configuration file.

   - On a system with the BIOS firmware, use:

   ```
   # grub2-mkconfig -o /boot/grub2/grub.cfg
   ```

   - On a system with the UEFI firmware, use:

   ```
   # grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
   ```

4. Reboot the system for the changes to take effect.

> **NOTE**
>
> In Red Hat Enterprise Linux 7, discard is fully supported by the ext4 and XFS file systems only.

In Red Hat Enterprise Linux 6.3 and earlier, only the ext4 file system fully supports discard. Starting with Red Hat Enterprise Linux 6.4, both ext4 and XFS file systems fully support discard. To enable discard commands on a device, use the **discard** option of the **mount** command. For example, to mount **/dev/sda2** to **/mnt** with discard enabled, use:

```
# mount -t ext4 -o discard /dev/sda2 /mnt
```

By default, ext4 does not issue the **discard** command to, primarily, avoid problems on devices which might not properly implement discard. The Linux **swap** code issues **discard** commands to discard-enabled devices, and there is no option to control this behavior.

Performance Tuning Considerations

For information on performance tuning considerations regarding solid-state disks, see the Solid-State Disks section in the *Red Hat Enterprise Linux 7 Performance Tuning Guide*.

CHAPTER 22. WRITE BARRIERS

A *write barrier* is a kernel mechanism used to ensure that file system metadata is correctly written and ordered on persistent storage, even when storage devices with volatile write caches lose power. File systems with write barriers enabled ensures that data transmitted via **fsync()** is persistent throughout a power loss.

Enabling write barriers incurs a substantial performance penalty for some applications. Specifically, applications that use **fsync()** heavily or create and delete many small files will likely run much slower.

22.1. IMPORTANCE OF WRITE BARRIERS

File systems safely update metadata, ensuring consistency. Journalled file systems bundle metadata updates into transactions and send them to persistent storage in the following manner:

1. The file system sends the body of the transaction to the storage device.

2. The file system sends a commit block.

3. If the transaction and its corresponding commit block are written to disk, the file system assumes that the transaction will survive any power failure.

However, file system integrity during power failure becomes more complex for storage devices with extra caches. Storage target devices like local S-ATA or SAS drives may have write caches ranging from 32MB to 64MB in size (with modern drives). Hardware RAID controllers often contain internal write caches. Further, high end arrays, like those from NetApp, IBM, Hitachi and EMC (among others), also have large caches.

Storage devices with write caches report I/O as "complete" when the data is in cache; if the cache loses power, it loses its data as well. Worse, as the cache de-stages to persistent storage, it may change the original metadata ordering. When this occurs, the commit block may be present on disk without having the complete, associated transaction in place. As a result, the journal may replay these uninitialized transaction blocks into the file system during post-power-loss recovery; this will cause data inconsistency and corruption.

How Write Barriers Work

Write barriers are implemented in the Linux kernel via storage write cache flushes before and after the I/O, which is *order-critical*. After the transaction is written, the storage cache is flushed, the commit block is written, and the cache is flushed again. This ensures that:

- The disk contains all the data.

- No re-ordering has occurred.

With barriers enabled, an **fsync()** call also issues a storage cache flush. This guarantees that file data is persistent on disk even if power loss occurs shortly after **fsync()** returns.

22.2. ENABLING AND DISABLING WRITE BARRIERS

To mitigate the risk of data corruption during power loss, some storage devices use battery-backed write caches. Generally, high-end arrays and some hardware controllers use battery-backed write caches. However, because the cache's volatility is not visible to the kernel, Red Hat Enterprise Linux 7 enables write barriers by default on all supported journaling file systems.

NOTE

Write caches are designed to increase I/O performance. However, enabling write barriers means constantly flushing these caches, which can significantly reduce performance.

For devices with non-volatile, battery-backed write caches and those with write-caching disabled, you can safely disable write barriers at mount time using the **-o nobarrier** option for **mount**. However, some devices do not support write barriers; such devices log an error message to **/var/log/messages**. For more information, see Table 22.1, "Write Barrier Error Messages per File System".

Table 22.1. Write Barrier Error Messages per File System

File System	Error Message
ext3/ext4	`JBD: barrier-based sync failed on device - disabling barriers`
XFS	`Filesystem device - Disabling barriers, trial barrier write failed`
btrfs	`btrfs: disabling barriers on dev device`

22.3. WRITE BARRIER CONSIDERATIONS

Some system configurations do not need write barriers to protect data. In most cases, other methods are preferable to write barriers, since enabling write barriers causes a significant performance penalty.

Disabling Write Caches

One way to alternatively avoid data integrity issues is to ensure that no write caches lose data on power failures. When possible, the best way to configure this is to disable the write cache. On a simple server or desktop with one or more SATA drives (off a local SATA controller Intel AHCI part), you can disable the write cache on the target SATA drives with the following command:

```
# hdparm -W0 /device/
```

Battery-Backed Write Caches

Write barriers are also unnecessary whenever the system uses hardware RAID controllers with battery-backed write cache. If the system is equipped with such controllers and if its component drives have write caches disabled, the controller acts as a write-through cache; this informs the kernel that the write cache data survives a power loss.

Most controllers use vendor-specific tools to query and manipulate target drives. For example, the LSI Megaraid SAS controller uses a battery-backed write cache; this type of controller requires the **MegaCli64** tool to manage target drives. To show the state of all back-end drives for LSI Megaraid SAS, use:

```
# MegaCli64 -LDGetProp -DskCache -LAll -aALL
```

To disable the write cache of all back-end drives for LSI Megaraid SAS, use:

```
# MegaCli64 -LDSetProp -DisDskCache -Lall -aALL
```

NOTE

Hardware RAID cards recharge their batteries while the system is operational. If a system is powered off for an extended period of time, the batteries will lose their charge, leaving stored data vulnerable during a power failure.

High-End Arrays

High-end arrays have various ways of protecting data in the event of a power failure. As such, there is no need to verify the state of the internal drives in external RAID storage.

NFS

NFS clients do not need to enable write barriers, since data integrity is handled by the NFS server side. As such, NFS servers should be configured to ensure data persistence throughout a power loss (whether through write barriers or other means).

CHAPTER 23. STORAGE I/O ALIGNMENT AND SIZE

Recent enhancements to the SCSI and ATA standards allow storage devices to indicate their preferred (and in some cases, required) *I/O alignment* and *I/O size*. This information is particularly useful with newer disk drives that increase the physical sector size from 512 bytes to 4k bytes. This information may also be beneficial for RAID devices, where the chunk size and stripe size may impact performance.

The Linux I/O stack has been enhanced to process vendor-provided I/O alignment and I/O size information, allowing storage management tools (`parted`, `lvm`, `mkfs.*`, and the like) to optimize data placement and access. If a legacy device does not export I/O alignment and size data, then storage management tools in Red Hat Enterprise Linux 7 will conservatively align I/O on a 4k (or larger power of 2) boundary. This will ensure that 4k-sector devices operate correctly even if they do not indicate any required/preferred I/O alignment and size.

For information on determining the information that the operating system obtained from the device, see the Section 23.2, "Userspace Access". This data is subsequently used by the storage management tools to determine data placement.

The IO scheduler has changed for Red Hat Enterprise Linux 7. Default IO Scheduler is now *Deadline*, except for SATA drives. CFQ is the default IO scheduler for SATA drives. For faster storage, Deadline outperforms CFQ and when it is used there is a performance increase without the need of special tuning.

If default is not right for some disks (for example, SAS rotational disks), then change the IO scheduler to CFQ. This instance will depend on the workload.

23.1. PARAMETERS FOR STORAGE ACCESS

The operating system uses the following information to determine I/O alignment and size:

physical_block_size

Smallest internal unit on which the device can operate

logical_block_size

Used externally to address a location on the device

alignment_offset

The number of bytes that the beginning of the Linux block device (partition/MD/LVM device) is offset from the underlying physical alignment

minimum_io_size

The device's preferred minimum unit for random I/O

optimal_io_size

The device's preferred unit for streaming I/O

For example, certain 4K sector devices may use a 4K `physical_block_size` internally but expose a more granular 512-byte `logical_block_size` to Linux. This discrepancy introduces potential for misaligned I/O. To address this, the Red Hat Enterprise Linux 7 I/O stack will attempt to start all data areas on a naturally-aligned boundary (`physical_block_size`) by making sure it accounts for any alignment_offset if the beginning of the block device is offset from the underlying physical alignment.

Storage vendors can also supply *I/O hints* about the preferred minimum unit for random I/O

(`minimum_io_size`) and streaming I/O (`optimal_io_size`) of a device. For example, `minimum_io_size` and `optimal_io_size` may correspond to a RAID device's chunk size and stripe size respectively.

23.2. USERSPACE ACCESS

Always take care to use properly aligned and sized I/O. This is especially important for Direct I/O access. Direct I/O should be aligned on a `logical_block_size` boundary, and in multiples of the `logical_block_size`.

With native 4K devices (i.e. `logical_block_size` is 4K) it is now critical that applications perform direct I/O in multiples of the device's `logical_block_size`. This means that applications will fail with native 4k devices that perform 512-byte aligned I/O rather than 4k-aligned I/O.

To avoid this, an application should consult the I/O parameters of a device to ensure it is using the proper I/O alignment and size. As mentioned earlier, I/O parameters are exposed through the both `sysfs` and block device `ioctl` interfaces.

For more information, see `man libblkid`. This `man` page is provided by the `libblkid-devel` package.

sysfs Interface

- /sys/block/*disk*/alignment_offset

  or

  /sys/block/*disk*/*partition*/alignment_offset

  > **NOTE**
  >
  > The file location depends on whether the disk is a physical disk (be that a local disk, local RAID, or a multipath LUN) or a virtual disk. The first file location is applicable to physical disks while the second file location is applicable to virtual disks. The reason for this is because virtio-blk will always report an alignment value for the partition. Physical disks may or may not report an alignment value.

- /sys/block/*disk*/queue/physical_block_size

- /sys/block/*disk*/queue/logical_block_size

- /sys/block/*disk*/queue/minimum_io_size

- /sys/block/*disk*/queue/optimal_io_size

The kernel will still export these `sysfs` attributes for "legacy" devices that do not provide I/O parameters information, for example:

Example 23.1. `sysfs` Interface

```
alignment_offset:    0
physical_block_size: 512
logical_block_size:  512
```

```
minimum_io_size:      512
optimal_io_size:        0
```

Block Device ioctls

- **BLKALIGNOFF**: `alignment_offset`

- **BLKPBSZGET**: `physical_block_size`

- **BLKSSZGET**: `logical_block_size`

- **BLKIOMIN**: `minimum_io_size`

- **BLKIOOPT**: `optimal_io_size`

23.3. I/O STANDARDS

This section describes I/O standards used by ATA and SCSI devices.

ATA

ATA devices must report appropriate information via the **IDENTIFY DEVICE** command. ATA devices only report I/O parameters for **physical_block_size**, **logical_block_size**, and **alignment_offset**. The additional I/O hints are outside the scope of the ATA Command Set.

SCSI

I/O parameters support in Red Hat Enterprise Linux 7 requires at least *version 3* of the *SCSI Primary Commands* (SPC-3) protocol. The kernel will only send an *extended inquiry* (which gains access to the **BLOCK LIMITS VPD** page) and **READ CAPACITY(16)** command to devices which claim compliance with SPC-3.

The **READ CAPACITY(16)** command provides the block sizes and alignment offset:

- **LOGICAL BLOCK LENGTH IN BYTES** is used to derive **/sys/block/*disk*/queue/physical_block_size**

- **LOGICAL BLOCKS PER PHYSICAL BLOCK EXPONENT** is used to derive **/sys/block/*disk*/queue/logical_block_size**

- **LOWEST ALIGNED LOGICAL BLOCK ADDRESS** is used to derive:

  - **/sys/block/*disk*/alignment_offset**

  - **/sys/block/*disk*/*partition*/alignment_offset**

The **BLOCK LIMITS VPD** page (**0xb0**) provides the I/O hints. It also uses **OPTIMAL TRANSFER LENGTH GRANULARITY** and **OPTIMAL TRANSFER LENGTH** to derive:

- **/sys/block/*disk*/queue/minimum_io_size**

- **/sys/block/*disk*/queue/optimal_io_size**

The **sg3_utils** package provides the **sg_inq** utility, which can be used to access the **BLOCK LIMITS VPD** page. To do so, run:

```
# sg_inq -p 0xb0 disk
```

23.4. STACKING I/O PARAMETERS

All layers of the Linux I/O stack have been engineered to propagate the various I/O parameters up the stack. When a layer consumes an attribute or aggregates many devices, the layer must expose appropriate I/O parameters so that upper-layer devices or tools will have an accurate view of the storage as it transformed. Some practical examples are:

- Only one layer in the I/O stack should adjust for a non-zero **alignment_offset**; once a layer adjusts accordingly, it will export a device with an **alignment_offset** of zero.

- A striped Device Mapper (DM) device created with LVM must export a **minimum_io_size** and **optimal_io_size** relative to the stripe count (number of disks) and user-provided chunk size.

In Red Hat Enterprise Linux 7, Device Mapper and Software Raid (MD) device drivers can be used to arbitrarily combine devices with different I/O parameters. The kernel's block layer will attempt to reasonably combine the I/O parameters of the individual devices. The kernel will not prevent combining heterogeneous devices; however, be aware of the risks associated with doing so.

For instance, a 512-byte device and a 4K device may be combined into a single logical DM device, which would have a **logical_block_size** of 4K. File systems layered on such a hybrid device assume that 4K will be written atomically, but in reality it will span 8 logical block addresses when issued to the 512-byte device. Using a 4K **logical_block_size** for the higher-level DM device increases potential for a partial write to the 512-byte device if there is a system crash.

If combining the I/O parameters of multiple devices results in a conflict, the block layer may issue a warning that the device is susceptible to partial writes and/or is misaligned.

23.5. LOGICAL VOLUME MANAGER

LVM provides userspace tools that are used to manage the kernel's DM devices. LVM will shift the start of the data area (that a given DM device will use) to account for a non-zero **alignment_offset** associated with any device managed by LVM. This means logical volumes will be properly aligned (**alignment_offset=0**).

By default, LVM will adjust for any **alignment_offset**, but this behavior can be disabled by setting **data_alignment_offset_detection** to **0** in **/etc/lvm/lvm.conf**. Disabling this is not recommended.

LVM will also detect the I/O hints for a device. The start of a device's data area will be a multiple of the **minimum_io_size** or **optimal_io_size** exposed in sysfs. LVM will use the **minimum_io_size** if **optimal_io_size** is undefined (i.e. **0**).

By default, LVM will automatically determine these I/O hints, but this behavior can be disabled by setting **data_alignment_detection** to **0** in **/etc/lvm/lvm.conf**. Disabling this is not recommended.

23.6. PARTITION AND FILE SYSTEM TOOLS

This section describes how different partition and file system management tools interact with a device's I/O parameters.

util-linux-ng's libblkid and fdisk

The **libblkid** library provided with the **util-linux-ng** package includes a programmatic API to access a device's I/O parameters. **libblkid** allows applications, especially those that use Direct I/O, to properly size their I/O requests. The **fdisk** utility from **util-linux-ng** uses **libblkid** to determine the I/O parameters of a device for optimal placement of all partitions. The **fdisk** utility will align all partitions on a 1MB boundary.

parted and libparted

The **libparted** library from **parted** also uses the I/O parameters API of **libblkid**. **Anaconda**, the Red Hat Enterprise Linux 7 installer, uses **libparted**, which means that all partitions created by either the installer or **parted** will be properly aligned. For all partitions created on a device that does not appear to provide I/O parameters, the default alignment will be 1MB.

The heuristics **parted** uses are as follows:

- Always use the reported **alignment_offset** as the offset for the start of the first primary partition.

- If **optimal_io_size** is defined (i.e. not **0**), align all partitions on an **optimal_io_size** boundary.

- If **optimal_io_size** is undefined (i.e. **0**), **alignment_offset** is **0**, and **minimum_io_size** is a power of 2, use a 1MB default alignment.

  This is the catch-all for "legacy" devices which don't appear to provide I/O hints. As such, by default all partitions will be aligned on a 1MB boundary.
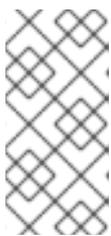
> **NOTE**
>
> Red Hat Enterprise Linux 7 cannot distinguish between devices that don't provide I/O hints and those that do so with **alignment_offset=0** and **optimal_io_size=0**. Such a device might be a single SAS 4K device; as such, at worst 1MB of space is lost at the start of the disk.

File System Tools

The different **mkfs.*filesystem*** utilities have also been enhanced to consume a device's I/O parameters. These utilities will not allow a file system to be formatted to use a block size smaller than the **logical_block_size** of the underlying storage device.

Except for **mkfs.gfs2**, all other **mkfs.*filesystem*** utilities also use the I/O hints to layout on-disk data structure and data areas relative to the **minimum_io_size** and **optimal_io_size** of the underlying storage device. This allows file systems to be optimally formatted for various RAID (striped) layouts.

CHAPTER 24. SETTING UP A REMOTE DISKLESS SYSTEM

To set up a basic remote diskless system booted over PXE, you need the following packages:

- **tftp-server**

- **xinetd**

- **dhcp**

- **syslinux**

- **dracut-network**

> **NOTE**
>
> After installing the **dracut-network** package, add the following line to **/etc/dracut.conf**:
>
> ```
> add_dracutmodules+="nfs"
> ```

Remote diskless system booting requires both a **tftp** service (provided by **tftp-server**) and a DHCP service (provided by **dhcp**). The **tftp** service is used to retrieve kernel image and **initrd** over the network via the PXE loader.

> **NOTE**
>
> SELinux is only supported over NFSv4.2. To use SELinux, NFS must be explicitly enabled in **/etc/sysconfig/nfs** by adding the line:
>
> **RPCNFSDARGS="-V 4.2"**
>
> Then, in **/var/lib/tftpboot/pxelinux.cfg/default**, change **root=nfs:server-ip:/exported/root/directory** to **root=nfs:server-ip:/exported/root/directory,vers=4.2**.
>
> Finally, reboot the NFS server.

The following sections outline the necessary procedures for deploying remote diskless systems in a network environment.

> **IMPORTANT**
>
> Some RPM packages have started using file capabilities (such as **setcap** and **getcap**). However, NFS does not currently support these so attempting to install or update any packages that use file capabilities will fail.

24.1. CONFIGURING A TFTP SERVICE FOR DISKLESS CLIENTS

Prerequisites

- Install the necessary packages. See Chapter 24, *Setting up a Remote Diskless System*

Procedure

To configure **tftp**, perform the following steps:

Procedure 24.1. To Configure tftp

1. Enable PXE booting over the network:

   ```
   # systemctl enable --now tftp
   ```

2. The **tftp** root directory (**chroot**) is located in **/var/lib/tftpboot**. Copy **/usr/share/syslinux/pxelinux.0** to **/var/lib/tftpboot/**:

   ```
   # cp /usr/share/syslinux/pxelinux.0 /var/lib/tftpboot/
   ```

3. Create a **pxelinux.cfg** directory inside the **tftp** root directory:

   ```
   # mkdir -p /var/lib/tftpboot/pxelinux.cfg/
   ```

4. Configure firewall rules to allow **tftp** traffic.

   As **tftp** supports TCP wrappers, you can configure host access to **tftp** in the **/etc/hosts.allow** configuration file. For more information on configuring TCP wrappers and the **/etc/hosts.allow** configuration file, see the Red Hat Enterprise Linux 7 Security Guide. The hosts_access(5) also provides information about **/etc/hosts.allow**.

Next Steps

After configuring **tftp** for diskless clients, configure DHCP, NFS, and the exported file system accordingly. For instructions on configuring the DHCP, NFS, and the exported file system, see Section 24.2, "Configuring DHCP for Diskless Clients" and Section 24.3, "Configuring an Exported File System for Diskless Clients".

24.2. CONFIGURING DHCP FOR DISKLESS CLIENTS

Prerequisites

- Install the necessary packages. See Chapter 24, *Setting up a Remote Diskless System*

- Configure the **tftp** service. See Section 24.1, "Configuring a tftp Service for Diskless Clients".

Procedure

1. After configuring a **tftp** server, you need to set up a DHCP service on the same host machine. For instructions on setting up a DHCP server, see the Configuring a DHCP Server.

2. Enable PXE booting on the DHCP server by adding the following configuration to **/etc/dhcp/dhcp.conf**:

   ```
   allow booting;
   allow bootp;
   class "pxeclients" {
       match if substring(option vendor-class-identifier, 0, 9) =
   "PXEClient";
   ```

```
    next-server server-ip;
    filename "pxelinux.0";
}
```

- Replace *server-ip* with the IP address of the host machine on which the **tftp** and DHCP services reside.

> **NOTE**
>
> When **libvirt** virtual machines are used as the diskless client, **libvirt** provides the DHCP service and the stand alone DHCP server is not used. In this situation, network booting must be enabled with the **bootp file='*filename*'** option in the **libvirt** network configuration, **virsh net-edit**.

Next Steps

Now that **tftp** and DHCP are configured, configure NFS and the exported file system. For instructions, see the Section 24.3, "Configuring an Exported File System for Diskless Clients".

24.3. CONFIGURING AN EXPORTED FILE SYSTEM FOR DISKLESS CLIENTS

Prerequisites

- Install the necessary packages. See Chapter 24, *Setting up a Remote Diskless System*

- Configure the **tftp** service. See Section 24.1, "Configuring a tftp Service for Diskless Clients".

- Configure DHCP. See Section 24.2, "Configuring DHCP for Diskless Clients".

Procedure

1. The root directory of the exported file system (used by diskless clients in the network) is shared via NFS. Configure the NFS service to export the root directory by adding it to **/etc/exports**. For instructions on how to do so, see the Section 8.7.1, "The **/etc/exports** Configuration File".

2. To accommodate completely diskless clients, the root directory should contain a complete Red Hat Enterprise Linux installation. You can either clone an existing installation or install a new base system:

   - To synchronize with a running system, use the **rsync** utility:

     ```
     # rsync -a -e ssh --exclude='/proc/*' --exclude='/sys/*' \
     hostname.com:/exported-root-directory
     ```

     - Replace *hostname.com* with the hostname of the running system with which to synchronize via **rsync**.

     - Replace *exported-root-directory* with the path to the exported file system.

   - To install Red Hat Enterprise Linux to the exported location, use the **yum** utility with the **--installroot** option:

```
# yum install @Base kernel dracut-network nfs-utils \ --
installroot=exported-root-directory --releasever=/
```

The file system to be exported still needs to be configured further before it can be used by diskless clients. To do this, perform the following procedure:

Procedure 24.2. Configure File System

1. Select the kernel that diskless clients should use (**vmlinuz-kernel-version**) and copy it to the **tftp** boot directory:

   ```
   # cp /boot/vmlinuz-kernel-version /var/lib/tftpboot/
   ```

2. Create the **initrd** (that is, **initramfs-kernel-version.img**) with network support:

   ```
   # dracut initramfs-kernel-version.img kernel-version
   ```

3. Change the initrd's file permissions to 644 using the following command:

   ```
   # chmod 644 initramfs-kernel-version.img
   ```

   > **WARNING**
   >
   > If the initrd's file permissions are not changed, the **pxelinux.0** boot loader will fail with a "file not found" error.

4. Copy the resulting **initramfs-kernel-version.img** into the **tftp** boot directory as well.

5. Edit the default boot configuration to use the **initrd** and kernel in the **/var/lib/tftpboot/** directory. This configuration should instruct the diskless client's root to mount the exported file system (**/exported/root/directory**) as read-write. Add the following configuration in the **/var/lib/tftpboot/pxelinux.cfg/default** file:

   ```
   default rhel7

   label rhel7
     kernel vmlinuz-kernel-version
     append initrd=initramfs-kernel-version.img root=nfs:server-
   ip:/exported/root/directory rw
   ```

   Replace **server-ip** with the IP address of the host machine on which the **tftp** and DHCP services reside.

The NFS share is now ready for exporting to diskless clients. These clients can boot over the network via PXE.

CHAPTER 25. ONLINE STORAGE MANAGEMENT

It is often desirable to add, remove or re-size storage devices while the operating system is running, and without rebooting. This chapter outlines the procedures that may be used to reconfigure storage devices on Red Hat Enterprise Linux 7 host systems while the system is running. It covers iSCSI and Fibre Channel storage interconnects; other interconnect types may be added it the future.

This chapter focuses on adding, removing, modifying, and monitoring storage devices. It does not discuss the Fibre Channel or iSCSI protocols in detail. For more information about these protocols, refer to other documentation.

This chapter makes reference to various **sysfs** objects. Red Hat advises that the **sysfs** object names and directory structure are subject to change in major Red Hat Enterprise Linux releases. This is because the upstream Linux kernel does not provide a stable internal API. For guidelines on how to reference **sysfs** objects in a transportable way, refer to the document **/usr/share/doc/kernel-doc-*version*/Documentation/sysfs-rules.txt** in the kernel source tree for guidelines.

> **⚠ WARNING**
>
> Online storage reconfiguration must be done carefully. System failures or interruptions during the process can lead to unexpected results. Red Hat advises that you reduce system load to the maximum extent possible during the change operations. This will reduce the chance of I/O errors, out-of-memory errors, or similar errors occurring in the midst of a configuration change. The following sections provide more specific guidelines regarding this.
>
> In addition, Red Hat recommends that you back up all data before reconfiguring online storage.

25.1. TARGET SETUP

Red Hat Enterprise Linux 7 uses the **targetcli** shell as a front end for viewing, editing, and saving the configuration of the Linux-IO Target without the need to manipulate the kernel target's configuration files directly. The **targetcli** tool is a command-line interface that allows an administrator to export local storage resources, which are backed by either files, volumes, local SCSI devices, or RAM disks, to remote systems. The **targetcli** tool has a tree-based layout, includes built-in tab completion, and provides full auto-complete support and inline documentation.

The hierarchy of **targetcli** does not always match the kernel interface exactly because **targetcli** is simplified where possible.
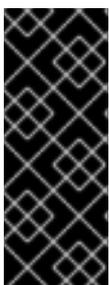
> **IMPORTANT**
>
> To ensure that the changes made in **targetcli** are persistent, start and enable the target service:
>
> ```
> # systemctl start target
> # systemctl enable target
> ```

25.1.1. Installing and Running targetcli

To install **targetcli**, use:

```
# yum install targetcli
```

Start the **target** service:

```
# systemctl start target
```

Configure **target** to start at boot time:

```
# systemctl enable target
```

Open port **3260** in the firewall and reload the firewall configuration:

```
# firewall-cmd --permanent --add-port=3260/tcp
Success
# firewall-cmd --reload
Success
```

Use the **targetcli** command, and then use the **ls** command for the layout of the tree interface:

```
# targetcli
:
/> ls
o- /........................................[...]
  o- backstores.............................[...]
  | o- block.................[Storage Objects: 0]
  | o- fileio................[Storage Objects: 0]
  | o- pscsi.................[Storage Objects: 0]
  | o- ramdisk...............[Storage Ojbects: 0]
  o- iscsi..........................[Targets: 0]
  o- loopback.......................[Targets: 0]
```

> **NOTE**
>
> In Red Hat Enterprise Linux 7.0, using the **targetcli** command from Bash, for example, **targetcli iscsi/ create**, does not work and does not return an error. Starting with Red Hat Enterprise Linux 7.1, an error status code is provided to make using **targetcli** with shell scripts more useful.

25.1.2. Creating a Backstore

Backstores enable support for different methods of storing an exported LUN's data on the local machine. Creating a storage object defines the resources the backstore uses.

NOTE

In Red Hat Enterprise Linux 6, the term 'backing-store' is used to refer to the mappings created. However, to avoid confusion between the various ways 'backstores' can be used, in Red Hat Enterprise Linux 7 the term 'storage objects' refers to the mappings created and 'backstores' is used to describe the different types of backing devices.

The backstore devices that LIO supports are:

FILEIO (Linux file-backed storage)

FILEIO storage objects can support either **write_back** or **write_thru** operation. The **write_back** enables the local file system cache. This improves performance but increases the risk of data loss. It is recommended to use **write_back=false** to disable **write_back** in favor of **write_thru**.

To create a fileio storage object, run the command **/backstores/fileio create *file_name* *file_location file_size* write_back=false**. For example:

```
/> /backstores/fileio create file1 /tmp/disk1.img 200M write_back=false
Created fileio file1 with size 209715200
```

BLOCK (Linux BLOCK devices)

The block driver allows the use of any block device that appears in the **/sys/block** to be used with LIO. This includes physical devices (for example, HDDs, SSDs, CDs, DVDs) and logical devices (for example, software or hardware RAID volumes, or LVM volumes).

NOTE

BLOCK backstores usually provide the best performance.

To create a BLOCK backstore using any block device, use the following command:

```
# fdisk /dev/vdb
Welcome to fdisk (util-linux 2.23.2).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table
Building a new DOS disklabel with disk identifier 0x39dc48fb.

Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): *Enter*
Using default response p
Partition number (1-4, default 1): *Enter*
First sector (2048-2097151, default 2048): *Enter*
Using default value 2048
Last sector, +sectors or +size{K,M,G} (2048-2097151, default 2097151):
+250M
Partition 1 of type Linux and of size 250 MiB is set
```

```
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

```
/> /backstores/block create name=block_backend dev=/dev/vdb
Generating a wwn serial.
Created block storage object block_backend using /dev/vdb.
```

> **NOTE**
>
> You can also create a BLOCK backstore on a logical volume.

PSCSI (Linux pass-through SCSI devices)

Any storage object that supports direct pass-through of SCSI commands without SCSI emulation, and with an underlying SCSI device that appears with lsscsi in **/proc/scsi/scsi** (such as a SAS hard drive) can be configured as a backstore. SCSI-3 and higher is supported with this subsystem.

> **WARNING**
>
> PSCSI should only be used by advanced users. Advanced SCSI commands such as for Aysmmetric Logical Unit Assignment (ALUAs) or Persistent Reservations (for example, those used by VMware ESX, and vSphere) are usually not implemented in the device firmware and can cause malfunctions or crashes. When in doubt, use BLOCK for production setups instead.

To create a PSCSI backstore for a physical SCSI device, a **TYPE_ROM** device using **/dev/sr0** in this example, use:

```
/> backstores/pscsi/ create name=pscsi_backend dev=/dev/sr0
Generating a wwn serial.
Created pscsi storage object pscsi_backend using /dev/sr0
```

Memory Copy RAM disk (Linux RAMDISK_MCP)

Memory Copy RAM disks (**ramdisk**) provide RAM disks with full SCSI emulation and separate memory mappings using memory copy for initiators. This provides capability for multi-sessions and is particularly useful for fast, volatile mass storage for production purposes.

To create a 1GB RAM disk backstore, use the following command:

```
/> backstores/ramdisk/ create name=rd_backend size=1GB
Generating a wwn serial.
Created rd_mcp ramdisk rd_backend with size 1GB.
```

25.1.3. Creating an iSCSI Target

To create an iSCSI target:

Procedure 25.1. Creating an iSCSI target

1. Run **targetcli**.

2. Move into the iSCSI configuration path:

   ```
   /> iscsi/
   ```

   > **NOTE**
   >
   > The **cd** command is also accepted to change directories, as well as simply listing the path to move into.

3. Create an iSCSI target using a default target name.

   ```
   /iscsi> create
   Created target
   iqn.2003-01.org.linux-iscsi.hostname.x8664:sn.78b473f296ff
   Created TPG1
   ```

   Or create an iSCSI target using a specified name.

   ```
   /iscsi > create iqn.2006-04.com.example:444
   Created target iqn.2006-04.com.example:444
   Created TPG1
   ```

4. Verify that the newly created target is visible when targets are listed with **ls**.

   ```
   /iscsi > ls
   o- iscsi.......................................[1 Target]
       o- iqn.2006-04.com.example:444................[1 TPG]
           o- tpg1...........................[enabled, auth]
               o- acls...............................[0 ACL]
               o- luns...............................[0 LUN]
               o- portals.........................[0 Portal]
   ```

> **NOTE**
>
> As of Red Hat Enterprise Linux 7.1, whenever a target is created, a default portal is also created.

25.1.4. Configuring an iSCSI Portal

To configure an iSCSI portal, an iSCSI target must first be created and associated with a TPG. For instructions on how to do this, refer to Section 25.1.3, "Creating an iSCSI Target".

NOTE

As of Red Hat Enterprise Linux 7.1 when an iSCSI target is created, a default portal is created as well. This portal is set to listen on all IP addresses with the default port number (that is, 0.0.0.0:3260). To remove this and add only specified portals, use */iscsi/iqn-name*/tpg1/portals delete ip_address=0.0.0.0 ip_port=3260 then create a new portal with the required information.

Procedure 25.2. Creating an iSCSI Portal

1. Move into the TPG.

   ```
   /iscsi> iqn.2006-04.example:444/tpg1/
   ```

2. There are two ways to create a portal: create a default portal, or create a portal specifying what IP address to listen to.

   Creating a default portal uses the default iSCSI port 3260 and allows the target to listen on all IP addresses on that port.

   ```
   /iscsi/iqn.20...mple:444/tpg1> portals/ create
   Using default IP port 3260
   Binding to INADDR_Any (0.0.0.0)
   Created network portal 0.0.0.0:3260
   ```

   To create a portal specifying what IP address to listen to, use the following command.

   ```
   /iscsi/iqn.20...mple:444/tpg1> portals/ create 192.168.122.137
   Using default IP port 3260
   Created network portal 192.168.122.137:3260
   ```

3. Verify that the newly created portal is visible with the **ls** command.

   ```
   /iscsi/iqn.20...mple:444/tpg1> ls
   o- tpg................................. [enambled, auth]
       o- acls ......................................[0 ACL]
       o- luns ......................................[0 LUN]
       o- portals ...............................[1 Portal]
           o- 192.168.122.137:3260......................[OK]
   ```

25.1.5. Configuring LUNs

To configure LUNs, first create storage objects. See Section 25.1.2, "Creating a Backstore" for more information.

Procedure 25.3. Configuring LUNs

1. Create LUNs of already created storage objects.

   ```
   /iscsi/iqn.20...mple:444/tpg1> luns/ create
   /backstores/ramdisk/rd_backend
   Created LUN 0.
   ```

```
/iscsi/iqn.20...mple:444/tpg1> luns/ create
/backstores/block/block_backend
Created LUN 1.

/iscsi/iqn.20...mple:444/tpg1> luns/ create /backstores/fileio/file1
Created LUN 2.
```

2. Show the changes.

