Red Hat Enterprise Linux 6

Logical Volume Manager Administration

LVM Administrator Guide

Red Hat Enterprise Linux 6 Logical Volume Manager Administration
LVM Administrator Guide
Edition 1

Copyright © 2011 Red Hat, Inc. and others.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at *http://creativecommons.org/licenses/by-sa/3.0/*. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, MetaMatrix, Fedora, the Infinity Logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

XFS® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

All other trademarks are the property of their respective owners.

1801 Varsity Drive
Raleigh, NC 27606-2072 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701

This book describes the LVM logical volume manager, including information on running LVM in a clustered environment.

Introduction vii
- 1. About This Guide ... vii
- 2. Audience ... vii
- 3. Software Versions ... vii
- 4. Related Documentation ... vii
- 5. We Need Feedback! ... viii
- 6. Document Conventions ... viii
  - 6.1. Typographic Conventions ... viii
  - 6.2. Pull-quote Conventions ... x
  - 6.3. Notes and Warnings ... x

1. The LVM Logical Volume Manager 1
- 1.1. New and Changed Features ... 1
  - 1.1.1. New and Changed Features for Red Hat Enterprise Linux 6.0 ... 1
  - 1.1.2. New and Changed Features for Red Hat Enterprise Linux 6.1 ... 2
  - 1.1.3. New and Changed Features for Red Hat Enterprise Linux 6.2 ... 2
- 1.2. Logical Volumes ... 2
- 1.3. LVM Architecture Overview ... 3
- 1.4. The Clustered Logical Volume Manager (CLVM) ... 4
- 1.5. Document Overview ... 6

2. LVM Components 9
- 2.1. Physical Volumes ... 9
  - 2.1.1. LVM Physical Volume Layout ... 9
  - 2.1.2. Multiple Partitions on a Disk ... 10
- 2.2. Volume Groups ... 10
- 2.3. LVM Logical Volumes ... 11
  - 2.3.1. Linear Volumes ... 11
  - 2.3.2. Striped Logical Volumes ... 12
  - 2.3.3. Mirrored Logical Volumes ... 13
  - 2.3.4. Snapshot Volumes ... 14

3. LVM Administration Overview 17
- 3.1. Creating LVM Volumes in a Cluster ... 17
- 3.2. Logical Volume Creation Overview ... 17
- 3.3. Growing a File System on a Logical Volume ... 18
- 3.4. Logical Volume Backup ... 18
- 3.5. Logging ... 19

4. LVM Administration with CLI Commands 21
- 4.1. Using CLI Commands ... 21
- 4.2. Physical Volume Administration ... 22
  - 4.2.1. Creating Physical Volumes ... 22
  - 4.2.2. Displaying Physical Volumes ... 24
  - 4.2.3. Preventing Allocation on a Physical Volume ... 25
  - 4.2.4. Resizing a Physical Volume ... 25
  - 4.2.5. Removing Physical Volumes ... 25
- 4.3. Volume Group Administration ... 25
  - 4.3.1. Creating Volume Groups ... 25
  - 4.3.2. Creating Volume Groups in a Cluster ... 26
  - 4.3.3. Adding Physical Volumes to a Volume Group ... 27
  - 4.3.4. Displaying Volume Groups ... 27
  - 4.3.5. Scanning Disks for Volume Groups to Build the Cache File ... 28
  - 4.3.6. Removing Physical Volumes from a Volume Group ... 29
  - 4.3.7. Changing the Parameters of a Volume Group ... 29
  - 4.3.8. Activating and Deactivating Volume Groups ... 30

- 4.3.9. Removing Volume Groups 30
- 4.3.10. Splitting a Volume Group 30
- 4.3.11. Combining Volume Groups 30
- 4.3.12. Backing Up Volume Group Metadata 31
- 4.3.13. Renaming a Volume Group 31
- 4.3.14. Moving a Volume Group to Another System 31
- 4.3.15. Recreating a Volume Group Directory 32

4.4. Logical Volume Administration 32
- 4.4.1. Creating Linear Logical Volumes 32
- 4.4.2. Creating Striped Volumes 34
- 4.4.3. Creating Mirrored Volumes 34
- 4.4.4. Creating Snapshot Volumes 39
- 4.4.5. Merging Snapshot Volumes 41
- 4.4.6. Persistent Device Numbers 41
- 4.4.7. Resizing Logical Volumes 42
- 4.4.8. Changing the Parameters of a Logical Volume Group 42
- 4.4.9. Renaming Logical Volumes 42
- 4.4.10. Removing Logical Volumes 42
- 4.4.11. Displaying Logical Volumes 43
- 4.4.12. Growing Logical Volumes 43
- 4.4.13. Shrinking Logical Volumes 47

4.5. Controlling LVM Device Scans with Filters 47

4.6. Online Data Relocation 48

4.7. Activating Logical Volumes on Individual Nodes in a Cluster 49

4.8. Customized Reporting for LVM 49
- 4.8.1. Format Control 49
- 4.8.2. Object Selection 51
- 4.8.3. Sorting LVM Reports 57
- 4.8.4. Specifying Units 57

5. LVM Configuration Examples 59

5.1. Creating an LVM Logical Volume on Three Disks 59
- 5.1.1. Creating the Physical Volumes 59
- 5.1.2. Creating the Volume Group 59
- 5.1.3. Creating the Logical Volume 59
- 5.1.4. Creating the File System 59

5.2. Creating a Striped Logical Volume 60
- 5.2.1. Creating the Physical Volumes 60
- 5.2.2. Creating the Volume Group 60
- 5.2.3. Creating the Logical Volume 61
- 5.2.4. Creating the File System 61

5.3. Splitting a Volume Group 61
- 5.3.1. Determining Free Space 62
- 5.3.2. Moving the Data 62
- 5.3.3. Splitting the Volume Group 62
- 5.3.4. Creating the New Logical Volume 63
- 5.3.5. Making a File System and Mounting the New Logical Volume 63
- 5.3.6. Activating and Mounting the Original Logical Volume 63

5.4. Removing a Disk from a Logical Volume 63
- 5.4.1. Moving Extents to Existing Physical Volumes 64
- 5.4.2. Moving Extents to a New Disk 64

5.5. Creating a Mirrored LVM Logical Volume in a Cluster 66

6. LVM Troubleshooting 69

6.1. Troubleshooting Diagnostics 69

6.2. Displaying Information on Failed Devices 69
6.3. Recovering from LVM Mirror Failure 70
6.4. Recovering Physical Volume Metadata 73
6.5. Replacing a Missing Physical Volume 74
6.6. Removing Lost Physical Volumes from a Volume Group 75
6.7. Insufficient Free Extents for a Logical Volume 75

7. LVM Administration with the LVM GUI 77

A. The Device Mapper 79
A.1. Device Table Mappings 79
  A.1.1. The linear Mapping Target 80
  A.1.2. The striped Mapping Target 80
  A.1.3. The mirror Mapping Target 82
  A.1.4. The snapshot and snapshot-origin Mapping Targets 83
  A.1.5. The error Mapping Target 85
  A.1.6. The zero Mapping Target 85
  A.1.7. The multipath Mapping Target 86
  A.1.8. The crypt Mapping Target 88
A.2. The dmsetup Command 89
  A.2.1. The dmsetup info Command 89
  A.2.2. The dmsetup ls Command 90
  A.2.3. The dmsetup status Command 91
  A.2.4. The dmsetup deps Command 91
A.3. Device Mapper Support for the udev Device Manager 92
  A.3.1. udev Integration with the Device Mapper 92
  A.3.2. Commands and Interfaces that Support udev 94

B. The LVM Configuration Files 97
B.1. The LVM Configuration Files 97
B.2. Sample lvm.conf File 97

C. LVM Object Tags 109
C.1. Adding and Removing Object Tags 109
C.2. Host Tags 109
C.3. Controlling Activation with Tags 110

D. LVM Volume Group Metadata 111
D.1. The Physical Volume Label 111
D.2. Metadata Contents 111
D.3. Sample Metadata 112

E. Revision History 115

Index 117

Introduction

1. About This Guide

This book describes the Logical Volume Manager (LVM), including information on running LVM in a clustered environment.

2. Audience

This book is intended to be used by system administrators managing systems running the Linux operating system. It requires familiarity with Red Hat Enterprise Linux 6 and GFS2 file system administration.

3. Software Versions

Table 1. Software Versions

Software	Description
Red Hat Enterprise Linux 6	refers to Red Hat Enterprise Linux 6 and higher
GFS2	refers to GFS2 for Red Hat Enterprise Linux 6 and higher

4. Related Documentation

For more information about using Red Hat Enterprise Linux, refer to the following resources:

- *Installation Guide* — Documents relevant information regarding the installation of Red Hat Enterprise Linux 6.
- *Deployment Guide* — Documents relevant information regarding the deployment, configuration and administration of Red Hat Enterprise Linux 6.
- *Storage Administration Guide* — Provides instructions on how to effectively manage storage devices and file systems on Red Hat Enterprise Linux 6.

For more information about the High Availability Add-On and the Resilient Storage Add-On for Red Hat Enterprise Linux 6, refer to the following resources:

- *High Availability Add-On Overview* — Provides a high-level overview of the Red Hat High Availability Add-On.
- *Cluster Administration* — Provides information about installing, configuring and managing the Red Hat High Availability Add-On,
- *Global File System 2: Configuration and Administration* — Provides information about installing, configuring, and maintaining Red Hat GFS2 (Red Hat Global File System 2), which is included in the Resilient Storage Add-On.
- *DM Multipath* — Provides information about using the Device-Mapper Multipath feature of Red Hat Enterprise Linux 6.
- *Load Balancer Administration* — Provides information on configuring high-performance systems and services with the Load Balancer Add-On, a set of integrated software components that provide Linux Virtual Servers (LVS) for balancing IP load across a set of real servers.

- *Release Notes* — Provides information about the current release of Red Hat products.

High Availability Add-On documentation and other Red Hat documents are available in HTML, PDF, and RPM versions on the Red Hat Enterprise Linux Documentation CD and online at *http://docs.redhat.com/docs/en-US/index.html*.

5. We Need Feedback!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in Bugzilla: *http://bugzilla.redhat.com/* against the product **Red Hat Enterprise Linux 6** and the component **doc-Logical_Volume_Manager**. When submitting a bug report, be sure to mention the manual's identifier:

```
Logical_Volume_Manager_Administration(EN)-6 (2011-11-14-15:20)
```

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

6. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts*[1] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

6.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight keycaps and key combinations. For example:

> To see the contents of the file **`my_next_bestselling_novel`** in your current working directory, enter the **`cat my_next_bestselling_novel`** command at the shell prompt and press **`Enter`** to execute the command.

The above includes a file name, a shell command and a keycap, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from keycaps by the hyphen connecting each part of a key combination. For example:

> Press **`Enter`** to execute the command.

---

[1] https://fedorahosted.org/liberation-fonts/

> Press **`Ctrl`**+**`Alt`**+**`F2`** to switch to the first virtual terminal. Press **`Ctrl`**+**`Alt`**+**`F1`** to return to your X-Windows session.

The first paragraph highlights the particular keycap to press. The second highlights two key combinations (each a set of three keycaps with each set pressed simultaneously).

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in **`mono-spaced bold`**. For example:

> File-related classes include **`filesystem`** for file systems, **`file`** for files, and **`dir`** for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialog box text; labeled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System** → **Preferences** → **Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications** → **Accessories** → **Character Map** from the main menu bar. Next, choose **Search** → **Find…** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit** → **Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in proportional bold and all distinguishable by context.

`Mono-spaced Bold Italic` or ***`Proportional Bold Italic`***

Whether mono-spaced bold or proportional bold, the addition of italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type **`ssh`** ***`username@domain.name`*** at a shell prompt. If the remote machine is **`example.com`** and your username on that machine is john, type **`ssh john@example.com`**.
>
> The **`mount -o remount`** ***`file-system`*** command remounts the named file system. For example, to remount the **`/home`** file system, the command is **`mount -o remount /home`**.
>
> To see the version of a currently installed package, use the **`rpm -q`** ***`package`*** command. It will return a result as follows: ***`package-version-release`***.

Note the words in bold italics above — username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> Publican is a *DocBook* publishing system.

6.2. Pull-quote Conventions

Terminal output and source code listings are set off visually from the surrounding text.