```
/iscsi/iqn.20...mple:444/tpg1> ls
o- tpg............................... [enambled, auth]
   o- acls ...................................[0 ACL]
   o- luns ...................................[3 LUNs]
   |  o- lun0.........................[ramdisk/ramdisk1]
   |  o- lun1................[block/block1 (/dev/vdb1)]
   |  o- lun2..................[fileio/file1 (/foo.img)]
   o- portals ...............................[1 Portal]
      o- 192.168.122.137:3260.....................[OK]
```

> **NOTE**
>
> Be aware that the default LUN name starts at 0, as opposed to 1 as was the case when using **tgtd** in Red Hat Enterprise Linux 6.

> **IMPORTANT**
>
> By default, LUNs are created with read-write permissions. In the event that a new LUN is added after ACLs have been created that LUN will be automatically mapped to all available ACLs. This can cause a security risk. Use the following procedure to create a LUN as read-only.

Procedure 25.4. Create a Read-only LUN

1. To create a LUN with read-only permissions, first use the following command:

```
/> set global auto_add_mapped_luns=false
Parameter auto_add_mapped_luns is now 'false'.
```

This prevents the auto mapping of LUNs to existing ACLs allowing the manual mapping of LUNs.

2. Next, manually create the LUN with the command **iscsi/*target_iqn_name*/tpg1/acls/*initiator_iqn_name*/ create mapped_lun=*next_sequential_LUN_number* tpg_lun_or_backstore=*backstore* write_protect=1**.

```
/> iscsi/iqn.2015-06.com.redhat:target/tpg1/acls/iqn.2015-
06.com.redhat:initiator/ create mapped_lun=1
tpg_lun_or_backstore=/backstores/block/block2 write_protect=1
Created LUN 1.
Created Mapped LUN 1.
/> ls
o- / ............................................... [...]
  o- backstores ................................... [...]
```

```
      <snip>
      o- iscsi ......................................... [Targets: 1]
      | o- iqn.2015-06.com.redhat:target .................. [TPGs: 1]
      |    o- tpg1 ........................... [no-gen-acls, no-auth]
      |       o- acls ..................................... [ACLs: 2]
      |       | o- iqn.2015-06.com.redhat:initiator .. [Mapped LUNs: 2]
      |       | | o- mapped_lun0 .............. [lun0 block/disk1 (rw)]
      |       | | o- mapped_lun1 ............. [lun1 block/disk2 (ro)]
      |       o- luns ...................................... [LUNs: 2]
      |       | o- lun0 ..................... [block/disk1 (/dev/vdb)]
      |       | o- lun1 ..................... [block/disk2 (/dev/vdc)]
      <snip>
```

The mapped_lun1 line now has (ro) at the end (unlike mapped_lun0's (rw)) stating that it is read-only.

25.1.6. Configuring ACLs

Create an ACL for each initiator that will be connecting. This enforces authentication when that initiator connects, allowing only LUNs to be exposed to each initiator. Usually each initator has exclusive access to a LUN. Both targets and initiators have unique identifying names. The initiator's unique name must be known to configure ACLs. For open-iscsi initiators, this can be found in **/etc/iscsi/initiatorname.iscsi**.

Procedure 25.5. Configuring ACLs

1. Move into the acls directory.

   ```
   /iscsi/iqn.20...mple:444/tpg1> acls/
   ```

2. Create an ACL. Either use the initiator name found in **/etc/iscsi/initiatorname.iscsi** on the initiator, or if using a name that is easier to remember, refer to Section 25.2, "Creating an iSCSI Initiator" to ensure ACL matches the initiator. For example:

   ```
   /iscsi/iqn.20...444/tpg1/acls> create iqn.2006-
   04.com.example.foo:888
   Created Node ACL for iqn.2006-04.com.example.foo:888
   Created mapped LUN 2.
   Created mapped LUN 1.
   Created mapped LUN 0.
   ```

   NOTE

   The given example's behavior depends on the setting used. In this case, the global setting **auto_add_mapped_luns** is used. This automatically maps LUNs to any created ACL.

   You can set user-created ACLs within the TPG node on the target server:

   ```
   /iscsi/iqn.20...scsi:444/tpg1> set attribute
   generate_node_acls=1
   ```

3. Show the changes.

```
/iscsi/iqn.20...444/tpg1/acls> ls
o- acls .............................................[1 ACL]
    o- iqn.2006-04.com.example.foo:888 ....[3 Mapped LUNs, auth]
        o- mapped_lun0 .............[lun0 ramdisk/ramdisk1 (rw)]
        o- mapped_lun1 ................[lun1 block/block1 (rw)]
        o- mapped_lun2 ................[lun2 fileio/file1 (rw)]
```

25.1.7. Configuring Fibre Channel over Ethernet (FCoE) Target

In addition to mounting LUNs over FCoE, as described in Section 25.4, "Configuring a Fibre Channel over Ethernet Interface", exporting LUNs to other machines over FCoE is also supported with the aid of **targetcli**.

> **IMPORTANT**
>
> Before proceeding, refer to Section 25.4, "Configuring a Fibre Channel over Ethernet Interface" and verify that basic FCoE setup is completed, and that **fcoeadm -i** displays configured FCoE interfaces.

Procedure 25.6. Configure FCoE target

1. Setting up an FCoE target requires the installation of the **targetcli** package, along with its dependencies. Refer to Section 25.1, "Target Setup" for more information on **targetcli** basics and set up.

2. Create an FCoE target instance on an FCoE interface.

   ```
   /> tcm_fc/ create 00:11:22:33:44:55:66:77
   ```

   If FCoE interfaces are present on the system, tab-completing after **create** will list available interfaces. If not, ensure **fcoeadm -i** shows active interfaces.

3. Map a backstore to the target instance.

   Example 25.1. Example of Mapping a Backstore to the Target Instance

   ```
   /> tcm_fc/00:11:22:33:44:55:66:77
   ```

   ```
   /> luns/ create /backstores/fileio/example2
   ```

4. Allow access to the LUN from an FCoE initiator.

   ```
   /> acls/ create 00:99:88:77:66:55:44:33
   ```

   The LUN should now be accessible to that initiator.

5. To make the changes persistent across reboots, use the **saveconfig** command and type **yes** when prompted. If this is not done the configuration will be lost after rebooting.

6. Exit **targetcli** by typing **exit** or entering **ctrl**+**D**.

25.1.8. Removing Objects with `targetcli`

To remove an backstore use the command:

```
/> /backstores/backstore-type/backstore-name
```

To remove parts of an iSCSI target, such as an ACL, use the following command:

```
/> /iscsi/iqn-name/tpg/acls/ delete iqn-name
```

To remove the entire target, including all ACLs, LUNs, and portals, use the following command:

```
/> /iscsi delete iqn-name
```

25.1.9. `targetcli` References

For more information on **targetcli**, refer to the following resources:

man targetcli

The **targetcli** man page. It includes an example walk through.

The Linux SCSI Target Wiki

http://linux-iscsi.org/wiki/Targetcli

Screencast by Andy Grover

https://www.youtube.com/watch?v=BkBGTBadOO8
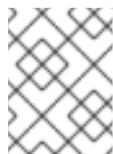
> **NOTE**
>
> This was uploaded on February 28, 2012. As such, the service name has changed from **targetcli** to **target**.

25.2. CREATING AN ISCSI INITIATOR

After creating a target with **targetcli** as in Section 25.1, "Target Setup", use the **iscsiadm** utility to set up an initiator.

In Red Hat Enterprise Linux 7, the iSCSI service is lazily started by default: the service starts after running the **iscsiadm** command.

Procedure 25.7. Creating an iSCSI Initiator

1. Install **iscsi-initiator-utils**:

   ```
   # yum install iscsi-initiator-utils -y
   ```

2. If the ACL was given a custom name in Section 25.1.6, "Configuring ACLs", modify the **/etc/iscsi/initiatorname.iscsi** file accordingly. For example:

```
# cat /etc/iscsi/initiatorname.iscsi
InitiatorName=iqn.2006-04.com.example.node1

# vi /etc/iscsi/initiatorname.iscsi
```

3. Discover the target:

```
# iscsiadm -m discovery -t st -p target-ip-address
10.64.24.179:3260,1 iqn.2006-04.com.example:3260
```

4. Log in to the target with the target IQN you discovered in step 3:

```
# iscsiadm -m node -T iqn.2006-04.com.example:3260 -l
Logging in to [iface: default, target: iqn.2006-04.com.example:3260,
portal: 10.64.24.179,3260] (multiple)
Login to [iface: default, target: iqn.2006-04.com.example:3260,
portal: 10.64.24.179,3260] successful.
```

This procedure can be followed for any number of initators connected to the same LUN so long as their specific initiator names are added to the ACL as described in Section 25.1.6, "Configuring ACLs".

5. Find the iSCSI disk name and create a file system on this iSCSI dick:

```
# grep "Attached SCSI" /var/log/messages

# mkfs.ext4 /dev/disk_name
```

Replace *disk_name* with the iSCSI disk name displayed in **/var/log/messages**.

6. Mount the file system:

```
# mkdir /mount/point
# mount /dev/disk_name /mount/point
```

Replace */mount/point* with the mount point of the partition.

7. Edit the **/etc/fstab** to mount the file system automatically when the system boots:

```
# vim /etc/fstab
/dev/disk_name /mount/point ext4 _netdev 0 0
```

Replace *disk_name* with the iSCSI disk name.

8. Log off from the target:

```
# iscsiadm -m node -T iqn.2006-04.com.example:3260 -u
```

25.3. FIBRE CHANNEL

This section discusses the Fibre Channel API, native Red Hat Enterprise Linux 7 Fibre Channel drivers, and the Fibre Channel capabilities of these drivers.

25.3.1. Fibre Channel API

Following is a list of **/sys/class/** directories that contain files used to provide the userspace API. In each item, host numbers are designated by **H**, bus numbers are **B**, targets are **T**, logical unit numbers (LUNs) are **L**, and remote port numbers are **R**.

> **IMPORTANT**
>
> If your system is using multipath software, Red Hat recommends that you consult your hardware vendor before changing any of the values described in this section.

Transport: /sys/class/fc_transport/targetH:B:T/

- **port_id** — 24-bit port ID/address

- **node_name** — 64-bit node name

- **port_name** — 64-bit port name

Remote Port: /sys/class/fc_remote_ports/rport-H:B-R/

- **port_id**

- **node_name**

- **port_name**

- **dev_loss_tmo**: controls when the scsi device gets removed from the system. After **dev_loss_tmo** triggers, the scsi device is removed.

  In **multipath.conf**, you can set **dev_loss_tmo** to **infinity**, which sets its value to 2,147,483,647 seconds, or 68 years, and is the maximum **dev_loss_tmo** value.

  In Red Hat Enterprise Linux 7, if you do not set the **fast_io_fail_tmo** option, **dev_loss_tmo** is capped to 600 seconds. By default, **fast_io_fail_tmo** is set to 5 seconds in Red Hat Enterprise Linux 7 if the **multipathd** service is running; otherwise, it is set to **off**.

- **fast_io_fail_tmo**: specifies the number of seconds to wait before it marks a link as "bad". Once a link is marked bad, existing running I/O or any new I/O on its corresponding path fails.

  If I/O is in a blocked queue, it will not be failed until **dev_loss_tmo** expires and the queue is unblocked.

  If **fast_io_fail_tmo** is set to any value except **off**, **dev_loss_tmo** is uncapped. If **fast_io_fail_tmo** is set to **off**, no I/O fails until the device is removed from the system. If **fast_io_fail_tmo** is set to a number, I/O fails immediately when the **fast_io_fail_tmo** timeout triggers.

Host: /sys/class/fc_host/hostH/

- **port_id**

- **issue_lip**: instructs the driver to rediscover remote ports.

25.3.2. Native Fibre Channel Drivers and Capabilities

Red Hat Enterprise Linux 7 ships with the following native Fibre Channel drivers:

- **lpfc**

- **qla2xxx**

- **zfcp**

- **bfa**

> **IMPORTANT**
>
> The **qla2xxx** driver runs in initiator mode by default. To use qla2xxx with Linux-IO, enable Fibre Channel target mode with the corresponding *qlini_mode* module parameter.
>
> First, make sure that the firmware package for your qla device, such as ql2200-firmware or similar, is installed.
>
> To enable target mode, add the following parameter to the **/usr/lib/modprobe.d/qla2xxx.conf** qla2xxx module configuration file:
>
> ```
> options qla2xxx qlini_mode=disabled
> ```
>
> Then, use the **dracut -f** command to rebuild the initial ramdisk (**initrd**), and reboot the system for the changes to take effect.

Table 25.1, "Fibre Channel API Capabilities" describes the different Fibre Channel API capabilities of each native Red Hat Enterprise Linux 7 driver. X denotes support for the capability.

Table 25.1. Fibre Channel API Capabilities

	lpfc	**qla2xxx**	**zfcp**	**bfa**
Transport **port_id**	X	X	X	X
Transport **node_name**	X	X	X	X
Transport **port_name**	X	X	X	X
Remote Port **dev_loss_tmo**	X	X	X	X

	lpfc	qla2xxx	zfcp	bfa
Remote Port **fast_io_fail _tmo**	X	X [a]	X [b]	X
Host **port_id**	X	X	X	X
Host **issue_lip**	X	X		X

[a] Supported as of Red Hat Enterprise Linux 5.4

[b] Supported as of Red Hat Enterprise Linux 6.0

25.4. CONFIGURING A FIBRE CHANNEL OVER ETHERNET INTERFACE

Setting up and deploying a Fibre Channel over Ethernet (FCoE) interface requires two packages:

- **fcoe-utils**

- **lldpad**

Once these packages are installed, perform the following procedure to enable FCoE over a virtual LAN (VLAN):

Procedure 25.8. Configuring an Ethernet Interface to Use FCoE

1. To configure a new VLAN, make a copy of an existing network script, for example **/etc/fcoe/cfg-eth0**, and change the name to the Ethernet device that supports FCoE. This provides you with a default file to configure. Given that the FCoE device is **ethX**, run:

   ```
   # cp /etc/fcoe/cfg-ethx  /etc/fcoe/cfg-ethX
   ```

   Modify the contents of **cfg-ethX** as needed. Notably, set **DCB_REQUIRED** to **no** for networking interfaces that implement a hardware Data Center Bridging Exchange (DCBX) protocol client.

2. If you want the device to automatically load during boot time, set **ONBOOT=yes** in the corresponding **/etc/sysconfig/network-scripts/ifcfg-ethX** file. For example, if the FCoE device is eth2, edit **/etc/sysconfig/network-scripts/ifcfg-eth2** accordingly.

3. Start the data center bridging daemon (**dcbd**) by running:

   ```
   # systemctl start lldpad
   ```

4. For networking interfaces that implement a hardware DCBX client, skip this step.

   For interfaces that require a software DCBX client, enable data center bridging on the Ethernet interface by running:

   ```
   # dcbtool sc ethX dcb on
   ```

Then, enable FCoE on the Ethernet interface by running:

```
# dcbtool sc ethX app:fcoe e:1
```

Note that these commands only work if the **dcbd** settings for the Ethernet interface were not changed.

5. Load the FCoE device now using:

```
# ip link set dev ethX up
```

6. Start FCoE using:

```
# systemctl start fcoe
```

The FCoE device appears soon if all other settings on the fabric are correct. To view configured FCoE devices, run:

```
# fcoeadm -i
```

After correctly configuring the Ethernet interface to use FCoE, Red Hat recommends that you set FCoE and the **lldpad** service to run at startup. To do so, use the **systemctl** utility:

```
# systemctl enable lldpad
```

```
# systemctl enable fcoe
```

> **NOTE**
>
> Running the # **systemctl stop fcoe** command stops the daemon, but does not reset the configuration of FCoE interfaces. To do so, run the # **systemctl -s SIGHUP kill fcoe** command.

As of Red Hat Enterprise Linux 7, Network Manager has the ability to query and set the DCB settings of a DCB capable Ethernet interface.

25.5. CONFIGURING AN FCOE INTERFACE TO AUTOMATICALLY MOUNT AT BOOT

> **NOTE**
>
> The instructions in this section are available in **/usr/share/doc/fcoe-utils-*version*/README** as of Red Hat Enterprise Linux 6.1. Refer to that document for any possible changes throughout minor releases.

You can mount newly discovered disks via **udev** rules, **autofs**, and other similar methods. Sometimes, however, a specific service might require the FCoE disk to be mounted at boot-time. In such cases, the FCoE disk should be mounted *as soon as* the **fcoe** service runs and *before* the initiation of any service that requires the FCoE disk.

To configure an FCoE disk to automatically mount at boot, add proper FCoE mounting code to the startup script for the **fcoe** service. The **fcoe** startup script is **/lib/systemd/system/fcoe.service**.

The FCoE mounting code is different per system configuration, whether you are using a simple formatted FCoE disk, LVM, or multipathed device node.

> **Example 25.2. FCoE Mounting Code**
>
> The following is a sample FCoE mounting code for mounting file systems specified via wild cards in **/etc/fstab**:
>
> ```
> mount_fcoe_disks_from_fstab()
>  {
>     local timeout=20
>     local done=1
>     local fcoe_disks=($(egrep 'by-path\/fc-.*_netdev' /etc/fstab | cut
> -d ' ' -f1))
>
>     test -z $fcoe_disks && return 0
>
>     echo -n "Waiting for fcoe disks . "
>     while [ $timeout -gt 0 ]; do
>   for disk in ${fcoe_disks[*]}; do
>    if ! test -b $disk; then
>     done=0
>     break
>    fi
>   done
>
>     test $done -eq 1 && break;
>     sleep 1
>     echo -n ". "
>     done=1
>     let timeout--
>        done
>
>     if test $timeout -eq 0; then
>   echo "timeout!"
>     else
>   echo "done!"
>     fi
>
>     # mount any newly discovered disk
>     mount -a 2>/dev/null
>  }
> ```

The **mount_fcoe_disks_from_fstab** function should be invoked *after* the **fcoe** service script starts the **fcoemon** daemon. This will mount FCoE disks specified by the following paths in **/etc/fstab**:

```
/dev/disk/by-path/fc-0xXX:0xXX /mnt/fcoe-disk1 ext3  defaults,_netdev    0
0
/dev/disk/by-path/fc-0xYY:0xYY /mnt/fcoe-disk2 ext3  defaults,_netdev    0
0
```

Entries with **fc-** and **_netdev** sub-strings enable the **mount_fcoe_disks_from_fstab** function to identify FCoE disk mount entries. For more information on **/etc/fstab** entries, refer to **man 5 fstab**.

> **NOTE**
>
> The **fcoe** service does not implement a timeout for FCoE disk discovery. As such, the FCoE mounting code should implement its own timeout period.

25.6. ISCSI

This section describes the iSCSI API and the **iscsiadm** utility. Before using the **iscsiadm** utility, install the **iscsi-initiator-utils** package first by running **yum install iscsi-initiator-utils**.

In Red Hat Enterprise Linux 7, the iSCSI service is lazily started by default. If root is not on an iSCSI device or there are no nodes marked with **node.startup = automatic** then the iSCSI service will not start until an **iscsiadm** command is run that requires iscsid or the iscsi kernel modules to be started. For example, running the discovery command **iscsiadm -m discovery -t st -p ip:port** will cause iscsiadmin to start the iSCSI service.

To force the iscsid daemon to run and iSCSI kernel modules to load, run **systemctl start iscsid.service**.

25.6.1. iSCSI API

To get information about running sessions, run:

```
# iscsiadm -m session -P 3
```

This command displays the session/device state, session ID (sid), some negotiated parameters, and the SCSI devices accessible through the session.

For shorter output (for example, to display only the sid-to-node mapping), run:

```
# iscsiadm -m session -P 0
```

or

```
# iscsiadm -m session
```

These commands print the list of running sessions with the format:

```
driver [sid] target_ip:port,target_portal_group_tag proper_target_name
```

Example 25.3. Output of the iscsisadm -m session Command

For example:

```
# iscsiadm -m session

tcp [2] 10.15.84.19:3260,2 iqn.1992-08.com.netapp:sn.33615311
tcp [3] 10.15.85.19:3260,3 iqn.1992-08.com.netapp:sn.33615311
```

For more information about the iSCSI API, refer to **/usr/share/doc/iscsi-initiator-utils-*version*/README**.

25.7. PERSISTENT NAMING

Red Hat Enterprise Linux provides a number of ways to identify storage devices. It is important to use the correct option to identify each device when used in order to avoid inadvertently accessing the wrong device, particularly when installing to or reformatting drives.

25.7.1. Major and Minor Numbers of Storage Devices

Storage devices managed by the **sd** driver are identified internally by a collection of major device numbers and their associated minor numbers. The major device numbers used for this purpose are not in a contiguous range. Each storage device is represented by a major number and a range of minor numbers, which are used to identify either the entire device or a partition within the device. There is a direct association between the major and minor numbers allocated to a device and numbers in the form of **sd<*letter(s)>[number(s)]***. Whenever the **sd** driver detects a new device, an available major number and minor number range is allocated. Whenever a device is removed from the operating system, the major number and minor number range is freed for later reuse.

The major and minor number range and associated **sd** names are allocated for each device when it is detected. This means that the association between the major and minor number range and associated **sd** names can change if the order of device detection changes. Although this is unusual with some hardware configurations (for example, with an internal SCSI controller and disks that have their SCSI target ID assigned by their physical location within a chassis), it can nevertheless occur. Examples of situations where this can happen are as follows:

- A disk may fail to power up or respond to the SCSI controller. This will result in it not being detected by the normal device probe. The disk will not be accessible to the system and subsequent devices will have their major and minor number range, including the associated **sd** names shifted down. For example, if a disk normally referred to as **sdb** is not detected, a disk that is normally referred to as **sdc** would instead appear as **sdb**.

- A SCSI controller (host bus adapter, or HBA) may fail to initialize, causing all disks connected to that HBA to not be detected. Any disks connected to subsequently probed HBAs would be assigned different major and minor number ranges, and different associated **sd** names.

- The order of driver initialization could change if different types of HBAs are present in the system. This would cause the disks connected to those HBAs to be detected in a different order. This can also occur if HBAs are moved to different PCI slots on the system.

- Disks connected to the system with Fibre Channel, iSCSI, or FCoE adapters might be inaccessible at the time the storage devices are probed, due to a storage array or intervening switch being powered off, for example. This could occur when a system reboots after a power failure, if the storage array takes longer to come online than the system take to boot. Although some Fibre Channel drivers support a mechanism to specify a persistent SCSI target ID to WWPN mapping, this will not cause the major and minor number ranges, and the associated **sd** names to be reserved, it will only provide consistent SCSI target ID numbers.

These reasons make it undesirable to use the major and minor number range or the associated **sd** names when referring to devices, such as in the **/etc/fstab** file. There is the possibility that the wrong device will be mounted and data corruption could result.

Occasionally, however, it is still necessary to refer to the **sd** names even when another mechanism is used (such as when errors are reported by a device). This is because the Linux kernel uses **sd** names (and also SCSI host/channel/target/LUN tuples) in kernel messages regarding the device.

25.7.2. World Wide Identifier (WWID)

The *World Wide Identifier* (WWID) can be used in reliably identifying devices. It is a persistent, system-independent ID that the SCSI Standard requires from all SCSI devices. The WWID identifier is guaranteed to be unique for every storage device, and independent of the path that is used to access the device.

This identifier can be obtained by issuing a SCSI Inquiry to retrieve the *Device Identification Vital Product Data* (page **0x83**) or *Unit Serial Number* (page **0x80**). The mappings from these WWIDs to the current **/dev/sd** names can be seen in the symlinks maintained in the **/dev/disk/by-id/** directory.

> **Example 25.4. WWID**
>
> For example, a device with a page **0x83** identifier would have:
>
> ```
> scsi-3600508b400105e210000900000490000 -> ../../sda
> ```
>
> Or, a device with a page **0x80** identifier would have:
>
> ```
> scsi-SSEAGATE_ST373453LW_3HW1RHM6 -> ../../sda
> ```

Red Hat Enterprise Linux automatically maintains the proper mapping from the WWID-based device name to a current **/dev/sd** name on that system. Applications can use the **/dev/disk/by-id/** name to reference the data on the disk, even if the path to the device changes, and even when accessing the device from different systems.

If there are multiple paths from a system to a device, **DM Multipath** uses the WWID to detect this. **DM Multipath** then presents a single "pseudo-device" in the **/dev/mapper/wwid** directory, such as **/dev/mapper/3600508b400105df70000e00000ac0000**.

The command **multipath -l** shows the mapping to the non-persistent identifiers: *Host*:*Channel*:*Target*:*LUN*, **/dev/sd** name, and the **major:minor** number.

```
3600508b400105df70000e00000ac0000 dm-2 vendor,product
[size=20G][features=1 queue_if_no_path][hwhandler=0][rw]
\_ round-robin 0 [prio=0][active]
 \_ 5:0:1:1 sdc 8:32  [active][undef]
 \_ 6:0:1:1 sdg 8:96  [active][undef]
\_ round-robin 0 [prio=0][enabled]
 \_ 5:0:0:1 sdb 8:16  [active][undef]
 \_ 6:0:0:1 sdf 8:80  [active][undef]
```

DM Multipath automatically maintains the proper mapping of each WWID-based device name to its corresponding **/dev/sd** name on the system. These names are persistent across path changes, and they are consistent when accessing the device from different systems.

When the **user_friendly_names** feature (of **DM Multipath**) is used, the WWID is mapped to a name of the form **/dev/mapper/mpath*n***. By default, this mapping is maintained in the file **/etc/multipath/bindings**. These **mpath*n*** names are persistent as long as that file is maintained.

> **IMPORTANT**
>
> If you use **user_friendly_names**, then additional steps are required to obtain consistent names in a cluster. Refer to the Consistent Multipath Device Names in a Cluster section in the DM Multipath book.

In addition to these persistent names provided by the system, you can also use **udev** rules to implement persistent names of your own, mapped to the WWID of the storage.

25.7.3. Device Names Managed by the **udev** Mechanism in **/dev/disk/by-***

The **udev** mechanism consists of three major components:

The kernel

Generates events that are sent to user space when devices are added, removed, or changed.

The udevd service

Receives the events.

The udev rules

Specifies the action to take when the **udev** service receives the kernel events.

This mechanism is used for all types of devices in Linux, not just for storage devices. In the case of storage devices, Red Hat Enterprise Linux contains **udev** rules that create symbolic links in the **/dev/disk/** directory allowing storage devices to be referred to by their contents, a unique identifier, their serial number, or the hardware path used to access the device.

/dev/disk/by-label/

Entries in this directory provide a symbolic name that refers to the storage device by a label in the contents (that is, the data) stored on the device. The **blkid** utility is used to read data from the device and determine a name (that is, a label) for the device. For example:

```
/dev/disk/by-label/Boot
```

> **NOTE**
>
> The information is obtained from the contents (that is, the data) on the device so if the contents are copied to another device, the label will remain the same.

The label can also be used to refer to the device in **/etc/fstab** using the following syntax:

```
LABEL=Boot
```

/dev/disk/by-uuid/

Entries in this directory provide a symbolic name that refers to the storage device by a unique identifier in the contents (that is, the data) stored on the device. The **blkid** utility is used to read data from the device and obtain a unique identifier (that is, the UUID) for the device. For example:

```
UUID=3e6be9de-8139-11d1-9106-a43f08d823a6
```

/dev/disk/by-id/

Entries in this directory provide a symbolic name that refers to the storage device by a unique identifier (different from all other storage devices). The identifier is a property of the device but is not stored in the contents (that is, the data) on the devices. For example:

```
/dev/disk/by-id/scsi-3600508e000000000ce506dc50ab0ad05
```

```
/dev/disk/by-id/wwn-0x600508e000000000ce506dc50ab0ad05
```

The id is obtained from the world-wide ID of the device, or the device serial number. The **/dev/disk/by-id/** entries may also include a partition number. For example:

```
/dev/disk/by-id/scsi-3600508e000000000ce506dc50ab0ad05-part1
```

```
/dev/disk/by-id/wwn-0x600508e000000000ce506dc50ab0ad05-part1
```

/dev/disk/by-path/

Entries in this directory provide a symbolic name that refers to the storage device by the hardware path used to access the device, beginning with a reference to the storage controller in the PCI hierarchy, and including the SCSI host, channel, target, and LUN numbers and, optionally, the partition number. Although these names are preferable to using major and minor numbers or **sd** names, caution must be used to ensure that the target numbers do not change in a Fibre Channel SAN environment (for example, through the use of persistent binding) and that the use of the names is updated if a host adapter is moved to a different PCI slot. In addition, there is the possibility that the SCSI host numbers could change if a HBA fails to probe, if drivers are loaded in a different order, or if a new HBA is installed on the system. An example of by-path listing is:

```
/dev/disk/by-path/pci-0000:03:00.0-scsi-0:1:0:0
```

The **/dev/disk/by-path/** entries may also include a partition number, such as:

```
/dev/disk/by-path/pci-0000:03:00.0-scsi-0:1:0:0-part1
```

25.7.3.1. Limitations of the **udev** Device Naming Convention

The following are some limitations of the **udev** naming convention.

- It is possible that the device may not be accessible at the time the query is performed because the **udev** mechanism may rely on the ability to query the storage device when the **udev** rules are processed for a **udev** event. This is more likely to occur with Fibre Channel, iSCSI or FCoE storage devices when the device is not located in the server chassis.

- The kernel may also send **udev** events at any time, causing the rules to be processed and possibly causing the **/dev/disk/by-*/** links to be removed if the device is not accessible.

- There can be a delay between when the **udev** event is generated and when it is processed, such as when a large number of devices are detected and the user-space **udevd** service takes some amount of time to process the rules for each one). This could cause a delay between when the kernel detects the device and when the **/dev/disk/by-*/** names are available.

- External programs such as **blkid** invoked by the rules may open the device for a brief period of time, making the device inaccessible for other uses.

25.7.3.2. Modifying Persistent Naming Attributes

Although **udev** naming attributes are persistent, in that they do not change on their own across system reboots, some are also configurable. You can set custom values for the following persistent naming attributes:

- **UUID**: file system UUID

- **LABEL**: file system label

Because the **UUID** and **LABEL** attributes are related to the file system, the tool you need to use depends on the file system on that partition.

- To change the **UUID** or **LABEL** attributes of an XFS file system, unmount the file system and then use the **xfs_admin** utility to change the attribute:

```
# umount /dev/device
# xfs_admin [-U new_uuid] [-L new_label] /dev/device
# udevadm settle
```

- To change the **UUID** or **LABEL** attributes of an ext4, ext3, or ext2 file system, use the **tune2fs** utility:

```
# tune2fs [-U new_uuid] [-L new_label] /dev/device
# udevadm settle
```

Replace *new_uuid* with the UUID you want to set; for example, **1cdfbc07-1c90-4984-b5ec-f61943f5ea50**. Replace *new_label* with a label; for example, **backup_data**.

> **NOTE**
>
> Changing **udev** attributes happens in the background and might take a long time. The **udevadm settle** command waits until the change is fully registered, which ensures that your next command will be able to utilize the new attribute correctly.
>
> You should also use the command after creating new devices; for example, after using the **parted** tool to create a partition with a custom **PARTUUID** or **PARTLABEL** attribute, or after creating a new file system.

25.8. REMOVING A STORAGE DEVICE

Before removing access to the storage device itself, it is advisable to back up data from the device first. Afterwards, flush I/O and remove all operating system references to the device (as described below). If the device uses multipathing, then do this for the multipath "pseudo device" (Section 25.7.2, "World Wide

Identifier (WWID)") and each of the identifiers that represent a path to the device. If you are only removing a path to a multipath device, and other paths will remain, then the procedure is simpler, as described in Section 25.10, "Adding a Storage Device or Path".

Removal of a storage device is not recommended when the system is under memory pressure, since the I/O flush will add to the load. To determine the level of memory pressure, run the command **vmstat 1 100**; device removal is not recommended if:

- Free memory is less than 5% of the total memory in more than 10 samples per 100 (the command **free** can also be used to display the total memory).

- Swapping is active (non-zero **si** and **so** columns in the **vmstat** output).

The general procedure for removing all access to a device is as follows:

Procedure 25.9. Ensuring a Clean Device Removal

1. Close all users of the device and backup device data as needed.

2. Use **umount** to unmount any file systems that mounted the device.

3. Remove the device from any **md** and LVM volume using it. If the device is a member of an LVM Volume group, then it may be necessary to move data off the device using the **pvmove** command, then use the **vgreduce** command to remove the physical volume, and (optionally) **pvremove** to remove the LVM metadata from the disk.

4. If the device uses multipathing, run **multipath -l** and note all the paths to the device. Afterwards, remove the multipathed device using **multipath -f device**.

5. Run **blockdev --flushbufs *device*** to flush any outstanding I/O to all paths to the device. This is particularly important for raw devices, where there is no **umount** or **vgreduce** operation to cause an I/O flush.

6. Remove any reference to the device's path-based name, like **/dev/sd**, **/dev/disk/by-path** or the **major:minor** number, in applications, scripts, or utilities on the system. This is important in ensuring that different devices added in the future will not be mistaken for the current device.

7. Finally, remove each path to the device from the SCSI subsystem. To do so, use the command **echo 1 > /sys/block/*device-name*/device/delete** where ***device-name*** may be **sde**, for example.

   Another variation of this operation is **echo 1 > /sys/class/scsi_device/*h:c:t:l*/device/delete**, where ***h*** is the HBA number, ***c*** is the channel on the HBA, ***t*** is the SCSI target ID, and ***l*** is the LUN.

   > **NOTE**
   >
   > The older form of these commands, **echo "scsi remove-single-device 0 0 0 0" > /proc/scsi/scsi**, is deprecated.

You can determine the ***device-name***, HBA number, HBA channel, SCSI target ID and LUN for a device from various commands, such as **lsscsi**, **scsi_id**, **multipath -l**, and **ls -l /dev/disk/by-***.

After performing Procedure 25.9, "Ensuring a Clean Device Removal", a device can be physically removed safely from a running system. It is not necessary to stop I/O to other devices while doing so.

Other procedures, such as the physical removal of the device, followed by a rescan of the SCSI bus (as described in Section 25.11, "Scanning Storage Interconnects") to cause the operating system state to be updated to reflect the change, are not recommended. This will cause delays due to I/O timeouts, and devices may be removed unexpectedly. If it is necessary to perform a rescan of an interconnect, it must be done while I/O is paused, as described in Section 25.11, "Scanning Storage Interconnects".

25.9. REMOVING A PATH TO A STORAGE DEVICE

If you are removing a path to a device that uses multipathing (without affecting other paths to the device), then the general procedure is as follows:

Procedure 25.10. Removing a Path to a Storage Device

1. Remove any reference to the device's path-based name, like **/dev/sd** or **/dev/disk/by-path** or the **major:minor** number, in applications, scripts, or utilities on the system. This is important in ensuring that different devices added in the future will not be mistaken for the current device.

2. Take the path offline using **echo offline > /sys/block/sda/device/state**.

   This will cause any subsequent I/O sent to the device on this path to be failed immediately. **Device-mapper-multipath** will continue to use the remaining paths to the device.

3. Remove the path from the SCSI subsystem. To do so, use the command **echo 1 > /sys/block/*device-name*/device/delete** where *device-name* may be **sde**, for example (as described in Procedure 25.9, "Ensuring a Clean Device Removal").

After performing Procedure 25.10, "Removing a Path to a Storage Device", the path can be safely removed from the running system. It is not necessary to stop I/O while this is done, as **device-mapper-multipath** will re-route I/O to remaining paths according to the configured path grouping and failover policies.

Other procedures, such as the physical removal of the cable, followed by a rescan of the SCSI bus to cause the operating system state to be updated to reflect the change, are not recommended. This will cause delays due to I/O timeouts, and devices may be removed unexpectedly. If it is necessary to perform a rescan of an interconnect, it must be done while I/O is paused, as described in Section 25.11, "Scanning Storage Interconnects".

25.10. ADDING A STORAGE DEVICE OR PATH

When adding a device, be aware that the path-based device name (**/dev/sd** name, **major:minor** number, and **/dev/disk/by-path** name, for example) the system assigns to the new device may have been previously in use by a device that has since been removed. As such, ensure that all old references to the path-based device name have been removed. Otherwise, the new device may be mistaken for the old device.

Procedure 25.11. Add a Storage Device or Path

1. The first step in adding a storage device or path is to physically enable access to the new storage device, or a new path to an existing device. This is done using vendor-specific commands at the Fibre Channel or iSCSI storage server. When doing so, note the LUN value for the new storage that will be presented to your host. If the storage server is Fibre Channel, also

take note of the *World Wide Node Name* (WWNN) of the storage server, and determine whether there is a single WWNN for all ports on the storage server. If this is not the case, note the *World Wide Port Name* (WWPN) for each port that will be used to access the new LUN.

2. Next, make the operating system aware of the new storage device, or path to an existing device. The recommended command to use is:

```
$ echo "c t l" >  /sys/class/scsi_host/hosth/scan
```

In the previous command, *h* is the HBA number, *c* is the channel on the HBA, *t* is the SCSI target ID, and *l* is the LUN.

> **NOTE**
>
> The older form of this command, **echo "scsi add-single-device 0 0 0 0" > /proc/scsi/scsi**, is deprecated.

a. In some Fibre Channel hardware, a newly created LUN on the RAID array may not be visible to the operating system until a *Loop Initialization Protocol* (LIP) operation is performed. Refer to Section 25.11, "Scanning Storage Interconnects" for instructions on how to do this.

> **IMPORTANT**
>
> It will be necessary to stop I/O while this operation is executed if an LIP is required.

b. If a new LUN has been added on the RAID array but is still not being configured by the operating system, confirm the list of LUNs being exported by the array using the **sg_luns** command, part of the sg3_utils package. This will issue the **SCSI REPORT LUNS** command to the RAID array and return a list of LUNs that are present.

For Fibre Channel storage servers that implement a single WWNN for all ports, you can determine the correct *h*,*c*,and *t* values (i.e. HBA number, HBA channel, and SCSI target ID) by searching for the WWNN in **sysfs**.

Example 25.5. Determine Correct h, c, and t Values

For example, if the WWNN of the storage server is **0x5006016090203181**, use:

```
$ grep 5006016090203181 /sys/class/fc_transport/*/node_name
```

This should display output similar to the following:

```
/sys/class/fc_transport/target5:0:2/node_name:0x5006016090203181
/sys/class/fc_transport/target5:0:3/node_name:0x5006016090203181
/sys/class/fc_transport/target6:0:2/node_name:0x5006016090203181
/sys/class/fc_transport/target6:0:3/node_name:0x5006016090203181
```

This indicates there are four Fibre Channel routes to this target (two single-channel HBAs, each leading to two storage ports). Assuming a LUN value is **56**, then the following command will configure the first path:

```
$ echo "0 2 56" >  /sys/class/scsi_host/host5/scan
```

This must be done for each path to the new device.

For Fibre Channel storage servers that do not implement a single WWNN for all ports, you can determine the correct HBA number, HBA channel, and SCSI target ID by searching for each of the WWPNs in **sysfs**.

Another way to determine the HBA number, HBA channel, and SCSI target ID is to refer to another device that is already configured on the same path as the new device. This can be done with various commands, such as **lsscsi**, **scsi_id**, **multipath -l**, and **ls -l /dev/disk/by-***. This information, plus the LUN number of the new device, can be used as shown above to probe and configure that path to the new device.

3. After adding all the SCSI paths to the device, execute the **multipath** command, and check to see that the device has been properly configured. At this point, the device can be added to **md**, LVM, **mkfs**, or **mount**, for example.

If the steps above are followed, then a device can safely be added to a running system. It is not necessary to stop I/O to other devices while this is done. Other procedures involving a rescan (or a reset) of the SCSI bus, which cause the operating system to update its state to reflect the current device connectivity, are not recommended while storage I/O is in progress.

25.11. SCANNING STORAGE INTERCONNECTS

Certain commands allow you to reset, scan, or both reset and scan one or more interconnects, which potentially adds and removes multiple devices in one operation. This type of scan can be disruptive, as it can cause delays while I/O operations time out, and remove devices unexpectedly. Red Hat recommends using interconnect scanning *only when necessary*. Observe the following restrictions when scanning storage interconnects:

- All I/O on the effected interconnects must be paused and flushed before executing the procedure, and the results of the scan checked before I/O is resumed.

- As with removing a device, interconnect scanning is not recommended when the system is under memory pressure. To determine the level of memory pressure, run the **vmstat 1 100** command. Interconnect scanning is not recommended if free memory is less than 5% of the total memory in more than 10 samples per 100. Also, interconnect scanning is not recommended if swapping is active (non-zero **si** and **so** columns in the **vmstat** output). The **free** command can also display the total memory.

The following commands can be used to scan storage interconnects:

echo "1" > /sys/class/fc_host/host/issue_lip

This operation performs a *Loop Initialization Protocol* (*LIP*), scans the interconnect, and causes the SCSI layer to be updated to reflect the devices currently on the bus. Essentially, an LIP is a bus reset, and causes device addition and removal. This procedure is necessary to configure a new SCSI target on a Fibre Channel interconnect.

Note that **issue_lip** is an asynchronous operation. The command can complete before the entire scan has completed. You must monitor **/var/log/messages** to determine when **issue_lip** finishes.

The **lpfc**, **qla2xxx**, and **bnx2fc** drivers support **issue_lip**. For more information about the API capabilities supported by each driver in Red Hat Enterprise Linux, see Table 25.1, "Fibre Channel API Capabilities".

/usr/bin/rescan-scsi-bus.sh

The **/usr/bin/rescan-scsi-bus.sh** script was introduced in Red Hat Enterprise Linux 5.4. By default, this script scans all the SCSI buses on the system, and updates the SCSI layer to reflect new devices on the bus. The script provides additional options to allow device removal, and the issuing of LIPs. For more information about this script, including known issues, see Section 25.17, "Adding/Removing a Logical Unit Through rescan-scsi-bus.sh".

echo "- - -" > /sys/class/scsi_host/host*h*/scan

This is the same command as described in Section 25.10, "Adding a Storage Device or Path" to add a storage device or path. In this case, however, the channel number, SCSI target ID, and LUN values are replaced by wildcards. Any combination of identifiers and wildcards is allowed, so you can make the command as specific or broad as needed. This procedure adds LUNs, but does not remove them.

modprobe --remove *driver-name*, modprobe *driver-name*

Running the **modprobe --remove *driver-name*** command followed by the **modprobe *driver-name*** command completely re-initializes the state of all interconnects controlled by the driver. Despite being rather extreme, using the described commands can be appropriate in certain situations. The commands can be used, for example, to restart the driver with a different module parameter value.

25.12. ISCSI DISCOVERY CONFIGURATION

The default iSCSI configuration file is **/etc/iscsi/iscsid.conf**. This file contains iSCSI settings used by **iscsid** and **iscsiadm**.

During target discovery, the **iscsiadm** tool uses the settings in **/etc/iscsi/iscsid.conf** to create two types of records:

Node records in /var/lib/iscsi/nodes

When logging into a target, **iscsiadm** uses the settings in this file.

Discovery records in /var/lib/iscsi/*discovery_type*

When performing discovery to the same destination, **iscsiadm** uses the settings in this file.

Before using different settings for discovery, delete the current discovery records (i.e. **/var/lib/iscsi/*discovery_type***) first. To do this, use the following command: [5]

```
# iscsiadm -m discovery -t discovery_type -p target_IP:port -o delete
```

Here, *discovery_type* can be either **sendtargets**, **isns**, or **fw**.

For details on different types of discovery, refer to the *DISCOVERY TYPES* section of the iscsiadm(8) man page.

There are two ways to reconfigure discovery record settings:

- Edit the **/etc/iscsi/iscsid.conf** file directly prior to performing a discovery. Discovery settings use the prefix **discovery**; to view them, run:

  ```
  # iscsiadm -m discovery -t discovery_type -p target_IP:port
  ```

- Alternatively, **iscsiadm** can also be used to directly change discovery record settings, as in:

  ```
  # iscsiadm -m discovery -t discovery_type -p target_IP:port -o
  update -n setting -v %value
  ```

  Refer to the iscsiadm(8) man page for more information on available *setting* options and valid *value* options for each.

After configuring discovery settings, any subsequent attempts to discover new targets will use the new settings. Refer to for details on how to scan for new iSCSI targets.

For more information on configuring iSCSI target discovery, refer to the **man** pages of **iscsiadm** and **iscsid**. The **/etc/iscsi/iscsid.conf** file also contains examples on proper configuration syntax.

25.13. CONFIGURING ISCSI OFFLOAD AND INTERFACE BINDING

This chapter describes how to set up iSCSI interfaces in order to bind a session to a NIC port when using software iSCSI. It also describes how to set up interfaces for use with network devices that support offloading.

The network subsystem can be configured to determine the path/NIC that iSCSI interfaces should use for binding. For example, if portals and NICs are set up on different subnets, then it is not necessary to manually configure iSCSI interfaces for binding.

Before attempting to configure an iSCSI interface for binding, run the following command first:

```
$ ping -I ethX target_IP
```

If **ping** fails, then you will not be able to bind a session to a NIC. If this is the case, check the network settings first.

25.13.1. Viewing Available iface Configurations

iSCSI offload and interface binding is supported for the following iSCSI initiator implementations:

Software iSCSI

This stack allocates an iSCSI host instance (that is, **scsi_host**) per session, with a single connection per session. As a result, **/sys/class_scsi_host** and **/proc/scsi** will report a **scsi_host** for each connection/session you are logged into.

Offload iSCSI

This stack allocates a **scsi_host** for each PCI device. As such, each port on a host bus adapter will show up as a different PCI device, with a different **scsi_host** per HBA port.

To manage both types of initiator implementations, **iscsiadm** uses the **iface** structure. With this structure, an **iface** configuration must be entered in **/var/lib/iscsi/ifaces** for each HBA port, software iSCSI, or network device (**ethX**) used to bind sessions.

To view available **iface** configurations, run **iscsiadm -m iface**. This will display **iface** information in the following format:

```
iface_name
transport_name,hardware_address,ip_address,net_ifacename,initiator_name
```

Refer to the following table for an explanation of each value/setting.

Table 25.2. iface Settings

Setting	Description
`iface_name`	**iface** configuration name.
`transport_name`	Name of driver
`hardware_address`	MAC address
`ip_address`	IP address to use for this port
`net_iface_name`	Name used for the **vlan** or alias binding of a software iSCSI session. For iSCSI offloads, **net_iface_name** will be **<empty>** because this value is not persistent across reboots.
`initiator_name`	This setting is used to override a default name for the initiator, which is defined in **/etc/iscsi/initiatorname.iscsi**

Example 25.6. Sample Output of the `iscsiadm -m iface` Command

The following is a sample output of the **iscsiadm -m iface** command:

```
iface0 qla4xxx,00:c0:dd:08:63:e8,20.15.0.7,default,iqn.2005-
06.com.redhat:madmax
iface1 qla4xxx,00:c0:dd:08:63:ea,20.15.0.9,default,iqn.2005-
06.com.redhat:madmax
```

For software iSCSI, each **iface** configuration must have a unique name (with less than 65 characters). The **iface_name** for network devices that support offloading appears in the format *transport_name.hardware_name*.

Example 25.7. `iscsiadm -m iface` Output with a Chelsio Network Card

For example, the sample output of **iscsiadm -m iface** on a system using a Chelsio network card might appear as:

```
default tcp,<empty>,<empty>,<empty>,<empty>
iser iser,<empty>,<empty>,<empty>,<empty>
cxgb3i.00:07:43:05:97:07 cxgb3i,00:07:43:05:97:07,<empty>,<empty>,
<empty>
```

It is also possible to display the settings of a specific **iface** configuration in a more friendly way. To do so, use the option **-I *iface_name***. This will display the settings in the following format:

```
iface.setting = value
```

Example 25.8. Using iface Settings with a Chelsio Converged Network Adapter

Using the previous example, the **iface** settings of the same Chelsio converged network adapter (i.e. **iscsiadm -m iface -I cxgb3i.00:07:43:05:97:07**) would appear as:

```
# BEGIN RECORD 2.0-871
iface.iscsi_ifacename = cxgb3i.00:07:43:05:97:07
iface.net_ifacename = <empty>
iface.ipaddress = <empty>
iface.hwaddress = 00:07:43:05:97:07
iface.transport_name = cxgb3i
iface.initiatorname = <empty>
# END RECORD
```

25.13.2. Configuring an iface for Software iSCSI

As mentioned earlier, an **iface** configuration is required for each network object that will be used to bind a session.

Before

To create an **iface** configuration for software iSCSI, run the following command:

```
# iscsiadm -m iface -I iface_name --op=new
```

This will create a new *empty* **iface** configuration with a specified ***iface_name***. If an existing **iface** configuration already has the same ***iface_name***, then it will be overwritten with a new, empty one.

To configure a specific setting of an **iface** configuration, use the following command:

```
# iscsiadm -m iface -I iface_name --op=update -n iface.setting -v
hw_address
```

Example 25.9. Set MAC Address of iface0

For example, to set the MAC address (**hardware_address**) of **iface0** to **00:0F:1F:92:6B:BF**, run:

```
# iscsiadm -m iface -I iface0 --op=update -n iface.hwaddress -v
00:0F:1F:92:6B:BF
```

> ■

> ⚠ **WARNING**
>
> Do not use **default** or **iser** as **iface** names. Both strings are special values
> used by **iscsiadm** for backward compatibility. Any manually-created **iface**
> configurations named **default** or **iser** will disable backwards compatibility.

25.13.3. Configuring an iface for iSCSI Offload

By default, **iscsiadm** creates an **iface** configuration for each port. To view available **iface**
configurations, use the same command for doing so in software iSCSI: **iscsiadm -m iface**.

Before using the **iface** of a network card for iSCSI offload, first set the **iface.ipaddress** value of the
offload interface to the initiator IP address that the interface should use:

- For devices that use the **be2iscsi** driver, the IP address is configured in the BIOS setup
  screen.

- For all other devices, to configure the IP address of the **iface**, use:

  ```
  # iscsiadm -m iface -I iface_name -o update -n iface.ipaddress -v
  initiator_ip_address
  ```

Example 25.10. Set the iface IP Address of a Chelsio Card

For example, to set the **iface** IP address to **20.15.0.66** when using a card with the **iface** name
of **cxgb3i.00:07:43:05:97:07**, use:

```
# iscsiadm -m iface -I cxgb3i.00:07:43:05:97:07 -o update -n
iface.ipaddress -v 20.15.0.66
```

25.13.4. Binding/Unbinding an iface to a Portal

Whenever **iscsiadm** is used to scan for interconnects, it will first check the **iface.transport**
settings of each **iface** configuration in **/var/lib/iscsi/ifaces**. The **iscsiadm** utility will then bind
discovered portals to any **iface** whose **iface.transport** is **tcp**.

This behavior was implemented for compatibility reasons. To override this, use the **-I iface_name** to
specify which portal to bind to an **iface**, as in:

```
# iscsiadm -m discovery -t st -p target_IP:port -I iface_name -P 1
[5]
```

By default, the **iscsiadm** utility will not automatically bind any portals to **iface** configurations that use

offloading. This is because such **iface** configurations will not have **iface.transport** set to **tcp**. As such, the **iface** configurations need to be manually bound to discovered portals.

It is also possible to prevent a portal from binding to any existing **iface**. To do so, use **default** as the *iface_name*, as in:

```
# iscsiadm -m discovery -t st -p IP:port -I default -P 1
```

To remove the binding between a target and **iface**, use:

```
# iscsiadm -m node -targetname proper_target_name -I iface0 --op=delete[6]
```

To delete all bindings for a specific **iface**, use:

```
# iscsiadm -m node -I iface_name --op=delete
```

To delete bindings for a specific portal (e.g. for Equalogic targets), use:

```
# iscsiadm -m node -p IP:port -I iface_name --op=delete
```

> **NOTE**
>
> If there are no **iface** configurations defined in **/var/lib/iscsi/iface** and the **-I** option is not used, **iscsiadm** will allow the network subsystem to decide which device a specific portal should use.

25.14. SCANNING ISCSI INTERCONNECTS

For iSCSI, if the targets send an iSCSI async event indicating new storage is added, then the scan is done automatically.

However, if the targets do not send an iSCSI async event, you need to manually scan them using the **iscsiadm** utility. Before doing so, however, you need to first retrieve the proper **--targetname** and the **--portal** values. If your device model supports only a single logical unit and portal per target, use **iscsiadm** to issue a **sendtargets** command to the host, as in:

```
# iscsiadm -m discovery -t sendtargets -p target_IP:port
[5]
```

The output will appear in the following format:

```
target_IP:port,target_portal_group_tag proper_target_name
```

Example 25.11. Using iscsiadm to issue a sendtargets Command

For example, on a target with a *proper_target_name* of **iqn.1992-08.com.netapp:sn.33615311** and a *target_IP:port* of **10.15.85.19:3260**, the output may appear as:

```
10.15.84.19:3260,2 iqn.1992-08.com.netapp:sn.33615311
```

```
10.15.85.19:3260,3 iqn.1992-08.com.netapp:sn.33615311
```

In this example, the target has two portals, each using **target_ip:port**s of **10.15.84.19:3260** and **10.15.85.19:3260**.

To see which **iface** configuration will be used for each session, add the **-P 1** option. This option will print also session information in tree format, as in:

```
Target: proper_target_name
    Portal: target_IP:port,target_portal_group_tag
        Iface Name: iface_name
```

Example 25.12. View `iface` Configuration

For example, with **iscsiadm -m discovery -t sendtargets -p 10.15.85.19:3260 -P 1**, the output may appear as:

```
Target: iqn.1992-08.com.netapp:sn.33615311
    Portal: 10.15.84.19:3260,2
        Iface Name: iface2
    Portal: 10.15.85.19:3260,3
        Iface Name: iface2
```

This means that the target **iqn.1992-08.com.netapp:sn.33615311** will use **iface2** as its **iface** configuration.

With some device models a single target may have multiple logical units and portals. In this case, issue a **sendtargets** command to the host first to find new portals on the target. Then, rescan the existing sessions using:

```
# iscsiadm -m session --rescan
```

You can also rescan a specific session by specifying the session's **SID** value, as in:

```
# iscsiadm -m session -r SID --rescan[7]
```

If your device supports multiple targets, you will need to issue a **sendtargets** command to the hosts to find new portals for *each* target. Rescan existing sessions to discover new logical units on existing sessions using the **--rescan** option.

> **IMPORTANT**
>
> The **sendtargets** command used to retrieve **--targetname** and **--portal** values overwrites the contents of the **/var/lib/iscsi/nodes** database. This database will then be repopulated using the settings in **/etc/iscsi/iscsid.conf**. However, this will not occur if a session is currently logged in and in use.
>
> To safely add new targets/portals or delete old ones, use the **-o new** or **-o delete** options, respectively. For example, to add new targets/portals without overwriting **/var/lib/iscsi/nodes**, use the following command:
>
> ```
> iscsiadm -m discovery -t st -p target_IP -o new
> ```
>
> To delete **/var/lib/iscsi/nodes** entries that the target did not display during discovery, use:
>
> ```
> iscsiadm -m discovery -t st -p target_IP -o delete
> ```
>
> You can also perform both tasks simultaneously, as in:
>
> ```
> iscsiadm -m discovery -t st -p target_IP -o delete -o new
> ```

The **sendtargets** command will yield the following output:

```
ip:port,target_portal_group_tag proper_target_name
```

Example 25.13. Output of the `sendtargets` Command

For example, given a device with a single target, logical unit, and portal, with **equallogic-iscsi1** as your *target_name*, the output should appear similar to the following:

```
10.16.41.155:3260,0 iqn.2001-05.com.equallogic:6-8a0900-ac3fe0101-
63aff113e344a4a2-dl585-03-1
```

Note that *proper_target_name* and *ip:port,target_portal_group_tag* are identical to the values of the same name in

At this point, you now have the proper **--targetname** and **--portal** values needed to manually scan for iSCSI devices. To do so, run the following command:

```
# iscsiadm --mode node --targetname proper_target_name --portal
ip:port,target_portal_group_tag \ --login
[8]
```

Example 25.14. Full `iscsiadm` Command

Using our previous example (where *proper_target_name* is **equallogic-iscsi1**), the full command would be:

```
# iscsiadm --mode node --targetname  \ iqn.2001-05.com.equallogic:6-
8a0900-ac3fe0101-63aff113e344a4a2-dl585-03-1  \ --portal
10.16.41.155:3260,0 --login[8]
```

25.15. LOGGING IN TO AN ISCSI TARGET

As mentioned in Section 25.6, "iSCSI", the iSCSI service must be running in order to discover or log into targets. To start the iSCSI service, run:

```
# systemctl start iscsi
```

When this command is executed, the iSCSI **init** scripts will automatically log into targets where the **node.startup** setting is configured as **automatic**. This is the default value of **node.startup** for all targets.

To prevent automatic login to a target, set **node.startup** to **manual**. To do this, run the following command:

```
# iscsiadm -m node --targetname proper_target_name -p target_IP:port -o
update -n node.startup -v manual
```

Deleting the entire record will also prevent automatic login. To do this, run:

```
# iscsiadm -m node --targetname proper_target_name -p target_IP:port -o
delete
```

To automatically mount a file system from an iSCSI device on the network, add a partition entry for the mount in **/etc/fstab** with the **_netdev** option. For example, to automatically mount the iSCSI device **sdb** to **/mount/iscsi** during startup, add the following line to **/etc/fstab**:

```
/dev/sdb /mnt/iscsi ext3 _netdev 0 0
```

To manually log in to an iSCSI target, use the following command:

```
# iscsiadm -m node --targetname proper_target_name -p target_IP:port -l
```

> **NOTE**
>
> The *proper_target_name* and *target_IP:port* refer to the full name and IP address/port combination of a target. For more information, refer to Section 25.6.1, "iSCSI API" and Section 25.14, "Scanning iSCSI Interconnects".

25.16. RESIZING AN ONLINE LOGICAL UNIT

In most cases, fully resizing an online *logical unit* involves two things: resizing the logical unit itself and reflecting the size change in the corresponding multipath device (if multipathing is enabled on the system).

To resize the online logical unit, start by modifying the logical unit size through the array management interface of your storage device. This procedure differs with each array; as such, consult your storage array vendor documentation for more information on this.

> **NOTE**
>
> In order to resize an online file system, the file system must not reside on a partitioned device.

25.16.1. Resizing Fibre Channel Logical Units

After modifying the online logical unit size, re-scan the logical unit to ensure that the system detects the updated size. To do this for Fibre Channel logical units, use the following command:

```
$ echo 1 > /sys/block/sdX/device/rescan
```

> **IMPORTANT**
>
> To re-scan Fibre Channel logical units on a system that uses multipathing, execute the aforementioned command for each sd device (i.e. **sd1**, **sd2**, and so on) that represents a path for the multipathed logical unit. To determine which devices are paths for a multipath logical unit, use **multipath -ll**; then, find the entry that matches the logical unit being resized. It is advisable that you refer to the WWID of each entry to make it easier to find which one matches the logical unit being resized.

25.16.2. Resizing an iSCSI Logical Unit

After modifying the online logical unit size, re-scan the logical unit to ensure that the system detects the updated size. To do this for iSCSI devices, use the following command:

```
# iscsiadm -m node --targetname target_name -R
[5]
```

Replace *target_name* with the name of the target where the device is located.

> **NOTE**
>
> You can also re-scan iSCSI logical units using the following command:
>
> ```
> # iscsiadm -m node -R -I interface
> ```
>
> Replace *interface* with the corresponding interface name of the resized logical unit (for example, **iface0**). This command performs two operations:
>
> - It scans for new devices in the same way that the command **echo "- - -" > /sys/class/scsi_host/*host*/scan** does (refer to Section 25.14, "Scanning iSCSI Interconnects").
>
> - It re-scans for new/modified logical units the same way that the command **echo 1 > /sys/block/sdX/device/rescan** does. Note that this command is the same one used for re-scanning Fibre Channel logical units.

25.16.3. Updating the Size of Your Multipath Device

If multipathing is enabled on your system, you will also need to reflect the change in logical unit size to the logical unit's corresponding multipath device (*after* resizing the logical unit). This can be done through **multipathd**. To do so, first ensure that **multipathd** is running using **service multipathd status**. Once you've verified that **multipathd** is operational, run the following command:

```
# multipathd -k"resize map multipath_device"
```

The *multipath_device* variable is the corresponding multipath entry of your device in **/dev/mapper**. Depending on how multipathing is set up on your system, *multipath_device* can be either of two formats:

- **mpath*X***, where *X* is the corresponding entry of your device (for example, **mpath0**)

- a WWID; for example, **3600508b400105e210000900000490000**

To determine which multipath entry corresponds to your resized logical unit, run **multipath -ll**. This displays a list of all existing multipath entries in the system, along with the major and minor numbers of their corresponding devices.

> **IMPORTANT**
>
> Do not use **multipathd -k"resize map *multipath_device*"** if there are any commands queued to *multipath_device*. That is, do not use this command when the **no_path_retry** parameter (in **/etc/multipath.conf**) is set to **"queue"**, and there are no active paths to the device.

For more information about multipathing, refer to the Red Hat Enterprise Linux 7 DM Multipath guide.

25.16.4. Changing the Read/Write State of an Online Logical Unit

Certain storage devices provide the user with the ability to change the state of the device from Read/Write (R/W) to Read-Only (RO), and from RO to R/W. This is typically done through a management interface on the storage device. The operating system will not automatically update its view of the state of the device when a change is made. Follow the procedures described in this chapter to make the operating system aware of the change.

Run the following command, replacing XYZ with the desired device designator, to determine the operating system's current view of the R/W state of a device:

```
# blockdev --getro /dev/sdXYZ
```

The following command is also available for Red Hat Enterprise Linux 7:

```
# cat /sys/block/sdXYZ/ro 1 = read-only 0 = read-write
```

When using multipath, refer to the *ro* or *rw* field in the second line of output from the **multipath -ll** command. For example:

```
36001438005deb4710000500000640000 dm-8 GZ,GZ500
[size=20G][features=0][hwhandler=0][ro]
\_ round-robin 0 [prio=200][active]
```

```
 \_ 6:0:4:1  sdax 67:16  [active][ready]
 \_ 6:0:5:1  sday 67:32  [active][ready]
\_ round-robin 0 [prio=40][enabled]
 \_ 6:0:6:1  sdaz 67:48  [active][ready]
 \_ 6:0:7:1  sdba 67:64  [active][ready]
```

To change the R/W state, use the following procedure:

Procedure 25.12. Change the R/W State

1. To move the device from RO to R/W, see step 2.

   To move the device from R/W to RO, ensure no further writes will be issued. Do this by stopping the application, or through the use of an appropriate, application-specific action.

   Ensure that all outstanding write I/Os are complete with the following command:

   ```
   # blockdev --flushbufs /dev/device
   ```

   Replace *device* with the desired designator; for a device mapper multipath, this is the entry for your device in **dev/mapper**. For example, **/dev/mapper/*mpath3***.

2. Use the management interface of the storage device to change the state of the logical unit from R/W to RO, or from RO to R/W. The procedure for this differs with each array. Consult applicable storage array vendor documentation for more information.

3. Perform a re-scan of the device to update the operating system's view of the R/W state of the device. If using a device mapper multipath, perform this re-scan for each path to the device before issuing the command telling multipath to reload its device maps.

   This process is explained in further detail in Section 25.16.4.1, "Rescanning Logical Units".

25.16.4.1. Rescanning Logical Units

After modifying the online logical unit Read/Write state, as described in Section 25.16.4, "Changing the Read/Write State of an Online Logical Unit", re-scan the logical unit to ensure the system detects the updated state with the following command:

```
# echo 1 > /sys/block/sdX/device/rescan
```

To re-scan logical units on a system that uses multipathing, execute the above command for each sd device that represents a path for the multipathed logical unit. For example, run the command on sd1, sd2 and all other sd devices. To determine which devices are paths for a multipath unit, use **multipath -11**, then find the entry that matches the logical unit to be changed.

Example 25.15. Use of the `multipath -11` Command

For example, the **multipath -11** above shows the path for the LUN with WWID 36001438005deb4710000500000640000. In this case, enter:

```
# echo 1 > /sys/block/sdax/device/rescan
# echo 1 > /sys/block/sday/device/rescan
# echo 1 > /sys/block/sdaz/device/rescan
# echo 1 > /sys/block/sdba/device/rescan
```

25.16.4.2. Updating the R/W State of a Multipath Device

If multipathing is enabled, after rescanning the logical unit, the change in its state will need to be reflected in the logical unit's corresponding multipath drive. Do this by reloading the multipath device maps with the following command:

```
# multipath -r
```

The **multipath -11** command can then be used to confirm the change.

25.16.4.3. Documentation

Further information can be found in the Red Hat Knowledgebase. To access this, navigate to
https://www.redhat.com/wapps/sso/login.html?redirect=https://access.redhat.com/knowledge/ and log in.
Then access the article at https://access.redhat.com/kb/docs/DOC-32850.

25.17. ADDING/REMOVING A LOGICAL UNIT THROUGH RESCAN-SCSI-BUS.SH

The **sg3_utils** package provides the **rescan-scsi-bus.sh** script, which can automatically update the logical unit configuration of the host as needed (after a device has been added to the system). The **rescan-scsi-bus.sh** script can also perform an **issue_lip** on supported devices. For more information about how to use this script, refer to **rescan-scsi-bus.sh --help**.

To install the **sg3_utils** package, run **yum install sg3_utils**.

Known Issues with rescan-scsi-bus.sh

When using the **rescan-scsi-bus.sh** script, take note of the following known issues:

- In order for **rescan-scsi-bus.sh** to work properly, **LUN0** must be the first mapped logical unit. The **rescan-scsi-bus.sh** can only detect the first mapped logical unit if it is **LUN0**. The **rescan-scsi-bus.sh** will not be able to scan any other logical unit unless it detects the first mapped logical unit even if you use the **--nooptscan** option.

- A race condition requires that **rescan-scsi-bus.sh** be run twice if logical units are mapped for the first time. During the first scan, **rescan-scsi-bus.sh** only adds **LUN0**; all other logical units are added in the second scan.

- A bug in the **rescan-scsi-bus.sh** script incorrectly executes the functionality for recognizing a change in logical unit size when the **--remove** option is used.

- The **rescan-scsi-bus.sh** script does not recognize ISCSI logical unit removals.

25.18. MODIFYING LINK LOSS BEHAVIOR

This section describes how to modify the link loss behavior of devices that use either Fibre Channel or iSCSI protocols.

25.18.1. Fibre Channel

If a driver implements the Transport **dev_loss_tmo** callback, access attempts to a device through a link will be blocked when a transport problem is detected. To verify if a device is blocked, run the following command:

```
$ cat /sys/block/device/device/state
```

This command will return **blocked** if the device is blocked. If the device is operating normally, this command will return **running**.

Procedure 25.13. Determining the State of a Remote Port

1. To determine the state of a remote port, run the following command:

   ```
   $ cat
   /sys/class/fc_remote_port/rport-H:B:R/port_state
   ```

2. This command will return **Blocked** when the remote port (along with devices accessed through it) are blocked. If the remote port is operating normally, the command will return **Online**.

3. If the problem is not resolved within **dev_loss_tmo** seconds, the rport and devices will be unblocked and all I/O running on that device (along with any new I/O sent to that device) will be failed.

Procedure 25.14. Changing `dev_loss_tmo`

- To change the **dev_loss_tmo** value, **echo** in the desired value to the file. For example, to set **dev_loss_tmo** to 30 seconds, run:

  ```
  $ echo 30 >
  /sys/class/fc_remote_port/rport-H:B:R/dev_loss_tmo
  ```

For more information about **dev_loss_tmo**, refer to Section 25.3.1, "Fibre Channel API".

When a link loss exceeds **dev_loss_tmo**, the **scsi_device** and **sd***N* devices are removed. Typically, the Fibre Channel class will leave the device as is; i.e. **/dev/sdx** will remain **/dev/sdx**. This is because the target binding is saved by the Fibre Channel driver so when the target port returns, the SCSI addresses are recreated faithfully. However, this cannot be guaranteed; the **sdx** will be restored only if no additional change on in-storage box configuration of LUNs is made.

25.18.2. iSCSI Settings with `dm-multipath`

If **dm-multipath** is implemented, it is advisable to set iSCSI timers to immediately defer commands to the multipath layer. To configure this, nest the following line under **device {** in **/etc/multipath.conf**:

```
features    "1 queue_if_no_path"
```

This ensures that I/O errors are retried and queued if all paths are failed in the **dm-multipath** layer.

You may need to adjust iSCSI timers further to better monitor your SAN for problems. Available iSCSI timers you can configure are *NOP-Out Interval/Timeouts* and **replacement_timeout**, which are discussed in the following sections.

25.18.2.1. NOP-Out Interval/Timeout

To help monitor problems the SAN, the iSCSI layer sends a NOP-Out request to each target. If a NOP-Out request times out, the iSCSI layer responds by failing any running commands and instructing the SCSI layer to requeue those commands when possible.

When **dm-multipath** is being used, the SCSI layer will fail those running commands and defer them to the multipath layer. The multipath layer then retries those commands on another path. If **dm-multipath** is *not* being used, those commands are retried five times before failing altogether.

Intervals between NOP-Out requests are 10 seconds by default. To adjust this, open **/etc/iscsi/iscsid.conf** and edit the following line:

```
node.conn[0].timeo.noop_out_interval = [interval value]
```

Once set, the iSCSI layer will send a NOP-Out request to each target every *[interval value]* seconds.

By default, NOP-Out requests time out in 10 seconds[9]. To adjust this, open **/etc/iscsi/iscsid.conf** and edit the following line:

```
node.conn[0].timeo.noop_out_timeout = [timeout value]
```

This sets the iSCSI layer to timeout a NOP-Out request after *[timeout value]* seconds.

SCSI Error Handler

If the SCSI Error Handler is running, running commands on a path will not be failed immediately when a NOP-Out request times out on that path. Instead, those commands will be failed *after* **replacement_timeout** seconds. For more information about **replacement_timeout**, refer to Section 25.18.2.2, "**replacement_timeout**".

To verify if the SCSI Error Handler is running, run:

```
# iscsiadm -m session -P 3
```

25.18.2.2. **replacement_timeout**

replacement_timeout controls how long the iSCSI layer should wait for a timed-out path/session to reestablish itself before failing any commands on it. The default **replacement_timeout** value is 120 seconds.

To adjust **replacement_timeout**, open **/etc/iscsi/iscsid.conf** and edit the following line:

```
node.session.timeo.replacement_timeout = [replacement_timeout]
```

The **1 queue_if_no_path** option in **/etc/multipath.conf** sets iSCSI timers to immediately defer commands to the multipath layer (refer to Section 25.18.2, "iSCSI Settings with **dm-multipath**"). This setting prevents I/O errors from propagating to the application; because of this, you can set **replacement_timeout** to 15-20 seconds.

By configuring a lower **replacement_timeout**, I/O is quickly sent to a new path and executed (in the event of a NOP-Out timeout) while the iSCSI layer attempts to re-establish the failed path/session. If all paths time out, then the multipath and device mapper layer will internally queue I/O based on the settings

in **/etc/multipath.conf** instead of **/etc/iscsi/iscsid.conf**.

> **IMPORTANT**
>
> Whether your considerations are failover speed or security, the recommended value for **replacement_timeout** will depend on other factors. These factors include the network, target, and system workload. As such, it is recommended that you thoroughly test any new configurations to **replacements_timeout** before applying it to a mission-critical system.

25.18.3. iSCSI Root

When accessing the root partition directly through an iSCSI disk, the iSCSI timers should be set so that iSCSI layer has several chances to try to reestablish a path/session. In addition, commands should not be quickly re-queued to the SCSI layer. This is the opposite of what should be done when **dm-multipath** is implemented.

To start with, NOP-Outs should be disabled. You can do this by setting both NOP-Out interval and timeout to zero. To set this, open **/etc/iscsi/iscsid.conf** and edit as follows:

```
node.conn[0].timeo.noop_out_interval = 0
node.conn[0].timeo.noop_out_timeout = 0
```

In line with this, **replacement_timeout** should be set to a high number. This will instruct the system to wait a long time for a path/session to reestablish itself. To adjust **replacement_timeout**, open **/etc/iscsi/iscsid.conf** and edit the following line:

```
node.session.timeo.replacement_timeout = replacement_timeout
```

After configuring **/etc/iscsi/iscsid.conf**, you must perform a re-discovery of the affected storage. This will allow the system to load and use any new values in **/etc/iscsi/iscsid.conf**. For more information on how to discover iSCSI devices, refer to Section 25.14, "Scanning iSCSI Interconnects".

Configuring Timeouts for a Specific Session

You can also configure timeouts for a specific session and make them non-persistent (instead of using **/etc/iscsi/iscsid.conf**). To do so, run the following command (replace the variables accordingly):

```
# iscsiadm -m node -T target_name -p target_IP:port -o update -n
node.session.timeo.replacement_timeout -v $timeout_value
```
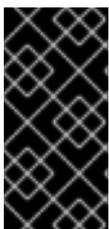
> **IMPORTANT**
>
> The configuration described here is recommended for iSCSI sessions involving root partition access. For iSCSI sessions involving access to other types of storage (namely, in systems that use **dm-multipath**), refer to Section 25.18.2, "iSCSI Settings with **dm-multipath**".

25.19. CONTROLLING THE SCSI COMMAND TIMER AND DEVICE STATUS

The Linux SCSI layer sets a timer on each command. When this timer expires, the SCSI layer will quiesce the *host bus adapter* (HBA) and wait for all outstanding commands to either time out or complete. Afterwards, the SCSI layer will activate the driver's error handler.

When the error handler is triggered, it attempts the following operations in order (until one successfully executes):

1. Abort the command.

2. Reset the device.

3. Reset the bus.

4. Reset the host.

If all of these operations fail, the device will be set to the **offline** state. When this occurs, all I/O to that device will be failed, until the problem is corrected and the user sets the device to **running**.

The process is different, however, if a device uses the Fibre Channel protocol and the **rport** is blocked. In such cases, the drivers wait for several seconds for the **rport** to become online again before activating the error handler. This prevents devices from becoming offline due to temporary transport problems.

Device States

To display the state of a device, use:

```
$ cat /sys/block/device-name/device/state
```

To set a device to the **running** state, use:

```
# echo running > /sys/block/device-name/device/state
```

Command Timer

To control the command timer, modify the **/sys/block/*device-name*/device/timeout** file:

```
# echo value > /sys/block/device-name/device/timeout
```

Replace *value* in the command with the timeout value, in seconds, that you want to implement.

25.20. TROUBLESHOOTING ONLINE STORAGE CONFIGURATION

This section provides solution to common problems users experience during online storage reconfiguration.

Logical unit removal status is not reflected on the host.

When a logical unit is deleted on a configured filer, the change is not reflected on the host. In such cases, **lvm** commands will hang indefinitely when **dm-multipath** is used, as the logical unit has now become *stale*.

To work around this, perform the following procedure:

Procedure 25.15. Working Around Stale Logical Units

1. Determine which **mpath** link entries in **/etc/lvm/cache/.cache** are specific to the stale logical unit. To do this, run the following command:

   ```
   $ ls -l /dev/mpath | grep stale-logical-unit
   ```

   Example 25.16. Determine Specific mpath Link Entries

   For example, if *stale-logical-unit* is 3600d0230003414f30000203a7bc41a00, the following results may appear:

   ```
   lrwxrwxrwx 1 root root 7 Aug  2 10:33
   /3600d0230003414f30000203a7bc41a00 -> ../dm-4
   lrwxrwxrwx 1 root root 7 Aug  2 10:33
   /3600d0230003414f30000203a7bc41a00p1 -> ../dm-5
   ```

   This means that 3600d0230003414f30000203a7bc41a00 is mapped to two **mpath** links: **dm-4** and **dm-5**.

2. Next, open **/etc/lvm/cache/.cache**. Delete all lines containing *stale-logical-unit* and the **mpath** links that *stale-logical-unit* maps to.

   Example 25.17. Delete Relevant Lines

   Using the same example in the previous step, the lines you need to delete are:

   ```
   /dev/dm-4
   /dev/dm-5
   /dev/mapper/3600d0230003414f30000203a7bc41a00
   /dev/mapper/3600d0230003414f30000203a7bc41a00p1
   /dev/mpath/3600d0230003414f30000203a7bc41a00
   /dev/mpath/3600d0230003414f30000203a7bc41a00p1
   ```

25.21. CONFIGURING MAXIMUM TIME FOR ERROR RECOVERY WITH EH_DEADLINE

IMPORTANT

In most scenarios, you do not need to enable the *eh_deadline* parameter. Using the *eh_deadline* parameter can be useful in certain specific scenarios, for example if a link loss occurs between a Fibre Channel switch and a target port, and the Host Bus Adapter (HBA) does not receive Registered State Change Notifications (RSCNs). In such a case, I/O requests and error recovery commands all time out rather than encounter an error. Setting *eh_deadline* in this environment puts an upper limit on the recovery time, which enables the failed I/O to be retried on another available path by multipath.

However, if RSCNs are enabled, the HBA does not register the link becoming unavailable, or both, the *eh_deadline* functionality provides no additional benefit, as the I/O and error recovery commands fail immediately, which allows multipath to retry.

The SCSI host object *eh_deadline* parameter enables you to configure the maximum amount of time that the SCSI error handling mechanism attempts to perform error recovery before stopping and resetting the entire HBA.

The value of the *eh_deadline* is specified in seconds. The default setting is **off**, which disables the time limit and allows all of the error recovery to take place. In addition to using **sysfs**, a default value can be set for all SCSI HBAs by using the *scsi_mod.eh_deadline* kernel parameter.

Note that when *eh_deadline* expires, the HBA is reset, which affects all target paths on that HBA, not only the failing one. As a consequence, I/O errors can occur if some of the redundant paths are not available for other reasons. Enable *eh_deadline* only if you have a fully redundant multipath configuration on all targets.

---

[5] The *target_IP* and *port* variables refer to the IP address and port combination of a target/portal, respectively. For more information, refer to Section 25.6.1, "iSCSI API" and Section 25.14, "Scanning iSCSI Interconnects".

[6] Refer to Section 25.14, "Scanning iSCSI Interconnects" for information on *proper_target_name*.

[7] For information on how to retrieve a session's SID value, refer to Section 25.6.1, "iSCSI API".

[8] This is a single command split into multiple lines, to accommodate printed and PDF versions of this document. All concatenated lines — preceded by the backslash (\) — should be treated as one command, sans backslashes.

[9] Prior to Red Hat Enterprise Linux 5.4, the default NOP-Out requests time out was 15 seconds.

CHAPTER 26. DEVICE MAPPER MULTIPATHING AND VIRTUAL STORAGE

Red Hat Enterprise Linux 7 supports *DM-Multipath* and *virtual storage*. Both features are documented in detail in the Red Hat books DM Multipath and Virtualization Deployment and Administration Guide.

26.1. VIRTUAL STORAGE

Red Hat Enterprise Linux 7 supports the following file systems/online storage methods for virtual storage:

- Fibre Channel

- iSCSI

- NFS

- GFS2

Virtualization in Red Hat Enterprise Linux 7 uses `libvirt` to manage virtual instances. The `libvirt` utility uses the concept of *storage pools* to manage storage for virtualized guests. A storage pool is storage that can be divided up into smaller volumes or allocated directly to a guest. Volumes of a storage pool can be allocated to virtualized guests. There are two categories of storage pools available:

Local storage pools

Local storage covers storage devices, files or directories directly attached to a host. Local storage includes local directories, directly attached disks, and LVM Volume Groups.

Networked (shared) storage pools

Networked storage covers storage devices shared over a network using standard protocols. It includes shared storage devices using Fibre Channel, iSCSI, NFS, GFS2, and SCSI RDMA protocols, and is a requirement for migrating guest virtualized guests between hosts.

> **IMPORTANT**
>
> For comprehensive information on the deployment and configuration of virtual storage instances in your environment, refer to the Virtualization Deployment and Administration Guide provided by Red Hat.

26.2. DM-MULTIPATH

Device Mapper Multipathing (DM-Multipath) is a feature that allows you to configure multiple I/O paths between server nodes and storage arrays into a single device. These I/O paths are physical SAN connections that can include separate cables, switches, and controllers. Multipathing aggregates the I/O paths, creating a new device that consists of the aggregated paths.

DM-Multipath are used primarily for the following reasons:

Redundancy

DM-Multipath can provide failover in an active/passive configuration. In an active/passive configuration, only half the paths are used at any time for I/O. If any element of an I/O path (the cable, switch, or controller) fails, DM-Multipath switches to an alternate path.

Improved Performance

DM-Multipath can be configured in active/active mode, where I/O is spread over the paths in a round-robin fashion. In some configurations, DM-Multipath can detect loading on the I/O paths and dynamically re-balance the load.

> **IMPORTANT**
>
> For comprehensive information on the deployment and configuration of DM Multipath in your environment, refer to the DM Multipath guide provided by Red Hat.

CHAPTER 27. EXTERNAL ARRAY MANAGEMENT (LIBSTORAGEMGMT)

Red Hat Enterprise Linux 7 ships with a new external array management library called **libStorageMgmt**.

27.1. INTRODUCTION TO LIBSTORAGEMGMT

The **libStorageMgmt** library is a storage array independent Application Programming Interface (API). It provides a stable and consistent API that allows developers the ability to programmatically manage different storage arrays and leverage the hardware accelerated features provided.

This library is used as a building block for other higher level management tools and applications. End system administrators can also use it as a tool to manually manage storage and automate storage management tasks with the use of scripts.

The **libStorageMgmt** library allows operations such as:

- List storage pools, volumes, access groups, or file systems.

- Create and delete volumes, access groups, file systems, or NFS exports.

- Grant and remove access to volumes, access groups, or initiators.

- Replicate volumes with snapshots, clones, and copies.

- Create and delete access groups and edit members of a group.

Server resources such as CPU and interconnect bandwidth are not utilized because the operations are all done on the array.

The libstoragemgmt package provides:

- A stable C and Python API for client application and plug-in developers.

- A command-line interface that utilizes the library (**lsmcli**).

- A daemon that executes the plug-in (**lsmd**).

- A simulator plug-in that allows the testing of client applications (**sim**).

- Plug-in architecture for interfacing with arrays.

> **WARNING**
>
> This library and its associated tool have the ability to destroy any and all data located on the arrays it manages. It is highly recommended to develop and test applications and scripts against the storage simulator plug-in to remove any logic errors before working with production systems. Testing applications and scripts on actual non-production hardware before deploying to production is also strongly encouraged if possible.

The **libStorageMgmt** library in Red Hat Enterprise Linux 7 adds a default udev rule to handle the REPORTED LUNS DATA HAS CHANGED unit attention.

When a storage configuration change has taken place, one of several Unit Attention ASC/ASCQ codes reports the change. A uevent is then generated and is rescanned automatically with **sysfs**.

The file **/lib/udev/rules.d/90-scsi-ua.rules** contains example rules to enumerate other events that the kernel can generate.

The **libStorageMgmt** library uses a plug-in architecture to accommodate differences in storage arrays. For more information on **libStorageMgmt** plug-ins and how to write them, refer to the Red Hat Developer Guide.

27.2. LIBSTORAGEMGMT TERMINOLOGY

Different array vendors and storage standards use different terminology to refer to similar functionality. This library uses the following terminology.

Storage array

Any storage system that provides block access (FC, FCoE, iSCSI) or file access through Network Attached Storage (NAS).

Volume

Storage Area Network (SAN) Storage Arrays can expose a volume to the Host Bus Adapter (HBA) over different transports, such as FC, iSCSI, or FCoE. The host OS treats it as block devices. One volume can be exposed to many disks if **multipath[2]** is enabled).

This is also known as the Logical Unit Number (LUN), StorageVolume with SNIA terminology, or virtual disk.

Pool

A group of storage spaces. File systems or volumes can be created from a pool. Pools can be created from disks, volumes, and other pools. A pool may also hold RAID settings or thin provisioning settings.

This is also known as a StoragePool with SNIA Terminology.

Snapshot

A point in time, read only, space efficient copy of data.

This is also known as a read only snapshot.

Clone

A point in time, read writeable, space efficient copy of data.

This is also known as a read writeable snapshot.

Copy

A full bitwise copy of the data. It occupies the full space.

Mirror

A continuously updated copy (synchronous and asynchronous).

Access group

Collections of iSCSI, FC, and FCoE initiators which are granted access to one or more storage volumes. This ensures that only storage volumes are accessibly by the specified initiators.

This is also known as an initiator group.

Access Grant

Exposing a volume to a specified access group or initiator. The `libStorageMgmt` library currently does not support LUN mapping with the ability to choose a specific logical unit number. The `libStorageMgmt` library allows the storage array to select the next available LUN for assignment. If configuring a boot from SAN or masking more than 256 volumes be sure to read the OS, Storage Array, or HBA documents.

Access grant is also known as LUN Masking.

System

Represents a storage array or a direct attached storage RAID.

File system

A Network Attached Storage (NAS) storage array can expose a file system to host an OS through an IP network, using either NFS or CIFS protocol. The host OS treats it as a mount point or a folder containing files depending on the client operating system.

Disk

The physical disk holding the data. This is normally used when creating a pool with RAID settings.

This is also known as a DiskDrive using SNIA Terminology.

Initiator

In Fibre Channel (FC) or Fibre Channel over Ethernet (FCoE), the intiator is the World Wide Port Name (WWPN) or World Wide Node Name (WWNN). In iSCSI, the initiator is the iSCSI Qualified Name (IQN). In NFS or CIFS, the initiator is the host name or the IP address of the host.

Child dependency

Some arrays have an implicit relationship between the origin (parent volume or file system) and the child (such as a snapshot or a clone). For example, it is impossible to delete the parent if it has one or more depend children. The API provides methods to determine if any such relationship exists and

a method to remove the dependency by replicating the required blocks.

27.3. INSTALLING LIBSTORAGEMGMT

To install **libStorageMgmt** for use of the command line, required run-time libraries and simulator plug-ins, use the following command:

```
$ sudo yum install libstoragemgmt libstoragemgmt-python
```

To develop C applications that utilize the library, install the libstoragemgmt-devel package with the following command:

```
# yum install libstoragemgmt-devel
```

To install **libStorageMgmt** for use with hardware arrays, select one or more of the appropriate plug-in packages with the following command:

```
$ sudo yum install libstoragemgmt-name-plugin
```

The following plug-ins that are available include:

libstoragemgmt-smis-plugin

Generic SMI-S array support.

libstoragemgmt-netapp-plugin

Specific support for NetApp files.

libstoragemgmt-nstor-plugin

Specific support for NexentaStor.

libstoragemgmt-targetd-plugin

Specific support for targetd.

The daemon is then installed and configured to run at start up but will not do so until the next reboot. To use it immediately without rebooting, start the daemon manually.

To manage an array requires support through a plug-in. The base install package includes open source plug-ins for a number of different vendors. Additional plug-in packages will be available separately as array support improves. Currently supported hardware is constantly changing and improving.

The libStorageMgmt daemon (**lsmd**) behaves like any standard service for the system.

To check on the status of the libStorageMgmt service, use:

```
$ sudo systemctl status libstoragemgmt
```

To stop the service use:

```
$ sudo systemctl stop libstoragemgmt
```

To start the service use:

```
$ sudo systemctl start libstoragemgmt
```

27.4. USING LIBSTORAGEMGMT

To use **libStorageMgmt** interactively, use the **lsmcli** tool.

The **lsmcli** tool requires two things to run:

- A Uniform Resource Identifier (URI) which is used to identify the plug-in to connect to the array and any configurable options the array requires.

- A valid user name and password for the array.

URI has the following form:

plugin+*optional-transport*://*user-name*@*host*:*port*/?*query-string-parameters*

Each plug-in has different requirements for what is needed.

> **Example 27.1. Examples of Different Plug-in Requirements**
>
> **Simulator Plug-in That Requires No User Name or Password**
>
> **sim://**
>
> **NetApp Plug-in over SSL with User Name root**
>
> **ontap+ssl://root@*filer.company*.com/**
>
> **SMI-S Plug-in over SSL for EMC Array**
>
> **smis+ssl://admin@*provider*.com:5989/?namespace=root/emc**

There are three options to use the URI:

1. Pass the URI as part of the command.

   ```
   $ lsmcli -u sim://...
   ```

2. Store the URI in an environmental vairable.

   ```
   $ export LSMCLI_URI=sim:// && lsmcli ...
   ```

3. Place the URI in the file **~/.lsmcli**, which contains name-value pairs separated by "=". The only currently supported configuration is 'uri'.

Determining which URI to use needs to be done in this order. If all three are supplied, only the first one on the command line will be used.

Supply the password by specifying the **-P** option on the command line or by placing it in an environmental variable **LSMCLI_PASSWORD**.

Example 27.2. Examples of lsmcli

An example for using the command line to create a new volume and making it visible to an initiator.

List arrays that are serviced by this connection.

```
$ lsmcli list --type SYSTEMS
ID     | Name                        | Status
-------+-----------------------------+--------
sim-01 | LSM simulated storage plug-in | OK
```

List storage pools.

```
$ lsmcli list --type POOLS -H
ID   | Name          | Total space          | Free space           |
System ID
-----+---------------+----------------------+----------------------
+-----------
POO2 | Pool 2        | 18446744073709551616 | 18446744073709551616 |
sim-01
POO3 | Pool 3        | 18446744073709551616 | 18446744073709551616 |
sim-01
POO1 | Pool 1        | 18446744073709551616 | 18446744073709551616 |
sim-01
POO4 | lsm_test_aggr | 18446744073709551616 | 18446744073709551616 |
sim-01
```

Create a volume.

```
$ lsmcli volume-create --name volume_name --size 20G --pool POO1 -H
ID   | Name        | vpd83                            | bs  | #blocks
| status | ...
-----+-------------+----------------------------------+-----+-------
---+--------+----
Vol1 | volume_name | F7DDF7CA945C66238F593BC38137BD2F | 512 | 41943040 |
OK      | ...
```

Create an access group with an iSCSI initiator in it.

```
$ lsmcli --create-access-group example_ag --id iqn.1994-
05.com.domain:01.89bd01 --type ISCSI --system sim-01
ID                               | Name       | Initiator ID
|SystemID
---------------------------------+------------+---------------------
-------------+--------
782d00c8ac63819d6cca7069282e03a0 | example_ag | iqn.1994-
05.com.domain:01.89bd01 |sim-01
```

Create an access group with an iSCSI intiator in it:

```
$ lsmcli access-group-create --name example_ag --init iqn.1994-
05.com.domain:01.89bd01 --init-type ISCSI --sys sim-01
ID                               | Name       | Initiator IDs
| System ID
```

```
--------------------------------+-----------+--------------------
-------------+-----------
782d00c8ac63819d6cca7069282e03a0 | example_ag | iqn.1994-
05.com.domain:01.89bd01 | sim-01
```

Allow the access group visibility to the newly created volume:

```
$ lsmcli access-group-grant --ag 782d00c8ac63819d6cca7069282e03a0 --vol
Vol1 --access RW
```

The design of the library provides for a process separation between the client and the plug-in by means of *inter-process communication* (IPC). This prevents bugs in the plug-in from crashing the client application. It also provides a means for plug-in writers to write plug-ins with a license of their own choosing. When a client opens the library passing a URI, the client library looks at the URI to determine which plug-in should be used.

The plug-ins are technically stand alone applications but they are designed to have a file descriptor passed to them on the command line. The client library then opens the appropriate Unix domain socket which causes the daemon to fork and execute the plug-in. This gives the client library a point to point communcation channel with the plug-in. The daemon can be restarted without affecting existing clients. While the client has the library open for that plug-in, the plug-in process is running. After one or more commands are sent and the plug-in is closed, the plug-in process cleans up and then exits.

The default behavior of lsmcli is to wait until the operation is completee. Depending on the requested operations, this could potentially could take many hours. To allow a return to normal usage, it is possible to use the **-b** option on the command line. If the exit code is 0 the command is completed. If the exit code is 7 the command is in progress and a job identifier is written to standard output. The user or script can then take the job ID and query the status of the command as needed by using **lsmcli --jobstatus *JobID***. If the job is now completed, the exit value will be 0 and the results printed to standard output. If the command is still in progress, the return value will be 7 and the percentage complete will be printed to the standard output.

Example 27.3. An Asynchronous Example

Create a volume passing the **-b** option so that the command returns immediately.

```
$ lsmcli volume-create --name async_created --size 20G --pool POO1 -b
JOB_3
```

Check to see what the exit value was, remembering that 7 indicates the job is still in progress.

```
$ echo $?
7
```

Check to see if the job is completed.

```
$ lsmcli job-status --job JOB_3
33
```

Check to see what the exit value was, remembering that 7 indicates the job is still in progress so the standard output is the percentage done or 33% based on the above screen.

```
$ echo $?
```

```
7
```

Wait some more and check it again, remembering that exit 0 means success and standard out displays the new volume.

```
$ lsmcli job-status --job JOB_3
ID   | Name            | vpd83                            | Block Size
| ...
-----+--------------+--------------------------------+----------
--+-----
Vol2 | async_created | 855C9BA51991B0CC122A3791996F6B15 | 512        |
...
```

For scripting, pass the **-t** *SeparatorCharacters* option. This will make it easier to parse the output.

Example 27.4. Scripting Examples

```
$ lsmcli list --type volumes -t#
Vol1#volume_name#049167B5D09EC0A173E92A63F6C3EA2A#512#41943040#214748364
80#OK#sim-01#POO1
Vol2#async_created#3E771A2E807F68A32FA5E15C235B60CC#512#41943040#2147483
6480#OK#sim-01#POO1
```

```
$ lsmcli list --type volumes -t " | "
Vol1 | volume_name | 049167B5D09EC0A173E92A63F6C3EA2A | 512 | 41943040 |
21474836480 | OK | 21474836480 | sim-01 | POO1
Vol2 | async_created | 3E771A2E807F68A32FA5E15C235B60CC | 512 | 41943040
| 21474836480 | OK | sim-01 | POO1
```

```
$ lsmcli list --type volumes -s
------------------------------------------------
ID          | Vol1
Name        | volume_name
VPD83       | 049167B5D09EC0A173E92A63F6C3EA2A
Block Size  | 512
#blocks     | 41943040
Size        | 21474836480
Status      | OK
System ID   | sim-01
Pool ID     | POO1
------------------------------------------------
ID          | Vol2
Name        | async_created
VPD83       | 3E771A2E807F68A32FA5E15C235B60CC
Block Size  | 512
#blocks     | 41943040
Size        | 21474836480
Status      | OK
System ID   | sim-01
Pool ID     | POO1
------------------------------------------------
```

It is recommended to use the Python library for non-trivial scripting.

For more information on **lsmcli**, refer to the man pages or the command `lsmcli --help`.

CHAPTER 28. PERSISTENT MEMORY: NVDIMMS

Persistent memory (**pmem**), also called as storage class memory, is a combination of memory and storage. pmem combines the durability of storage with the low access latency and the high bandwidth of dynamic RAM (DRAM):

- Persistent memory is byte-addressable, so it can be accessed by using CPU load and store instructions. In addition to **read()** or **write()** system calls that are required for accessing traditional block-based storage, pmem also supports direct load and store programming model.

- The performance characteristics of persistent memory are similar to DRAM with very low access latency, typically in the tens to hundreds of nanoseconds.

- Contents of persistent memory are preserved when the power is off, like with storage.

Using persistent memory is beneficial for use cases like:

Rapid start: data set is already in memory.

Rapid start is also called the *warm cache* effect. A file server has none of the file contents in memory after starting. As clients connect and read and write data, that data is cached in the page cache. Eventually, the cache contains mostly hot data. After a reboot, the system must start the process again.

Persistent memory allows an application to keep the warm cache across reboots if the application is designed properly. In this instance, there would be no page cache involved: the application would cache data directly in the persistent memory.

Fast write-cache

File servers often do not acknowledge a client's write request until the data is on durable media. Using persistent memory as a fast write cache enables a file server to acknowledge the write request quickly thanks to the low latency of pmem.

NVDIMMs Interleaving

Non-Volatile Dual In-line Memory Modules (NVDIMMs) can be grouped into interleave sets in the same way as regular DRAM. An interleave set is like a RAID 0 (stripe) across multiple DIMMs.

Following are the advantages of NVDIMMS interleaving:

- Like DRAM, NVDIMMs benefit from increased performance when they are configured into interleave sets.

- It can be used to combine multiple smaller NVDIMMs into one larger logical device.

Use the system BIOS or UEFI firmware to configure interleave sets.

In Linux, one region device is created per interleave set.

Following is the relation between region devices and labels:

- If your NVDIMMs support labels, the region device can be further subdivided into namespaces.

- If your NVDIMMs do not support labels, the region devices can only contain a single namespace. In this case, the kernel creates a default namespace which covers the entire region.

Persistent Memory Access Modes

You can use persistent memory in **sector**, **memory**, **dax** (Direct Access) or **raw** mode:

sector mode

It presents the storage as a fast block device. Using sector mode is useful for legacy applications that have not been modified to use persistent memory, or for applications that make use of the full I/O stack, including the Device Mapper.

memory mode

It enables persistent memory devices to support direct access programming as described in the Storage Networking Industry Association (SNIA) Non-Volatile Memory (NVM) Programming Model specification. In memory mode, I/O bypasses the storage stack of the kernel, and many **Device Mapper** drivers therefore cannot be used.

dax mode

The dax mode,also called device DAX, provides raw access to persistent memory by using a DAX character device node. Data on a DAX device can be made durable using CPU cache flushing and fencing instructions. Certain databases and virtual machine hypervisors might benefit from DAX mode. File systems cannot be created on device dax instances.

raw mode

The raw mode namespaces have several limitations and should not be used.

28.1. CONFIGURING PERSISTENT MEMORY WITH NDCTL

Use the **ndctl** utility to configure persistent memory devices. To install **ndctl** utility, use the following command:

```
# yum install ndctl
```

Procedure 28.1. Configuring Persistent Memory for device that does not support labels

1. List the available pmem regions on your system. In the following example, the command lists an NVDIMM-N device that does not support labels:

```
# ndctl list --regions
[
  {
    "dev":"region1",
    "size":34359738368,
    "available_size":0,
    "type":"pmem"
  },
  {
    "dev":"region0",
    "size":34359738368,
    "available_size":0,
    "type":"pmem"
  }
]
```

OS creates a default namespace for each region because the NVDIMM-N device here does not support labels. Hence, the available size is 0 bytes.

2. List all the inactive namespaces on your system:

```
# ndctl list --namespaces --idle
[
  {
    "dev":"namespace1.0",
    "mode":"raw",
    "size":34359738368,
    "state":"disabled",
    "numa_node":1
  },
  {
    "dev":"namespace0.0",
    "mode":"raw",
    "size":34359738368,
    "state":"disabled",
    "numa_node":0
  }
]
```

3. Reconfigure the inactive namespaces in order to make use of this space. For example, to use *namespace0.0* for a file system that supports DAX, use the following command:

```
# ndctl create-namespace --force --reconfig=namespace0.0 --
mode=memory --map=mem
{
  "dev":"namespace0.0",
  "mode":"memory",
  "size":"32.00 GiB (34.36 GB)",
  "uuid":"ab91cc8f-4c3e-482e-a86f-78d177ac655d",
  "blockdev":"pmem0",
  "numa_node":0
}
```

Procedure 28.2. Configuring Persistent Memory for device that support labels

1. List the available pmem regions on your system. In the following example, the command lists an NVDIMM-N device that support labels:

```
# ndctl list --regions
[
  {
    "dev":"region5",
    "size":270582939648,
    "available_size":270582939648,
    "type":"pmem",
    "iset_id":-7337419320239190016
  },
  {
    "dev":"region4",
    "size":270582939648,
    "available_size":270582939648,
```

```
            "type":"pmem",
            "iset_id":-137289417188962304
        }
    ]
```

2. If an NVDIMM support labels, default namespaces are not created, and you can allocate one or more namespaces from a region without using the **--force** or **--reconfigure** flags:

```
# ndctl create-namespace --region=region4 --mode=memory --map=dev --
size=36G
{
   "dev":"namespace4.0",
   "mode":"memory",
   "size":"35.44 GiB (38.05 GB)",
   "uuid":"9c5330b5-dc90-4f7a-bccd-5b558fa881fe",
   "blockdev":"pmem4",
   "numa_node":0
}
```

Now, you can create another namespace from the same region:

```
# ndctl create-namespace --region=region4 --mode=memory --map=dev --
size=36G
{
   "dev":"namespace4.1",
   "mode":"memory",
   "size":"35.44 GiB (38.05 GB)",
   "uuid":"91868e21-830c-4b8f-a472-353bf482a26d",
   "blockdev":"pmem4.1",
   "numa_node":0
}
```

You can also create namespaces of different types from the same region, using the following command:

```
# ndctl create-namespace --region=region4 --mode=dax --align=2M --
size=36G
{
   "dev":"namespace4.2",
   "mode":"dax",
   "size":"35.44 GiB (38.05 GB)",
   "uuid":"a188c847-4153-4477-81bb-7143e32ffc5c",
   "daxregion":
   {
      "id":4,
      "size":"35.44 GiB (38.05 GB)",
      "align":2097152,
      "devices":[
        {
           "chardev":"dax4.2",
           "size":"35.44 GiB (38.05 GB)"
        }]
   },
      "numa_node":0
}
```

For more information on **ndctl** utility, see `man ndctl`.

28.2. CONFIGURING PERSISTENT MEMORY FOR USE AS A BLOCK DEVICE (LEGACY MODE)

To use persistent memory as a fast block device, set the namespace to sector mode.