Output sent to a terminal is set in **`mono-spaced roman`** and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in **`mono-spaced roman`** but add syntax highlighting as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }
}
```

6.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled 'Important' will not cause data loss but may cause irritation and frustration.

Warning

Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

The LVM Logical Volume Manager

This chapter provides a summary of the features of the LVM logical volume manager that are new for the initial and subsequent releases of Red Hat Enterprise Linux 6. Following that, this chapter provides a high-level overview of the components of the Logical Volume Manager (LVM).

1.1. New and Changed Features

This section lists new and changed features of the LVM logical volume manager that are included with the initial and subsequent releases of Red Hat Enterprise Linux 6.

1.1.1. New and Changed Features for Red Hat Enterprise Linux 6.0

Red Hat Enterprise Linux 6.0 includes the following documentation and feature updates and changes.

- You can define how a mirrored logical volume behaves in the event of a device failure with the **`mirror_image_fault_policy`** and **`mirror_log_fault_policy`** parameters in the **`activation`** section of the **`lvm.conf`** file. When this parameter is set to **`remove`**, the system attempts to remove the faulty device and run without it. When this parameter is set to **`allocate`**, the system attempts to remove the faulty device and tries to allocate space on a new device to be a replacement for the failed device; this policy acts like the **`remove`** policy if no suitable device and space can be allocated for the replacement. For information on the LVM mirror failure policies, see *Section 4.4.3.1, “Mirrored Logical Volume Failure Policy”*.

- For the Red Hat Enterprise Linux 6 release, the Linux I/O stack has been enhanced to process vendor-provided I/O limit information. This allows storage management tools, including LVM, to optimize data placement and access. This support can be disabled by changing the default values of **`data_alignment_detection`** and **`data_alignment_offset_detection`** in the **`lvm.conf`** file, although disabling this support is not recommended.

  For information on data alignment in LVM as well as information on changing the default values of **`data_alignment_detection`** and **`data_alignment_offset_detection`**, see the inline documentation for the **`/etc/lvm/lvm.conf`** file, which is also documented in *Appendix B, The LVM Configuration Files*. For general information on support for the I/O Stack and I/O limits in Red Hat Enterprise Linux 6, see the *Storage Administration Guide*.

- In Red Hat Enterprise Linux 6, the Device Mapper provides direct support for **`udev`** integration. This synchronizes the Device Mapper with all **`udev`** processing related to Device Mapper devices, including LVM devices. For information on Device Mapper support for the **`udev`** device manager, see *Section A.3, “Device Mapper Support for the udev Device Manager”*.

- For the Red Hat Enterprise Linux 6 release, you can use the **`lvconvert --repair`** command to repair a mirror after disk failure. This brings the mirror back into a consistent state. For information on the **`lvconvert --repair`** command, see *Section 4.4.3.3, “Repairing a Mirrored Logical Device”*.

- As of the Red Hat Enterprise Linux 6 release, you can use the **`--merge`** option of the **`lvconvert`** command to merge a snapshot into its origin volume. For information on merging snapshots, see *Section 4.4.5, “Merging Snapshot Volumes”*.

- As of the Red Hat Enterprise Linux 6 release, you can use the **`--splitmirrors`** argument of the **`lvconvert`** command to split off a redundant image of a mirrored logical volume to form a new logical volume. For information on using this option, see *Section 4.4.3.2, “Splitting Off a Redundant Image of a Mirrored Logical Volume”*.

- You can now create a mirror log for a mirrored logical device that is itself mirrored by using the **`--mirrorlog mirrored`** argument of the **`lvcreate`** command when creating a mirrored logical device. For information on using this option, see *Section 4.4.3, “Creating Mirrored Volumes”*.

1.1.2. New and Changed Features for Red Hat Enterprise Linux 6.1

Red Hat Enterprise Linux 6.1 includes the following documentation and feature updates and changes.

- The Red Hat Enterprise Linux 6.1 release supports the creation of snapshot logical volumes of mirrored logical volumes. You create a snapshot of a mirrored volume just as you would create a snapshot of a linear or striped logical volume. For information on creating snapshot volumes, see *Section 4.4.4, “Creating Snapshot Volumes”*.

- When extending an LVM volume, you can now use the **`--alloc cling`** option of the **`lvextend`** command to specify the **`cling`** allocation policy. This policy will choose space on the same physical volumes as the last segment of the existing logical volume. If there is insufficient space on the physical volumes and a list of tags is defined in the **`lvm.conf`** file, LVM will check whether any of the tags are attached to the physical volumes and seek to match those physical volume tags between existing extents and new extents.

  For information on extending LVM mirrored volumes with the **`--alloc cling`** option of the **`lvextend`** command, see *Section 4.4.12.2, “Extending a Logical Volume with the **`cling`** Allocation Policy”*.

- You can now specify multiple **`--addtag`** and **`--deltag`** arguments within a single **`pvchange`**, **`vgchange`**, or **`lvchange`** command. For information on adding and removing object tags, see *Section C.1, “Adding and Removing Object Tags”*.

- The list of allowed characters in LVM object tags has been extended, and tags can contain the "/", "=", "!", ":", "#", and "&" characters. For information on LVM object tags, see *Appendix C, LVM Object Tags*.

- You can now combine RAID0 (striping) and RAID1 (mirroring) in a single logical volume. Creating a logical volume while simultaneously specifying the number of mirrors (**`--mirrors X`**) and the number of stripes (**`--stripes Y`**) results in a mirror device whose constituent devices are striped. For information on creating mirrored logical volumes, see *Section 4.4.3, “Creating Mirrored Volumes”*.

- As of the Red Hat Enterprise Linux 6.1 release, if you need to create a consistent backup of data on a clustered logical volume you can activate the volume exclusively and then create the snapshot. For information on activating logical volumes exclusively on one node, see *Section 4.7, “Activating Logical Volumes on Individual Nodes in a Cluster”*.

1.1.3. New and Changed Features for Red Hat Enterprise Linux 6.2

Red Hat Enterprise Linux 6.2 includes the following documentation and feature updates and changes.

- The Red Hat Enterprise Linux 6.2 release supports the **`issue_discards`** parameter in the **`lvm.conf`** configuration file. When this parameter is set, LVM will issue discards to a logical volume's underlying physical volumes when the logical volume is no longer using the space on the physical volumes. For information on this parameter, refer to the inline documentation for the **`/etc/lvm/lvm.conf`** file, which is also documented in *Appendix B, The LVM Configuration Files*.

1.2. Logical Volumes

Volume management creates a layer of abstraction over physical storage, allowing you to create logical storage volumes. This provides much greater flexibility in a number of ways than using physical storage directly. With a logical volume, you are not restricted to physical disk sizes. In addition, the hardware storage configuration is hidden from the software so it can be resized and moved without stopping applications or unmounting file systems. This can reduce operational costs.

Logical volumes provide the following advantages over using physical storage directly:

- Flexible capacity

  When using logical volumes, file systems can extend across multiple disks, since you can aggregate disks and partitions into a single logical volume.

- Resizeable storage pools

  You can extend logical volumes or reduce logical volumes in size with simple software commands, without reformatting and repartitioning the underlying disk devices.

- Online data relocation

  To deploy newer, faster, or more resilient storage subsystems, you can move data while your system is active. Data can be rearranged on disks while the disks are in use. For example, you can empty a hot-swappable disk before removing it.

- Convenient device naming

  Logical storage volumes can be managed in user-defined groups, which you can name according to your convenience.

- Disk striping

  You can create a logical volume that stripes data across two or more disks. This can dramatically increase throughput.

- Mirroring volumes

  Logical volumes provide a convenient way to configure a mirror for your data.

- Volume Snapshots

  Using logical volumes, you can take device snapshots for consistent backups or to test the effect of changes without affecting the real data.

The implementation of these features in LVM is described in the remainder of this document.

1.3. LVM Architecture Overview

For the Red Hat Enterprise Linux 4 release of the Linux operating system, the original LVM1 logical volume manager was replaced by LVM2, which has a more generic kernel framework than LVM1. LVM2 provides the following improvements over LVM1:

- flexible capacity
- more efficient metadata storage
- better recovery format
- new ASCII metadata format

- atomic changes to metadata
- redundant copies of metadata

LVM2 is backwards compatible with LVM1, with the exception of snapshot and cluster support. You can convert a volume group from LVM1 format to LVM2 format with the **`vgconvert`** command. For information on converting LVM metadata format, see the **`vgconvert`**(8) man page.

The underlying physical storage unit of an LVM logical volume is a block device such as a partition or whole disk. This device is initialized as an LVM *physical volume* (PV).

To create an LVM logical volume, the physical volumes are combined into a *volume group* (VG). This creates a pool of disk space out of which LVM logical volumes (LVs) can be allocated. This process is analogous to the way in which disks are divided into partitions. A logical volume is used by file systems and applications (such as databases).

Figure 1.1, “LVM Logical Volume Components” shows the components of a simple LVM logical volume:

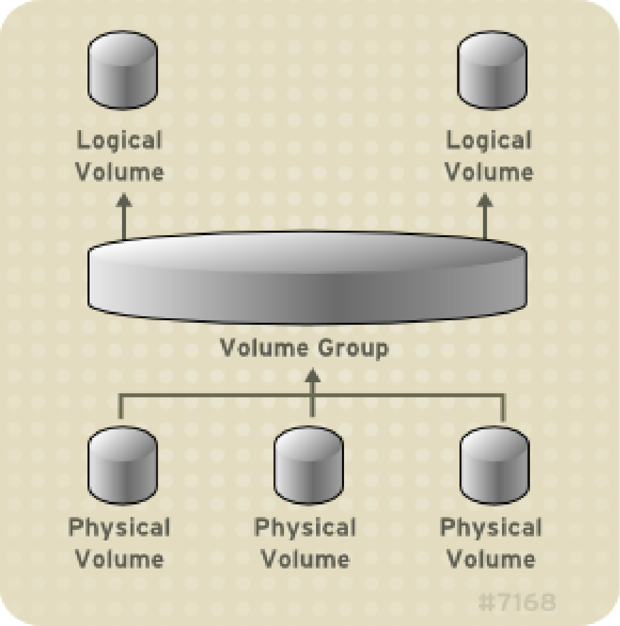

Figure 1.1. LVM Logical Volume Components

For detailed information on the components of an LVM logical volume, see *Chapter 2, LVM Components*.

1.4. The Clustered Logical Volume Manager (CLVM)

The Clustered Logical Volume Manager (CLVM) is a set of clustering extensions to LVM. These extensions allow a cluster of computers to manage shared storage (for example, on a SAN) using LVM. CLVM is part of the Resilient Storage Add-On.

Whether you should use CLVM depends on your system requirements:

- If only one node of your system requires access to the storage you are configuring as logical volumes, then you can use LVM without the CLVM extensions and the logical volumes created with that node are all local to the node.

- If you are using a clustered system for failover where only a single node that accesses the storage is active at any one time, you should use High Availability Logical Volume Management agents (HA-LVM).
- If more than one node of your cluster will require access to your storage which is then shared among the active nodes, then you must use CLVM. CLVM allows a user to configure logical volumes on shared storage by locking access to physical storage while a logical volume is being configured, and uses clustered locking services to manage the shared storage.

In order to use CLVM, the High Availability Add-On and Resilient Storage Add-On software, including the **`clvmd`** daemon, must be running. The **`clvmd`** daemon is the key clustering extension to LVM. The **`clvmd`** daemon runs in each cluster computer and distributes LVM metadata updates in a cluster, presenting each cluster computer with the same view of the logical volumes. For information on installing and administering the High Availability Add-On see *Cluster Administration*.

To ensure that **`clvmd`** is started at boot time, you can execute a **`chkconfig ... on`** command on the **`clvmd`** service, as follows:

```
# chkconfig clvmd on
```

If the **`clvmd`** daemon has not been started, you can execute a **`service ... start`** command on the **`clvmd`** service, as follows:

```
# service clvmd start
```

Creating LVM logical volumes in a cluster environment is identical to creating LVM logical volumes on a single node. There is no difference in the LVM commands themselves, or in the LVM graphical user interface, as described in *Chapter 4, LVM Administration with CLI Commands* and *Chapter 7, LVM Administration with the LVM GUI*. In order to enable the LVM volumes you are creating in a cluster, the cluster infrastructure must be running and the cluster must be quorate.

By default, logical volumes created with CLVM on shared storage are visible to all systems that have access to the shared storage. It is possible to create volume groups in which all of the storage devices are visible to only one node in the cluster. It is also possible to change the status of a volume group from a local volume group to a clustered volume group. For information, see *Section 4.3.2, “Creating Volume Groups in a Cluster”* and *Section 4.3.7, “Changing the Parameters of a Volume Group”*.

Warning

When you create volume groups with CLVM on shared storage, you must ensure that all nodes in the cluster have access to the physical volumes that constitute the volume group. Asymmetric cluster configurations in which some nodes have access to the storage and others do not are not supported.

Figure 1.2, “CLVM Overview” shows a CLVM overview in a cluster.

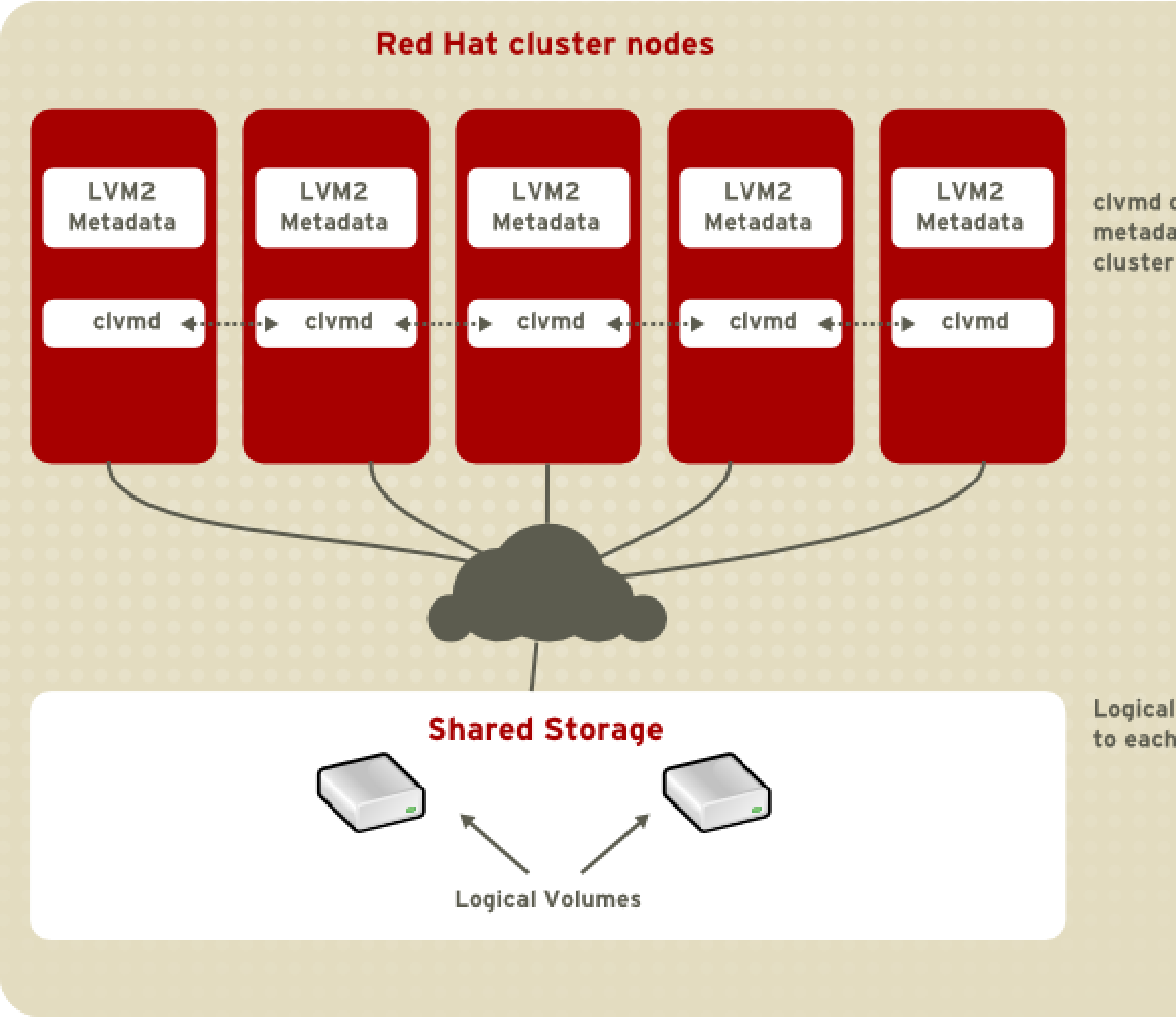

Figure 1.2. CLVM Overview

Note

CLVM requires changes to the **`lvm.conf`** file for cluster-wide locking. Information on configuring the **`lvm.conf`** file to support clustered locking is provided within the **`lvm.conf`** file itself. For information about the **`lvm.conf`** file, see *Appendix B, The LVM Configuration Files*.

1.5. Document Overview

This remainder of this document includes the following chapters:

- *Chapter 2, LVM Components* describes the components that make up an LVM logical volume.

- *Chapter 3, LVM Administration Overview* provides an overview of the basic steps you perform to configure LVM logical volumes, whether you are using the LVM Command Line Interface (CLI) commands or the LVM Graphical User Interface (GUI).
- *Chapter 4, LVM Administration with CLI Commands* summarizes the individual administrative tasks you can perform with the LVM CLI commands to create and maintain logical volumes.
- *Chapter 5, LVM Configuration Examples* provides a variety of LVM configuration examples.
- *Chapter 6, LVM Troubleshooting* provides instructions for troubleshooting a variety of LVM issues.
- *Chapter 7, LVM Administration with the LVM GUI* summarizes the operating of the LVM GUI.
- *Appendix A, The Device Mapper* describes the Device Mapper that LVM uses to map logical and physical volumes.
- *Appendix B, The LVM Configuration Files* describes the LVM configuration files.
- *Appendix C, LVM Object Tags* describes LVM object tags and host tags.
- *Appendix D, LVM Volume Group Metadata* describes LVM volume group metadata, and includes a sample copy of metadata for an LVM volume group.

LVM Components

This chapter describes the components of an LVM Logical volume.

2.1. Physical Volumes

The underlying physical storage unit of an LVM logical volume is a block device such as a partition or whole disk. To use the device for an LVM logical volume the device must be initialized as a physical volume (PV). Initializing a block device as a physical volume places a label near the start of the device.

By default, the LVM label is placed in the second 512-byte sector. You can overwrite this default by placing the label on any of the first 4 sectors. This allows LVM volumes to co-exist with other users of these sectors, if necessary.

An LVM label provides correct identification and device ordering for a physical device, since devices can come up in any order when the system is booted. An LVM label remains persistent across reboots and throughout a cluster.

The LVM label identifies the device as an LVM physical volume. It contains a random unique identifier (the UUID) for the physical volume. It also stores the size of the block device in bytes, and it records where the LVM metadata will be stored on the device.

The LVM metadata contains the configuration details of the LVM volume groups on your system. By default, an identical copy of the metadata is maintained in every metadata area in every physical volume within the volume group. LVM metadata is small and stored as ASCII.

Currently LVM allows you to store 0, 1 or 2 identical copies of its metadata on each physical volume. The default is 1 copy. Once you configure the number of metadata copies on the physical volume, you cannot change that number at a later time. The first copy is stored at the start of the device, shortly after the label. If there is a second copy, it is placed at the end of the device. If you accidentally overwrite the area at the beginning of your disk by writing to a different disk than you intend, a second copy of the metadata at the end of the device will allow you to recover the metadata.

For detailed information about the LVM metadata and changing the metadata parameters, see *Appendix D, LVM Volume Group Metadata*.

2.1.1. LVM Physical Volume Layout

Figure 2.1, “Physical Volume layout” shows the layout of an LVM physical volume. The LVM label is on the second sector, followed by the metadata area, followed by the usable space on the device.

Note

In the Linux kernel (and throughout this document), sectors are considered to be 512 bytes in size.

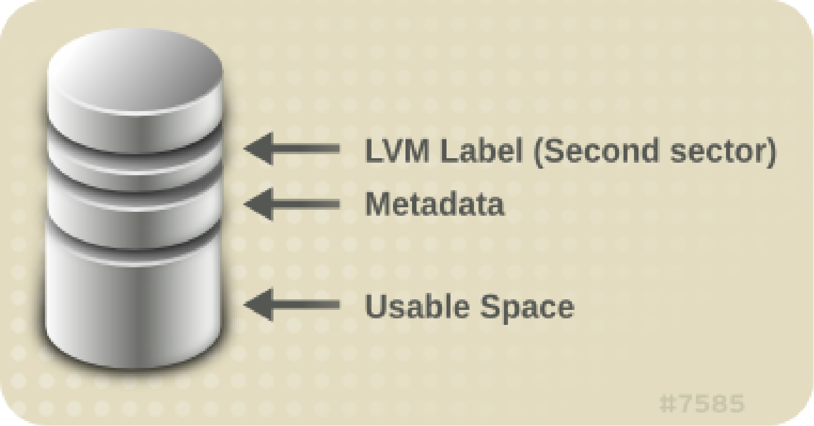

Figure 2.1. Physical Volume layout

2.1.2. Multiple Partitions on a Disk

LVM allows you to create physical volumes out of disk partitions. It is generally recommended that you create a single partition that covers the whole disk to label as an LVM physical volume for the following reasons:

- Administrative convenience

  It is easier to keep track of the hardware in a system if each real disk only appears once. This becomes particularly true if a disk fails. In addition, multiple physical volumes on a single disk may cause a kernel warning about unknown partition types at boot-up.

- Striping performance

  LVM cannot tell that two physical volumes are on the same physical disk. If you create a striped logical volume when two physical volumes are on the same physical disk, the stripes could be on different partitions on the same disk. This would result in a decrease in performance rather than an increase.

Although it is not recommended, there may be specific circumstances when you will need to divide a disk into separate LVM physical volumes. For example, on a system with few disks it may be necessary to move data around partitions when you are migrating an existing system to LVM volumes. Additionally, if you have a very large disk and want to have more than one volume group for administrative purposes then it is necessary to partition the disk. If you do have a disk with more than one partition and both of those partitions are in the same volume group, take care to specify which partitions are to be included in a logical volume when creating striped volumes.

2.2. Volume Groups

Physical volumes are combined into volume groups (VGs). This creates a pool of disk space out of which logical volumes can be allocated.

Within a volume group, the disk space available for allocation is divided into units of a fixed-size called extents. An extent is the smallest unit of space that can be allocated. Within a physical volume, extents are referred to as physical extents.

A logical volume is allocated into logical extents of the same size as the physical extents. The extent size is thus the same for all logical volumes in the volume group. The volume group maps the logical extents to physical extents.

2.3. LVM Logical Volumes

In LVM, a volume group is divided up into logical volumes. There are three types of LVM logical volumes: *linear* volumes, *striped* volumes, and *mirrored* volumes. These are described in the following sections.

2.3.1. Linear Volumes

A linear volume aggregates multiple physical volumes into one logical volume. For example, if you have two 60GB disks, you can create a 120GB logical volume. The physical storage is concatenated.

Creating a linear volume assigns a range of physical extents to an area of a logical volume in order. For example, as shown in *Figure 2.2, “Extent Mapping”* logical extents 1 to 99 could map to one physical volume and logical extents 100 to 198 could map to a second physical volume. From the point of view of the application, there is one device that is 198 extents in size.

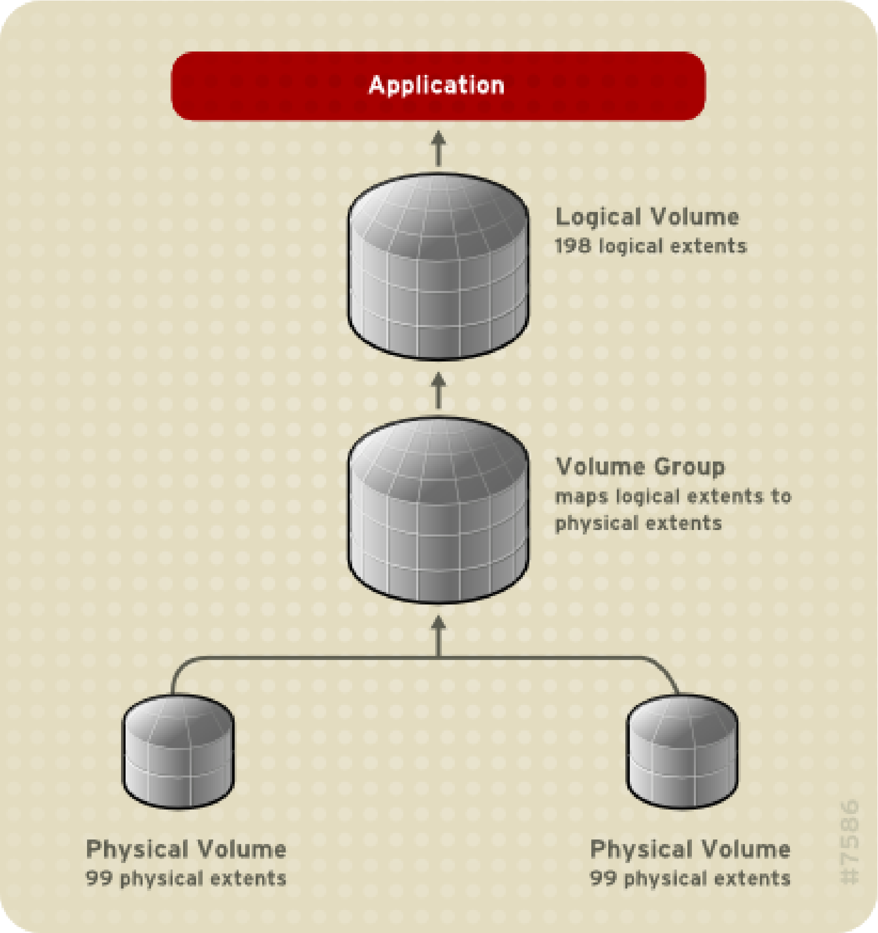

Figure 2.2. Extent Mapping

The physical volumes that make up a logical volume do not have to be the same size. *Figure 2.3, “Linear Volume with Unequal Physical Volumes”* shows volume group **VG1** with a physical extent size of 4MB. This volume group includes 2 physical volumes named **PV1** and **PV2**. The physical volumes are divided into 4MB units, since that is the extent size. In this example, **PV1** is 100 extents in size (400MB) and **PV2** is 200 extents in size (800MB). You can create a linear volume any size between 1 and 300 extents (4MB to 1200MB). In this example, the linear volume named **LV1** is 300 extents in size.

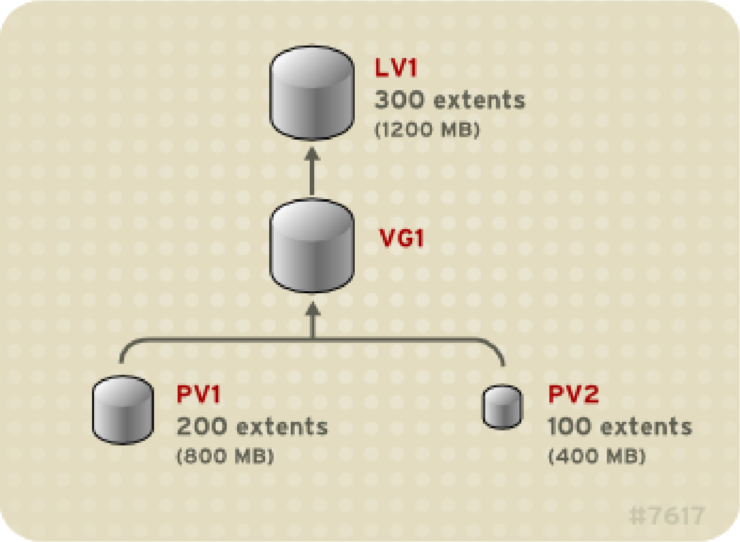

Figure 2.3. Linear Volume with Unequal Physical Volumes

You can configure more than one linear logical volume of whatever size you require from the pool of physical extents. *Figure 2.4, "Multiple Logical Volumes"* shows the same volume group as in *Figure 2.3, "Linear Volume with Unequal Physical Volumes"*, but in this case two logical volumes have been carved out of the volume group: **LV1**, which is 250 extents in size (1000MB) and **LV2** which is 50 extents in size (200MB).

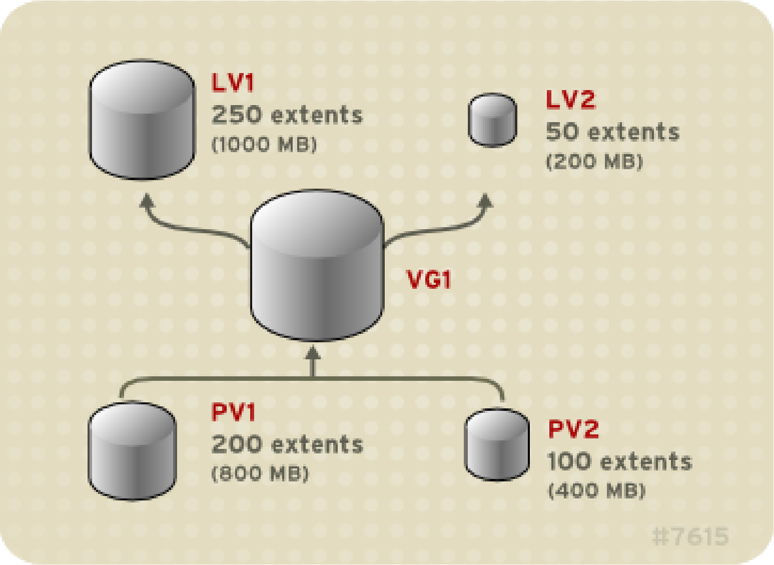

Figure 2.4. Multiple Logical Volumes

2.3.2. Striped Logical Volumes

When you write data to an LVM logical volume, the file system lays the data out across the underlying physical volumes. You can control the way the data is written to the physical volumes by creating a striped logical volume. For large sequential reads and writes, this can improve the efficiency of the data I/O.

Striping enhances performance by writing data to a predetermined number of physical volumes in round-robin fashion. With striping, I/O can be done in parallel. In some situations, this can result in near-linear performance gain for each additional physical volume in the stripe.

The following illustration shows data being striped across three physical volumes. In this figure:

- the first stripe of data is written to PV1
- the second stripe of data is written to PV2
- the third stripe of data is written to PV3
- the fourth stripe of data is written to PV1

In a striped logical volume, the size of the stripe cannot exceed the size of an extent.

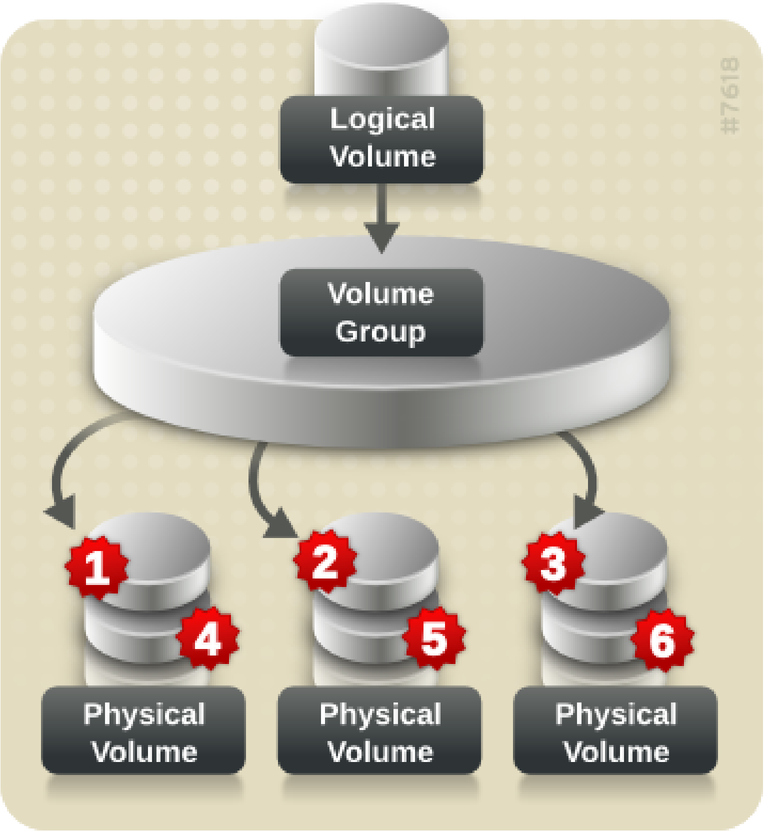

Figure 2.5. Striping Data Across Three PVs

Striped logical volumes can be extended by concatenating another set of devices onto the end of the first set. In order extend a striped logical volume, however, there must be enough free space on the underlying physical volumes that make up the volume group to support the stripe. For example, if you have a two-way stripe that uses up an entire volume group, adding a single physical volume to the volume group will not enable you to extend the stripe. Instead, you must add at least two physical volumes to the volume group. For more information on extending a striped volume, see *Section 4.4.12.1, “Extending a Striped Volume”*.

2.3.3. Mirrored Logical Volumes

A mirror maintains identical copies of data on different devices. When data is written to one device, it is written to a second device as well, mirroring the data. This provides protection for device failures. When one leg of a mirror fails, the logical volume becomes a linear volume and can still be accessed.

LVM supports mirrored volumes. When you create a mirrored logical volume, LVM ensures that data written to an underlying physical volume is mirrored onto a separate physical volume. With LVM, you can create mirrored logical volumes with multiple mirrors.

An LVM mirror divides the device being copied into regions that are typically 512KB in size. LVM maintains a small log which it uses to keep track of which regions are in sync with the mirror or mirrors. This log can be kept on disk, which will keep it persistent across reboots, or it can be maintained in memory.

Figure 2.6, “Mirrored Logical Volume” shows a mirrored logical volume with one mirror. In this configuration, the log is maintained on disk.

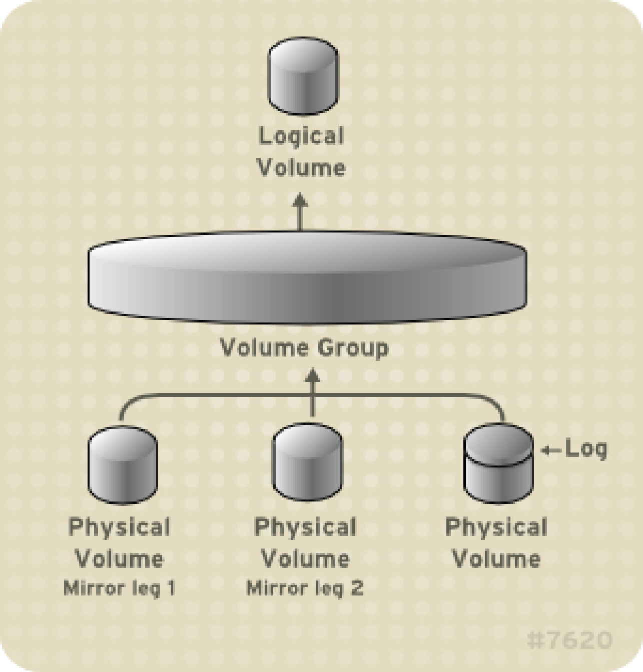

Figure 2.6. Mirrored Logical Volume

For information on creating and modifying mirrors, see *Section 4.4.3, “Creating Mirrored Volumes”*.

2.3.4. Snapshot Volumes

The LVM snapshot feature provides the ability to create virtual images of a device at a particular instant without causing a service interruption. When a change is made to the original device (the origin) after a snapshot is taken, the snapshot feature makes a copy of the changed data area as it was prior to the change so that it can reconstruct the state of the device.

Note

LVM snapshots are not supported across the nodes in a cluster. You cannot create a snapshot volume in a clustered volume group.

Note

LVM snapshots are not supported for LVM mirrored logical volumes.

Because a snapshot copies only the data areas that change after the snapshot is created, the snapshot feature requires a minimal amount of storage. For example, with a rarely updated origin, 3-5 % of the origin's capacity is sufficient to maintain the snapshot.

Note

Snapshot copies of a file system are virtual copies, not actual media backup for a file system. Snapshots do not provide a substitute for a backup procedure.

The size of the snapshot governs the amount of space set aside for storing the changes to the origin volume. For example, if you made a snapshot and then completely overwrote the origin the snapshot would have to be at least as big as the origin volume to hold the changes. You need to dimension a snapshot according to the expected level of change. So for example a short-lived snapshot of a read-mostly volume, such as **`/usr`**, would need less space than a long-lived snapshot of a volume that sees a greater number of writes, such as **`/home`**.

If a snapshot runs full, the snapshot becomes invalid, since it can no longer track changes on the origin volume. You should regularly monitor the size of the snapshot. Snapshots are fully resizeable, however, so if you have the storage capacity you can increase the size of the snapshot volume to prevent it from getting dropped. Conversely, if you find that the snapshot volume is larger than you need, you can reduce the size of the volume to free up space that is needed by other logical volumes.

When you create a snapshot file system, full read and write access to the origin stays possible. If a chunk on a snapshot is changed, that chunk is marked and never gets copied from the original volume.

There are several uses for the snapshot feature:

- Most typically, a snapshot is taken when you need to perform a backup on a logical volume without halting the live system that is continuously updating the data.
- You can execute the **`fsck`** command on a snapshot file system to check the file system integrity and determine whether the original file system requires file system repair.
- Because the snapshot is read/write, you can test applications against production data by taking a snapshot and running tests against the snapshot, leaving the real data untouched.
- You can create LVM volumes for use with Red Hat virtualization. LVM snapshots can be used to create snapshots of virtual guest images. These snapshots can provide a convenient way to modify existing guests or create new guests with minimal additional storage. For information on creating LVM-based storage pools with Red Hat Virtualization, see the *Virtualization Administration Guide*.

For information on creating snapshot volumes, see *Section 4.4.4, “Creating Snapshot Volumes”*.

As of the Red Hat Enterprise Linux 6 release, you can use the **`--merge`** option of the **`lvconvert`** command to merge a snapshot into its origin volume. One use for this feature is to perform system rollback if you have lost data or files or otherwise need to restore your system to a previous state. After you merge the snapshot volume, the resulting logical volume will have the origin volume's name, minor number, and UUID and the merged snapshot is removed. For information on using this option, see *Section 4.4.5, “Merging Snapshot Volumes”*.

LVM Administration Overview

This chapter provides an overview of the administrative procedures you use to configure LVM logical volumes. This chapter is intended to provide a general understanding of the steps involved. For specific step-by-step examples of common LVM configuration procedures, see *Chapter 5, LVM Configuration Examples*.

For descriptions of the CLI commands you can use to perform LVM administration, see *Chapter 4, LVM Administration with CLI Commands*. Alternately, you can use the LVM GUI, which is described in *Chapter 7, LVM Administration with the LVM GUI*.

3.1. Creating LVM Volumes in a Cluster

To create logical volumes in a cluster environment, you use the Clustered Logical Volume Manager (CLVM), which is a set of clustering extensions to LVM. These extensions allow a cluster of computers to manage shared storage (for example, on a SAN) using LVM. In order to use CLVM, the High Availability Add-On and Resilient Storage Add-On software, including the **`clvmd`** daemon, must be started at boot time, as described in *Section 1.4, “The Clustered Logical Volume Manager (CLVM)”*.

Creating LVM logical volumes in a cluster environment is identical to creating LVM logical volumes on a single node. There is no difference in the LVM commands themselves, or in the LVM GUI interface. In order to enable the LVM volumes you are creating in a cluster, the cluster infrastructure must be running and the cluster must be quorate.

CLVM requires changes to the **`lvm.conf`** file for cluster-wide locking. Information on configuring the **`lvm.conf`** file to support clustered locking is provided within the **`lvm.conf`** file itself. For information about the **`lvm.conf`** file, see *Appendix B, The LVM Configuration Files*.

By default, logical volumes created with CLVM on shared storage are visible to all systems that have access to the shared storage. It is possible to create volume groups in which all of the storage devices are visible to only one node in the cluster. It is also possible to change the status of a volume group from a local volume group to a clustered volume group. For information, see *Section 4.3.2, “Creating Volume Groups in a Cluster”* and *Section 4.3.7, “Changing the Parameters of a Volume Group”*

Warning

When you create volume groups with CLVM on shared storage, you must ensure that all nodes in the cluster have access to the physical volumes that constitute the volume group. Asymmetric cluster configurations in which some nodes have access to the storage and others do not are not supported.

For information on how to install the High Availability Add-On and set up the cluster infrastructure, see *Cluster Administration*.

For an example of creating a mirrored logical volume in a cluster, see *Section 5.5, “Creating a Mirrored LVM Logical Volume in a Cluster”*.

3.2. Logical Volume Creation Overview

The following is a summary of the steps to perform to create an LVM logical volume.

1. Initialize the partitions you will use for the LVM volume as physical volumes (this labels them).
2. Create a volume group.
3. Create a logical volume.

After creating the logical volume you can create and mount the file system. The examples in this document use GFS2 file systems.

Note

Although a GFS2 file system can be implemented in a standalone system or as part of a cluster configuration, for the Red Hat Enterprise Linux 6 release Red Hat does not support the use of GFS2 as a single-node file system. Red Hat will continue to support single-node GFS2 file systems for mounting snapshots of cluster file systems (for example, for backup purposes).

1. Create a GFS2 file system on the logical volume with the **`mkfs.gfs2`** command.
2. Create a new mount point with the **`mkdir`** command. In a clustered system, create the mount point on all nodes in the cluster.
3. Mount the file system. You may want to add a line to the **`fstab`** file for each node in the system.

Alternately, you can create and mount the GFS2 file system with the LVM GUI.

Creating the LVM volume is machine independent, since the storage area for LVM setup information is on the physical volumes and not the machine where the volume was created. Servers that use the storage have local copies, but can recreate that from what is on the physical volumes. You can attach physical volumes to a different server if the LVM versions are compatible.

3.3. Growing a File System on a Logical Volume

To grow a file system on a logical volume, perform the following steps:

1. Make a new physical volume.
2. Extend the volume group that contains the logical volume with the file system you are growing to include the new physical volume.
3. Extend the logical volume to include the new physical volume.
4. Grow the file system.

If you have sufficient unallocated space in the volume group, you can use that space to extend the logical volume instead of performing steps 1 and 2.

3.4. Logical Volume Backup

Metadata backups and archives are automatically created on every volume group and logical volume configuration change unless disabled in the **`lvm.conf`** file. By default, the metadata backup is stored in the **`/etc/lvm/backup`** file and the metadata archives are stored in the **`/etc/lvm/archive`** file. How long the metadata archives stored in the **`/etc/lvm/archive`** file are kept and how many archive files are kept is determined by parameters you can set in the **`lvm.conf`** file. A daily system backup should include the contents of the **`/etc/lvm`** directory in the backup.

Note that a metadata backup does not back up the user and system data contained in the logical volumes.

You can manually back up the metadata to the **`/etc/lvm/backup`** file with the **`vgcfgbackup`** command. You can restore metadata with the **`vgcfgrestore`** command. The **`vgcfgbackup`** and **`vgcfgrestore`** commands are described in *Section 4.3.12, "Backing Up Volume Group Metadata"*.

3.5. Logging

All message output passes through a logging module with independent choices of logging levels for:

- standard output/error
- syslog
- log file
- external log function

The logging levels are set in the **`/etc/lvm/lvm.conf`** file, which is described in *Appendix B, The LVM Configuration Files*.

LVM Administration with CLI Commands

This chapter summarizes the individual administrative tasks you can perform with the LVM Command Line Interface (CLI) commands to create and maintain logical volumes.

Note

If you are creating or modifying an LVM volume for a clustered environment, you must ensure that you are running the **`clvmd`** daemon. For information, see *Section 3.1, “Creating LVM Volumes in a Cluster”*.

4.1. Using CLI Commands

There are several general features of all LVM CLI commands.

When sizes are required in a command line argument, units can always be specified explicitly. If you do not specify a unit, then a default is assumed, usually KB or MB. LVM CLI commands do not accept fractions.

When specifying units in a command line argument, LVM is case-insensitive; specifying M or m is equivalent, for example, and powers of 2 (multiples of 1024) are used. However, when specifying the **`--units`** argument in a command, lower-case indicates that units are in multiples of 1024 while upper-case indicates that units are in multiples of 1000.

Where commands take volume group or logical volume names as arguments, the full path name is optional. A logical volume called **`lvol0`** in a volume group called **`vg0`** can be specified as **`vg0/lvol0`**. Where a list of volume groups is required but is left empty, a list of all volume groups will be substituted. Where a list of logical volumes is required but a volume group is given, a list of all the logical volumes in that volume group will be substituted. For example, the **`lvdisplay vg0`** command will display all the logical volumes in volume group **`vg0`**.

All LVM commands accept a **`-v`** argument, which can be entered multiple times to increase the output verbosity. For example, the following examples shows the default output of the **`lvcreate`** command.