```
# ndctl create-namespace --force --reconfig=namespace1.0 --mode=sector
{
  "dev":"namespace1.0",
  "mode":"sector",
  "size":17162027008,
  "uuid":"029caa76-7be3-4439-8890-9c2e374bcc76",
  "sector_size":4096,
  "blockdev":"pmem1s"
}
```

In the example, *namespace1.0* is reconfigured to sector mode. Note that the block device name changed from **pmem1** to **pmem1s**. This device can be used in the same way as any other block device on the system. For example, the device can be partitioned, you can create a file system on the device, the device can be configured as part of a software RAID set, and the device can be the cache device for **dm-cache**.

28.3. CONFIGURING PERSISTENT MEMORY FOR FILE SYSTEM DIRECT ACCESS (DAX)

Direct access requires the namespace to be configured to memory mode. Memory mode allows for the direct access programming model. When a device is configured in memory mode, a file system can be created on top of it, and then mounted with the **-o dax** mount option. Then, any application that performs an **mmap()** operation on a file on this file system gets direct access to its storage. See the following example:

```
# ndctl create-namespace --force --reconfig=namespace0.0 --mode=memory --
map=mem
{
  "dev":"namespace0.0",
  "mode":"memory",
  "size":17177772032,
  "uuid":"e6944638-46aa-4e06-a722-0b3f16a5acbf",
  "blockdev":"pmem0"
}
```

In the example, *namespace0.0* is converted to namespace memory mode. With the **--map=mem** argument, **ndctl** puts operating system data structures used for Direct Memory Access (DMA) in system DRAM.

To perform DMA, the kernel requires a data structure for each page in the memory region. The overhead of this data structure is 64 bytes per 4-KiB page. For small devices, the amount of overhead is small enough to fit in DRAM with no problems. For example, the 16-GiB namespace only requires 256MiB for page structures. Because the NVDIMM is small and expensive, storing the kernel's page tracking data structures in DRAM is preferable, as indicated by the **--map=mem** parameter.

In the future, NVDIMM devices might be terabytes in size. For such devices, the amount of memory required to store the page tracking data structures might exceed the amount of DRAM in the system. One TiB of persistent memory requires 16 GiB just for page structures. As a result, specifying the **`--map=dev`** parameter to store the data structures in the persistent memory itself is preferable in such cases.

After configuring the namespace in memory mode, the namespace is ready for a file system. Starting with Red Hat Enterprise Linux 7.3, both the Ext4 and XFS file system enable using persistent memory as a Technology Preview. File system creation requires no special arguments. To get the DAX functionality, **`mount`** the file system with the **`dax`** mount option. For example:

```
# mkfs -t xfs /dev/pmem0
# mount -o dax /dev/pmem0 /mnt/pmem/
```

Then, applications can use persistent memory and create files in the **`/mnt/pmem/`** directory, open the files, and use the **`mmap`** operation to map the files for direct access.

When creating partitions on a pmem device to be used for direct access, partitions must be aligned on page boundaries. On the Intel 64 and AMD64 architecture, at least 4KiB alignment for the start and end of the partition, but 2MiB is the preferred alignment. By default, the **parted** tool aligns partitions on 1MiB boundaries. For the first partition, specify 2MiB as the start of the partition. If the size of the partition is a multiple of 2MiB, all other partitions are also aligned.

28.4. CONFIGURING PERSISTENT MEMORY FOR USE IN DEVICE DAX MODE

Device DAX provides a means for applications to directly access storage, without the involvement of a file system. The benefit of device DAX is that it provides a guaranteed fault granularity, which can be configured using the **`--align`** option with the **`ndctl`** utilty:

```
# ndctl create-namespace --force --reconfig=namespace0.0 --mode=dax --
align=2M
```

The given command ensures that the operating system would fault in 2MiB pages at a time. For the Intel 64 and AMD64 architecture, the following fault granularities are supported:

- 4KiB

- 2MiB

- 1GiB

Device DAX nodes (**`/dev/daxN.M`**) only supports the following system call:

- **`open()`**

- **`close()`**

- **`mmap()`**

- **`fallocate()`**

`read()` and **`write()`** variants are not supported because the use case is tied to persistent memory programming.

28.5. TROUBLESHOOTING

Some NVDIMMs support Self-Monitoring, Analysis and Reporting Technology (S.M.A.R.T.) interfaces for retrieving health information.

> **NOTE**
>
> On some systems, the acpi_ipmi driver must be loaded to retrieve health information using the following command:
>
> ```
> # modprobe acpi_ipmi
> ```

To access the health information, use the following command:

```
# ndctl list --dimms --health --dimm=nmem0
[
  {
    {
      "dev":"nmem0",
      "id":"802c-01-1513-b3009166",
      "handle":1,
      "phys_id":22,
      "health":
      {
        "health_state":"ok",
        "temperature_celsius":25.000000,
        "spares_percentage":99,
        "alarm_temperature":false,
        "alarm_spares":false,
        "temperature_threshold":50.000000,
        "spares_threshold":20,
        "life_used_percentage":1,
        "shutdown_state":"clean"
      }
    }
  }
]
```

PART III. DATA DEDUPLICATION AND COMPRESSION WITH VDO

This part describes how to provide deduplicated block storage capabilities to existing storage management applications by enabling them to utilize Virtual Data Optimizer (VDO).

CHAPTER 29. VDO INTEGRATION

29.1. THEORETICAL OVERVIEW OF VDO

Virtual Data Optimizer (VDO) is a block virtualization technology that allows you to easily create compressed and deduplicated pools of block storage.

- **Deduplication** is a technique for reducing the consumption of storage resources by eliminating multiple copies of duplicate blocks.

  Instead of writing the same data more than once, VDO detects each duplicate block and records it as a reference to the original block. VDO maintains a mapping from logical block addresses, which are used by the storage layer above VDO, to physical block addresses, which are used by the storage layer under VDO.

  After deduplication, multiple logical block addresses may be mapped to the same physical block address; these are called *shared blocks*. Block sharing is invisible to users of the storage, who read and write blocks as they would if VDO were not present. When a shared block is overwritten, a new physical block is allocated for storing the new block data to ensure that other logical block addresses that are mapped to the shared physical block are not modified.

- **Compression** is a data-reduction technique that works well with file formats that do not necessarily exhibit block-level redundancy, such as log files and databases. See Section 29.4.8, "Using Compression" for more detail.

The VDO solution consists of the following components:

kvdo

A kernel module that loads into the Linux Device Mapper layer to provide a deduplicated, compressed, and thinly provisioned block storage volume

uds

A kernel module that communicates with the Universal Deduplication Service (UDS) index on the volume and analyzes data for duplicates.

Command line tools

For configuring and managing optimized storage.

29.1.1. The UDS Kernel Module (`uds`)

The UDS index provides the foundation of the VDO product. For each new piece of data, it quickly determines if that piece is identical to any previously stored piece of data. If the index finds match, the storage system can then internally reference the existing item to avoid storing the same information more than once.

The UDS index runs inside the kernel as the **uds** kernel module.

29.1.2. The VDO Kernel Module (`kvdo`)

The **kvdo** Linux kernel module provides block-layer deduplication services within the Linux Device Mapper layer. In the Linux kernel, Device Mapper serves as a generic framework for managing pools of block storage, allowing the insertion of block-processing modules into the storage stack between the

kernel's block interface and the actual storage device drivers.

The **kvdo** module is exposed as a block device that can be accessed directly for block storage or presented through one of the many available Linux file systems, such as XFS or ext4. When **kvdo** receives a request to read a (logical) block of data from a VDO volume, it maps the requested logical block to the underlying physical block and then reads and returns the requested data.

When **kvdo** receives a request to write a block of data to a VDO volume, it first checks whether it is a `DISCARD` or `TRIM` request or whether the data is uniformly zero. If either of these conditions holds, **kvdo** updates its block map and acknowledges the request. Otherwise, a physical block is allocated for use by the request.

Overview of VDO Write Policies

If the **kvdo** module is operating in synchronous mode:

1. It temporarily writes the data in the request to the allocated block and then acknowledges the request.

2. Once the acknowledgment is complete, an attempt is made to deduplicate the block by computing a MurmurHash-3 signature of the block data, which is sent to the VDO index.

3. If the VDO index contains an entry for a block with the same signature, **kvdo** reads the indicated block and does a byte-by-byte comparison of the two blocks to verify that they are identical.

4. If they are indeed identical, then **kvdo** updates its block map so that the logical block points to the corresponding physical block and releases the allocated physical block.

5. If the VDO index did not contain an entry for the signature of the block being written, or the indicated block does not actually contain the same data, **kvdo** updates its block map to make the temporary physical block permanent.

If **kvdo** is operating in asynchronous mode:

1. Instead of writing the data, it will immediately acknowledge the request.

2. It will then attempt to deduplicate the block in same manner as described above.

3. If the block turns out to be a duplicate, **kvdo** will update its block map and release the allocated block. Otherwise, it will write the data in the request to the allocated block and update the block map to make the physical block permanent.

29.1.3. VDO Volume

VDO uses a block device as a backing store, which can include an aggregation of physical storage consisting of one or more disks, partitions, or even flat files. When a VDO volume is created by a storage management tool, VDO reserves space from the volume for both a UDS index and the VDO volume, which interact together to provide deduplicated block storage to users and applications. Figure 29.1, "VDO Disk Organization" illustrates how these pieces fit together.
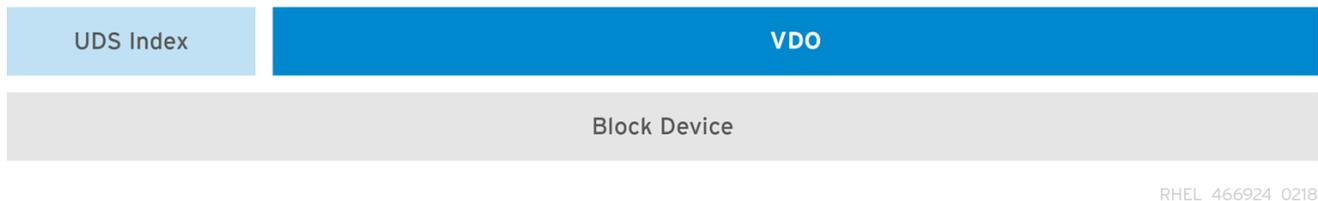
Figure 29.1. VDO Disk Organization

Slabs

The physical storage of the VDO volume is divided into a number of slabs, each of which is a contiguous region of the physical space. All of the slabs for a given volume will be of the same size, which may be any power of 2 multiple of 128 MB up to 32 GB.

The default slab size is 2 GB in order to facilitate evaluating VDO on smaller test systems. A single VDO volume may have up to 8096 slabs. Therefore, in the default configuration with 2 GB slabs, the maximum allowed physical storage is 16 TB. When using 32 GB slabs, the maximum allowed physical storage is 256 TB.

For a recommendation on what slab size to choose depending on your physical volume size, see Table 29.1, "Recommended VDO Slab Sizes by Physical Volume Size".

At least one entire slab will be reserved by VDO for metadata, and therefore cannot be used for storing user data.

The size of a slab can be controlled by providing the `--vdoSlabSize=`*`megabytes`* option to the **vdo create** command.

Table 29.1. Recommended VDO Slab Sizes by Physical Volume Size

Physical Volume Size	10–99 GB	100 GB – 1 TB	2–10 TB	11–50 TB	51–100 TB	101–256 TB
Slab Size	1 GB	2 GB	32 GB	32 GB	32 GB	32 GB

Physical Size and Available Physical Size

Both physical size and available physical size describe the amount of disk space on the block device that VDO can utilize:

- **Physical size** is the same size as the underlying block device. VDO uses this storage for:

    - User data, which might be deduplicated and compressed

    - VDO metadata, such as the UDS index

- **Available physical size** is the portion of the physical size that VDO is able to use for user data.

    It is equivalent to the physical size minus the size of the metadata, minus the remainder after dividing the volume into slabs by the given slab size.

For examples of how much storage VDO metadata require on block devices of different sizes, see Section 29.2.3, "Examples of VDO System Requirements by Physical Volume Size".

Logical Size

If the `--vdoLogicalSize` option is not specified, the logical volume size defaults to the available physical volume size. Note that, in Figure 29.1, "VDO Disk Organization", the VDO deduplicated storage target sits completely on top of the block device, meaning the physical size of the VDO volume is the same size as the underlying block device.

VDO currently supports any logical size up to 254 times the size of the physical volume with an absolute maximum logical size of 4PB.

29.1.4. Command Line Tools

VDO includes the following command line tools for configuration and management:

`vdo`

> Creates, configures, and controls VDO volumes

`vdostats`

> Provides utilization and performance statistics

29.2. SYSTEM REQUIREMENTS

Processor Architectures
One or more processors implementing the Intel 64 instruction set are required: that is, a processor of the AMD64 or Intel 64 architecture.

RAM
Each VDO volume has two distinct memory requirements:

- The VDO module requires 370 MB plus an additional 268 MB per each 1 TB of physical storage managed.

- The Universal Deduplication Service (UDS) index requires a minimum of 250 MB of DRAM, which is also the default amount that deduplication uses. For details on the memory usage of UDS, see Section 29.2.1, "UDS Index Memory Requirements".

Storage
A single VDO volume can be configured to use up to 256 TB of physical storage. See Section 29.2.2, "VDO Storage Requirements" for the calculations to determine the usable size of a VDO-managed volume from the physical size of the storage pool the VDO is given.

Additional System Software
VDO depends on the following software:

- LVM

- Python 2.7

The **yum** package manager will install all necessary software dependencies automatically.

Placement of VDO in the Storage Stack

As a general rule, you should place certain storage layers under VDO and others on top of VDO:

- Under VDO: DM-Multipath, DM-Crypt, and software RAID (LVM or `mdraid`).

- On top of VDO: LVM cache, LVM Logical Volumes, LVM snapshots, and LVM Thin Provisioning.

The following configurations are not supported:

- VDO on top of VDO volumes: storage → VDO → LVM → VDO

- VDO on top of LVM Snapshots

- VDO on top of LVM Cache

- VDO on top of the loopback device

- VDO on top of LVM Thin Provisioning

- Encrypted volumes on top of VDO: storage → VDO → DM-Crypt

- Partitions on a VDO volume: `fdisk`, `parted`, and similar partitions

- RAID (LVM, MD, or any other type) on top of a VDO volume

> **IMPORTANT**
>
> VDO supports two write modes: `sync` and `async`. When VDO is in `sync` mode, writes to the VDO device are acknowledged when the underlying storage has written the data permanently. When VDO is in `async` mode, writes are acknowledged before being written to persistent storage.
>
> It is critical to set the VDO write policy to match the behavior of the underlying storage. By default, VDO write policy is set to the `auto` option, which selects the appropriate policy automatically.
>
> For more information, see Section 29.4.2, "Selecting VDO Write Modes".

29.2.1. UDS Index Memory Requirements

The UDS index consists of two parts:

- A compact representation is used in memory that contains at most one entry per unique block.

- An on-disk component which records the associated block names presented to the index as they occur, in order.

UDS uses an average of 4 bytes per entry in memory (including cache).

The on-disk component maintains a bounded history of data passed to UDS. UDS provides deduplication advice for data that falls within this deduplication window, containing the names of the most recently seen blocks. The deduplication window allows UDS to index data as efficiently as possible while limiting the amount of memory required to index large data repositories. Despite the bounded nature of the deduplication window, most datasets which have high levels of deduplication also exhibit a high degree of temporal locality — in other words, most deduplication occurs among sets of blocks that were written at about the same time. Furthermore, in general, data being written is more likely to duplicate data that

was recently written than data that was written a long time ago. Therefore, for a given workload over a given time interval, deduplication rates will often be the same whether UDS indexes only the most recent data or all the data.

Because duplicate data tends to exhibit temporal locality, it is rarely necessary to index every block in the storage system. Were this not so, the cost of index memory would outstrip the savings of reduced storage costs from deduplication. Index size requirements are more closely related to the rate of data ingestion. For example, consider a storage system with 100 TB of total capacity but with an ingestion rate of 1 TB per week. With a deduplication window of 4 TB, UDS can detect most redundancy among the data written within the last month.

UDS's Sparse Indexing feature (the recommended mode for VDO) further exploits temporal locality by attempting to retain only the most relevant index entries in memory. UDS can maintain a deduplication window that is ten times larger while using the same amount of memory. While the sparse index provides the greatest coverage, the dense index provides more advice. For most workloads, given the same amount of memory, the difference in deduplication rates between dense and sparse indexes is negligible.

The memory required for the index is determined by the desired size of the deduplication window:

- For a dense index, UDS will provide a deduplication window of 1 TB per 1 GB of RAM. A 1 GB index is generally sufficient for storage systems of up to 4 TB.

- For a sparse index, UDS will provide a deduplication window of 10 TB per 1 GB of RAM. A 1 GB sparse index is generally sufficient for up to 40 TB of physical storage.

For concrete examples of UDS Index memory requirements, see Section 29.2.3, "Examples of VDO System Requirements by Physical Volume Size"

29.2.2. VDO Storage Requirements

VDO requires storage both for VDO metadata and for the actual UDS deduplication index:

- VDO writes two types of metadata to its underlying physical storage:

  - The first type scales with the physical size of the VDO volume and uses approximately 1 MB for each 4 GB of physical storage plus an additional 1 MB per slab.

  - The second type scales with the logical size of the VDO volume and consumes approximately 1.25 MB for each 1 GB of logical storage, rounded up to the nearest slab.

    See Section 29.1.3, "VDO Volume" for a description of slabs.

- The UDS index is stored within the VDO volume group and is managed by the associated VDO instance. The amount of storage required depends on the type of index and the amount of RAM allocated to the index. For each 1 GB of RAM, a dense UDS index will use 17 GB of storage, and a sparse UDS index will use 170 GB of storage.

For concrete examples of VDO storage requirements, see Section 29.2.3, "Examples of VDO System Requirements by Physical Volume Size"

29.2.3. Examples of VDO System Requirements by Physical Volume Size

The following tables provide approximate system requirements of VDO based on the size of the underlying physical volume. Each table lists requirements appropriate to the intended deployment, such as primary storage or backup storage.

The exact numbers depend on your configuration of the VDO volume.

Primary Storage Deployment

In the primary storage case, the UDS index is between 0.01% to 25% the size of the physical volume.

Table 29.2. VDO Storage and Memory Requirements for Primary Storage

Physical Volume Size	10 GB – 1–TB	2–10 TB	11–50 TB	51–100 TB	101–256 TB
RAM Usage	250 MB	Dense: 1 GB Sparse: 250 MB	2 GB	3 GB	12 GB
Disk Usage	2.5 GB	Dense: 10 GB Sparse: 22 GB	170 GB	255 GB	1020 GB
Index Type	Dense	Dense or Sparse	Sparse	Sparse	Sparse

Backup Storage Deployment

In the backup storage case, the UDS index covers the size of the backup set but is not bigger than the physical volume. If you expect the backup set or the physical size to grow in the future, factor this into the index size.

Table 29.3. VDO Storage and Memory Requirements for Backup Storage

Physical Volume Size	10 GB – 1 TB	2–10 TB	11–50 TB	51–100 TB	101–256 TB
RAM Usage	250 MB	2 GB	10 GB	20 GB	26 GB
Disk Usage	2.5 GB	170 GB	850 GB	1700 GB	3400 GB
Index Type	Dense	Sparse	Sparse	Sparse	Sparse

29.3. GETTING STARTED WITH VDO

29.3.1. Introduction

Virtual Data Optimizer (VDO) provides inline data reduction for Linux in the form of deduplication, compression, and thin provisioning. When you set up a VDO volume, you specify a block device on which to construct your VDO volume and the amount of logical storage you plan to present.

- When hosting active VMs or containers, Red Hat recommends provisioning storage at a 10:1 logical to physical ratio: that is, if you are utilizing 1 TB of physical storage, you would present it as 10 TB of logical storage.

- For object storage, such as the type provided by Ceph, Red Hat recommends using a 3:1 logical to physical ratio: that is, 1 TB of physical storage would present as 3 TB logical storage.

In either case, you can simply put a file system on top of the logical device presented by VDO and then use it directly or as part of a distributed cloud storage architecture.

This chapter describes the following use cases of VDO deployment:

- the direct-attached use case for virtualization servers, such as those built using Red Hat Virtualization, and

- the cloud storage use case for object-based distributed storage clusters, such as those built using Ceph Storage.

> **NOTE**
>
> VDO deployment with Ceph is currently not supported.

This chapter provides examples for configuring VDO for use with a standard Linux file system that can be easily deployed for either use case; see the diagrams in Section 29.3.5, "Deployment Examples".

29.3.2. Installing VDO

VDO is deployed using the following RPM packages:

- vdo

- kmod-kvdo

To install VDO, use the **yum** package manager to install the RPM packages:

```
# yum install vdo kmod-kvdo
```

29.3.3. Creating a VDO Volume

Create a VDO volume for your block device. Note that multiple VDO volumes can be created for separate devices on the same machine. If you choose this approach, you must supply a different name and device for each instance of VDO on the system.

In all the following steps, replace *vdo_name* with the identifier you want to use for your VDO volume; for example, **vdo1**.

> **NOTE**
>
> Before creating volumes, VDO uses LVM utilities such as, **pvcreate --test** to validate block device.

1. Create the VDO volume using the VDO Manager:

```
# vdo create \
        --name=vdo_name \
        --device=block_device \
        --vdoLogicalSize=logical_size
```

- Replace *block_device* with the persistent name of the block device where you want to create the VDO volume. For example, **/dev/disk/by-id/scsi-3600508b1001c264ad2af21e903ad031f**.

  > **IMPORTANT**
  >
  > Use a persistent device name. If you use a non-persistent device name, then VDO might fail to start properly in the future if the device name changes.
  >
  > For more information on persistent names, see Section 25.7, "Persistent Naming".

- Replace *logical_size* with the amount of logical storage that the VDO volume should present:

  - For active VMs or container storage, use logical size that is *ten* times the physical size of your block device. For example, if your block device is 1 TB in size, use **10T** here.

  - For object storage, use logical size that is *three* times the physical size of your block device. For example, if your block device is 1 TB in size, use **3T** here.

  Example 29.1. Creating VDO for Container Storage

  For example, to create a VDO volume for container storage on a 1 TB block device, you might use:

  ```
  # vdo create \
        --name=vdo1 \
        --device=/dev/disk/by-id/scsi-
  3600508b1001c264ad2af21e903ad031f \
        --vdoLogicalSize=10T
  ```

When a VDO volume is created, VDO adds an entry to the **/etc/vdoconf.yml** configuration file. The **vdo.service** systemd unit then uses the entry to start the volume by default.

> **IMPORTANT**
>
> If a failure occurs when creating the VDO volume, remove the volume to clean up. See Section 29.4.3.1, "Removing an Unsuccessfully Created Volume" for details.

2. Create a file system:

   - For the XFS file system:

     ```
     # mkfs.xfs -K /dev/mapper/vdo_name
     ```

   - For the ext4 file system:

     ```
     # mkfs.ext4 -E nodiscard /dev/mapper/vdo_name
     ```

3. Mount the file system:

```
# mkdir -m 1777 /mnt/vdo_name
# mount /dev/mapper/vdo_name /mnt/vdo_name
```

4. To configure the file system to mount automatically, use either the **/etc/fstab** file or a systemd mount unit:

   - If you decide to use the **/etc/fstab** configuration file, add one of the following lines to the file:

     - For the XFS file system:

       ```
       /dev/mapper/vdo_name /mnt/vdo_name xfs defaults,x-
       systemd.requires=vdo.service 0 0
       ```

     - For the ext4 file system:

       ```
       /dev/mapper/vdo_name /mnt/vdo_name ext4 defaults,x-
       systemd.requires=vdo.service 0 0
       ```

   - Alternatively, if you decide to use a systemd unit, create a systemd mount unit file with the appropriate filename. For the mount point of your VDO volume, create the **/etc/systemd/system/mnt-vdo_name.mount** file with the following content:

     ```
     [Unit]
     Description = VDO unit file to mount file system
     name = vdo_name.mount
     Requires = vdo.service
     After = multi-user.target
     Conflicts = umount.target

     [Mount]
     What = /dev/mapper/vdo_name
     Where = /mnt/vdo_name
     Type = xfs

     [Install]
     WantedBy = multi-user.target
     ```

     An example systemd unit file is also installed at **/usr/share/doc/vdo/examples/systemd/VDO.mount.example**.

5. Enable the **discard** feature for the file system on your VDO device. Both batch and online operations work with VDO.

   For information on how to set up the **discard** feature, see Section 2.4, "Discard Unused Blocks".

29.3.4. Monitoring VDO

Because VDO is thin provisioned, the file system and applications will only see the logical space in use and will not be aware of the actual physical space available. Scripting should be used to monitor the actual available space and generate an alert if use exceeds a threshold: for example, when the file system is 80% full.

VDO space usage and efficiency can be monitored using the **vdostats** utility:

```
# vdostats --human-readable

Device                  1K-blocks    Used       Available    Use%
Space saving%
/dev/mapper/node1osd1   926.5G       21.0G      905.5G       2%      73%
/dev/mapper/node1osd2   926.5G       28.2G      898.3G       3%      64%
```

29.3.5. Deployment Examples

The following examples illustrate how VDO might be used in KVM and other deployments.

VDO Deployment with KVM

To see how VDO can be deployed successfully on a KVM server configured with Direct Attached Storage, see Figure 29.2, "VDO Deployment with KVM".

RHEL_462492_1117
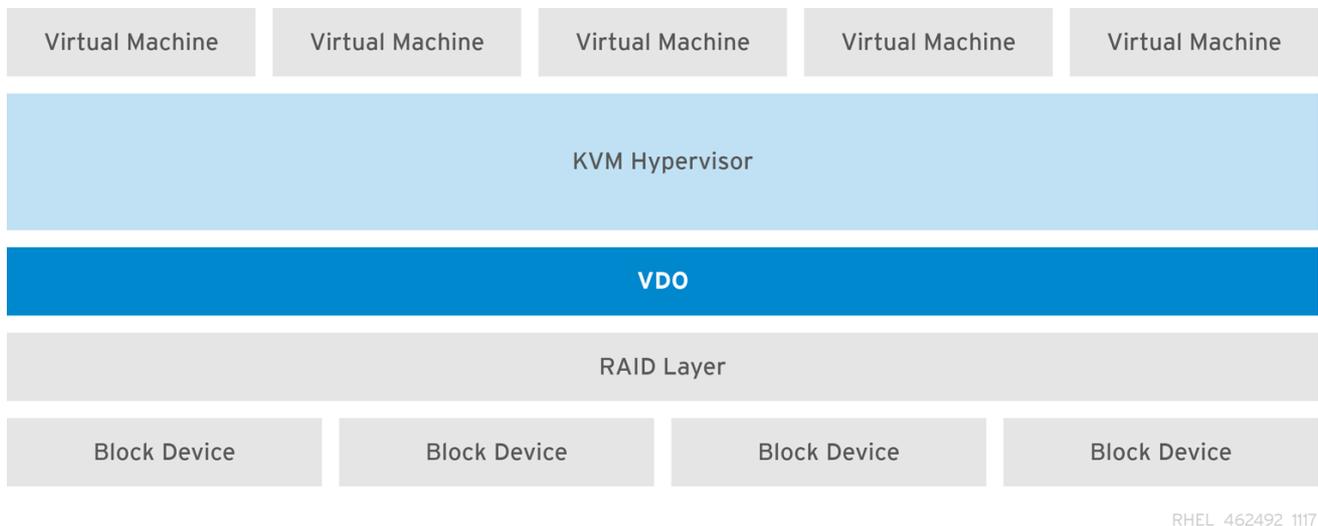
Figure 29.2. VDO Deployment with KVM

More Deployment Scenarios

For more information on VDO deployment, see Section 29.5, "Deployment Scenarios".

29.4. ADMINISTERING VDO

29.4.1. Starting or Stopping VDO

To start a given VDO volume, or all VDO volumes, and the associated UDS index(es), storage management utilities should invoke one of these commands:

```
# vdo start --name=my_vdo
# vdo start --all
```

The VDO systemd unit is installed and enabled by default when the vdo package is installed. This unit automatically runs the **vdo start --all** command at system startup to bring up all *activated* VDO volumes. See Section 29.4.6, "Automatically Starting VDO Volumes at System Boot" for more

information.

To stop a given VDO volume, or all VDO volumes, and the associated UDS index(es), use one of these commands:

```
# vdo stop --name=my_vdo
# vdo stop --all
```

If restarted after an unclean shutdown, VDO will perform a rebuild to verify the consistency of its metadata and will repair it if necessary. Rebuilds are automatic and do not require user intervention. See Section 29.4.5, "Recovering a VDO Volume After an Unclean Shutdown" for more information on the rebuild process.

VDO might rebuild different writes depending on the write mode:

- In synchronous mode, all writes that were acknowledged by VDO prior to the shutdown will be rebuilt.

- In asynchronous mode, all writes that were acknowledged prior to the last acknowledged flush request will be rebuilt.

In either mode, some writes that were either unacknowledged or not followed by a flush may also be rebuilt.

For details on VDO write modes, see Section 29.4.2, "Selecting VDO Write Modes".

29.4.2. Selecting VDO Write Modes

VDO supports three write modes, **sync**, **async**, and **auto**:

- When VDO is in **sync** mode, the layers above it assume that a write command writes data to persistent storage. As a result, it is not necessary for the file system or application, for example, to issue FLUSH or Force Unit Access (FUA) requests to cause the data to become persistent at critical points.

  VDO must be set to **sync** mode only when the underlying storage guarantees that data is written to persistent storage when the write command completes. That is, the storage must either have no volatile write cache, or have a write through cache.

- When VDO is in **async** mode, the data is not guaranteed to be written to persistent storage when a write command is acknowledged. The file system or application must issue FLUSH or FUA requests to ensure data persistence at critical points in each transaction.

  VDO must be set to **async** mode if the underlying storage does not guarantee that data is written to persistent storage when the write command completes; that is, when the storage has a volatile write back cache.

  For information on how to find out if a device uses volatile cache or not, see the section called "Checking for a Volatile Cache".

- The **auto** mode automatically selects **sync** or **async** based on the characteristics of each device. This is the default option.

For a more detailed theoretical overview of how write policies operate, see the section called "Overview of VDO Write Policies".

To set a write policy, use the **--writePolicy** option. This can be specified either when creating a VDO volume as in Section 29.3.3, "Creating a VDO Volume" or when modifying an existing VDO volume with the **changeWritePolicy** subcommand:

```
# vdo changeWritePolicy --writePolicy=sync|async|auto --name=vdo_name
```

> **IMPORTANT**
>
> Using the incorrect write policy might result in data loss on power failure.

Checking for a Volatile Cache

To see whether a device has a writeback cache, read the **/sys/block/*block_device*/device/scsi_disk/*identifier*/cache_type** sysfs file. For example:

- Device **sda** indicates that it *has* a writeback cache:

  ```
  $ cat '/sys/block/sda/device/scsi_disk/7:0:0:0/cache_type'

  write back
  ```

- Device **sdb** indicates that it *does not* have a writeback cache:

  ```
  $ cat '/sys/block/sdb/device/scsi_disk/1:2:0:0/cache_type'

  None
  ```

Additionally, in the kernel boot log, you can find whether the above mentioned devices have a write cache or not:

```
sd 7:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't
support DPO or FUA
sd 1:2:0:0: [sdb] Write cache: disabled, read cache: disabled, supports
DPO and FUA
```

See the Viewing and Managing Log Files chapter in the *System Administrator's Guide* for more information on reading the system log.

In these examples, use the following write policies for VDO:

- **async** mode for the **sda** device

- **sync** mode for the **sdb** device

> **NOTE**
>
> You should configure VDO to use the **sync** write policy if the **cache_type** value is **none** or **write through**.

29.4.3. Removing VDO Volumes

A VDO volume can be removed from the system by running:

```
# vdo remove --name=my_vdo
```

Prior to removing a VDO volume, unmount file systems and stop applications that are using the storage. The **vdo remove** command removes the VDO volume and its associated UDS index, as well as logical volumes where they reside.

29.4.3.1. Removing an Unsuccessfully Created Volume

If a failure occurs when the **vdo** utility is creating a VDO volume, the volume is left in an intermediate state. This might happen when, for example, the system crashes, power fails, or the administrator interrupts a running **vdo create** command.

To clean up from this situation, remove the unsuccessfully created volume with the **--force** option:

```
# vdo remove --force --name=my_vdo
```

The **--force** option is required because the administrator might have caused a conflict by changing the system configuration since the volume was unsuccessfully created. Without the **--force** option, the **vdo remove** command fails with the following message:

```
[...]
A previous operation failed.
Recovery from the failure either failed or was interrupted.
Add '--force' to 'remove' to perform the following cleanup.
Steps to clean up VDO my_vdo:
umount -f /dev/mapper/my_vdo
udevadm settle
dmsetup remove my_vdo
vdo: ERROR - VDO volume my_vdo previous operation (create) is incomplete
```

29.4.4. Configuring the UDS Index

VDO uses a high-performance deduplication index called UDS to detect duplicate blocks of data as they are being stored. The *deduplication window* is the number of previously written blocks which the index remembers. The size of the deduplication window is configurable. For a given window size, the index will requires a specific amount of RAM and a specific amount of disk space. The size of the window is usually determined by specifying the size of the index memory using the **--indexMem=*size*** option. The amount of disk space to use will then be determined automatically.

In general, Red Hat recommends using a *sparse* UDS index for all production use cases. This is an extremely efficient indexing data structure, requiring approximately one-tenth of a byte of DRAM per block in its deduplication window. On disk, it requires approximately 72 bytes of disk space per block. The minimum configuration of this index uses 256 MB of DRAM and approximately 25 GB of space on disk. To use this configuration, specify the **--sparseIndex=enabled --indexMem=0.25** options to the **vdo create** command. This configuration results in a deduplication window of 2.5 TB (meaning it will remember a history of 2.5 TB). For most use cases, a deduplication window of 2.5 TB is appropriate for deduplicating storage pools that are up to 10 TB in size.

The default configuration of the index, however, is to use a *dense* index. This index is considerably less efficient (by a factor of 10) in DRAM, but it has much lower (also by a factor of 10) minimum required disk space, making it more convenient for evaluation in constrained environments.

In general, a deduplication window which is one quarter of the physical size of a VDO volume is a recommended configuration. However, this is not an actual requirement. Even small deduplication windows (compared to the amount of physical storage) can find significant amounts of duplicate data in many use cases. Larger windows may also be used, but it in most cases, there will be little additional benefit to doing so.

Speak with your Red Hat Technical Account Manager representative for additional guidelines on tuning this important system parameter.

29.4.5. Recovering a VDO Volume After an Unclean Shutdown

If a volume is restarted without having been shut down cleanly, VDO will need to rebuild a portion of its metadata to continue operating, which occurs automatically when the volume is started. (Also see Section 29.4.5.2, "Forcing a Rebuild" to invoke this process on a volume that was cleanly shut down.)

Data recovery depends on the write policy of the device:

- If VDO was running on synchronous storage and write policy was set to **sync**, then all data written to the volume will be fully recovered.

- If the write policy was **async**, then some writes may not be recovered if they were not made durable by sending VDO a **FLUSH** command, or a write I/O tagged with the **FUA** flag (force unit access). This is accomplished from user mode by invoking a data integrity operation like **fsync**, **fdatasync**, **sync**, or **umount**.

29.4.5.1. Online Recovery

In the majority of cases, most of the work of rebuilding an unclean VDO volume can be done after the VDO volume has come back online and while it is servicing read and write requests. Initially, the amount of space available for write requests may be limited. As more of the volume's metadata is recovered, more free space may become available. Furthermore, data written while the VDO is recovering may fail to deduplicate against data written before the crash if that data is in a portion of the volume which has not yet been recovered. Data may be compressed while the volume is being recovered. Previously compressed blocks may still be read or overwritten.

During an online recovery, a number of statistics will be unavailable: for example, **blocks in use** and **blocks free**. These statistics will become available once the rebuild is complete.

29.4.5.2. Forcing a Rebuild

VDO can recover from most hardware and software errors. If a VDO volume cannot be recovered successfully, it is placed in a read-only mode that persists across volume restarts. Once a volume is in read-only mode, there is no guarantee that data has not been lost or corrupted. In such cases, Red Hat recommends copying the data out of the read-only volume and possibly restoring the volume from backup. (The **operating mode** attribute of **vdostats** indicates whether a VDO volume is in read-only mode.)

If the risk of data corruption is acceptable, it is possible to force an offline rebuild of the VDO volume metadata so the volume can be brought back online and made available. Again, the integrity of the rebuilt data cannot be guaranteed.

To force a rebuild of a read-only VDO volume, first stop the volume if it is running:

```
# vdo stop --name=my_vdo
```

Then restart the volume using the **--forceRebuild** option:

```
# vdo start --name=my_vdo --forceRebuild
```

29.4.6. Automatically Starting VDO Volumes at System Boot

During system boot, the **vdo** systemd unit automatically starts all VDO devices that are configured as *activated*.

To prevent certain existing volumes from being started automatically, *deactivate* those volumes by running either of these commands:

- To deactivate a specific volume:

  ```
  # vdo deactivate --name=my_vdo
  ```

- To deactivate all volumes:

  ```
  # vdo deactivate --all
  ```

Conversely, to activate volumes, use one of these commands:

- To activate a specific volume:

  ```
  # vdo activate --name=my_vdo
  ```

- To activate all volumes:

  ```
  # vdo activate --all
  ```

You can also create a VDO volume that does not start automatically by adding the **--activate=disabled** option to the **vdo create** command.

For systems that place LVM volumes on top of VDO volumes as well as beneath them (for example, Figure 29.5, "Deduplicated Unified Storage"), it is vital to start services in the right order:

1. The lower layer of LVM must be started first (in most systems, starting this layer is configured automatically when the LVM2 package is installed).

2. The **vdo** systemd unit must then be started.

3. Finally, additional scripts must be run in order to start LVM volumes or other services on top of the now running VDO volumes.

29.4.7. Disabling and Re-enabling Deduplication

In some instances, it may be desirable to temporarily disable deduplication of data being written to a VDO volume while still retaining the ability to read to and write from the volume. While disabling deduplication will prevent subsequent writes from being deduplicated, data which was already deduplicated will remain so.

- To stop deduplication on a VDO volume, use the following command:

```
# vdo disableDeduplication --name=my_vdo
```

This stops the associated UDS index and informs the VDO volume that deduplication is no longer active.

- To restart deduplication on a VDO volume, use the following command:

```
# vdo enableDeduplication --name=my_vdo
```

This restarts the associated UDS index and informs the VDO volume that deduplication is active again.

You can also disable deduplication when creating a new VDO volume by adding the **--deduplication=disabled** option to the **vdo create** command.

29.4.8. Using Compression

29.4.8.1. Introduction

In addition to block-level deduplication, VDO also provides inline block-level compression using the HIOPS Compression™ technology. While deduplication is the optimal solution for virtual machine environments and backup applications, compression works very well with structured and unstructured file formats that do not typically exhibit block-level redundancy, such as log files and databases.

Compression operates on blocks that have not been identified as duplicates. When unique data is seen for the first time, it is compressed. Subsequent copies of data that have already been stored are deduplicated without requiring an additional compression step. The compression feature is based on a parallelized packaging algorithm that enables it to handle many compression operations at once. After first storing the block and responding to the requestor, a best-fit packing algorithm finds multiple blocks that, when compressed, can fit into a single physical block. After it is determined that a particular physical block is unlikely to hold additional compressed blocks, it is written to storage and the uncompressed blocks are freed and reused. By performing the compression and packaging operations after having already responded to the requestor, using compression imposes a minimal latency penalty.

29.4.8.2. Enabling and Disabling Compression

VDO volume compression is on by default.

When creating a volume, you can disable compression by adding the **--compression=disabled** option to the **vdo create** command.

Compression can be stopped on an existing VDO volume if necessary to maximize performance or to speed processing of data that is unlikely to compress.

- To stop compression on a VDO volume, use the following command:

```
# vdo disableCompression --name=my_vdo
```

- To start it again, use the following command:

```
# vdo enableCompression --name=my_vdo
```

29.4.9. Managing Free Space

Because VDO is a thinly provisioned block storage target, the amount of physical space VDO uses may differ from the size of the volume presented to users of the storage. Integrators and systems administrators can exploit this disparity to save on storage costs but must take care to avoid unexpectedly running out of storage space if the data written does not achieve the expected rate of deduplication.

Whenever the number of logical blocks (virtual storage) exceeds the number of physical blocks (actual storage), it becomes possible for file systems and applications to unexpectedly run out of space. For that reason, storage systems using VDO must provide storage administrators with a way of monitoring the size of the VDO's free pool. The size of this free pool may be determined by using the **vdostats** utility; see Section 29.7.2, "vdostats" for details. The default output of this utility lists information for all running VDO volumes in a format similar to the Linux **df** utility. For example:

```
Device                1K-blocks    Used         Available    Use%
/dev/mapper/my_vdo    211812352    105906176    105906176    50%
```

If the size of VDO's free pool drops below a certain level, the storage administrator can take action by deleting data (which will reclaim space whenever the deleted data is not duplicated), adding physical storage, or even deleting LUNs.

Reclaiming Space on File Systems

VDO cannot reclaim space unless file systems communicate that blocks are free using **DISCARD**, **TRIM**, or **UNMAP** commands. For file systems that do not use **DISCARD**, **TRIM**, or **UNMAP**, free space may be manually reclaimed by storing a file consisting of binary zeros and then deleting that file.

File systems may generally be configured to issue **DISCARD** requests in one of two ways:

Realtime discard (also online discard or inline discard)

When realtime discard is enabled, file systems send **REQ_DISCARD** requests to the block layer whenever a user deletes a file and frees space. VDO recieves these requests and returns space to its free pool, assuming the block was not shared.

For file systems that support online discard, you can enable it by setting the **discard** option at mount time.

Batch discard

Batch discard is a user-initiated operation that causes the file system to notify the block layer (VDO) of any unused blocks. This is accomplished by sending the file system an **ioctl** request called **FITRIM**.

You can use the **fstrim** utility (for example from **cron**) to send this **ioctl** to the file system.

For more information on the **discard** feature, see Section 2.4, "Discard Unused Blocks".

Reclaiming Space Without a File System

It is also possible to manage free space when the storage is being used as a block storage target without a file system. For example, a single VDO volume can be carved up into multiple subvolumes by installing the Logical Volume Manager (LVM) on top of it. Before deprovisioning a volume, the **blkdiscard** command can be used in order to free the space previously used by that logical volume.

LVM supports the **REQ_DISCARD** command and will forward the requests to VDO at the appropriate logical block addresses in order to free the space. If other volume managers are being used, they would also need to support **REQ_DISCARD**, or equivalently, **UNMAP** for SCSI devices or **TRIM** for ATA devices.

Reclaiming Space on Fibre Channel or Ethernet Network

VDO volumes (or portions of volumes) can also be provisioned to hosts on a Fibre Channel storage fabric or an Ethernet network using SCSI target frameworks such as LIO or SCST. SCSI initiators can use the **UNMAP** command to free space on thinly provisioned storage targets, but the SCSI target framework will need to be configured to advertise support for this command. This is typically done by enabling *thin provisioning* on these volumes. Support for **UNMAP** can be verified on Linux-based SCSI initiators by running the following command:

```
# sg_vpd --page=0xb0 /dev/device
```

In the output, verify that the "Maximum unmap LBA count" value is greater than zero.

29.4.10. Increasing Logical Volume Size

Management applications can increase the logical size of a VDO volume using the **vdo growLogical** subcommand. Once the volume has been grown, the management should inform any devices or file systems on top of the VDO volume of its new size. The volume may be grown as follows:

```
# vdo growLogical --name=my_vdo --vdoLogicalSize=new_logical_size
```

The use of this command allows storage administrators to initially create VDO volumes which have a logical size small enough to be safe from running out of space. After some period of time, the actual rate of data reduction can be evaluated, and if sufficient, the logical size of the VDO volume can be grown to take advantage of the space savings.

29.4.11. Increasing Physical Volume Size

To increase the amount of physical storage available to a VDO volume:

1. Increase the size of the underlying device.

   The exact procedure depends on the type of the device. For example, to resize an MBR partition, use the **fdisk** utility as described in Section 13.5, "Resizing a Partition with fdisk".

2. Use the **growPhysical** option to add the new physical storage space to the VDO volume:

   ```
   # vdo growPhysical --name=my_vdo
   ```

It is not possible to shrink a VDO volume with this command.

29.4.12. Automating VDO with Ansible

You can use the Ansible tool to automate VDO deployment and administration. For details, see:

- Ansible documentation: https://docs.ansible.com/

- VDO Ansible module documentation:
  https://docs.ansible.com/ansible/latest/modules/vdo_module.html

29.5. DEPLOYMENT SCENARIOS

VDO can be deployed in a variety of ways to provide deduplicated storage for both block and file access and for both local and remote storage. Because VDO exposes its deduplicated storage as a standard Linux block device, it can be used with standard file systems, iSCSI and FC target drivers, or as unified storage.

29.5.1. iSCSI Target

As a simple example, the entirety of the VDO storage target can be exported as an iSCSI Target to remote iSCSI initiators.

Figure 29.3. Deduplicated Block Storage Target

See http://linux-iscsi.org/ for more information on iSCSI Target.

29.5.2. File Systems

If file access is desired instead, file systems can be created on top of VDO and exposed to NFS or CIFS users via either the Linux NFS server or Samba.
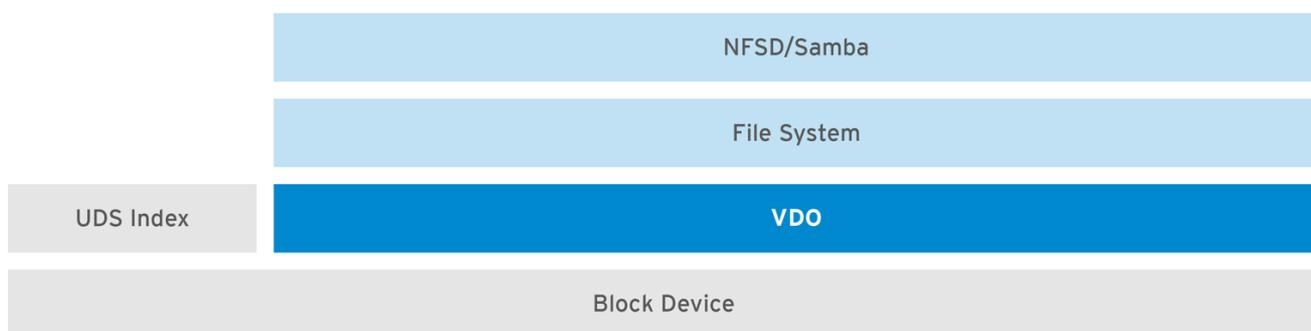
Figure 29.4. Deduplicated NAS

29.5.3. LVM

More feature-rich systems may make further use of LVM to provide multiple LUNs that are all backed by the same deduplicated storage pool. In Figure 29.5, "Deduplicated Unified Storage", the VDO target is registered as a physical volume so that it can be managed by LVM. Multiple logical volumes (**LV1** to **LV4**) are created out of the deduplicated storage pool. In this way, VDO can support multiprotocol unified block/file access to the underlying deduplicated storage pool.
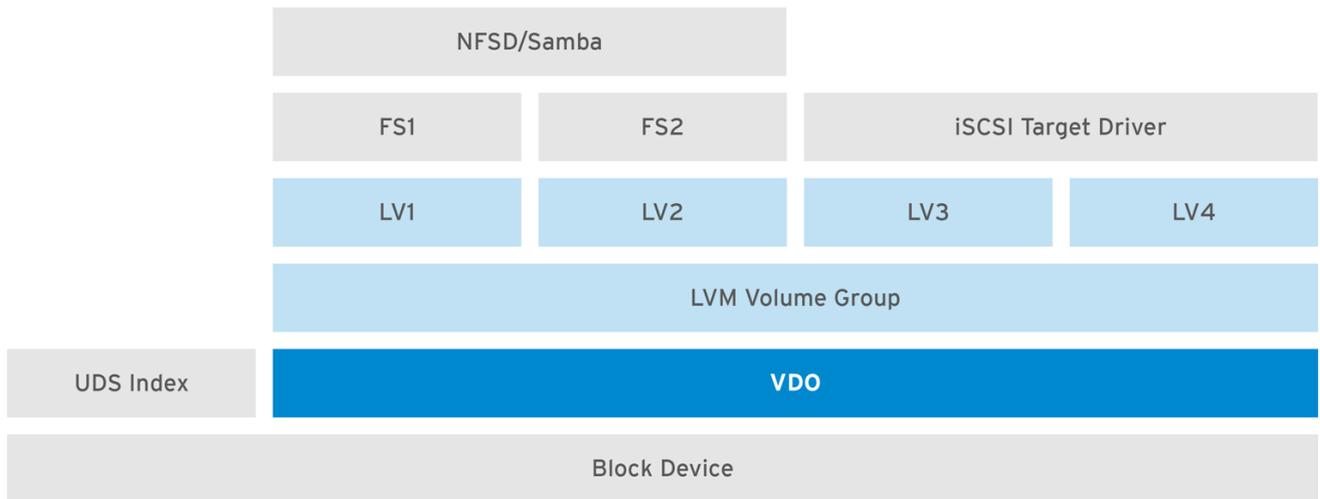
Figure 29.5. Deduplicated Unified Storage

Deduplicated unified storage design allows for multiple file systems to collectively use the same deduplication domain through the LVM tools. Also, file systems can take advantage of LVM snapshot, copy-on-write, and shrink or grow features, all on top of VDO.

29.5.4. Encryption

Data security is critical today. More and more companies have internal policies regarding data encryption. Linux Device Mapper mechanisms such as DM-Crypt are compatible with VDO. Encrypting VDO volumes will help ensure data security, and any file systems above VDO still gain the deduplication feature for disk optimization. Note that applying encryption above VDO results in little if any data deduplication; encryption renders duplicate blocks different before VDO can deduplicate them.
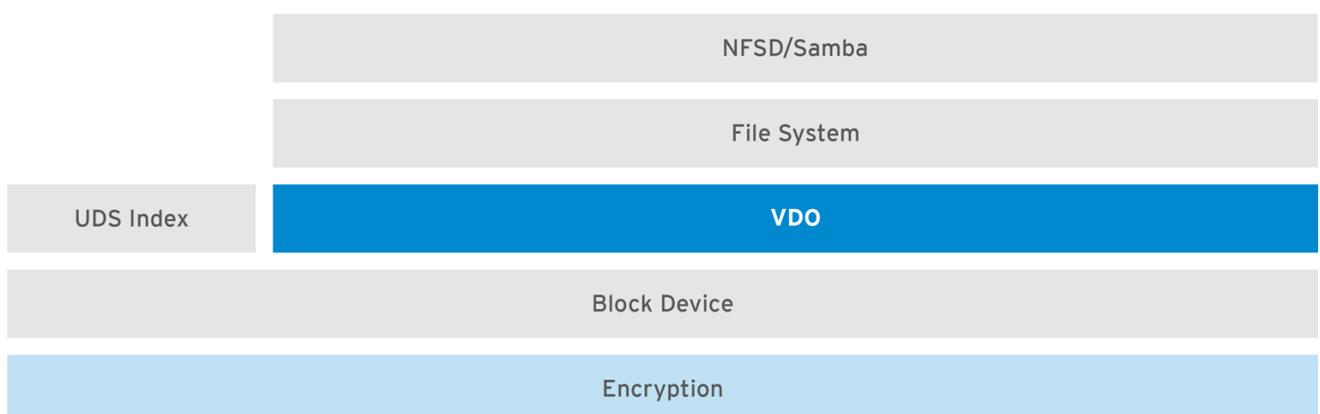
Figure 29.6. Using VDO with Encryption

29.6. TUNING VDO

29.6.1. Introduction to VDO Tuning

As with tuning databases or other complex software, tuning VDO involves making trade-offs between numerous system constraints, and some experimentation is required. The primary controls available for

tuning VDO are the number of threads assigned to different types of work, the CPU affinity settings for those threads, and cache settings.

29.6.2. Background on VDO Architecture

The VDO kernel driver is multi-threaded to improve performance by amortizing processing costs across multiple concurrent I/O requests. Rather than have one thread process an I/O request from start to finish, it delegates different stages of work to one or more threads or groups of threads, with messages passed between them as the I/O request makes its way through the pipeline. This way, one thread can serialize all access to a global data structure without having to lock and unlock it each time an I/O operation is processed. If the VDO driver is well-tuned, each time a thread completes a requested processing stage there will usually be another request queued up for that same processing. Keeping these threads busy reduces the overhead of context switching and scheduling, improving performance. Separate threads are also used for parts of the operating system that can block, such as enqueueing I/O operations to the underlying storage system or messages to UDS.

The various worker thread types used by VDO are:

Logical zone threads

The *logical* threads, with process names including the string **kvdo:logQ**, maintain the mapping between the logical block numbers (LBNs) presented to the user of the VDO device and the physical block numbers (PBNs) in the underlying storage system. They also implement locking such that two I/O operations attempting to write to the same block will not be processed concurrently. Logical zone threads are active during both read and write operations.

LBNs are divided into chunks (a *block map page* contains a bit over 3 MB of LBNs) and these chunks are grouped into *zones* that are divided up among the threads.

Processing should be distributed fairly evenly across the threads, though some unlucky access patterns may occasionally concentrate work in one thread or another. For example, frequent access to LBNs within a given block map page will cause one of the logical threads to process all of those operations.

The number of logical zone threads can be controlled using the **--vdoLogicalThreads=*thread count*** option of the **vdo** command

Physical zone threads

Physical, or **kvdo:physQ**, threads manage data block allocation and maintain reference counts. They are active during write operations.

Like LBNs, PBNs are divided into chunks called *slabs*, which are further divided into zones and assigned to worker threads that distribute the processing load.

The number of physical zone threads can be controlled using the **--vdoPhysicalThreads=*thread count*** option of the **vdo** command.

I/O submission threads

kvdo:bioQ threads submit block I/O (*bio*) operations from VDO to the storage system. They take I/O requests enqueued by other VDO threads and pass them to the underlying device driver. These threads may communicate with and update data structures associated with the device, or set up requests for the device driver's kernel threads to process. Submitting I/O requests can block if the underlying device's request queue is full, so this work is done by dedicated threads to avoid processing delays.

If these threads are frequently shown in **D** state by **ps** or **top** utilities, then VDO is frequently keeping the storage system busy with I/O requests. This is generally good if the storage system can service multiple requests in parallel, as some SSDs can, or if the request processing is pipelined. If thread CPU utilization is very low during these periods, it may be possible to reduce the number of I/O submission threads.

CPU usage and memory contention are dependent on the device driver(s) beneath VDO. If CPU utilization per I/O request increases as more threads are added then check for CPU, memory, or lock contention in those device drivers.

The number of I/O submission threads can be controlled using the **--vdoBioThreads=*thread count*** option of the **vdo** command.

CPU-processing threads

kvdo:cpuQ threads exist to perform any CPU-intensive work such as computing hash values or compressing data blocks that do not block or require exclusive access to data structures associated with other thread types.

The number of CPU-processing threads can be controlled using the **--vdoCpuThreads=*thread count*** option of the **vdo** command.

I/O acknowledgement threads

The **kvdo:ackQ** threads issue the callbacks to whatever sits atop VDO (for example, the kernel page cache, or application program threads doing direct I/O) to report completion of an I/O request. CPU time requirements and memory contention will be dependent on this other kernel-level code.

The number of acknowledgement threads can be controlled using the **--vdoAckThreads=*thread count*** option of the **vdo** command.

Non-scalable VDO kernel threads:

Deduplication thread

The **kvdo:dedupeQ** thread takes queued I/O requests and contacts UDS. Since the socket buffer can fill up if the server cannot process requests quickly enough or if kernel memory is constrained by other system activity, this work is done by a separate thread so if a thread should block, other VDO processing can continue. There is also a timeout mechanism in place to skip an I/O request after a long delay (several seconds).

Journal thread

The **kvdo:journalQ** thread updates the recovery journal and schedules journal blocks for writing. A VDO device uses only one journal, so this work cannot be split across threads.

Packer thread

The **kvdo:packerQ** thread, active in the write path when compression is enabled, collects data blocks compressed by the **kvdo:cpuQ** threads to minimize wasted space. There is one packer data structure, and thus one packer thread, per VDO device.

29.6.3. Values to tune

29.6.3.1. CPU/memory

29.6.3.1.1. Logical, physical, cpu, ack thread counts

The logical, physical, cpu, and I/O acknowledgement work can be spread across multiple threads, the number of which can be specified during initial configuration or later if the VDO device is restarted.

One core, or one thread, can do a finite amount of work during a given time. Having one thread compute all data-block hash values, for example, would impose a hard limit on the number of data blocks that could be processed per second. Dividing the work across multiple threads (and cores) relieves that bottleneck.

As a thread or core approaches 100% usage, more work items will tend to queue up for processing. While this may result in CPU having fewer idle cycles, queueing delays and latency for individual I/O requests will typically increase. According to some queueing theory models, utilization levels above 70% or 80% can lead to excessive delays that can be several times longer than the normal processing time. Thus it may be helpful to distribute work further for a thread or core with 50% or higher utilization, even if those threads or cores are not always busy.

In the opposite case, where a thread or CPU is very lightly loaded (and thus very often asleep), supplying work for it to do is more likely to incur some additional cost. (A thread attempting to wake another thread must acquire a global lock on the scheduler's data structures, and may potentially send an inter-processor interrupt to transfer work to another core). As more cores are configured to run VDO threads, it becomes less likely that a given piece of data will be cached as work is moved between threads or as threads are moved between cores — so too much work distribution can also degrade performance.

The work performed by the logical, physical, and CPU threads per I/O request will vary based on the type of workload, so systems should be tested with the different types of workloads they are expected to service.

Write operations in sync mode involving successful deduplication will entail extra I/O operations (reading the previously stored data block), some CPU cycles (comparing the new data block to confirm that they match), and journal updates (remapping the LBN to the previously-stored data block's PBN) compared to writes of new data. When duplication is detected in async mode, data write operations are avoided at the cost of the read and compare operations described above; only one journal update can happen per write, whether or not duplication is detected.

If compression is enabled, reads and writes of compressible data will require more processing by the CPU threads.

Blocks containing all zero bytes (a *zero block*) are treated specially, as they commonly occur. A special entry is used to represent such data in the block map, and the zero block is not written to or read from the storage device. Thus, tests that write or read all-zero blocks may produce misleading results. The same is true, to a lesser degree, of tests that write over zero blocks or uninitialized blocks (those that were never written since the VDO device was created) because reference count updates done by the physical threads are not required for zero or uninitialized blocks.

Acknowledging I/O operations is the only task that is not significantly affected by the type of work being done or the data being operated upon, as one callback is issued per I/O operation.

29.6.3.1.2. CPU Affinity and NUMA

Accessing memory across NUMA node boundaries takes longer than accessing memory on the local node. With Intel processors sharing the last-level cache between cores on a node, cache contention between nodes is a much greater problem than cache contention within a node.

Tools such as `top` can not distinguish between CPU cycles that do work and cycles that are stalled. These tools interpret cache contention and slow memory accesses as actual work. As a result, moving a thread between nodes may appear to reduce the thread's apparent CPU utilization while increasing the

number of operations it performs per second.

While many of VDO's kernel threads maintain data structures that are accessed by only one thread, they do frequently exchange messages about the I/O requests themselves. Contention may be high if VDO threads are run on multiple nodes, or if threads are reassigned from one node to another by the scheduler. If it is possible to run other VDO-related work (such as I/O submissions to VDO, or interrupt processing for the storage device) on the same node as the VDO threads, contention may be further reduced. If one node does not have sufficient cycles to run all VDO-related work, memory contention should be considered when selecting threads to move onto other nodes.

If practical, collect VDO threads on one node using the **taskset** utility. If other VDO-related work can also be run on the same node, that may further reduce contention. In that case, if one node lacks the CPU power to keep up with processing demands then memory contention must be considered when choosing threads to move onto other nodes. For example, if a storage device's driver has a significant number of data structures to maintain, it may help to move both the device's interrupt handling and VDO's I/O submissions (the bio threads that call the device's driver code) to another node. Keeping I/O acknowledgment (ack threads) and higher-level I/O submission threads (user-mode threads doing direct I/O, or the kernel's page cache **flush** thread) paired is also good practice.

29.6.3.1.3. Frequency throttling

If power consumption is not an issue, writing the string **performance** to the **/sys/devices/system/cpu/cpu*/cpufreq/scaling_governor** files if they exist might produce better results. If these **sysfs** nodes do not exist, Linux or the system's BIOS may provide other options for configuring CPU frequency management.

Performance measurements are further complicated by CPUs that dynamically vary their frequencies based on workload, because the time needed to accomplish a specific piece of work may vary due to other work the CPU has been doing, even without task switching or cache contention.

29.6.3.2. Caching

29.6.3.2.1. Block Map Cache

VDO caches a number of block map pages for efficiency. The cache size defaults to 128 MB, but it can be increased with the **--blockMapCacheSize=*megabytes*** option of the **vdo** command. Using a larger cache may produce significant benefits for random-access workloads.

29.6.3.2.2. Read Cache

A second cache may be used for caching data blocks read from the storage system to verify VDO's deduplication advice. If similar data blocks are seen within a short time span, the number of I/O operations needed may be reduced.

The read cache also holds storage blocks containing compressed user data. If multiple compressible blocks were written within a short period of time, their compressed versions may be located within the same storage system block. Likewise, if they are read within a short time, caching may avoid the need for additional reads from the storage system.

The **vdo** command's **--readCache={enabled | disabled}** option controls whether a read cache is used. If enabled, the cache has a minimum size of 8 MB, but it can be increased with the **--readCacheSize=*megabytes*** option. Managing the read cache incurs a slight overhead, so it may not increase performance if the storage system is fast enough. The read cache is disabled by default.

29.6.3.3. Storage System I/O

29.6.3.3.1. Bio Threads

For generic hard drives in a RAID configuration, one or two bio threads may be sufficient for submitting I/O operations. If the storage device driver requires its I/O submission threads to do significantly more work (updating driver data structures or communicating with the device) such that one or two threads are very busy and storage devices are often idle, the bio thread count can be increased to compensate. However, depending on the driver implementation, raising the thread count too high may lead to cache or spin lock contention. If device access timing is not uniform across all NUMA nodes, it may be helpful to run bio threads on the node "closest" to the storage device controllers.

29.6.3.3.2. IRQ Handling

If a device driver does significant work in its interrupt handler and does not use a threaded IRQ handler, it may prevent the scheduler from providing the best performance. CPU time spent servicing hardware interrupts may look like normal VDO (or other) kernel thread execution in some ways. For example, if hardware IRQ handling required 30% of a core's cycles, a busy kernel thread on the same core could only use the remaining 70%. However, if the work queued up for that thread demanded 80% of the core's cycles, the thread would never catch up, and the scheduler might simply leave that thread to run impeded on that core instead of switching that thread to a less busy core.

Using such a device driver under a heavy VDO workload may require a large number of cycles to service hardware interrupts (the **%hi** indicator in the header of the **top** display). In that case it may help to assign IRQ handling to certain cores and adjust the CPU affinity of VDO kernel threads not to run on those cores.

29.6.3.4. Maximum Discard Sectors

The maximum allowed size of DISCARD (TRIM) operations to a VDO device can be tuned via **/sys/kvdo/max_discard_sectors**, based on system usage. The default is 8 sectors (that is, one 4 KB block). Larger sizes may be specified, though VDO will still process them in a loop, one block at a time, ensuring that metadata updates for one discarded block are written to the journal and flushed to disk before starting on the next block.

When using a VDO volume as a local file system, Red Hat testing found that a small discard size works best, as the generic block-device code in the Linux kernel will break large discard requests into multiple smaller ones and submit them in parallel. If there is low I/O activity on the device, VDO can process many smaller requests concurrently and much more quickly than one large request.

If the VDO device is to be used as a SCSI target, the initiator and target software introduce additional factors to consider. If the target SCSI software is SCST, it reads the maximum discard size and relays it to the initiator. (Red Hat has not attempted to tune VDO configurations in conjunction with LIO SCSI target code.)

Because the Linux SCSI initiator code allows only one discard operation at a time, discard requests that exceed the maximum size would be broken into multiple smaller discards and sent, one at a time, to the target system (and to VDO). So, in addition to VDO processing a number of small discard operations in serial, the round-trip communication time between the two systems adds additional latency.

Setting a larger maximum discard size can reduce this communication overhead, though that larger request is passed in its entirety to VDO and processed one 4 KB block at a time. While there is no per-block communication delay, additional processing time for the larger block may cause the SCSI initiator software to time out.

For SCSI target usage, Red Hat recommends configuring the maximum discard size to be moderately

large while still keeping the typical discard time well within the initiator's timeout setting. An extra round-trip cost every few seconds, for example, should not significantly affect performance and SCSI initiators with timeouts of 30 or 60 seconds should not time out.

29.6.4. Identifying Bottlenecks

There are several key factors that affect VDO performance, and many tools available to identify those having the most impact.

Thread or CPU utilization above 70%, as seen in utilities such as **top** or **ps**, generally implies that too much work is being concentrated in one thread or on one CPU. However, in some cases it could mean that a VDO thread was scheduled to run on the CPU but no work actually happened; this scenario could occur with excessive hardware interrupt handler processing, memory contention between cores or NUMA nodes, or contention for a spin lock.

When using the **top** utility to examine system performance, Red Hat suggests running **top -H** to show all process threads separately and then entering the **1 f j** keys, followed by the Enter/Return key; the **top** command then displays the load on individual CPU cores and identifies the CPU on which each process or thread last ran. This information can provide the following insights:

- If a core has low **%id** (idle) and **%wa** (waiting-for-I/O) values, it is being kept busy with work of some kind.

- If the **%hi** value for a core is very low, that core is doing normal processing work, which is being load-balanced by the kernel scheduler. Adding more cores to that set may reduce the load as long as it does not introduce NUMA contention.

- If the **%hi** for a core is more than a few percent and only one thread is assigned to that core, and **%id** and **%wa** are zero, the core is over-committed and the scheduler is not addressing the situation. In this case the kernel thread or the device interrupt handling should be reassigned to keep them on separate cores.

The **perf** utility can examine the performance counters of many CPUs. Red Hat suggests using the **perf top** subcommand as a starting point to examine the work a thread or processor is doing. If, for example, the **bioQ** threads are spending many cycles trying to acquire spin locks, there may be too much contention in the device driver below VDO, and reducing the number of **bioQ** threads might alleviate the situation. High CPU use (in acquiring spin locks or elsewhere) could also indicate contention between NUMA nodes if, for example, the **bioQ** threads and the device interrupt handler are running on different nodes. If the processor supports them, counters such as **stalled-cycles-backend**, **cache-misses**, and **node-load-misses** may be of interest.

The **sar** utility can provide periodic reports on multiple system statistics. The **sar -d 1** command reports block device utilization levels (percentage of the time they have at least one I/O operation in progress) and queue lengths (number of I/O requests waiting) once per second. However, not all block device drivers can report such information, so the **sar** usefulness might depend on the device drivers in use.

29.7. VDO COMMANDS

This section describes the following VDO utilities:

vdo

The **vdo** utility manages both the **kvdo** and UDS components of VDO.

It is also used to enable or disable compression.

vdostats

The **vdostats** utility displays statistics for each configured (or specified) device in a format similar to the Linux **df** utility.

29.7.1. vdo

The **vdo** utility manages both the **kvdo** and UDS components of VDO.

Synopsis