```
# lvcreate -L 50MB new_vg
  Rounding up size to full physical extent 52.00 MB
  Logical volume "lvol0" created
```

The following command shows the output of the **`lvcreate`** command with the **`-v`** argument.

```
# lvcreate -v -L 50MB new_vg
    Finding volume group "new_vg"
  Rounding up size to full physical extent 52.00 MB
    Archiving volume group "new_vg" metadata (seqno 4).
    Creating logical volume lvol0
    Creating volume group backup "/etc/lvm/backup/new_vg" (seqno 5).
    Found volume group "new_vg"
    Creating new_vg-lvol0
    Loading new_vg-lvol0 table
    Resuming new_vg-lvol0 (253:2)
    Clearing start of logical volume "lvol0"
    Creating volume group backup "/etc/lvm/backup/new_vg" (seqno 5).
  Logical volume "lvol0" created
```

You could also have used the **`-vv`**, **`-vvv`** or the **`-vvvv`** argument to display increasingly more details about the command execution. The **`-vvvv`** argument provides the maximum amount of information at this time. The following example shows only the first few lines of output for the **`lvcreate`** command with the **`-vvvv`** argument specified.

```
# lvcreate -vvvv -L 50MB new_vg
#lvmcmdline.c:913         Processing: lvcreate -vvvv -L 50MB new_vg
#lvmcmdline.c:916         O_DIRECT will be used
#config/config.c:864       Setting global/locking_type to 1
#locking/locking.c:138       File-based locking selected.
#config/config.c:841       Setting global/locking_dir to /var/lock/lvm
#activate/activate.c:358       Getting target version for linear
#ioctl/libdm-iface.c:1569         dm version   OF   [16384]
#ioctl/libdm-iface.c:1569         dm versions   OF   [16384]
#activate/activate.c:358       Getting target version for striped
#ioctl/libdm-iface.c:1569         dm versions   OF   [16384]
#config/config.c:864       Setting activation/mirror_region_size to 512
...
```

You can display help for any of the LVM CLI commands with the **`--help`** argument of the command.

```
# commandname --help
```

To display the man page for a command, execute the **`man`** command:

```
# man commandname
```

The **`man lvm`** command provides general online information about LVM.

All LVM objects are referenced internally by a UUID, which is assigned when you create the object. This can be useful in a situation where you remove a physical volume called **`/dev/sdf`** which is part of a volume group and, when you plug it back in, you find that it is now **`/dev/sdk`**. LVM will still find the physical volume because it identifies the physical volume by its UUID and not its device name. For information on specifying the UUID of a physical volume when creating a physical volume, see *Section 6.4, “Recovering Physical Volume Metadata”*.

4.2. Physical Volume Administration

This section describes the commands that perform the various aspects of physical volume administration.

4.2.1. Creating Physical Volumes

The following subsections describe the commands used for creating physical volumes.

4.2.1.1. Setting the Partition Type

If you are using a whole disk device for your physical volume, the disk must have no partition table. For DOS disk partitions, the partition id should be set to 0x8e using the **`fdisk`** or **`cfdisk`** command or an equivalent. For whole disk devices only the partition table must be erased, which will effectively destroy all data on that disk. You can remove an existing partition table by zeroing the first sector with the following command:

```
# dd if=/dev/zero of=PhysicalVolume bs=512 count=1
```

4.2.1.2. Initializing Physical Volumes

Use the **`pvcreate`** command to initialize a block device to be used as a physical volume. Initialization is analogous to formatting a file system.

The following command initializes **`/dev/sdd`**, **`/dev/sde`**, and **`/dev/sdf`** as LVM physical volumes for later use as part of LVM logical volumes.

```
# pvcreate /dev/sdd /dev/sde /dev/sdf
```

To initialize partitions rather than whole disks: run the **`pvcreate`** command on the partition. The following example initializes the partition **`/dev/hdb1`** as an LVM physical volume for later use as part of an LVM logical volume.

```
# pvcreate /dev/hdb1
```

4.2.1.3. Scanning for Block Devices

You can scan for block devices that may be used as physical volumes with the **`lvmdiskscan`** command, as shown in the following example.

```
# lvmdiskscan
/dev/ram0                    [       16.00 MB]
  /dev/sda                     [       17.15 GB]
  /dev/root                    [       13.69 GB]
  /dev/ram                     [       16.00 MB]
  /dev/sda1                    [       17.14 GB] LVM physical volume
  /dev/VolGroup00/LogVol01     [      512.00 MB]
  /dev/ram2                    [       16.00 MB]
  /dev/new_vg/lvol0            [       52.00 MB]
  /dev/ram3                    [       16.00 MB]
  /dev/pkl_new_vg/sparkie_lv   [        7.14 GB]
  /dev/ram4                    [       16.00 MB]
  /dev/ram5                    [       16.00 MB]
  /dev/ram6                    [       16.00 MB]
  /dev/ram7                    [       16.00 MB]
  /dev/ram8                    [        16.00 MB]
  /dev/ram9                    [        16.00 MB]
  /dev/ram10                   [        16.00 MB]
  /dev/ram11                   [        16.00 MB]
  /dev/ram12                   [        16.00 MB]
  /dev/ram13                   [        16.00 MB]
  /dev/ram14                   [        16.00 MB]
  /dev/ram15                   [        16.00 MB]
  /dev/sdb                     [        17.15 GB]
  /dev/sdb1                    [        17.14 GB] LVM physical volume
  /dev/sdc                     [        17.15 GB]
  /dev/sdc1                    [        17.14 GB] LVM physical volume
  /dev/sdd                     [        17.15 GB]
  /dev/sdd1                    [        17.14 GB] LVM physical volume
  7 disks
  17 partitions
  0 LVM physical volume whole disks
  4 LVM physical volumes
```

4.2.2. Displaying Physical Volumes

There are three commands you can use to display properties of LVM physical volumes: **`pvs`**, **`pvdisplay`**, and **`pvscan`**.

The **`pvs`** command provides physical volume information in a configurable form, displaying one line per physical volume. The **`pvs`** command provides a great deal of format control, and is useful for scripting. For information on using the **`pvs`** command to customize your output, see *Section 4.8, “Customized Reporting for LVM”*.

The **`pvdisplay`** command provides a verbose multi-line output for each physical volume. It displays physical properties (size, extents, volume group, etc.) in a fixed format.

The following example shows the output of the **`pvdisplay`** command for a single physical volume.

```
# pvdisplay
  --- Physical volume ---
  PV Name               /dev/sdc1
  VG Name               new_vg
  PV Size               17.14 GB / not usable 3.40 MB
  Allocatable           yes
  PE Size (KByte)       4096
  Total PE              4388
  Free PE               4375
  Allocated PE          13
  PV UUID               Joqlch-yWSj-kuEn-IdwM-01S9-XO8M-mcpsVe
```

The **`pvscan`** command scans all supported LVM block devices in the system for physical volumes.

The following command shows all physical devices found:

```
# pvscan
 PV /dev/sdb2   VG vg0   lvm2 [964.00 MB / 0    free]
 PV /dev/sdc1   VG vg0   lvm2 [964.00 MB / 428.00 MB free]
 PV /dev/sdc2            lvm2 [964.84 MB]
 Total: 3 [2.83 GB] / in use: 2 [1.88 GB] / in no VG: 1 [964.84 MB]
```

You can define a filter in the **`lvm.conf`** so that this command will avoid scanning specific physical volumes. For information on using filters to control which devices are scanned, see *Section 4.5, “Controlling LVM Device Scans with Filters”*.

4.2.3. Preventing Allocation on a Physical Volume

You can prevent allocation of physical extents on the free space of one or more physical volumes with the **`pvchange`** command. This may be necessary if there are disk errors, or if you will be removing the physical volume.

The following command disallows the allocation of physical extents on **`/dev/sdk1`**.

```
# pvchange -x n /dev/sdk1
```

You can also use the **`-xy`** arguments of the **`pvchange`** command to allow allocation where it had previously been disallowed.

4.2.4. Resizing a Physical Volume

If you need to change the size of an underlying block device for any reason, use the **`pvresize`** command to update LVM with the new size. You can execute this command while LVM is using the physical volume.

4.2.5. Removing Physical Volumes

If a device is no longer required for use by LVM, you can remove the LVM label with the **`pvremove`** command. Executing the **`pvremove`** command zeroes the LVM metadata on an empty physical volume.

If the physical volume you want to remove is currently part of a volume group, you must remove it from the volume group with the **`vgreduce`** command, as described in *Section 4.3.6, “Removing Physical Volumes from a Volume Group”*.

```
# pvremove /dev/ram15
  Labels on physical volume "/dev/ram15" successfully wiped
```

4.3. Volume Group Administration

This section describes the commands that perform the various aspects of volume group administration.

4.3.1. Creating Volume Groups

To create a volume group from one or more physical volumes, use the **`vgcreate`** command. The **`vgcreate`** command creates a new volume group by name and adds at least one physical volume to it.

The following command creates a volume group named **`vg1`** that contains physical volumes **`/dev/sdd1`** and **`/dev/sde1`**.

```
# vgcreate vg1 /dev/sdd1 /dev/sde1
```

When physical volumes are used to create a volume group, its disk space is divided into 4MB extents, by default. This extent is the minimum amount by which the logical volume may be increased or decreased in size. Large numbers of extents will have no impact on I/O performance of the logical volume.

You can specify the extent size with the **`-s`** option to the **`vgcreate`** command if the default extent size is not suitable. You can put limits on the number of physical or logical volumes the volume group can have by using the **`-p`** and **`-l`** arguments of the **`vgcreate`** command.

By default, a volume group allocates physical extents according to common-sense rules such as not placing parallel stripes on the same physical volume. This is the **`normal`** allocation policy. You can use the **`--alloc`** argument of the **`vgcreate`** command to specify an allocation policy of **`contiguous`**, **`anywhere`**, or **`cling`**.

The **`contiguous`** policy requires that new extents are adjacent to existing extents. If there are sufficient free extents to satisfy an allocation request but a **`normal`** allocation policy would not use them, the **`anywhere`** allocation policy will, even if that reduces performance by placing two stripes on the same physical volume. The **`cling`** policy places new extents on the same physical volume as existing extents in the same stripe of the logical volume. These policies can be changed using the **`vgchange`** command.

For information on using the **`cling`** policy in conjunction with LVM tags to specify which additional physical volumes to use when extending an LVM volume, see *Section 4.4.12.2, “Extending a Logical Volume with the **`cling`** Allocation Policy”*.

In general, allocation policies other than **`normal`** are required only in special cases where you need to specify unusual or nonstandard extent allocation.

LVM volume groups and underlying logical volumes are included in the device special file directory tree in the **`/dev`** directory with the following layout:

```
/dev/vg/lv/
```

For example, if you create two volume groups **`myvg1`** and **`myvg2`**, each with three logical volumes named **`lvo1`**, **`lvo2`**, and **`lvo3`**, this create six device special files:

```
/dev/myvg1/lv01
/dev/myvg1/lv02
/dev/myvg1/lv03
/dev/myvg2/lv01
/dev/myvg2/lv02
/dev/myvg2/lv03
```

The maximum device size with LVM is 8 Exabytes on 64-bit CPUs.

4.3.2. Creating Volume Groups in a Cluster

You create volume groups in a cluster environment with the **`vgcreate`** command, just as you create them on a single node.

By default, volume groups created with CLVM on shared storage are visible to all computers that have access to the shared storage. It is possible, however, to create volume groups that are local, visible only to one node in the cluster, by using the **`-c n`** of the **`vgcreate`** command.

The following command, when executed in a cluster environment, creates a volume group that is local to the node from which the command was executed. The command creates a local volume named **`vg1`** that contains physical volumes **`/dev/sdd1`** and **`/dev/sde1`**.

```
# vgcreate -c n vg1 /dev/sdd1 /dev/sde1
```

You can change whether an existing volume group is local or clustered with the **`-c`** option of the **`vgchange`** command, which is described in *Section 4.3.7, “Changing the Parameters of a Volume Group”*.

You can check whether an existing volume group is a clustered volume group with the **`vgs`** command, which displays the **`c`** attribute if the volume is clustered. The following command displays the attributes of the volume groups **`VolGroup00`** and **`testvg1`**. In this example, **`VolGroup00`** is not clustered, while **`testvg1`** is clustered, as indicated by the **`c`** attribute under the **`Attr`** heading.

```
[root@doc-07]# vgs
  VG            #PV #LV #SN Attr   VSize  VFree
  VolGroup00      1   2   0 wz--n- 19.88G    0
  testvg1         1   1   0 wz--nc 46.00G 8.00M
```

For more information on the **`vgs`** command, see *Section 4.3.4, “Displaying Volume Groups”Section 4.8, “Customized Reporting for LVM”*, and the **`vgs`** man page.

4.3.3. Adding Physical Volumes to a Volume Group

To add additional physical volumes to an existing volume group, use the **`vgextend`** command. The **`vgextend`** command increases a volume group's capacity by adding one or more free physical volumes.

The following command adds the physical volume **`/dev/sdf1`** to the volume group **`vg1`**.

```
# vgextend vg1 /dev/sdf1
```

4.3.4. Displaying Volume Groups

There are two commands you can use to display properties of LVM volume groups: **`vgs`** and **`vgdisplay`**.

The **`vgscan`** command, which scans all the disks for volume groups and rebuilds the LVM cache file, also displays the volume groups. For information on the **`vgscan`** command, see *Section 4.3.5, “Scanning Disks for Volume Groups to Build the Cache File”*.

The **`vgs`** command provides volume group information in a configurable form, displaying one line per volume group. The **`vgs`** command provides a great deal of format control, and is useful for scripting.

For information on using the **vgs** command to customize your output, see *Section 4.8, “Customized Reporting for LVM”*.

The **vgdisplay** command displays volume group properties (such as size, extents, number of physical volumes, etc.) in a fixed form. The following example shows the output of a **vgdisplay** command for the volume group **new_vg**. If you do not specify a volume group, all existing volume groups are displayed.

```
# vgdisplay new_vg
  --- Volume group ---
  VG Name               new_vg
  System ID
  Format                lvm2
  Metadata Areas        3
  Metadata Sequence No  11
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                1
  Open LV               0
  Max PV                0
  Cur PV                3
  Act PV                3
  VG Size               51.42 GB
  PE Size               4.00 MB
  Total PE              13164
  Alloc PE / Size       13 / 52.00 MB
  Free  PE / Size       13151 / 51.37 GB
  VG UUID               jxQJ0a-ZKk0-OpMO-0118-nlwO-wwqd-fD5D32
```

4.3.5. Scanning Disks for Volume Groups to Build the Cache File

The **vgscan** command scans all supported disk devices in the system looking for LVM physical volumes and volume groups. This builds the LVM cache in the **/etc/lvm/cache/.cache** file, which maintains a listing of current LVM devices.

LVM runs the **vgscan** command automatically at system startup and at other times during LVM operation, such as when you execute a **vgcreate** command or when LVM detects an inconsistency.

Note

You may need to run the **vgscan** command manually when you change your hardware configuration and add or delete a device from a node, causing new devices to be visible to the system that were not present at system bootup. This may be necessary, for example, when you add new disks to the system on a SAN or hotplug a new disk that has been labeled as a physical volume.

You can define a filter in the **lvm.conf** file to restrict the scan to avoid specific devices. For information on using filters to control which devices are scanned, see *Section 4.5, “Controlling LVM Device Scans with Filters”*.

The following example shows the output of a **vgscan** command.

```
# vgscan
  Reading all physical volumes.  This may take a while...
  Found volume group "new_vg" using metadata type lvm2
  Found volume group "officevg" using metadata type lvm2
```

4.3.6. Removing Physical Volumes from a Volume Group

To remove unused physical volumes from a volume group, use the **`vgreduce`** command. The **`vgreduce`** command shrinks a volume group's capacity by removing one or more empty physical volumes. This frees those physical volumes to be used in different volume groups or to be removed from the system.

Before removing a physical volume from a volume group, you can make sure that the physical volume is not used by any logical volumes by using the **`pvdisplay`** command.

```
# pvdisplay /dev/hda1

-- Physical volume ---
PV Name               /dev/hda1
VG Name               myvg
PV Size               1.95 GB / NOT usable 4 MB [LVM: 122 KB]
PV#                   1
PV Status             available
Allocatable           yes (but full)
Cur LV                1
PE Size (KByte)       4096
Total PE              499
Free PE               0
Allocated PE          499
PV UUID               Sd44tK-9IRw-SrMC-MOkn-76iP-iftz-OVSen7
```

If the physical volume is still being used you will have to migrate the data to another physical volume using the **`pvmove`** command. Then use the **`vgreduce`** command to remove the physical volume:

The following command removes the physical volume **`/dev/hda1`** from the volume group **`my_volume_group`**.

```
# vgreduce my_volume_group /dev/hda1
```

4.3.7. Changing the Parameters of a Volume Group

The **`vgchange`** command is used to deactivate and activate volume groups, as described in *Section 4.3.8, “Activating and Deactivating Volume Groups”*. You can also use this command to change several volume group parameters for an existing volume group.

The following command changes the maximum number of logical volumes of volume group **`vg00`** to 128.

```
# vgchange -l 128 /dev/vg00
```

For a description of the volume group parameters you can change with the **`vgchange`** command, see the **`vgchange`**(8) man page.

4.3.8. Activating and Deactivating Volume Groups

When you create a volume group it is, by default, activated. This means that the logical volumes in that group are accessible and subject to change.

There are various circumstances for which you need to make a volume group inactive and thus unknown to the kernel. To deactivate or activate a volume group, use the **`-a`** (**`--available`**) argument of the **`vgchange`** command.

The following example deactivates the volume group **`my_volume_group`**.

```
# vgchange -a n my_volume_group
```

If clustered locking is enabled, add ’e’ to activate or deactivate a volume group exclusively on one node or ’l’ to activate or/deactivate a volume group only on the local node. Logical volumes with single-host snapshots are always activated exclusively because they can only be used on one node at once.

You can deactivate individual logical volumes with the **`lvchange`** command, as described in *Section 4.4.8, “Changing the Parameters of a Logical Volume Group”*, For information on activating logical volumes on individual nodes in a cluster, see *Section 4.7, “Activating Logical Volumes on Individual Nodes in a Cluster”*.

4.3.9. Removing Volume Groups

To remove a volume group that contains no logical volumes, use the **`vgremove`** command.

```
# vgremove officevg
  Volume group "officevg" successfully removed
```

4.3.10. Splitting a Volume Group

To split the physical volumes of a volume group and create a new volume group, use the **`vgsplit`** command.

Logical volumes cannot be split between volume groups. Each existing logical volume must be entirely on the physical volumes forming either the old or the new volume group. If necessary, however, you can use the **`pvmove`** command to force the split.

The following example splits off the new volume group **`smallvg`** from the original volume group **`bigvg`**.

```
# vgsplit bigvg smallvg /dev/ram15
  Volume group "smallvg" successfully split from "bigvg"
```

4.3.11. Combining Volume Groups

To combine two volume groups into a single volume group, use the **`vgmerge`** command. You can merge an inactive "source" volume with an active or an inactive "destination" volume if the physical extent sizes of the volume are equal and the physical and logical volume summaries of both volume groups fit into the destination volume groups limits.

The following command merges the inactive volume group **my_vg** into the active or inactive volume group **databases** giving verbose runtime information.