```
vdo { activate | changeWritePolicy | create | deactivate |
disableCompression | disableDeduplication | enableCompression |
enableDeduplication | growLogical | growPhysical | list | modify |
printConfigFile | remove | start | status | stop }
[ options... ]
```

Sub-Commands

Table 29.4. VDO Sub-Commands

Sub-Command	Description

Sub-Command	Description
create	Creates a VDO volume and its associated index and makes it available. If **--activate=disabled** is specified the VDO volume is created but not made available. Will not overwrite an existing file system or formatted VDO volume unless **--force** is given. This command must be run with root privileges. Applicable options include: • **--name=*volume*** (required) • **--device=*device*** (required) • **--activate={enabled | disabled}** • **--indexMem=*gigabytes*** • **--blockMapCacheSize=*megabytes*** • **--blockMapPeriod=*period*** • **--compression={enabled | disabled}** • **--confFile=*file*** • **--deduplication={enabled | disabled}** • **--emulate512={enabled | disabled}** • **--sparseIndex={enabled | disabled}** • **--vdoAckThreads=*thread count*** • **--vdoBioRotationInterval=*I/O count*** • **--vdoBioThreads=*thread count*** • **--vdoCpuThreads=*thread count*** • **--vdoHashZoneThreads=*thread count*** • **--vdoLogicalThreads=*thread count*** • **--vdoLogLevel=*level*** • **--vdoLogicalSize=*megabytes*** • **--vdoPhysicalThreads=*thread count*** • **--readCache={enabled | disabled}** • **--readCacheSize=*megabytes*** • **--vdoSlabSize=*megabytes*** • **--verbose** • **--writePolicy={ auto | sync | async }** • **--logfile=pathname**
create	

Sub-Command	Description
remove	Removes one or more stopped VDO volumes and associated indexes. This command must be run with root privileges. Applicable options include: • { **--name=**_volume_ | **--all** } (required) • **--confFile=**_file_ • **--force** • **--verbose** • **--logfile=pathname**
start	Starts one or more stopped, activated VDO volumes and associated services. This command must be run with root privileges. Applicable options include: • { **--name=**_volume_ | **--all** } (required) • **--confFile=**_file_ • **--forceRebuild** • **--verbose** • **--logfile=pathname**
stop	Stops one or more running VDO volumes and associated services. This command must be run with root privileges. Applicable options include: • { **--name=**_volume_ | **--all** } (required) • **--confFile=**_file_ • **--force** • **--verbose** • **--logfile=pathname**
activate	Activates one or more VDO volumes. Activated volumes can be started using the **start** command. This command must be run with root privileges. Applicable options include: • { **--name=**_volume_ | **--all** } (required) • **--confFile=**_file_ • **--logfile=pathname** • **--verbose**

Sub-Command	Description
deactivate	Deactivates one or more VDO volumes. Deactivated volumes cannot be started by the **start** command. Deactivating a currently running volume does not stop it. Once stopped a deactivated VDO volume must be activated before it can be started again. This command must be run with root privileges. Applicable options include: • { **--name=***volume* | **--all** } (required) • **--confFile=***file* • **--verbose** • **--logfile=pathname**
status	Reports VDO system and volume status in YAML format. This command does not require root privileges though information will be incomplete if run without. Applicable options include: • { **--name=***volume* | **--all** } (required) • **--confFile=***file* • **--verbose** • **--logfile=pathname** See Table 29.6, "VDO Status Output" for the output provided.
list	Displays a list of started VDO volumes. If **--all** is specified it displays both started and non-started volumes. This command must be run with root privileges. Applicable options include: • **--all** • **--confFile=***file* • **--logfile=pathname** • **--verbose**

Sub-Command	Description
modify	Modifies configuration parameters of one or all VDO volumes. Changes take effect the next time the VDO device is started; already-running devices are not affected. Applicable options include: • { **--name=***volume* | **--all** } (required) • **--blockMapCacheSize=***megabytes* • **--blockMapPeriod=***period* • **--confFile=***file* • **--vdoAckThreads=***thread count* • **--vdoBioThreads=***thread count* • **--vdoCpuThreads=***thread count* • **--vdoHashZoneThreads=***thread count* • **--vdoLogicalThreads=***thread count* • **--vdoPhysicalThreads=***thread count* • **--readCache={enabled | disabled}** • **--readCacheSize=***megabytes* • **--verbose** • **--logfile=pathname**
changeWritePolicy	Modifies the write policy of one or all running VDO volumes. This command must be run with root privileges. • { **--name=***volume* | **--all** } (required) • **--writePolicy={ auto | sync | async }** (required) • **--confFile=***file* • **--logfile=pathname** • **--verbose**

Sub-Command	Description
enableDeduplication	Enables deduplication on one or more VDO volumes. This command must be run with root privileges. Applicable options include: • { **--name=*volume*** | **--all** } (required) • **--confFile=*file*** • **--verbose** • **--logfile=pathname**
disableDeduplication	Disables deduplication on one or more VDO volumes. This command must be run with root privileges. Applicable options include: • { **--name=*volume*** | **--all** } (required) • **--confFile=*file*** • **--verbose** • **--logfile=pathname**
enableCompression	Enables compression on one or more VDO volumes. If the VDO volume is running, takes effect immediately. If the VDO volume is not running compression will be enabled the next time the VDO volume is started. This command must be run with root privileges. Applicable options include: • { **--name=*volume*** | **--all** } (required) • **--confFile=*file*** • **--verbose** • **--logfile=pathname**
disableCompression	Disables compression on one or more VDO volumes. If the VDO volume is running, takes effect immediately. If the VDO volume is not running compression will be disabled the next time the VDO volume is started. This command must be run with root privileges. Applicable options include: • { **--name=*volume*** | **--all** } (required) • **--confFile=*file*** • **--verbose** • **--logfile=pathname**

Sub-Command	Description
growLogical	Adds logical space to a VDO volume. The volume must exist and must be running. This command must be run with root privileges. Applicable options include: • **--name=*volume*** (required) • **--vdoLogicalSize=*megabytes*** (required) • **--confFile=*file*** • **--verbose** • **--logfile=pathname**
growPhysical	Adds physical space to a VDO volume. The volume must exist and must be running. This command must be run with root privileges. Applicable options include: • **--name=*volume*** (required) • **--confFile=*file*** • **--verbose** • **--logfile=pathname**
printConfigFile	Prints the configuration file to **stdout**. This command require root privileges. Applicable options include: • **--confFile=*file*** • **--logfile=pathname** • **--verbose**

Options

Table 29.5. VDO Options

Option	Description
--indexMem=*gigabytes*	Specifies the amount of UDS server memory in gigabytes; the default size is 1 GB. The special decimal values 0.25, 0.5, 0.75 can be used, as can any positive integer.
--sparseIndex={enabled | disabled}	Enables or disables sparse indexing. The default is **disabled**.
--all	Indicates that the command should be applied to all configured VDO volumes. May not be used with **--name**.

Option	Description
`--blockMapCacheSize=`*`megabytes`*	Specifies the amount of memory allocated for caching block map pages; the value must be a multiple of 4096. Using a value with a **B**(ytes), **K**(ilobytes), **M**(egabytes), **G**(igabytes), **T**(erabytes), **P**(etabytes) or **E**(xabytes) suffix is optional. If no suffix is supplied, the value will be interpreted as megabytes. The default is 128M; the value must be at least 128M and less than 16T. Note that there is a memory overhead of 15%.
`--blockMapPeriod=`*`period`*	A value between 1 and 16380 which determines the number of block map updates which may accumulate before cached pages are flushed to disk. Higher values decrease recovery time after a crash at the expense of decreased performance during normal operation. The default value is 16380. Speak with your Red Hat representative before tuning this parameter.
`--compression={enabled	disabled}`	Enables or disables compression within the VDO device. The default is enabled. Compression may be disabled if necessary to maximize performance or to speed processing of data that is unlikely to compress.
`--confFile=`*`file`*	Specifies an alternate configuration file. The default is `/etc/vdoconf.yml`.
`--deduplication={enabled	disabled}`	Enables or disables deduplication within the VDO device. The default is **enabled**. Deduplication may be disabled in instances where data is not expected to have good deduplication rates but compression is still desired.
`--emulate512={enabled	disabled}`	Enables 512-byte block device emulation mode. The default is **disabled**.
`--force`	Unmounts mounted file systems before stopping a VDO volume.
`--forceRebuild`	Forces an offline rebuild before starting a read-only VDO volume so that it may be brought back online and made available. **This option may result in data loss or corruption.**
`--help`	Displays documentation for the **vdo** utility.
`--logfile=pathname`	Specify the file to which this script's log messages are directed. Warning and error messages are always logged to syslog as well.
`--name=`*`volume`*	Operates on the specified VDO volume. May not be used with `--all`.
`--device=`*`device`*	Specifies the absolute path of the device to use for VDO storage.
`--activate={enabled	disabled}`	The argument **disabled** indicates that the VDO volume should only be created. The volume will not be started or enabled. The default is **enabled**.

Option	Description
`--vdoAckThreads=`*`thread count`*	Specifies the number of threads to use for acknowledging completion of requested VDO I/O operations. The default is 1; the value must be at least 0 and less than or equal to 100.
`--vdoBioRotationInterval=`*`I/O count`*	Specifies the number of I/O operations to enqueue for each bio-submission thread before directing work to the next. The default is 64; the value must be at least 1 and less than or equal to 1024.
`--vdoBioThreads=`*`thread count`*	Specifies the number of threads to use for submitting I/O operations to the storage device. Minimum is 1; maximum is 100. The default is 4; the value must be at least 1 and less than or equal to 100.
`--vdoCpuThreads=`*`thread count`*	Specifies the number of threads to use for CPU- intensive work such as hashing or compression. The default is 2; the value must be at least 1 and less than or equal to 100.
`--vdoHashZoneThreads=`*`thread count`*	Specifies the number of threads across which to subdivide parts of the VDO processing based on the hash value computed from the block data. The default is **1**; the value must be at least 0 and less than or equal to 100. **vdoHashZoneThreads**, **vdoLogicalThreads** and **vdoPhysicalThreads** must be either all zero or all non-zero.
`--vdoLogicalThreads=`*`thread count`*	Specifies the number of threads across which to subdivide parts of the VDO processing based on the hash value computed from the block data. The value must be at least 0 and less than or equal to 100. A logical thread count of 9 or more will require explicitly specifying a sufficiently large block map cache size, as well. **vdoHashZoneThreads**, **vdoLogicalThreads**, and **vdoPhysicalThreads** must be either all zero or all non-zero. The default is 1.
`--vdoLogLevel=`*`level`*	Specifies the VDO driver log level: **critical**, **error**, **warning**, **notice**, **info**, or **debug**. Levels are case sensitive; the default is **info**.
`--vdoLogicalSize=`*`megabytes`*	Specifies the logical VDO volume size in megabytes. Using a value with a **S**(ectors), **B**(ytes), **K**(ilobytes), **M**(egabytes), **G**(igabytes), **T**(erabytes), **P**(etabytes) or **E**(xabytes) suffix is optional. Used for over- provisioning volumes. This defaults to the size of the storage device.
`--vdoPhysicalThreads=`*`thread count`*	Specifies the number of threads across which to subdivide parts of the VDO processing based on physical block addresses. The value must be at least 0 and less than or equal to 16. Each additional thread after the first will use an additional 10 MB of RAM. **vdoPhysicalThreads**, **vdoHashZoneThreads**, and **vdoLogicalThreads** must be either all zero or all non-zero. The default is 1.

Option	Description
`--readCache={enabled | disabled}`	Enables or disables the read cache within the VDO device. The default is **disabled**. The cache should be enabled if write workloads are expected to have high levels of deduplication, or for read intensive workloads of highly compressible data.
`--readCacheSize=megabytes`	Specifies the extra VDO device read cache size in megabytes. This space is in addition to a system- defined minimum. Using a value with a **B**(ytes), **K**(ilobytes), **M**(egabytes), **G**(igabytes), **T**(erabytes), **P**(etabytes) or **E**(xabytes) suffix is optional. The default is 0M. 1.12 MB of memory will be used per MB of read cache specified, per bio thread.
`--vdoSlabSize=megabytes`	Specifies the size of the increment by which a VDO is grown. Using a smaller size constrains the total maximum physical size that can be accommodated. Must be a power of two between 128M and 32G; the default is 2G. Using a value with a **S**(ectors), **B**(ytes), **K**(ilobytes), **M**(egabytes), **G**(igabytes), **T**(erabytes), **P**(etabytes) or **E**(xabytes) suffix is optional. If no suffix is used, the value will be interpreted as megabytes.
`--verbose`	Prints commands before executing them.
`--writePolicy={ auto | sync | async }`	Specifies the write policy: • **auto**: Select **sync** or **async** based on the storage layer underneath VDO. If a write back cache is present, **async** will be chosen. Otherwise, **sync** will be chosen. • **sync**: Writes are acknowledged only after data is stably written. This is the default policy. **This policy is not supported if the underlying storage is not also synchronous.** • **async**: Writes are acknowledged after data has been cached for writing to stable storage. Data which has not been flushed is not guaranteed to persist in this mode.

The `status` subcommand returns the following information in YAML format, divided into keys as follows:

Table 29.6. VDO Status Output

Key		Description
VDO Status		Information in this key covers the name of the host and date and time at which the status inquiry is being made. Parameters reported in this area include:
	Node	The host name of the system on which VDO is running.
	Date	The date and time at which the vdo status command is run.

Key	Description	
Kernel Module	Information in this key covers the configured kernel.	
	Loaded	Whether or not the kernel module is loaded (True or False).
	Version Information	Information on the version of kvdo that is configured.
Configuration	Information in this key covers the location and status of the VDO configuration file.	
	File	Location of the VDO configuration file.
	Last modified	The last-modified date of the VDO configuration file.
VDOs	Provides configuration information for all VDO volumes. Parameters reported for each VDO volume include:	
	Block size	The block size of the VDO volume, in bytes.
	512 byte emulation	Indicates whether the volume is running in 512-byte emulation mode.
	Enable deduplication	Whether deduplication is enabled for the volume.
	Logical size	The logical size of the VDO volume.
	Physical size	The size of a VDO volume's underlying physical storage.
	Write policy	The configured value of the write policy (sync or async).
	VDO Statistics	Output of the vdostats utility.

29.7.2. vdostats

The **vdostats** utility displays statistics for each configured (or specified) device in a format similar to the Linux **df** utility.

The output of the **vdostats** utility may be incomplete if it is not run with root privileges.

Synopsis

```
vdostats [ --verbose | --human-readable | --si | --all ] [ --version ] [
device ...]
```

■

Options

Table 29.7. `vdostats` Options

Option	Description
--verbose	Displays the utilization and block I/O (bios) statistics for one (or more) VDO devices. See Table 29.9, "vdostats --verbose Output" for details.
--human-readable	Displays block values in readable form (Base 2: 1 KB = 2^{10} bytes = 1024 bytes).
--si	The **--si** option modifies the output of the **--human-readable** option to use SI units (Base 10: 1 KB = 10^3 bytes = 1000 bytes). If the **--human-readable** option is not supplied, the **--si** option has no effect.
--all	This option is only for backwards compatibility. It is now equivalent to the **--verbose** option.
--version	Displays the **vdostats** version.
device ...	Specifies one or more specific volumes to report on. If this argument is omitted, **vdostats** will report on all devices.

Output

The following example shows sample output if no options are provided, which is described in Table 29.8, "Default vdostats Output":

```
Device                1K-blocks    Used          Available      Use%
Space Saving%
/dev/mapper/my_vdo    1932562432   427698104     1504864328     22%    21%
```

Table 29.8. Default vdostats Output

Item	Description
Device	The path to the VDO volume.
1K-blocks	The total number of 1K blocks allocated for a VDO volume (= physical volume size * block size / 1024)
Used	The total number of 1K blocks used on a VDO volume (= physical blocks used * block size / 1024)
Available	The total number of 1K blocks available on a VDO volume (= physical blocks free * block size / 1024)

Item	Description
Use%	The percentage of physical blocks used on a VDO volume (= used blocks / allocated blocks * 100)
Space Saving%	The percentage of physical blocks saved on a VDO volume (= [logical blocks used - physical blocks used] / logical blocks used)

The **--human-readable** option converts block counts into conventional units (1 KB = 1024 bytes):

```
Device                Size    Used      Available     Use%    Space Saving%
/dev/mapper/my_vdo    1.8T    407.9G    1.4T          22%     21%
```

The **--human-readable** and **--si** options convert block counts into SI units (1 KB = 1000 bytes):

```
Device                Size    Used      Available     Use%    Space
Saving%
/dev/mapper/my_vdo    2.0T    438G      1.5T          22%     21%
```

The **--verbose** (Table 29.9, "vdostats --verbose Output") option displays VDO device statistics in YAML format for one (or all) VDO devices.

Statistics printed in **bold** in Table 29.9, "vdostats --verbose Output" will continue to be reported in future releases. The remaining fields are primarily intended for software support and are subject to change in future releases; management tools should not rely upon them. Management tools should also not rely upon the order in which any of the statistics are reported.

Table 29.9. vdostats --verbose Output

Item	Description
Version	The version of these statistics.
Release version	The release version of the VDO.
Data blocks used	The number of physical blocks currently in use by a VDO volume to store data.
Overhead blocks used	The number of physical blocks currently in use by a VDO volume to store VDO metadata.
Logical blocks used	The number of logical blocks currently mapped.
Physical blocks	The total number of physical blocks allocated for a VDO volume.
Logical blocks	The maximum number of logical blocks that can be mapped by a VDO volume.
1K-blocks	The total number of 1K blocks allocated for a VDO volume (= physical volume size * block size / 1024)

Item	Description
1K-blocks used	The total number of 1K blocks used on a VDO volume (= physical blocks used * block size / 1024)
1K-blocks available	The total number of 1K blocks available on a VDO volume (= physical blocks free * block size / 1024)
Used percent	The percentage of physical blocks used on a VDO volume (= used blocks / allocated blocks * 100)
Saving percent	The percentage of physical blocks saved on a VDO volume (= [logical blocks used - physical blocks used] / logical blocks used)
Block map cache size	The size of the block map cache, in bytes.
Write policy	The active write policy (sync or async). This is configured via **vdo changeWritePolicy --writePolicy=auto|sync|async**.
Block size	The block size of a VDO volume, in bytes.
Completed recovery count	The number of times a VDO volume has recovered from an unclean shutdown.
Read-only recovery count	The number of times a VDO volume has been recovered from read-only mode (via **vdo start --forceRebuild**).
Operating mode	Indicates whether a VDO volume is operating normally, is in recovery mode, or is in read-only mode.
Recovery progress (%)	Indicates online recovery progress, or **N/A** if the volume is not in recovery mode.
Compressed fragments written	The number of compressed fragments that have been written since the VDO volume was last restarted.
Compressed blocks written	The number of physical blocks of compressed data that have been written since the VDO volume was last restarted.
Compressed fragments in packer	The number of compressed fragments being processed that have not yet been written.
Slab count	The total number of slabs.
Slabs opened	The total number of slabs from which blocks have ever been allocated.
Slabs reopened	The number of times slabs have been re-opened since the VDO was started.

Item	Description
Journal disk full count	The number of times a request could not make a recovery journal entry because the recovery journal was full.
Journal commits requested count	The number of times the recovery journal requested slab journal commits.
Journal entries batching	The number of journal entry writes started minus the number of journal entries written.
Journal entries started	The number of journal entries which have been made in memory.
Journal entries writing	The number of journal entries in submitted writes minus the number of journal entries committed to storage.
Journal entries written	The total number of journal entries for which a write has been issued.
Journal entries committed	The number of journal entries written to storage.
Journal blocks batching	The number of journal block writes started minus the number of journal blocks written.
Journal blocks started	The number of journal blocks which have been touched in memory.
Journal blocks writing	The number of journal blocks written (with metadatata in active memory) minus the number of journal blocks committed.
Journal entries written	The total number of journal blocks for which a write has been issued.
Journal blocks committed	The number of journal blocks written to storage.
Slab journal disk full count	The number of times an on-disk slab journal was full.
Slab journal flush count	The number of times an entry was added to a slab journal that was over the flush threshold.
Slab journal blocked count	The number of times an entry was added to a slab journal that was over the blocking threshold.
Slab journal blocks written	The number of slab journal block writes issued.
Slab journal tail busy count	The number of times write requests blocked waiting for a slab journal write.

Item	Description
Slab summary blocks written	The number of slab summary block writes issued.
Reference blocks written	The number of reference block writes issued.
Block map dirty pages	The number of dirty pages in the block map cache.
Block map clean pages	The number of clean pages in the block map cache.
Block map free pages	The number of free pages in the block map cache.
Block map failed pages	The number of block map cache pages that have write errors.
Block map incoming pages	The number of block map cache pages that are being read into the cache.
Block map outgoing pages	The number of block map cache pages that are being written.
Block map cache pressure	The number of times a free page was not available when needed.
Block map read count	The total number of block map page reads.
Block map write count	The total number of block map page writes.
Block map failed reads	The total number of block map read errors.
Block map failed writes	The total number of block map write errors.
Block map reclaimed	The total number of block map pages that were reclaimed.
Block map read outgoing	The total number of block map reads for pages that were being written.
Block map found in cache	The total number of block map cache hits.
Block map discard required	The total number of block map requests that required a page to be discarded.
Block map wait for page	The total number of requests that had to wait for a page.
Block map fetch required	The total number of requests that required a page fetch.
Block map pages loaded	The total number of page fetches.

Item	Description
Block map pages saved	The total number of page saves.
Block map flush count	The total number of flushes issued by the block map.
Invalid advice PBN count	The number of times the index returned invalid advice
No space error count.	The number of write requests which failed due to the VDO volume being out of space.
Read only error count	The number of write requests which failed due to the VDO volume being in read-only mode.
Instance	The VDO instance.
512 byte emulation	Indicates whether 512 byte emulation is on or off for the volume.
Current VDO IO requests in progress.	The number of I/O requests the VDO is current processing.
Maximum VDO IO requests in progress	The maximum number of simultaneous I/O requests the VDO has processed.
Current dedupe queries	The number of deduplication queries currently in flight.
Maximum dedupe queries	The maximum number of in-flight deduplication queries.
Dedupe advice valid	The number of times deduplication advice was correct.
Dedupe advice stale	The number of times deduplication advice was incorrect.
Dedupe advice timeouts	The number of times deduplication queries timed out.
Flush out	The number of flush requests submitted by VDO to the underlying storage.

Item	Description
Bios in... Bios in partial... Bios out... Bios meta... Bios journal... Bios page cache... Bios out completed... Bio meta completed... Bios journal completed... Bios page cache completed... Bios acknowledged... Bios acknowledged partial... Bios in progress...	These statistics count the number of bios in each category with a given flag. The categories are: • **bios in:** The number of block I/O requests received by VDO. • **bios in partial:** The number of partial block I/O requests received by VDO. Applies only to 512-byte emulation mode. • **bios out:** The number of non-metadata block I/O requests submitted by VDO to the storage device. • **bios meta:** The number of metadata block I/O requests submitted by VDO to the storage device. • **bios journal:** The number of recovery journal block I/O requests submitted by VDO to the storage device. • **bios page cache:** The number of block map I/O requests submitted by VDO to the storage device. • **bios out completed:** The number of non-metadata block I/O requests completed by the storage device. • **bios meta completed:** The number of metadata block I/O requests completed by the storage device. • **bios journal completed:** The number of recovery journal block I/O requests completed by the storage device. • **bios page cache completed:** The number of block map I/O requests completed by the storage device. • **bios acknowledged:** The number of block I/O requests acknowledged by VDO. • **bios acknowledged partial:** The number of partial block I/O requests acknowledged by VDO. Applies only to 512-byte emulation mode. • **bios in progress:** The number of bios submitted to the VDO which have not yet been acknowledged. There are three types of flags: • **read:** The number of non-write bios (bios without the REQ_WRITE flag set) • **write:** The number of write bios (bios with the REQ_WRITE flag set) • **discard:** The number of bios with a REQ_DISCARD flag set
Read cache accesses	The number of times VDO searched the read cache.
Read cache hits	The number of read cache hits.

29.8. STATISTICS FILES IN `/SYS`

Statistics for a running VDO volume may be read from files in the

/sys/kvdo/*volume_name*/statistics directory, where *volume_name* is the name of thhe VDO volume. This provides an alternate interface to the data produced by the **vdostats** utility suitable for access by shell scripts and management software.

There are files in the **statistics** directory in addition to the ones listed in the table below. These additional statistics files are not guaranteed to be supported in future releases.

Table 29.10. Statistics files

File	Description
dataBlocksUsed	The number of physical blocks currently in use by a VDO volume to store data.
logicalBlocksUsed	The number of logical blocks currently mapped.
physicalBlocks	The total number of physical blocks allocated for a VDO volume.
logicalBlocks	The maximum number of logical blocks that can be mapped by a VDO volume.
mode	Indicates whether a VDO volume is operating normally, is in recovery mode, or is in read-only mode.

CHAPTER 30. VDO EVALUATION

30.1. INTRODUCTION

VDO is software that provides inline block-level deduplication, compression, and thin provisioning capabilities for primary storage. VDO installs within the Linux device mapper framework, where it takes ownership of existing physical block devices and remaps these to new, higher-level block devices with data-efficiency properties. Specifically, VDO can multiply the effective capacity of these devices by ten or more. These benefits require additional system resources, so it is therefore necessary to measure VDO's impact on system performance.

Storage vendors undoubtedly have existing in-house test plans and expertise that they use to evaluate new storage products. Since the VDO layer helps to identify deduplication and compression, different tests may be required. An effective test plan requires studying the VDO architecture and exploring these items:

- VDO-specific configurable properties (performance tuning end-user applications)

- Impact of being a native 4 KB block device

- Response to access patterns and distributions of deduplication and compression

- Performance in high-load environments (very important)

- Analyze cost vs. capacity vs. performance, based on application

Failure to consider such factors up front has created situations that have invalidated certain tests and required customers to repeat testing and data collection efforts.

30.1.1. Expectations and Deliverables

This Evaluation Guide is meant to augment, not replace, a vendor's internal evaluation effort. With a modest investment of time, it will help evaluators produce an accurate assessment of VDO's integration into existing storage devices. This guide is designed to:

- Help engineers identify configuration settings that elicit optimal responses from the test device

- Provide an understanding of basic tuning parameters to help avoid product misconfigurations

- Create a performance results portfolio as a reference to compare against "real" application results

- Identify how different workloads affect performance and data efficiency

- Expedite time-to-market with VDO implementations

The test results will help Red Hat engineers assist in understanding VDO's behavior when integrated into specific storage environments. OEMs will understand how to design their deduplication and compression capable devices, and also how their customers can tune their applications to best use those devices.

Be aware that the procedures in this document are designed to provide conditions under which VDO can be most realistically evaluated. Altering test procedures or parameters may invalidate results. Red Hat Sales Engineers are available to offer guidance when modifying test plans.

30.2. TEST ENVIRONMENT PREPARATIONS

Before evaluating VDO, it is important to consider the host system configuration, VDO configuration, and the workloads that will be used during testing. These choices will affect benchmarking both in terms of data optimization (space efficiency) and performance (bandwidth and latency). Items that should be considered when developing test plans are listed in the following sections.

30.2.1. System Configuration

- Number and type of CPU cores available. This can be controlled by using the `taskset` utility.

- Available memory and total installed memory.

- Configuration of storage devices.

- Linux kernel version. Note that Red Hat Enterprise Linux 7 provides only one Linux kernel version.

- Packages installed.

30.2.2. VDO Configuration

- Partitioning scheme

- File system(s) used on VDO volumes

- Size of the physical storage assigned to a VDO volume

- Size of the logical VDO volume created

- Sparse or dense indexing

- UDS Index in memory size

- VDO's thread configuration

30.2.3. Workloads

- Types of tools used to generate test data

- Number of concurrent clients

- The quantity of duplicate 4 KB blocks in the written data

- Read and write patterns

- The working set size

VDO volumes may need to be re-created in between certain tests to ensure that each test is performed on the same disk environment. Read more about this in the testing section.

30.2.4. Supported System Configurations

Red Hat has tested VDO with Red Hat Enterprise Linux 7 on the Intel 64 architecture.

For the system requirements of VDO, see Section 29.2, "System Requirements".

The following utilities are recommended when evaluating VDO:

- **Flexible I/O Tester** version 2.08 or higher; available from the fio package

- `sysstat` version 8.1.2-2 or higher; available from the sysstat package

30.2.5. Pre-Test System Preparations

This section describes how to configure system settings to achieve optimal performance during the evaluation. Testing beyond the implicit bounds established in any particular test may result in loss of testing time due to abnormal results. For example, this guide describes a test that conducts random reads over a 100 GB address range. To test a working set of 500 GB, the amount of DRAM allocated for the VDO block map cache should be increased accordingly.

- System Configuration

  - Ensure that your CPU is running at its highest performance setting.

  - Disable frequency scaling if possible using the BIOS configuration or the Linux `cpupower` utility.

  - Enable Turbo mode if possible to achieve maximum throughput. Turbo mode introduces some variability in test results, but performance will meet or exceed that of testing without Turbo.

- Linux Configuration

  - For disk-based solutions, Linux offers several I/O scheduler algorithms to handle multiple read/write requests as they are queued. By default, Red Hat Enterprise Linux uses the CFQ (completely fair queuing) scheduler, which arranges requests in a way that improves rotational disk (hard disk) access in many situations. We instead suggest using the Deadline scheduler for rotational disks, having found that it provides better throughput and latency in Red Hat lab testing. Change the device settings as follows:

    ```
    # echo "deadline" > /sys/block/device/queue/scheduler
    ```

  - For flash-based solutions, the **noop** scheduler demonstrates superior random access throughput and latency in Red Hat lab testing. Change the device settings as follows:

    ```
    # echo "noop" > /sys/block/device/queue/scheduler
    ```

- Storage device configuration

  File systems (ext4, XFS, etc.) may have unique impacts on performance; they often skew performance measurements, making it harder to isolate VDO's impact on the results. If reasonable, we recommend measuring performance on the raw block device. If this is not possible, format the device using the file system that would be used in the target implementation.

30.2.6. VDO Internal Structures

We believe that a general understanding of VDO mechanisms is essential for a complete and successful evaluation. This understanding becomes especially important when testers wish to deviate from the test plan or devise new stimuli to emulate a particular application or use case. For more information, see Chapter 29, *VDO Integration*.

The Red Hat test plan was written to operate with a default VDO configuration. When developing new tests, some of the VDO parameters listed in the next section must be adjusted.

30.2.7. VDO Optimizations

High Load

Perhaps the most important strategy for producing optimal performance is determining the best I/O queue depth, a characteristic that represents the load on the storage system. Most modern storage systems perform optimally with high I/O depth. VDO's performance is best demonstrated with many concurrent requests.

Synchronous vs. Asynchronous Write Policy

VDO might operate with either of two write policies, synchronous or asynchronous. By default, VDO automatically chooses the appropriate write policy for your underlying storage device.

When testing performance, you need to know which write policy VDO selected. The following command shows the write policy of your VDO volume:

```
# vdo status --name=my_vdo
```

For more information on write policies, see the section called "Overview of VDO Write Policies" and Section 29.4.2, "Selecting VDO Write Modes".

Metadata Caching

VDO maintains a table of mappings from logical block addresses to physical block addresses, and VDO must look up the relevant mapping when accessing any particular block. By default, VDO allocates 128 MB of metadata cache in DRAM to support efficient access to 100 GB of logical space at a time. The test plan generates workloads appropriate to this configuration option.

Working sets larger than the configured cache size will require additional I/Os to service requests, in which case performance degradation will occur. If additional memory is available, the block map cache should be made larger. If the working set is larger than what the block map cache can hold in memory, additional I/O hover head can occur to lookup associated block map pages.

VDO Multithreading Configuration

VDO's thread configuration must be tuned to achieve optimal performance. Review the VDO Integration Guide for information on how to modify these settings when creating a VDO volume. Contact your Red Hat Sales Engineer to discuss how to design a test to find the optimal setting.

Data Content

Because VDO performs deduplication and compression, test data sets must be chosen to effectively exercise these capabilities.

30.2.8. Special Considerations for Testing Read Performance

When testing read performance, these factors must be considered:

1. If a 4 KB block has never been written, VDO will not perform I/O to the storage and will immediately respond with a zero block.

2. If a 4 KB block has been written but contains all zeros, VDO will not perform I/O to the storage and will immediately respond with a zero block.

This behavior results in very fast read performance when there is no data to read. **This makes it imperative that read tests prefill with actual data.**

30.2.9. Cross Talk

To prevent one test from affecting the results of another, it is suggested that a new VDO volume be created for each iteration of each test.

30.3. DATA EFFICIENCY TESTING PROCEDURES

Successful validation of VDO is dependent upon following a well-structured test procedure. This section provides a series of steps to follow, along with the expected results, as examples of tests to consider when participating in an evaluation.

Test Environment

The test cases in the next section make the following assumptions about the test environment:

- One or more Linux physical block devices are available.

- The target block device (for example, **/dev/sdb**) is larger than 512 GB.

- Flexible I/O Tester (**fio**) version 2.1.1 or later is installed.

- VDO is installed.

The following information should be recorded at the start of each test in order to ensure that the test environment is fully understood:

- The Linux build used, including the kernel build number.

- A complete list of installed packages, as obtained from the **rpm -qa** command.

- Complete system specifications:

    - CPU type and quantity (available in **/proc/cpuinfo**).

    - Installed memory and the amount available after the base OS is running (available in **/proc/meminfo**).

    - Type(s) of drive controller(s) used.

    - Type(s) and quantity of disk(s) used.

- A complete list of running processes (from **ps aux** or a similar listing).

- Name of the Physical Volume and the Volume Group created for use with VDO (**pvs** and **vgs** listings).

- File system used when formatting the VDO volume (if any).

- Permissions on the mounted directory.

- Contents of **/etc/vdoconf.yaml**.

- Location of the VDO files.

You can capture much of the required information by running **sosreport**.

Workloads

Effectively testing VDO requires the use of data sets that simulate real world workloads. The data sets should provide a balance between data that can be deduplicated and/or compressed and data that cannot in order to demonstrate performance under different conditions.

There are several tools that can synthetically generate data with repeatable characteristics. Two utilities in particular, VDbench and **fio**, are recommended for use during testing.

This guide uses **fio**. Understanding the arguments is critical to a successful evaluation:

Table 30.1. fio Options

Argument	Description	Value
--size	The quantity of data fio will send to the target per job (see **numjobs** below).	100 GB
--bs	The block size of each read/write request produced by fio. Red Hat recommends a 4 KB block size to match VDO's 4 KB default	4k
--numjobs	The number of jobs that fio will create to run the benchmark. Each job sends the amount of data specified by the **--size** parameter. The first job sends data to the device at the offset specified by the **--offset** parameter. Subsequent jobs write the same region of the disk (overwriting) unless the **--offset_increment** parameter is provided, which will offset each job from where the previous job began by that value. To achieve peak performance on flash at least two jobs are recommended. One job is typically enough to saturate rotational disk (HDD) throughput.	1 (HDD) 2 (SSD)
--thread	Instructs fio jobs to be run in threads rather than being forked, which may provide better performance by limiting context switching.	\<N/A\>
--ioengine	There are several I/O engines available in Linux that are able to be tested using fio. Red Hat testing uses the asynchronous unbuffered engine (**libaio**). If you are interested in another engine, discuss that with your Red Hat Sales Engineer. The Linux **libaio** engine is used to evaluate workloads in which one or more processes are making random requests simultaneously. **libaio** allows multiple requests to be made asynchronously from a single thread before any data has been retrieved, which limits the number of context switches that would be required if the requests were provided by manythreads via a synchronous engine.	libaio

Argument	Description	Value
`--direct`	When set, direct allows requests to be submitted to the device bypassing the Linux Kernel's page cache. Libaio Engine: `libaio` must be used with `direct` enabled (=1) or the kernel may resort to the `sync` API for all I/O requests.	1 (libaio)
`--iodepth`	The number of I/O buffers in flight at any time. A high `iodepth` will usually increase performance, particularly for random reads or writes. High depths ensure that the controller always has requests to batch. However, setting `iodepth` too high (greater than 1K, typically) may cause undesirable latency. While Red Hat recommends an `iodepth` between 128 and 512, the final value is a trade-off and depends on how your application tolerates latency.	128 (minimum)
`--iodepth_batch_submit`	The number of I/Os to create when the `iodepth` buffer pool begins to empty. This parameter limits task switching from I/O to buffer creation during the test.	16
`--iodepth_batch_complete`	The number of I/Os to complete before submitting a batch (`iodepth_batch_complete`). This parameter limits task switching from I/O to buffer creation during the test.	16
`--gtod_reduce`	Disables time-of-day calls to calculate latency. This setting will lower throughput if enabled (=0), so it should be enabled (=1) unless latency measurement is necessary.	1

30.3.1. Configuring a VDO Test Volume

1. Create a VDO Volume with a Logical Size of 1 TB on a 512 GB Physical Volume

1. Create a VDO volume.

   - To test the VDO **async** mode on top of synchronous storage, create an asynchronous volume using the **--writePolicy=async** option:

     ```
     # vdo create --name=vdo0 --device=/dev/sdb \
                  --vdoLogicalSize=1T --writePolicy=async --verbose
     ```

   - To test the VDO **sync** mode on top of synchronous storage, create a synchronous volume using the **--writePolicy=sync** option:

     ```
     # vdo create --name=vdo0 --device=/dev/sdb \
                  --vdoLogicalSize=1T --writePolicy=sync --verbose
     ```

2. Format the new device with an XFS or ext4 file system.

   - For XFS:

```
# mkfs.xfs -K /dev/mapper/vdo0
```

○ For ext4:

```
# mkfs.ext4 -E nodiscard /dev/mapper/vdo0
```

3. Mount the formatted device:

```
# mkdir /mnt/VDOVolume
# mount /dev/mapper/vdo0 /mnt/VDOVolume && \
   chmod a+rwx /mnt/VDOVolume
```

30.3.2. Testing VDO Efficiency

2. Test Reading and Writing to the VDO Volume

1. Write 32 GB of random data to the VDO volume:

```
$ dd if=/dev/urandom of=/mnt/VDOVolume/testfile bs=4096
count=8388608
```

2. Read the data from the VDO volume and write it to another location not on the VDO volume:

```
$ dd if=/mnt/VDOVolume/testfile of=/home/user/testfile bs=4096
```

3. Compare the two files using **diff**, which should report that the files are the same:

```
$ diff -s /mnt/VDOVolume/testfile /home/user/testfile
```

4. Copy the file to a second location on the VDO volume:

```
$ dd if=/home/user/testfile of=/mnt/VDOVolume/testfile2 bs=4096
```

5. Compare the third file to the second file. This should report that the files are the same:

```
$ diff -s /mnt/VDOVolume/testfile2 /home/user/testfile
```

3. Remove the VDO Volume

1. Unmount the file system created on the VDO volume:

```
# umount /mnt/VDOVolume
```

2. Run the command to remove the VDO volume **vdo0** from the system:

```
# vdo remove --name=vdo0
```

3. Verify that the volume has been removed. There should be no listing in **vdo list** for the VDO partition:

```
# vdo list --all | grep vdo
```

4. Measure Deduplication

1. Create and mount a VDO volume following Section 30.3.1, "Configuring a VDO Test Volume".

2. Create 10 directories on the VDO volume named **vdo1** through **vdo10** to hold 10 copies of the test data set:

```
$ mkdir /mnt/VDOVolume/vdo{01..10}
```

3. Examine the amount of disk space used according to the file system:

```
$ df -h /mnt/VDOVolume

Filesystem              Size  Used Avail Use% Mounted on
/dev/mapper/vdo0        1.5T  198M  1.4T   1% /mnt/VDOVolume
```

Consider tabulating the results in a table:

Statistic	Bare File System	After Seed	After 10 Copies
File System Used Size	198 MB		
VDO Data Used			
VDO Logical Used			

4. Run the following command and record the values. "Data blocks used" is the number of blocks used by user data on the physical device running under VDO. "Logical blocks used" is the number of blocks used before optimization. It will be used as the starting point for measurements

```
# vdostats --verbose | grep "blocks used"

data blocks used                : 1090
overhead blocks used            : 538846
logical blocks used             : 6059434
```

5. Create a data source file in the top level of the VDO volume

```
$ dd if=/dev/urandom of=/mnt/VDOVolume/sourcefile bs=4096
count=1048576

4294967296 bytes (4.3 GB) copied, 540.538 s, 7.9 MB/s
```

6. Re-examine the amount of used physical disk space in use. This should show an increase in the number of blocks used corresponding to the file just written:

```
$ df -h /mnt/VDOVolume
```

```
Filesystem              Size  Used Avail Use% Mounted on
/dev/mapper/vdo0        1.5T  4.2G  1.4T   1% /mnt/VDOVolume

# vdostats --verbose | grep "blocks used"

data blocks used                   : 1050093 (increased by 4GB)
overhead blocks used               : 538846 (Did not change)
logical blocks used                : 7108036 (increased by 4GB)
```

7. Copy the file to each of the 10 subdirectories:

```
$ for i in {01..10}; do
   cp /mnt/VDOVolume/sourcefile /mnt/VDOVolume/vdo$i
   done
```

8. Once again, check the amount of physical disk space used (data blocks used). This number should be similar to the result of step 6 above, with only a slight increase due to file system journaling and metadata:

```
$ df -h /mnt/VDOVolume

Filesystem              Size  Used Avail Use% Mounted on
/dev/mapper/vdo0        1.5T   45G  1.3T   4% /mnt/VDOVolume

# vdostats --verbose | grep "blocks used"

data blocks used                   : 1050836 (increased by 3M)
overhead blocks used               : 538846
logical blocks used                : 17594127 (increased by 41G)
```

9. Subtract this new value of the space used by the file system from the value found before writing the test data. This is the amount of space consumed by this test from the file system's perspective.

10. Observe the space savings in your recorded statistics:

   Note: In the following table, values have been converted to MB/GB. vdostats "blocks" are 4,096 B.

Statistic	Bare File System	After Seed	After 10 Copies
File System Used Size	198 MB	4.2 GB	45 GB
VDO Data Used	4 MB	4.1 GB	4.1 GB
VDO Logical Used	23.6 GB*	27.8 GB	68.7 GB

* File system overhead for 1.6 TB formatted drive

5. Measure Compression

1. Create a VDO volume of at least 10 GB of physical and logical size. Add options to disable deduplication and enable compression:

```
# vdo create --name=vdo0 --device=/dev/sdb \
              --vdoLogicalSize=10G --verbose \
              --deduplication=disabled --compression=enabled
```

2. Inspect VDO statistics before transfer; make note of data blocks used and logical blocks used (both should be zero):

```
# vdostats --verbose | grep "blocks used"
```

3. Format the new device with an XFS or ext4 file system.

   - For XFS:

     ```
     # mkfs.xfs -K /dev/mapper/vdo0
     ```

   - For ext4:

     ```
     # mkfs.ext4 -E nodiscard /dev/mapper/vdo0
     ```

4. Mount the formatted device:

```
# mkdir /mnt/VDOVolume
# mount /dev/mapper/vdo0 /mnt/VDOVolume && \
  chmod a+rwx /mnt/VDOVolume
```

5. Synchronize the VDO volume to complete any unfinished compression:

```
# sync && dmsetup message vdo0 0 sync-dedupe
```

6. Inspect VDO statistics again. Logical blocks used — data blocks used is the number of 4 KB blocks saved by compression for the file system alone. VDO optimizes file system overhead as well as actual user data:

```
# vdostats --verbose | grep "blocks used"
```

7. Copy the contents of **/lib** to the VDO volume. Record the total size:

```
# cp -vR /lib /mnt/VDOVolume

...
sent 152508960 bytes  received 60448 bytes  61027763.20 bytes/sec
total size is 152293104  speedup is 1.00
```

8. Synchronize Linux caches and the VDO volume:

```
# sync && dmsetup message vdo0 0 sync-dedupe
```

9. Inspect VDO statistics once again. Observe the logical and data blocks used:

```
# vdostats --verbose | grep "blocks used"
```

- Logical blocks used - data blocks used represents the amount of space used (in units of 4 KB blocks) for the copy of your **/lib** files.

- The total size (from the table in the section called "4. Measure Deduplication") - (logical blocks used-data blocks used * 4096) = bytes saved by compression.

10. Remove the VDO volume:

```
# umount /mnt/VDOVolume && vdo remove --name=vdo0
```

6. Test VDO Compression Efficiency

1. Create and mount a VDO volume following Section 30.3.1, "Configuring a VDO Test Volume".

2. Repeat the experiments in the section called "4. Measure Deduplication" and the section called "5. Measure Compression" without removing the volume. Observe changes to space savings in **vdostats**.

3. Experiment with your own datasets.

7. Understanding TRIM and DISCARD

Thin provisioning allows a logical or virtual storage space to be larger than the underlying physical storage. Applications such as file systems benefit from running on the larger virtual layer of storage, and data-efficiency techniques such as data deduplication reduce the number of physical data blocks needed to store all of the data. To benefit from these storage savings, the physical storage layer needs to know when application data has been deleted.

Traditional file systems did not have to inform the underlying storage when data was deleted. File systems that work with thin provisioned storage send **TRIM** or **DISCARD** commands to inform the storage system when a logical block is no longer required. These commands can be sent whenever a block is deleted using the discard mount option, or these commands can be sent in a controlled manner by running utilities such as **fstrim** that tell the file system to detect which logical blocks are unused and send the information to the storage system in the form of a **TRIM** or **DISCARD** command.

To see how this works:

1. Create and mount a new VDO logical volume following Section 30.3.1, "Configuring a VDO Test Volume".

2. Trim the file system to remove any unneeded blocks (this may take a long time):

```
# fstrim /mnt/VDOVolume
```

3. Record the initial state in following table below by entering:

```
$ df -m /mnt/VDOVolume
```

to see how much capacity is used in the file system, and run vdostats to see how many physical and logical data blocks are being used.

4. Create a 1 GB file with non-duplicate data in the file system running on top of VDO:

```
$ dd if=/dev/urandom of=/mnt/VDOVolume/file bs=1M count=1K
```

and then collect the same data. The file system should have used an additional 1 GB, and the data blocks used and logical blocks used have increased similarly.

5. Run **fstrim /mnt/VDOVolume** and confirm that this has no impact after creating a new file.

6. Delete the 1 GB file:

```
$ rm /mnt/VDOVolume/file
```

Check and record the parameters. The file system is aware that a file has been deleted, but there has been no change to the number of physical or logical blocks because the file deletion has not been communicated to the underlying storage.

7. Run **fstrim /mnt/VDOVolume** and record the same parameters. **fstrim** looks for free blocks in the file system and sends a TRIM command to the VDO volume for unused addresses, which releases the associated logical blocks, and VDO processes the TRIM to release the underlying physical blocks.

Step	File Space Used (MB)	Data Blocks Used	Logical Blocks Used
Initial			
Add 1 GB File			
Run **fstrim**			
Delete 1 GB File			
Run **fstrim**			

From this exercise, the TRIM process is needed so the underlying storage can have an accurate knowledge of capacity utilization. **fstrim** is a command line tool that analyzes many blocks at once for greater efficiency. An alternative method is to use the file system discard option when mounting. The discard option will update the underlying storage after each file system block is deleted, which can slow throughput but provides for great utilization awareness. It is also important to understand that the need to TRIM or DISCARD unused blocks is not unique to VDO; any thin-provisioned storage system has the same challenge

30.4. PERFORMANCE TESTING PROCEDURES

The goal of this section is to construct a performance profile of the device with VDO installed. Each test should be run with and without VDO installed, so that VDO's performance can be evaluated relative to the performance of the base system.

30.4.1. Phase 1: Effects of I/O Depth, Fixed 4 KB Blocks

The goal of this test is to determine the I/O depth that produces the optimal throughput and the lowest

latency for your appliance. VDO uses a 4 KB sector size rather than the traditional 512 B used on legacy storage devices. The larger sector size allows it to support higher-capacity storage, improve performance, and match the cache buffer size used by most operating systems.

1. Perform four-corner testing at 4 KB I/O, and I/O depth of 1, 8, 16, 32, 64, 128, 256, 512, 1024:

   - Sequential 100% reads, at fixed 4 KB *

   - Sequential 100% write, at fixed 4 KB

   - Random 100% reads, at fixed 4 KB *

   - Random 100% write, at fixed 4 KB **

   * Prefill any areas that may be read during the read test by performing a write fio job first

   ** Re-create the VDO volume after 4 KB random write I/O runs

   Example shell test input stimulus (write):

   ```
   # for depth in 1 2 4 8 16 32 64 128 256 512 1024 2048; do
     fio --rw=write --bs=4096 --name=vdo --filename=/dev/mapper/vdo0 \
         --ioengine=libaio --numjobs=1 --thread --norandommap --
   runtime=300\
         --direct=1 --iodepth=$depth --scramble_buffers=1  --offset=0 \
         --size=100g
     done
   ```

2. Record throughput and latency at each data point, and then graph.

3. Repeat test to complete four-corner testing: **--rw=randwrite**, **--rw=read**, and **--rw=randread**.

The result is a graph as shown below. Points of interest are the behavior across the range and the points of inflection where increased I/O depth proves to provide diminishing throughput gains. Likely, sequential access and random access will peak at different values, but it may be different for all types of storage configurations. In Figure 30.1, "I/O Depth Analysis" notice the "knee" in each performance curve. Marker 1 identifies the peak sequential throughput at point X, and marker 2 identifies peak random 4 KB throughput at point Z.

- This particular appliance does not benefit from sequential 4 KB I/O depth > X. Beyond that depth, there are diminishing bandwidth bandwidth gains, and average request latency will increase 1:1 for each additional I/O request.

- This particular appliance does not benefit from random 4 KB I/O depth > Z. Beyond that depth, there are diminishing bandwidth gains, and average request latency will increase 1:1 for each additional I/O request.
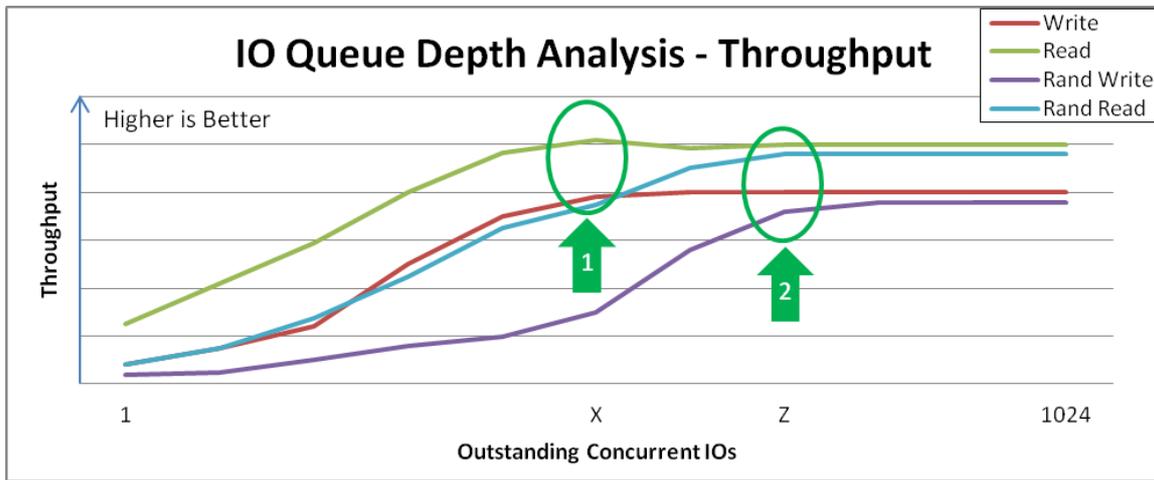
Figure 30.1. I/O Depth Analysis

Figure 30.2, "Latency Response of Increasing I/O for Random Writes" shows an example of the random write latency after the "knee" of the curve in Figure 30.1, "I/O Depth Analysis". Benchmarking practice should test at these points for maximum throughput that incurs the least response time penalty. As we move forward in the test plan for this example appliance, we will collect additional data with I/O depth = Z
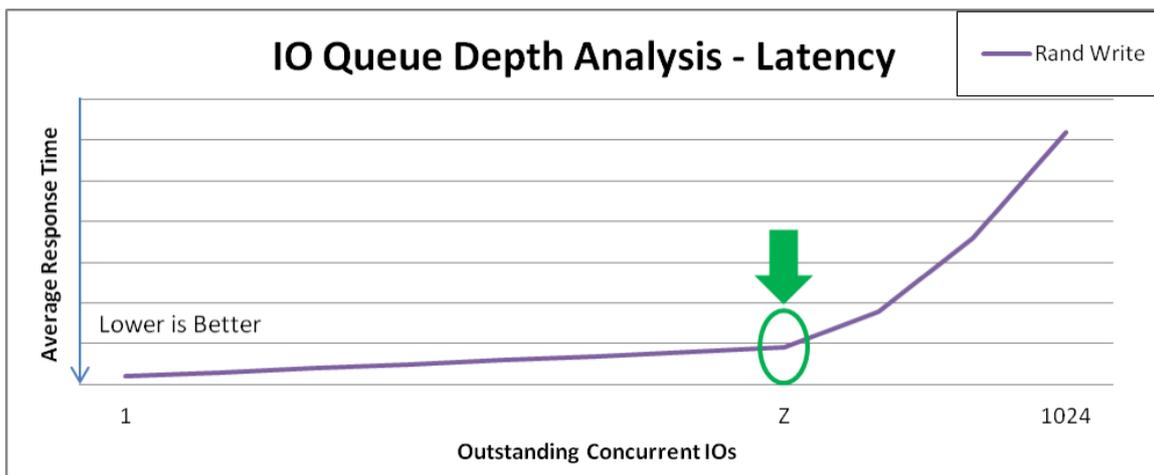
Figure 30.2. Latency Response of Increasing I/O for Random Writes

30.4.2. Phase 2: Effects of I/O Request Size

The goal of this test is to understand the block size that produces the best performance of the system under test at the optimal I/O depth determined in the previous step.

1. Perform four-corner testing at fixed I/O depth, with varied block size (powers of 2) over the range 8 KB to 1 MB. **Remember to prefill any areas to be read and to recreate volumes between tests.**

2. Set the I/O Depth to the value determined in Section 30.4.1, "Phase 1: Effects of I/O Depth, Fixed 4 KB Blocks".

   Example test input stimulus (write):

   ```
   # z=[see previous step]
   # for iosize in 4 8 16 32 64 128 256 512 1024; do
     fio --rw=write --bs=$iosize\k --name=vdo --
   filename=/dev/mapper/vdo0
   ```

```
        --ioengine=libaio --numjobs=1 --thread --norandommap --
runtime=300
        --direct=1 --iodepth=$z --scramble_buffers=1 --offset=0 --
size=100g
    done
```

3. Record throughput and latency at each data point, and then graph.

4. Repeat test to complete four-corner testing: **--rw=randwrite**, **--rw=read**, and **--rw=randread**.

There are several points of interest that you may find in the results. In this example:

- Sequential writes reach a peak throughput at request size Y. This curve demonstrates how applications that are configurable or naturally dominated by certain request sizes may perceive performance. Larger request sizes often provide more throughput because 4 KB I/Os may benefit from merging.

- Sequential reads reach a similar peak throughput at point Z. Remember that after these peaks, overall latency before the I/O completes will increase with no additional throughput. It would be wise to tune the device to not accept I/Os larger than this size.

- Random reads achieve peak throughput at point X. Some devices may achieve near-sequential throughput rates at large request size random accesses, while others suffer more penalty when varying from purely sequential access.

- Random writes achieve peak throughput at point Y. Random writes involve the most interaction of a deduplication device, and VDO achieves high performance especially when request sizes and/or I/O depths are large.

The results from this test Figure 30.3, "Request Size vs. Throughput Analysis and Key Inflection Points" help in understanding the characteristics of the storage device and the user experience for specific applications. Consult with a Red Hat Sales Engineer to determine if there may be further tuning needed to increase performance at different request sizes.
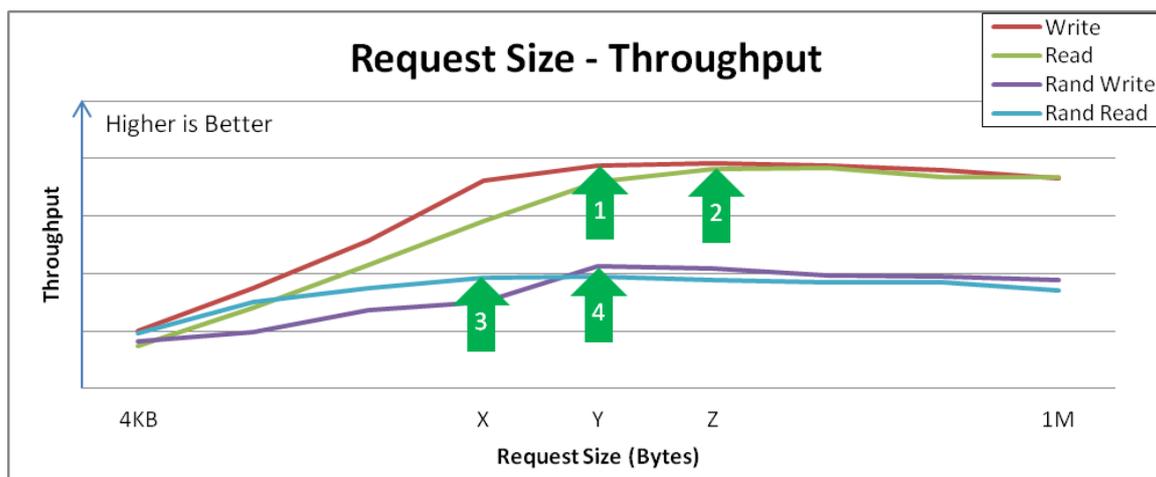
Figure 30.3. Request Size vs. Throughput Analysis and Key Inflection Points

30.4.3. Phase 3: Effects of Mixing Read & Write I/Os

The goal of this test is to understand how your appliance with VDO behaves when presented with mixed I/O loads (read/write), analyzing the effects of read/write mix at the optimal random queue depth and request sizes from 4 KB to 1 MB. You should use whatever is appropriate in your case.

1. Perform four-corner testing at fixed I/O depth, varied block size (powers of 2) over the 8 KB to 256 KB range, and set read percentage at 10% increments, beginning with 0%. **Remember to prefill any areas to be read and to recreate volumes between tests.**

2. Set the I/O Depth to the value determined in Section 30.4.1, "Phase 1: Effects of I/O Depth, Fixed 4 KB Blocks".

   Example test input stimulus (read/write mix):

   ```
   # z=[see previous step]
   # for readmix in 0 10 20 30 40 50 60 70 80 90 100; do
       for iosize in 4 8 16 32 64 128 256 512 1024; do
         fio --rw=rw --rwmixread=$readmix --bs=$iosize\k --name=vdo \
             --filename=/dev/mapper/vdo0 --ioengine=libaio --numjobs=1
   --thread \
             --norandommap --runtime=300 --direct=0 --iodepth=$z --
   scramble_buffers=1 \
             --offset=0 --size=100g
       done
     done
   ```

3. Record throughput and latency at each data point, and then graph.

Figure 30.4, "Performance Is Consistent across Varying Read/Write Mixes" shows an example of how VDO may respond to I/O loads:
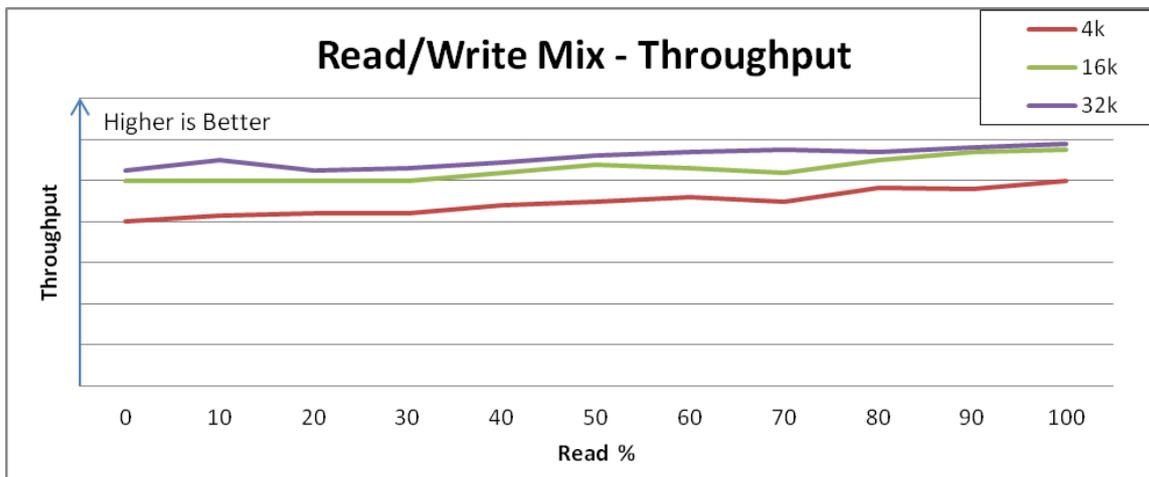
Figure 30.4. Performance Is Consistent across Varying Read/Write Mixes

Performance (aggregate) and latency (aggregate) are relatively consistent across the range of mixing reads and writes, trending from the lower max write throughput to the higher max read throughput.

This behavior may vary with different storage, but the important observation is that the performance is consistent under varying loads and/or that you can understand performance expectation for applications that demonstrate specific read/write mixes. If you discover any unexpected results, Red Hat Sales Engineers will be able to help you understand if it is VDO or the storage device itself that needs modification.

Note: Systems that do not exhibit a similar response consistency often signify a sub-optimal configuration. Contact your Red Hat Sales Engineer if this occurs.

30.4.4. Phase 4: Application Environments

The goal of these final tests is to understand how the system with VDO behaves when deployed in a real application environment. If possible, use real applications and use the knowledge learned so far; consider limiting the permissible queue depth on your appliance, and if possible tune the application to issue requests with those block sizes most beneficial to VDO performance.

Request sizes, I/O loads, read/write patterns, etc., are generally hard to predict, as they will vary by application use case (i.e., filers vs. virtual desktops vs. database), and applications often vary in the types of I/O based on the specific operation or due to multi-tenant access.

The final test shows general VDO performance in a mixed environment. If more specific details are known about your expected environment, test those settings as well.

Example test input stimulus (read/write mix):

```
# for readmix in 20 50 80; do
    for iosize in 4 8 16 32 64 128 256 512 1024; do
      fio --rw=rw --rwmixread=$readmix --bsrange=4k-256k --name=vdo \
          --filename=/dev/mapper/vdo0 --ioengine=libaio --numjobs=1 --
thread \
          --norandommap --runtime=300 --direct=0 --iodepth=$iosize \
          --scramble_buffers=1 --offset=0 --size=100g
    done
  done
```

Record throughput and latency at each data point, and then graph (Figure 30.5, "Mixed Environment Performance").
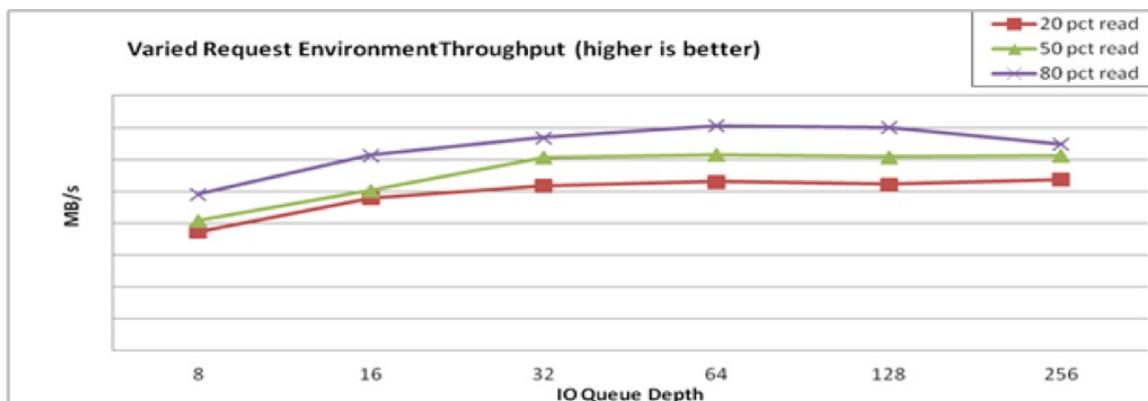
Figure 30.5. Mixed Environment Performance

30.5. ISSUE REPORTING

In the event that an issue is encountered while working with VDO, it is important to gather as much information as possible to assist Red Hat Sales Engineers in attempting to reproduce the issue.

Issue reports should include the following:

- A detailed description of the test environment; see the section called "Test Environment" for specifics

- The VDO configuration

- The use case that generated the issue

- The actions that were being performed at the time of the error

- The text of any error messages on the console or terminal

- The kernel log files

- Kernel crash dumps, if available

- The result of `sosreport`, which will capture data describing the entire Linux environment

30.6. CONCLUSION

Going through this or any other well-structured evaluation plan is an important step in integrating VDO into any storage system. The evaluation process is important to understanding performance and catching any potential compatibility issues. The collection of results from this evaluation not only demonstrates deduplication and compression, but also provides a performance profile of your system implementing VDO. The results help determine whether the results achieved in real applications are as expected and plausible or whether they fall short of expectations. Finally, we can also use these results to help predict the kinds of applications that will operate favorably with VDO.

APPENDIX A. RED HAT CUSTOMER PORTAL LABS RELEVANT TO STORAGE ADMINISTRATION

Red Hat Customer Portal Labs are tools designed to help you improve performance, troubleshoot issues, identify security problems, and optimize configuration. This appendix provides an overview of Red Hat Customer Portal Labs relevant to storage administration. All Red Hat Customer Portal Labs are available at https://access.redhat.com/labs/.

SCSI DECODER

The SCSI decoder is designed to decode SCSI error messages in the `/log/*` files or log file snippets, as these error messages can be hard to understand for the user.

Use the **SCSI decoder** to individually diagnose each SCSI error message and get solutions to resolve problems efficiently.

FILE SYSTEM LAYOUT CALCULATOR

The File System Layout Calculator determines the optimal parameters for creating ext3, ext4, and xfs file systems, after you provide storage options that describe your current or planned storage. Move the cursor over the question mark ("?") for a brief explanation of a particular option, or scroll down to read a summary of all options.

Use the **File System Layout Calculator** to generate a command that creates a file system with provided parameters on the specified RAID storage. Copy the generated command and execute it as **root** to create the required file system.

LVM RAID CALCULATOR

The LVM RAID Calculator determines the optimal parameters for creating logical volumes (LVMs) on a given RAID storage after you specify storage options. Move the cursor over the question mark ("?") for a brief explanation of a particular option, or scroll down to read a summary of all options.

The **LVM RAID Calculator** generates a sequence of commands that create LVMs on a given RAID storage. Copy and execute the generated commands one by one as **root** to create the required LVMs.

ISCSI HELPER

The iSCSI Helper provides a block-level storage over Internet Protocol (IP) networks, and enables the use of storage pools within server virtualization.

Use the **iSCSI Helper** to generate a script that prepares the system for its role of an iSCSI target (server) or an iSCSI initiator (client) configured according to the settings that you provide.

SAMBA CONFIGURATION HELPER

The Samba Configuration Helper creates a configuration that provides basic file and printer sharing through Samba:

- Click **Server** to specify basic server settings.

- Click **Shares** to add the directories that you want to share

- Click **Server** to add attached printers individually.

MULTIPATH HELPER

The Multipath Helper creates an optimal configuration for multipath devices on Red Hat Enterprise Linux 5, 6, and 7. By following the steps, you can create advanced multipath configurations, such as custom aliases or device blacklists.

The Multipath Helper also provides the `multipath.conf` file for a review. When you achieve the required configuration, download the installation script to run on your server.

NFS HELPER

The NFS Helper simplifies configuring a new NFS server or client. Follow the steps to specify the export and mount options. Then, generate a downloadable NFS configuration script.

MULTIPATH CONFIGURATION VISUALIZER

The Multipath Configuration Visualizer analyzes files in a sosreport and provides a diagram that visualizes the multipath configuration. Use the **Multipath Configuration Visualizer** to display:

- Hosts components including Host Bus Adapters (HBAs), local devices, and iSCSI devices on the server side

- Storage components on the storage side

- Fabric or Ethernet components between the server and the storage

- Paths to all mentioned components

You can either upload a sosreport compressed in the .xz, .gz, or .bz2 format, or extract a sosreport in a directory that you then select as the source for a client-side analysis.

RHEL BACKUP AND RESTORE ASSISTANT

The RHEL Backup and Restore Assistant provides information on back-up and restore tools, and common scenarios of Linux usage.

Described tools:

- **dump and restore**: for backing up the ext2, ext3, and ext4 file systems.

- **tar and cpio**: for archiving or restoring files and folders, especially when backing up the tape drives.

- **rsync**: for performing back-up operations and synchronizing files and directories between locations.

- **dd**: for copying files from a source to a destination block by block independently of the file systems or operating systems involved.

Described scenarios:

- Disaster recovery

- Hardware migration

- Partition table backup

- Important folder backup

- Incremental backup

- Differential backup

APPENDIX B. REVISION HISTORY

Revision 4-06	**Tue Aug 28 11 2018**	**Marek Suchanek**
An asynchronous update		
Revision 4-02	**Thu May 10 2018**	**Marek Suchanek**
An asynchronous update		
Revision 4-00	**Fri Apr 6 2018**	**Marek Suchanek**
Document version for 7.5 GA publication.		
Revision 3-95	**Thu Apr 5 2018**	**Marek Suchanek**
An asynchronous update		
Revision 3-93	**Mon Mar 5 2018**	**Marek Suchanek**
New chapter: VDO Integration		
Revision 3-92	**Fri Feb 9 2018**	**Marek Suchanek**
An asynchronous update		
Revision 3-90	**Wed Dec 6 2017**	**Marek Suchanek**
Version for 7.5 Alpha publication.		
Revision 3-86	**Mon Nov 6 2017**	**Marek Suchanek**
An asynchronous update.		
Revision 3-80	**Thu Jul 27 2017**	**Milan Navratil**
Document version for 7.4 GA publication.		
Revision 3-77	**Wed May 24 2017**	**Milan Navratil**
An asynchronous update.		
Revision 3-68	**Fri Oct 21 2016**	**Milan Navratil**
Version for 7.3 GA publication.		
Revision 3-67	**Fri Jun 17 2016**	**Milan Navratil**
An asynchronous update.		
Revision 3-64	**Wed Nov 11 2015**	**Jana Heves**
Version for 7.2 GA release.		
Revision 3-33	**Wed Feb 18 2015**	**Jacquelynn East**
Version for 7.1 GA		
Revision 3-26	**Wed Jan 21 2015**	**Jacquelynn East**
Added overview of Ceph		
Revision 3-22	**Thu Dec 4 2014**	**Jacquelynn East**
7.1 Beta		
Revision 3-4	**Thu Jul 17 2014**	**Jacquelynn East**
Added new chapter on targetcli		
Revision 3-1	**Tue Jun 3 2014**	**Jacquelynn East**
Version for 7.0 GA release		

INDEX

Symbols

/boot/ directory, The /boot/ Directory

/dev/shm, df Command

/etc/fstab, Converting to an ext3 File System, Mounting NFS File Systems Using /etc/fstab, Mounting a File System

/etc/fstab file

enabling disk quotas with, Enabling Quotas

/local/directory (client configuration, mounting)

NFS, Configuring NFS Client

/proc

/proc/devices, The /proc Virtual File System

/proc/filesystems, The /proc Virtual File System

/proc/mdstat, The /proc Virtual File System

/proc/mounts, The /proc Virtual File System

/proc/mounts/, The /proc Virtual File System

/proc/partitions, The /proc Virtual File System

/proc/devices

virtual file system (/proc), The /proc Virtual File System

/proc/filesystems

virtual file system (/proc), The /proc Virtual File System

/proc/mdstat

virtual file system (/proc), The /proc Virtual File System

/proc/mounts

virtual file system (/proc), The /proc Virtual File System

/proc/mounts/

virtual file system (/proc), The /proc Virtual File System

/proc/partitions

virtual file system (/proc), The /proc Virtual File System

/remote/export (client configuration, mounting)

NFS, Configuring NFS Client

A

adding paths to a storage device, Adding a Storage Device or Path

adding/removing