```
# vgmerge -v databases my_vg
```

4.3.12. Backing Up Volume Group Metadata

Metadata backups and archives are automatically created on every volume group and logical volume configuration change unless disabled in the **lvm.conf** file. By default, the metadata backup is stored in the **/etc/lvm/backup** file and the metadata archives are stored in the **/etc/lvm/archives** file. You can manually back up the metadata to the **/etc/lvm/backup** file with the **vgcfgbackup** command.

The **vgcfrestore** command restores the metadata of a volume group from the archive to all the physical volumes in the volume groups.

For an example of using the **vgcfgrestore** command to recover physical volume metadata, see *Section 6.4, “Recovering Physical Volume Metadata”*.

4.3.13. Renaming a Volume Group

Use the **vgrename** command to rename an existing volume group.

Either of the following commands renames the existing volume group **vg02** to **my_volume_group**

```
# vgrename /dev/vg02 /dev/my_volume_group
```

```
# vgrename vg02 my_volume_group
```

4.3.14. Moving a Volume Group to Another System

You can move an entire LVM volume group to another system. It is recommended that you use the **vgexport** and **vgimport** commands when you do this.

The **vgexport** command makes an inactive volume group inaccessible to the system, which allows you to detach its physical volumes. The **vgimport** command makes a volume group accessible to a machine again after the **vgexport** command has made it inactive.

To move a volume group form one system to another, perform the following steps:

1. Make sure that no users are accessing files on the active volumes in the volume group, then unmount the logical volumes.

2. Use the **-a n** argument of the **vgchange** command to mark the volume group as inactive, which prevents any further activity on the volume group.

3. Use the **`vgexport`** command to export the volume group. This prevents it from being accessed by the system from which you are removing it.

   After you export the volume group, the physical volume will show up as being in an exported volume group when you execute the **`pvscan`** command, as in the following example.

   ```
   [root@tng3-1]# pvscan
     PV /dev/sda1    is in exported VG myvg [17.15 GB / 7.15 GB free]
     PV /dev/sdc1    is in exported VG myvg [17.15 GB / 15.15 GB free]
     PV /dev/sdd1   is in exported VG myvg [17.15 GB / 15.15 GB free]
     ...
   ```

   When the system is next shut down, you can unplug the disks that constitute the volume group and connect them to the new system.

4. When the disks are plugged into the new system, use the **`vgimport`** command to import the volume group, making it accessible to the new system.

5. Activate the volume group with the **`-a y`** argument of the **`vgchange`** command.

6. Mount the file system to make it available for use.

4.3.15. Recreating a Volume Group Directory

To recreate a volume group directory and logical volume special files, use the **`vgmknodes`** command. This command checks the LVM2 special files in the **`/dev`** directory that are needed for active logical volumes. It creates any special files that are missing removes unused ones.

You can incorporate the **`vgmknodes`** command into the **`vgscan`** command by specifying the **`mknodes`** argument to the **`vgscan`** command.

4.4. Logical Volume Administration

This section describes the commands that perform the various aspects of logical volume administration.

4.4.1. Creating Linear Logical Volumes

To create a logical volume, use the **`lvcreate`** command. If you do not specify a name for the logical volume, the default name **`lvol#`** is used where # is the internal number of the logical volume.

When you create a logical volume, the logical volume is carved from a volume group using the free extents on the physical volumes that make up the volume group. Normally logical volumes use up any space available on the underlying physical volumes on a next-free basis. Modifying the logical volume frees and reallocates space in the physical volumes.

The following command creates a logical volume 10 gigabytes in size in the volume group **`vg1`**.

```
# lvcreate -L 10G vg1
```

The following command creates a 1500 MB linear logical volume named **`testlv`** in the volume group **`testvg`**, creating the block device **`/dev/testvg/testlv`**.

```
# lvcreate -L1500 -n testlv testvg
```

The following command creates a 50 gigabyte logical volume named **`gfslv`** from the free extents in volume group **`vg0`**.

```
# lvcreate -L 50G -n gfslv vg0
```

You can use the **`-l`** argument of the **`lvcreate`** command to specify the size of the logical volume in extents. You can also use this argument to specify the percentage of the volume group to use for the logical volume. The following command creates a logical volume called **`mylv`** that uses 60% of the total space in volume group **`testvol`**.

```
# lvcreate -l 60%VG -n mylv testvg
```

You can also use the **`-l`** argument of the **`lvcreate`** command to specify the percentage of the remaining free space in a volume group as the size of the logical volume. The following command creates a logical volume called **`yourlv`** that uses all of the unallocated space in the volume group **`testvol`**.

```
# lvcreate -l 100%FREE -n yourlv testvg
```

You can use **`-l`** argument of the **`lvcreate`** command to create a logical volume that uses the entire volume group. Another way to create a logical volume that uses the entire volume group is to use the **`vgdisplay`** command to find the "Total PE" size and to use those results as input to the **`lvcreate`** command.

The following commands create a logical volume called **`mylv`** that fills the volume group named **`testvg`**.

```
# vgdisplay testvg | grep "Total PE"
Total PE              10230
# lvcreate -l 10230 testvg -n mylv
```

The underlying physical volumes used to create a logical volume can be important if the physical volume needs to be removed, so you may need to consider this possibility when you create the logical volume. For information on removing a physical volume from a volume group, see *Section 4.3.6, “Removing Physical Volumes from a Volume Group”*.

To create a logical volume to be allocated from a specific physical volume in the volume group, specify the physical volume or volumes at the end at the **`lvcreate`** command line. The following command creates a logical volume named **`testlv`** in volume group **`testvg`** allocated from the physical volume **`/dev/sdg1`**,

```
# lvcreate -L 1500 -ntestlv testvg /dev/sdg1
```

You can specify which extents of a physical volume are to be used for a logical volume. The following example creates a linear logical volume out of extents 0 through 24 of physical volume **/dev/sda1** and extents 50 through 124 of physical volume **/dev/sdb1** in volume group **testvg**.

```
# lvcreate -l 100 -n testlv testvg /dev/sda1:0-24 /dev/sdb1:50-124
```

The following example creates a linear logical volume out of extents 0 through 25 of physical volume **/dev/sda1** and then continues laying out the logical volume at extent 100.

```
# lvcreate -l 100 -n testlv testvg /dev/sda1:0-25:100-
```

The default policy for how the extents of a logical volume are allocated is **inherit**, which applies the same policy as for the volume group. These policies can be changed using the **lvchange** command. For information on allocation policies, see *Section 4.3.1, “Creating Volume Groups”*.

4.4.2. Creating Striped Volumes

For large sequential reads and writes, creating a striped logical volume can improve the efficiency of the data I/O. For general information about striped volumes, see *Section 2.3.2, “Striped Logical Volumes”*.

When you create a striped logical volume, you specify the number of stripes with the **-i** argument of the **lvcreate** command. This determines over how many physical volumes the logical volume will be striped. The number of stripes cannot be greater than the number of physical volumes in the volume group (unless the **--alloc anywhere** argument is used).

If the underlying physical devices that make up a striped logical volume are different sizes, the maximum size of the striped volume is determined by the smallest underlying device. For example, in a two-legged stripe, the maximum size is twice the size of the smaller device. In a three-legged stripe, the maximum size is three times the size of the smallest device.

The following command creates a striped logical volume across 2 physical volumes with a stripe of 64kB. The logical volume is 50 gigabytes in size, is named **gfslv**, and is carved out of volume group **vg0**.

```
# lvcreate -L 50G -i2 -I64 -n gfslv vg0
```

As with linear volumes, you can specify the extents of the physical volume that you are using for the stripe. The following command creates a striped volume 100 extents in size that stripes across two physical volumes, is named **stripelv** and is in volume group **testvg**. The stripe will use sectors 0-49 of **/dev/sda1** and sectors 50-99 of **/dev/sdb1**.

```
# lvcreate -l 100 -i2 -nstripelv testvg /dev/sda1:0-49 /dev/sdb1:50-99
  Using default stripesize 64.00 KB
  Logical volume "stripelv" created
```

4.4.3. Creating Mirrored Volumes

Mirrored LVM Logical Volumes in a Cluster

Creating a mirrored LVM logical volume in a cluster requires the same commands and procedures as creating a mirrored LVM logical volume on a single node. However, in order to create a mirrored LVM volume in a cluster the cluster and cluster mirror infrastructure must be running, the cluster must be quorate, and the locking type in the **`lvm.conf`** file must be set correctly to enable cluster locking. For an example of creating a mirrored volume in a cluster, see *Section 5.5, “Creating a Mirrored LVM Logical Volume in a Cluster”*.

Attempting to run multiple LVM mirror creation and conversion commands in quick succession from multiple nodes in a cluster might cause a backlog of these commands. This might cause some of the requested operations to time-out and, subsequently, fail. To avoid this issue, it is recommended that cluster mirror creation commands be executed from one node of the cluster.

When you create a mirrored volume, you specify the number of copies of the data to make with the **`-m`** argument of the **`lvcreate`** command. Specifying **`-m1`** creates one mirror, which yields two copies of the file system: a linear logical volume plus one copy. Similarly, specifying **`-m2`** creates two mirrors, yielding three copies of the file system.

The following command creates a mirrored logical volume with a single mirror. The volume is 50 gigabytes in size, is named **`mirrorlv`**, and is carved out of volume group **`vg0`**:

```
# lvcreate -L 50G -m1 -n mirrorlv vg0
```

An LVM mirror divides the device being copied into regions that, by default, are 512KB in size. You can use the **`-R`** argument of the **`lvcreate`** command to specify the region size in megabytes. You can also change the default region size by editing the **`mirror_region_size`** setting in the **`lvm.conf`** file.

Note

Due to limitations in the cluster infrastructure, cluster mirrors greater than 1.5TB cannot be created with the default region size of 512KB. Users that require larger mirrors should increase the region size from its default to something larger. Failure to increase the region size will cause LVM creation to hang and may hang other LVM commands as well.

As a general guideline for specifying the region size for mirrors that are larger than 1.5TB, you could take your mirror size in terabytes and round up that number to the next power of 2, using that number as the **`-R`** argument to the **`lvcreate`** command. For example, if your mirror size is 1.5TB, you could specify **`-R 2`**. If your mirror size is 3TB, you could specify **`-R 4`**. For a mirror size of 5TB, you could specify **`-R 8`**.

The following command creates a mirrored logical volume with a region size of 2MB:

```
# lvcreate -m1 -L 2T -R 2 -n mirror vol_group
```

LVM maintains a small log which it uses to keep track of which regions are in sync with the mirror or mirrors. By default, this log is kept on disk, which keeps it persistent across reboots and ensures that the mirror does not need to be re-synced every time a machine reboots or crashes. You can specify instead that this log be kept in memory with the **`--mirrorlog core`** argument; this eliminates the need for an extra log device, but it requires that the entire mirror be resynchronized at every reboot.

The following command creates a mirrored logical volume from the volume group **`bigvg`**. The logical volume is named **`ondiskmirvol`** and has a single mirror. The volume is 12MB in size and keeps the mirror log in memory.

```
# lvcreate -L 12MB -m1 --mirrorlog core -n ondiskmirvol bigvg
  Logical volume "ondiskmirvol" created
```

The mirror log is created on a separate device from the devices on which any of the mirror legs are created. It is possible, however, to create the mirror log on the same device as one of the mirror legs by using the **`--alloc anywhere`** argument of the **`vgcreate`** command. This may degrade performance, but it allows you to create a mirror even if you have only two underlying devices.

The following command creates a mirrored logical volume with a single mirror for which the mirror log is on the same device as one of the mirror legs. In this example, the volume group **`vg0`** consists of only two devices. This command creates a 500 MB volume named **`mirrorlv`** in the **`vg0`** volume group.

```
# lvcreate -L 500M -m1 -n mirrorlv -alloc anywhere vg0
```

Note

With clustered mirrors, the mirror log management is completely the responsibility of the cluster node with the currently lowest cluster ID. Therefore, when the device holding the cluster mirror log becomes unavailable on a subset of the cluster, the clustered mirror can continue operating without any impact, as long as the cluster node with lowest ID retains access to the mirror log. Since the mirror is undisturbed, no automatic corrective action (repair) is issued, either. When the lowest-ID cluster node loses access to the mirror log, however, automatic action will kick in (regardless of accessibility of the log from other nodes).

To create a mirror log that is itself mirrored, you can specify the **`--mirrorlog mirrored`** argument. The following command creates a mirrored logical volume from the volume group **`bigvg`**. The logical volume is named **`twologvol`** and has a single mirror. The volume is 12MB in size and the mirror log is mirrored, with each log kept on a separate device.

```
# lvcreate -L 12MB -m1 --mirrorlog mirrored -n twologvol bigvg
  Logical volume "twologvol" created
```

Just as with a standard mirror log, it is possible to create the redundant mirror logs on the same device as the mirror legs by using the **`--alloc anywhere`** argument of the **`vgcreate`** command. This may degrade performance, but it allows you to create a redundant mirror log even if you do not have sufficient underlying devices for each log to be kept on a separate device than the mirror legs.

When a mirror is created, the mirror regions are synchronized. For large mirror components, the sync process may take a long time. When you are creating a new mirror that does not need to be revived, you can specify the **`--nosync`** argument to indicate that an initial synchronization from the first device is not required.

You can specify which devices to use for the mirror legs and log, and which extents of the devices to use. To force the log onto a particular disk, specify exactly one extent on the disk on which it will be placed. LVM does not necessary respect the order in which devices are listed in the command line. If any physical volumes are listed that is the only space on which allocation will take place. Any physical extents included in the list that are already allocated will get ignored.

The following command creates a mirrored logical volume with a single mirror and a single log that is not mirrored. The volume is 500 MB in size, it is named **`mirrorlv`**, and it is carved out of volume group **`vg0`**. The first leg of the mirror is on device **`/dev/sda1`**, the second leg of the mirror is on device **`/dev/sdb1`**, and the mirror log is on **`/dev/sdc1`**.

```
# lvcreate -L 500M -m1 -n mirrorlv vg0 /dev/sda1 /dev/sdb1 /dev/sdc1
```

The following command creates a mirrored logical volume with a single mirror. The volume is 500 MB in size, it is named **`mirrorlv`**, and it is carved out of volume group **`vg0`**. The first leg of the mirror is on extents 0 through 499 of device **`/dev/sda1`**, the second leg of the mirror is on extents 0 through 499 of device **`/dev/sdb1`**, and the mirror log starts on extent 0 of device **`/dev/sdc1`**. These are 1MB extents. If any of the specified extents have already been allocated, they will be ignored.

```
# lvcreate -L 500M -m1 -n mirrorlv vg0 /dev/sda1:0-499 /dev/sdb1:0-499 /dev/sdc1:0
```

Note

As of the Red Hat Enterprise Linux 6.1 release, you can combine RAID0 (striping) and RAID1 (mirroring) in a single logical volume. Creating a logical volume while simultaneously specifying the number of mirrors (**`--mirrors X`**) and the number of stripes (**`--stripes Y`**) results in a mirror device whose constituent devices are striped.

4.4.3.1. Mirrored Logical Volume Failure Policy

You can define how a mirrored logical volume behaves in the event of a device failure with the **`mirror_image_fault_policy`** and **`mirror_log_fault_policy`** parameters in the **`activation`** section of the **`lvm.conf`** file. When these parameters are set to **`remove`**, the system attempts to remove the faulty device and run without it. When this parameter is set to **`allocate`**, the system attempts to remove the faulty device and tries to allocate space on a new device to be a replacement for the failed device; this policy acts like the **`remove`** policy if no suitable device and space can be allocated for the replacement.

By default, the **`mirror_log_fault_policy`** parameter is set to **`allocate`**. Using this policy for the log is fast and maintains the ability to remember the sync state through crashes and reboots. If you set this policy to **`remove`**, when a log device fails the mirror converts to using an in-memory log and the mirror will not remember its sync status across crashes and reboots and the entire mirror will be re-synced.

By default, the **`mirror_image_fault_policy`** parameter is set to **`remove`**. With this policy, if a mirror image fails the mirror will convert to a non-mirrored device if there is only one remaining good copy. Setting this policy to **`allocate`** for a mirror device requires the mirror to resynchronize the devices; this is a slow process, but it preserves the mirror characteristic of the device.

Note

When an LVM mirror suffers a device failure, a two-stage recovery takes place. The first stage involves removing the failed devices. This can result in the mirror being reduced to a linear device. The second stage, if the **`mirror_log_fault_policy`** parameter is set to **`allocate`**, is to attempt to replace any of the failed devices. Note, however, that there is no guarantee that the second stage will choose devices previously in-use by the mirror that had not been part of the failure if others are available.

For information on manually recovering from an LVM mirror failure, refer to *Section 6.3, “Recovering from LVM Mirror Failure”*.

4.4.3.2. Splitting Off a Redundant Image of a Mirrored Logical Volume

You can split off a redundant image of a mirrored logical volume to form a new logical volume. To split off an image, you use the **`--splitmirrors`** argument of the **`lvconvert`** command, specifying the number of redundant images to split off. You must use the **`--name`** argument of the command to specify a name for the newly-split-off logical volume.

The following command splits off a new logical volume named **`copy`** from the mirrored logical volume **`vg/lv`**. The new logical volume contains two mirror legs. In this example, LVM selects which devices to split off.

```
# lvconvert --splitmirrors 2 --name copy vg/lv
```

You can specify which devices to split off. The following command splits off a new logical volume named **`copy`** from the mirrored logical volume **`vg/lv`**. The new logical volume contains two mirror legs consisting of devices **`/dev/sdc1`** and **`/dev/sde1`**.

```
# lvconvert --splitmirrors 2 --name copy vg/lv /dev/sd[ce]1
```

4.4.3.3. Repairing a Mirrored Logical Device

You can use the **`lvconvert --repair`** command to repair a mirror after a disk failure. This brings the mirror back into a consistent state. The **`lvconvert --repair`** command is an interactive command that prompts you to indicate whether you want the system to attempt to replace any failed devices.

- To skip the prompts and replace all of the failed devices, specify the **`-y`** option on the command line.

- To skip the prompts and replace none of the failed devices, specify the **`-f`** option on the command line.

- To skip the prompts and still indicate different replacement policies for the mirror image and the mirror log, you can specify the **`--use-policies`** argument to use the device replacement policies specified by the **`mirror_log_fault_policy`** and **`mirror_device_fault_policy`** parameters in the **`lvm.conf`** file.

4.4.3.4. Changing Mirrored Volume Configuration

You can convert a logical volume from a mirrored volume to a linear volume or from a linear volume to a mirrored volume with the **`lvconvert`** command. You can also use this command to reconfigure other mirror parameters of an existing logical volume, such as **`corelog`**.

When you convert a logical volume to a mirrored volume, you are basically creating mirror legs for an existing volume. This means that your volume group must contain the devices and space for the mirror legs and for the mirror log.

If you lose a leg of a mirror, LVM converts the volume to a linear volume so that you still have access to the volume, without the mirror redundancy. After you replace the leg, you can use the **`lvconvert`** command to restore the mirror. This procedure is provided in *Section 6.3, “Recovering from LVM Mirror Failure”*.

The following command converts the linear logical volume **`vg00/lvol1`** to a mirrored logical volume.

```
# lvconvert -m1 vg00/lvol1
```

The following command converts the mirrored logical volume **`vg00/lvol1`** to a linear logical volume, removing the mirror leg.

```
# lvconvert -m0 vg00/lvol1
```

4.4.4. Creating Snapshot Volumes

Use the **`-s`** argument of the **`lvcreate`** command to create a snapshot volume. A snapshot volume is writable.

Note

LVM snapshots are not supported across the nodes in a cluster. You cannot create a snapshot volume in a clustered volume group. As of the Red Hat Enterprise Linux 6.1 release, however, if you need to create a consistent backup of data on a clustered logical volume you can activate the volume exclusively and then create the snapshot. For information on activating logical volumes exclusively on one node, see *Section 4.7, “Activating Logical Volumes on Individual Nodes in a Cluster”*.

Note

As of the Red Hat Enterprise Linux 6.1 release, LVM snapshots are supported for mirrored logical volumes.

The following command creates a snapshot logical volume that is 100 MB in size named **`/dev/vg00/snap`**. This creates a snapshot of the origin logical volume named **`/dev/vg00/lvol1`**. If the original logical volume contains a file system, you can mount the snapshot logical volume on an arbitrary directory in order to access the contents of the file system to run a backup while the original file system continues to get updated.

```
# lvcreate --size 100M --snapshot --name snap /dev/vg00/lvol1
```

After you create a snapshot logical volume, specifying the origin volume on the **`lvdisplay`** command yields output that includes a list of all snapshot logical volumes and their status (active or inactive).

The following example shows the status of the logical volume **`/dev/new_vg/lvol0`**, for which a snapshot volume **`/dev/new_vg/newvgsnap`** has been created.

```
# lvdisplay /dev/new_vg/lvol0
  --- Logical volume ---
  LV Name                /dev/new_vg/lvol0
  VG Name                new_vg
  LV UUID                LBy1Tz-sr23-OjsI-LT03-nHLC-y8XW-EhCl78
  LV Write Access        read/write
  LV snapshot status     source of
                         /dev/new_vg/newvgsnap1 [active]
  LV Status              available
  # open                 0
  LV Size                52.00 MB
  Current LE             13
  Segments               1
  Allocation             inherit
  Read ahead sectors     0
  Block device           253:2
```

The **`lvs`** command, by default, displays the origin volume and the current percentage of the snapshot volume being used for each snapshot volume. The following example shows the default output for the **`lvs`** command for a system that includes the logical volume **`/dev/new_vg/lvol0`**, for which a snapshot volume **`/dev/new_vg/newvgsnap`** has been created.

```
# lvs
  LV         VG     Attr   LSize  Origin Snap%  Move Log Copy%
  lvol0      new_vg owi-a- 52.00M
  newvgsnap1 new_vg swi-a-  8.00M lvol0    0.20
```

Warning

Because the snapshot increases in size as the origin volume changes, it is important to monitor the percentage of the snapshot volume regularly with the **`lvs`** command to be sure it does not fill. A snapshot that is 100% full is lost completely, as a write to unchanged parts of the origin would be unable to succeed without corrupting the snapshot.

4.4.5. Merging Snapshot Volumes

As of the Red Hat Enterprise Linux 6 release, you can use the **`--merge`** option of the **`lvconvert`** command to merge a snapshot into its origin volume. If both the origin and snapshot volume are not open, the merge will start immediately. Otherwise, the merge will start the first time either the origin or snapshot are activated and both are closed. Merging a snapshot into an origin that cannot be closed, for example a root file system, is deferred until the next time the origin volume is activated. When merging starts, the resulting logical volume will have the origin's name, minor number and UUID. While the merge is in progress, reads or writes to the origin appear as they were directed to the snapshot being merged. When the merge finishes, the merged snapshot is removed.

The following command merges snapshot volume **`vg00/lvol1_snap`** into its origin.

```
# lvconvert --merge vg00/lvol1_snap
```

You can specify multiple snapshots on the command line, or you can use LVM object tags to specify that multiple snapshots be merged to their respective origins. In the following example, logical volumes **`vg00/lvol1`**, **`vg00/lvol2`**, and **`vg00/lvol3`** are all tagged with the tag **`@some_tag`**. The following command merges the snapshot logical volumes for all three volumes serially: **`vg00/lvol1`**, then **`vg00/lvol2`**, then **`vg00/lvol3`**. If the **`--background`** option were used, all snapshot logical volume merges would start in parallel.

```
# lvconvert --merge @some_tag
```

For information on tagging LVM objects, see *Appendix C, LVM Object Tags*. For further information on the **`lvconvert --merge`** command, see the **`lvconvert`**(8) man page.

4.4.6. Persistent Device Numbers

Major and minor device numbers are allocated dynamically at module load. Some applications work best if the block device always is activated with the same device (major and minor) number. You can specify these with the **`lvcreate`** and the **`lvchange`** commands by using the following arguments:

```
--persistent y --major major --minor minor
```

Use a large minor number to be sure that it has not already been allocated to another device dynamically.

If you are exporting a file system using NFS, specifying the **`fsid`** parameter in the exports file may avoid the need to set a persistent device number within LVM.

4.4.7. Resizing Logical Volumes

To reduce the size of a logical volume, use the **`lvreduce`** command. If the logical volume contains a file system, be sure to reduce the file system first (or use the LVM GUI) so that the logical volume is always at least as large as the file system expects it to be.

The following command reduces the size of logical volume **`lvol1`** in volume group **`vg00`** by 3 logical extents.

```
# lvreduce -l -3 vg00/lvol1
```

4.4.8. Changing the Parameters of a Logical Volume Group

To change the parameters of a logical volume, use the **`lvchange`** command. For a listing of the parameters you can change, see the **`lvchange`**(8) man page.

You can use the **`lvchange`** command to activate and deactivate logical volumes. To activate and deactivate all the logical volumes in a volume group at the same time, use the **`vgchange`** command, as described in *Section 4.3.7, “Changing the Parameters of a Volume Group”*.

The following command changes the permission on volume **`lvol1`** in volume group **`vg00`** to be read-only.

```
# lvchange -pr vg00/lvol1
```

4.4.9. Renaming Logical Volumes

To rename an existing logical volume, use the **`lvrename`** command.

Either of the following commands renames logical volume **`lvold`** in volume group **`vg02`** to **`lvnew`**.

```
# lvrename /dev/vg02/lvold /dev/vg02/lvnew
```

```
# lvrename vg02 lvold lvnew
```

For more information on activating logical volumes on individual nodes in a cluster, see *Section 4.7, “Activating Logical Volumes on Individual Nodes in a Cluster”*.

4.4.10. Removing Logical Volumes

To remove an inactive logical volume, use the **`lvremove`** command. If the logical volume is currently mounted, unmount the volume before removing it. In addition, in a clustered environment you must deactivate a logical volume before it can be removed.

The following command removes the logical volume **`/dev/testvg/testlv`** from the volume group **`testvg`**. Note that in this case the logical volume has not been deactivated.

```
[root@tng3-1 lvm]# lvremove /dev/testvg/testlv
Do you really want to remove active logical volume "testlv"? [y/n]: y
  Logical volume "testlv" successfully removed
```

You could explicitly deactivate the logical volume before removing it with the **`lvchange -an`** command, in which case you would not see the prompt verifying whether you want to remove an active logical volume.

4.4.11. Displaying Logical Volumes

There are three commands you can use to display properties of LVM logical volumes: **`lvs`**, **`lvdisplay`**, and **`lvscan`**.

The **`lvs`** command provides logical volume information in a configurable form, displaying one line per logical volume. The **`lvs`** command provides a great deal of format control, and is useful for scripting. For information on using the **`lvs`** command to customize your output, see *Section 4.8, “Customized Reporting for LVM”*.

The **`lvdisplay`** command displays logical volume properties (such as size, layout, and mapping) in a fixed format.

The following command shows the attributes of **`lvol2`** in **`vg00`**. If snapshot logical volumes have been created for this original logical volume, this command shows a list of all snapshot logical volumes and their status (active or inactive) as well.

```
# lvdisplay -v /dev/vg00/lvol2
```

The **`lvscan`** command scans for all logical volumes in the system and lists them, as in the following example.

```
# lvscan
 ACTIVE                   '/dev/vg0/gfslv' [1.46 GB] inherit
```

4.4.12. Growing Logical Volumes

To increase the size of a logical volume, use the **`lvextend`** command.

When you extend the logical volume, you can indicate how much you want to extend the volume, or how large you want it to be after you extend it.

The following command extends the logical volume **`/dev/myvg/homevol`** to 12 gigabytes.

```
# lvextend -L12G /dev/myvg/homevol
lvextend -- extending logical volume "/dev/myvg/homevol" to 12 GB
lvextend -- doing automatic backup of volume group "myvg"
lvextend -- logical volume "/dev/myvg/homevol" successfully extended
```

The following command adds another gigabyte to the logical volume **`/dev/myvg/homevol`**.

```
# lvextend -L+1G /dev/myvg/homevol
lvextend -- extending logical volume "/dev/myvg/homevol" to 13 GB
lvextend -- doing automatic backup of volume group "myvg"
lvextend -- logical volume "/dev/myvg/homevol" successfully extended
```

As with the **`lvcreate`** command, you can use the **`-l`** argument of the **`lvextend`** command to specify the number of extents by which to increase the size of the logical volume. You can also use this argument to specify a percentage of the volume group, or a percentage of the remaining free space in the volume group. The following command extends the logical volume called **`testlv`** to fill all of the unallocated space in the volume group **`myvg`**.

```
[root@tng3-1 ~]# lvextend -l +100%FREE /dev/myvg/testlv
  Extending logical volume testlv to 68.59 GB
  Logical volume testlv successfully resized
```

After you have extended the logical volume it is necessary to increase the file system size to match.

By default, most file system resizing tools will increase the size of the file system to be the size of the underlying logical volume so you do not need to worry about specifying the same size for each of the two commands.

4.4.12.1. Extending a Striped Volume

In order to increase the size of a striped logical volume, there must be enough free space on the underlying physical volumes that make up the volume group to support the stripe. For example, if you have a two-way stripe that that uses up an entire volume group, adding a single physical volume to the volume group will not enable you to extend the stripe. Instead, you must add at least two physical volumes to the volume group.

For example, consider a volume group **`vg`** that consists of two underlying physical volumes, as displayed with the following **`vgs`** command.

```
# vgs
  VG   #PV #LV #SN Attr   VSize   VFree
  vg     2   0   0 wz--n- 271.31G 271.31G
```

You can create a stripe using the entire amount of space in the volume group.

```
# lvcreate -n stripe1 -L 271.31G -i 2 vg
  Using default stripesize 64.00 KB
  Rounding up size to full physical extent 271.31 GB
  Logical volume "stripe1" created
# lvs -a -o +devices
  LV      VG   Attr   LSize   Origin Snap%  Move Log Copy%  Devices
  stripe1 vg   -wi-a- 271.31G                               /dev/sda1(0),/dev/sdb1(0)
```

Note that the volume group now has no more free space.

```
# vgs
  VG   #PV #LV #SN Attr   VSize   VFree
  vg     2   1   0 wz--n- 271.31G    0
```

The following command adds another physical volume to the volume group, which then has 135G of additional space.

```
# vgextend vg /dev/sdc1
  Volume group "vg" successfully extended
# vgs
  VG   #PV #LV #SN Attr   VSize   VFree
  vg     3   1   0 wz--n- 406.97G 135.66G
```

At this point you cannot extend the striped logical volume to the full size of the volume group, because two underlying devices are needed in order to stripe the data.

```
# lvextend vg/stripe1 -L 406G
  Using stripesize of last segment 64.00 KB
  Extending logical volume stripe1 to 406.00 GB
  Insufficient suitable allocatable extents for logical volume stripe1: 34480
more required
```

To extend the striped logical volume, add another physical volume and then extend the logical volume. In this example, having added two physical volumes to the volume group we can extend the logical volume to the full size of the volume group.

```
# vgextend vg /dev/sdd1
  Volume group "vg" successfully extended
# vgs
  VG   #PV #LV #SN Attr   VSize   VFree
  vg     4   1   0 wz--n- 542.62G 271.31G
# lvextend vg/stripe1 -L 542G
  Using stripesize of last segment 64.00 KB
  Extending logical volume stripe1 to 542.00 GB
  Logical volume stripe1 successfully resized
```

If you do not have enough underlying physical devices to extend the striped logical volume, it is possible to extend the volume anyway if it does not matter that the extension is not striped, which may result in uneven performance. When adding space to the logical volume, the default operation is to use the same striping parameters of the last segment of the existing logical volume, but you can override those parameters. The following example extends the existing striped logical volume to use the remaining free space after the initial **`lvextend`** command fails.

```
# lvextend vg/stripe1 -L 406G
  Using stripesize of last segment 64.00 KB
  Extending logical volume stripe1 to 406.00 GB
  Insufficient suitable allocatable extents for logical volume stripe1: 34480
more required
# lvextend -i1 -l+100%FREE vg/stripe1
```

4.4.12.2. Extending a Logical Volume with the `cling` Allocation Policy

When extending an LVM volume, you can use the **`--alloc cling`** option of the **`lvextend`** command to specify the **`cling`** allocation policy. This policy will choose space on the same physical volumes as the last segment of the existing logical volume. If there is insufficient space on the physical volumes and a list of tags is defined in the **`lvm.conf`** file, LVM will check whether any of the tags are attached to the physical volumes and seek to match those physical volume tags between existing extents and new extents.

For example, if you have logical volumes that are mirrored between two sites within a single volume group, you can tag the physical volumes according to where they are situated by tagging the physical volumes with @site1 and @site2 tags and specify the following line in the **`lvm.conf`** file:

```
cling_tag_list = [ "@site1", "@site2" ]
```

For information on tagging physical volumes, see *Appendix C, LVM Object Tags*.

In the following example, the **`lvm.conf`** file has been modified to contain the following line:

```
cling_tag_list = [ "@A", "@B" ]
```

Also in this example, a volume group **`taft`** has been created that consists of the physical volumes **`/dev/sdb1`**, **`/dev/sdc1`**, **`/dev/sdd1`**, **`/dev/sde1`**, **`/dev/sdf1`**, **`/dev/sdg1`**, and **`/dev/sdh1`**. These physical volumes have been tagged with tags **`A`**, **`B`**, and **`C`**. The example does not use the **`C`** tag, but this will show that LVM uses the tags to select which physical volumes to use for the mirror legs.

```
[root@taft-03 ~]# pvs -a -o +pv_tags /dev/sd[bcdefgh]1
  PV         VG   Fmt  Attr PSize   PFree   PV Tags
  /dev/sdb1  taft lvm2 a-   135.66g 135.66g A
  /dev/sdc1  taft lvm2 a-   135.66g 135.66g B
  /dev/sdd1  taft lvm2 a-   135.66g 135.66g B
  /dev/sde1  taft lvm2 a-   135.66g 135.66g C
  /dev/sdf1  taft lvm2 a-   135.66g 135.66g C
  /dev/sdg1  taft lvm2 a-   135.66g 135.66g A
  /dev/sdh1  taft lvm2 a-   135.66g 135.66g A
```

The following command creates a 100GB mirrored volume from the volume group **`taft`**.

```
[root@taft-03 ~]# lvcreate -m 1 -n mirror --nosync -L 100G taft
```

The following command shows which devices are used for the mirror legs and mirror log.

```
[root@taft-03 ~]# lvs -a -o +devices
  LV                 VG        Attr   LSize   Log         Copy%  Devices
  mirror             taft      Mwi-a- 100.00g mirror_mlog 100.00
mirror_mimage_0(0),mirror_mimage_1(0)
  [mirror_mimage_0] taft      iwi-ao 100.00g                    /dev/sdb1(0)
  [mirror_mimage_1] taft      iwi-ao 100.00g                    /dev/sdc1(0)
  [mirror_mlog]     taft      lwi-ao   4.00m                    /dev/sdh1(0)
```

The following command extends the size of the mirrored volume, using the **`cling`** allocation policy to indicate that the mirror legs should be extended using physical volumes with the same tag.

```
[root@taft-03 ~]# lvextend --alloc cling -L +100G taft/mirror
  Extending 2 mirror images.
  Extending logical volume mirror to 200.00 GiB
  Logical volume mirror successfully resized
```

The following display command shows that the mirror legs have been extended using physical volumes with the same tag as the leg. Note that the physical volumes with a tag of **`C`** were ignored.

```
[root@taft-03 ~]# lvs -a -o +devices
  LV                VG        Attr   LSize   Log         Copy%  Devices
  mirror            taft      Mwi-a- 200.00g mirror_mlog  50.16
mirror_mimage_0(0),mirror_mimage_1(0)
  [mirror_mimage_0] taft      Iwi-ao 200.00g                    /dev/sdb1(0)
  [mirror_mimage_0] taft      Iwi-ao 200.00g                    /dev/sdg1(0)
  [mirror_mimage_1] taft      Iwi-ao 200.00g                    /dev/sdc1(0)
  [mirror_mimage_1] taft      Iwi-ao 200.00g                    /dev/sdd1(0)
  [mirror_mlog]     taft      lwi-ao    4.00m                   /dev/sdh1(0)
```

4.4.13. Shrinking Logical Volumes

To reduce the size of a logical volume, first unmount the file system. You can then use the **`lvreduce`** command to shrink the volume. After shrinking the volume, remount the file system.

Warning

It is important to reduce the size of the file system or whatever is residing in the volume before shrinking the volume itself, otherwise you risk losing data.

Shrinking a logical volume frees some of the volume group to be allocated to other logical volumes in the volume group.

The following example reduces the size of logical volume **`lvol1`** in volume group **`vg00`** by 3 logical extents.

```
# lvreduce -l -3 vg00/lvol1
```

4.5. Controlling LVM Device Scans with Filters

At startup, the **`vgscan`** command is run to scan the block devices on the system looking for LVM labels, to determine which of them are physical volumes and to read the metadata and build up a list of volume groups. The names of the physical volumes are stored in the cache file of each node in the system, **`/etc/lvm/cache/.cache`**. Subsequent commands may read that file to avoiding rescanning.

You can control which devices LVM scans by setting up filters in the **`lvm.conf`** configuration file. The filters in the **`lvm.conf`** file consist of a series of simple regular expressions that get applied to the device names that are in the **`/dev`** directory to decide whether to accept or reject each block device found.

The following examples show the use of filters to control which devices LVM scans. Note that some of these examples do not necessarily represent best practice, as the regular expressions are matched freely against the complete pathname. For example, **`a/loop/`** is equivalent to **`a/.*loop.*/`** and would match **`/dev/solooperation/lvol1`**.

The following filter adds all discovered devices, which is the default behavior as there is no filter configured in the configuration file:

```
filter = [ "a/.*/" ]
```

The following filter removes the cdrom device in order to avoid delays if the drive contains no media:

```
filter = [ "r|/dev/cdrom|" ]
```

The following filter adds all loop and removes all other block devices:

```
filter = [ "a/loop.*/", "r/.*/" ]
```

The following filter adds all loop and IDE and removes all other block devices:

```
filter =[ "a|loop.*|", "a|/dev/hd.*|", "r|.*|" ]
```

The following filter adds just partition 8 on the first IDE drive and removes all other block devices:

```
filter = [ "a|^/dev/hda8$|", "r/.*/" ]
```

For more information on the **`lvm.conf`** file, see *Appendix B, The LVM Configuration Files* and the **`lvm.conf`**(5) man page.

4.6. Online Data Relocation

You can move data while the system is in use with the **`pvmove`** command.

The **`pvmove`** command breaks up the data to be moved into sections and creates a temporary mirror to move each section. For more information on the operation of the **`pvmove`** command, see the **`pvmove`**(8) man page.

The following command moves all allocated space off the physical volume **`/dev/sdc1`** to other free physical volumes in the volume group:

```
# pvmove /dev/sdc1
```

The following command moves just the extents of the logical volume **`MyLV`**.

```
# pvmove -n MyLV /dev/sdc1
```

Since the **`pvmove`** command can take a long time to execute, you may want to run the command in the background to avoid display of progress updates in the foreground. The following command moves all extents allocated to the physical volume **`/dev/sdc1`** over to **`/dev/sdf1`** in the background.

```
# pvmove -b /dev/sdc1 /dev/sdf1
```

The following command reports the progress of the move as a percentage at five second intervals.

```
# pvmove -i5 /dev/sdd1
```

4.7. Activating Logical Volumes on Individual Nodes in a Cluster

If you have LVM installed in a cluster environment, you may at times need to activate logical volumes exclusively on one node.

To activate logical volumes exclusively on one node, use the **`lvchange -aey`** command. Alternatively, you can use **`lvchange -aly`** command to activate logical volumes only on the local node but not exclusively. You can later activate them on additional nodes concurrently.

You can also activate logical volumes on individual nodes by using LVM tags, which are described in *Appendix C, LVM Object Tags*. You can also specify activation of nodes in the configuration file, which is described in *Appendix B, The LVM Configuration Files*.

4.8. Customized Reporting for LVM

You can produce concise and customizable reports of LVM objects with the **`pvs`**, **`lvs`**, and **`vgs`** commands. The reports that these commands generate include one line of output for each object. Each line contains an ordered list of fields of properties related to the object. There are five ways to select the objects to be reported: by physical volume, volume group, logical volume, physical volume segment, and logical volume segment.

The following sections provide:

- A summary of command arguments you can use to control the format of the generated report.
- A list of the fields you can select for each LVM object.
- A summary of command arguments you can use to sort the generated report.
- Instructions for specifying the units of the report output.

4.8.1. Format Control

Whether you use the **`pvs`**, **`lvs`**, or **`vgs`** command determines the default set of fields displayed and the sort order. You can control the output of these commands with the following arguments:

- You can change what fields are displayed to something other than the default by using the **`-o`** argument. For example, the following output is the default display for the **`pvs`** command (which displays information about physical volumes).

  ```
  # pvs
    PV         VG     Fmt  Attr PSize  PFree
    /dev/sdb1  new_vg lvm2 a-   17.14G 17.14G
    /dev/sdc1  new_vg lvm2 a-   17.14G 17.09G
    /dev/sdd1  new_vg lvm2 a-   17.14G 17.14G
  ```

  The following command displays only the physical volume name and size.

  ```
  # pvs -o pv_name,pv_size
    PV         PSize
  ```

```
  /dev/sdb1  17.14G
  /dev/sdc1  17.14G
  /dev/sdd1  17.14G
```

- You can append a field to the output with the plus sign (+), which is used in combination with the -o argument.

  The following example displays the UUID of the physical volume in addition to the default fields.

  ```
  # pvs -o +pv_uuid
    PV         VG     Fmt  Attr PSize  PFree  PV UUID
    /dev/sdb1  new_vg lvm2 a-   17.14G 17.14G onFF2w-1fLC-ughJ-D9eB-M7iv-6XqA-dqGeXY
    /dev/sdc1  new_vg lvm2 a-   17.14G 17.09G Joqlch-yWSj-kuEn-IdwM-01S9-X08M-mcpsVe
    /dev/sdd1  new_vg lvm2 a-   17.14G 17.14G yvfvZK-Cf31-j75k-dECm-0RZ3-0dGW-UqkCS
  ```

- Adding the **`-v`** argument to a command includes some extra fields. For example, the **`pvs -v`** command will display the **`DevSize`** and **`PV UUID`** fields in addition to the default fields.

  ```
  # pvs -v
      Scanning for physical volume names
    PV         VG     Fmt  Attr PSize  PFree  DevSize PV UUID
    /dev/sdb1  new_vg lvm2 a-   17.14G 17.14G  17.14G onFF2w-1fLC-ughJ-D9eB-M7iv-6XqA-dqGeXY
    /dev/sdc1  new_vg lvm2 a-   17.14G 17.09G  17.14G Joqlch-yWSj-kuEn-IdwM-01S9-XO8M-mcpsVe
    /dev/sdd1  new_vg lvm2 a-   17.14G 17.14G  17.14G yvfvZK-Cf31-j75k-dECm-0RZ3-0dGW-tUqkCS
  ```

- The **`--noheadings`** argument suppresses the headings line. This can be useful for writing scripts.

  The following example uses the **`--noheadings`** argument in combination with the **`pv_name`** argument, which will generate a list of all physical volumes.

  ```
  # pvs --noheadings -o pv_name
    /dev/sdb1
    /dev/sdc1
    /dev/sdd1
  ```

- The **`--separator`** ***`separator`*** argument uses *`separator`* to separate each field.

  The following example separates the default output fields of the **`pvs`** command with an equals sign (=).

  ```
  # pvs --separator =
    PV=VG=Fmt=Attr=PSize=PFree
    /dev/sdb1=new_vg=lvm2=a-=17.14G=17.14G
    /dev/sdc1=new_vg=lvm2=a-=17.14G=17.09G
    /dev/sdd1=new_vg=lvm2=a-=17.14G=17.14G
  ```

  To keep the fields aligned when using the **`separator`** argument, use the **`separator`** argument in conjunction with the **`--aligned`** argument.

  ```
  # pvs --separator = --aligned
    PV        =VG    =Fmt =Attr=PSize =PFree
    /dev/sdb1 =new_vg=lvm2=a-  =17.14G=17.14G
    /dev/sdc1 =new_vg=lvm2=a-  =17.14G=17.09G
  ```

```
  /dev/sdd1 =new_vg=lvm2=a-  =17.14G=17.14G
```

You can use the **-P** argument of the **lvs** or **vgs** command to display information about a failed volume that would otherwise not appear in the output. For information on the output this argument yields, see *Section 6.2, “Displaying Information on Failed Devices”*.

For a full listing of display arguments, see the **pvs**(8), **vgs**(8) and **lvs**(8) man pages.

Volume group fields can be mixed with either physical volume (and physical volume segment) fields or with logical volume (and logical volume segment) fields, but physical volume and logical volume fields cannot be mixed. For example, the following command will display one line of output for each physical volume.

```
# vgs -o +pv_name
  VG     #PV #LV #SN Attr   VSize  VFree  PV
  new_vg   3   1   0 wz--n- 51.42G 51.37G /dev/sdc1
  new_vg   3   1   0 wz--n- 51.42G 51.37G /dev/sdd1
  new_vg   3   1   0 wz--n- 51.42G 51.37G /dev/sdb1
```

4.8.2. Object Selection

This section provides a series of tables that list the information you can display about the LVM objects with the **pvs**, **vgs**, and **lvs** commands.

For convenience, a field name prefix can be dropped if it matches the default for the command. For example, with the **pvs** command, **name** means **pv_name**, but with the **vgs** command, **name** is interpreted as **vg_name**.

Executing the following command is the equivalent of executing **pvs -o pv_free**.

```
# pvs -o +free
  PFree
  17.14G
  17.09G
  17.14G
```

The pvs Command

Table 4.1, “pvs Display Fields” lists the display arguments of the **pvs** command, along with the field name as it appears in the header display and a description of the field.

Table 4.1. pvs Display Fields

Argument	Header	Description
dev_size	DevSize	The size of the underlying device on which the physical volume was created
pe_start	1st PE	Offset to the start of the first physical extent in the underlying device
pv_attr	Attr	Status of the physical volume: (a)llocatable or e(x)ported.
pv_fmt	Fmt	The metadata format of the physical volume (**lvm2** or **lvm1**)
pv_free	PFree	The free space remaining on the physical volume

Argument	Header	Description
`pv_name`	PV	The physical volume name
`pv_pe_alloc_count`	Alloc	Number of used physical extents
`pv_pe_count`	PE	Number of physical extents
`pvseg_size`	SSize	The segment size of the physical volume
`pvseg_start`	Start	The starting physical extent of the physical volume segment
`pv_size`	PSize	The size of the physical volume
`pv_tags`	PV Tags	LVM tags attached to the physical volume
`pv_used`	Used	The amount of space currently used on the physical volume
`pv_uuid`	PV UUID	The UUID of the physical volume

The **pvs** command displays the following fields by default: **`pv_name`**, **`vg_name`**, **`pv_fmt`**, **`pv_attr`**, **`pv_size`**, **`pv_free`**. The display is sorted by **`pv_name`**.

```
# pvs
  PV         VG     Fmt  Attr PSize  PFree
  /dev/sdb1  new_vg lvm2 a-   17.14G 17.14G
  /dev/sdc1  new_vg lvm2 a-   17.14G 17.09G
  /dev/sdd1  new_vg lvm2 a-   17.14G 17.13G
```

Using the **`-v`** argument with the **pvs** command adds the following fields to the default display: **`dev_size`**, **`pv_uuid`**.

```
# pvs -v
    Scanning for physical volume names
  PV         VG     Fmt  Attr PSize  PFree  DevSize PV UUID
  /dev/sdb1  new_vg lvm2 a-   17.14G 17.14G  17.14G onFF2w-1fLC-ughJ-D9eB-M7iv-6XqA-dqGeXY
  /dev/sdc1  new_vg lvm2 a-   17.14G 17.09G  17.14G Joqlch-yWSj-kuEn-IdwM-01S9-XO8M-mcpsVe
  /dev/sdd1  new_vg lvm2 a-   17.14G 17.13G  17.14G yvfvZK-Cf31-j75k-dECm-0RZ3-0dGW-tUqkCS
```

You can use the **`--segments`** argument of the **pvs** command to display information about each physical volume segment. A segment is a group of extents. A segment view can be useful if you want to see whether your logical volume is fragmented.

The **pvs `--segments`** command displays the following fields by default: **`pv_name`**, **`vg_name`**, **`pv_fmt`**, **`pv_attr`**, **`pv_size`**, **`pv_free`**, **`pvseg_start`**, **`pvseg_size`**. The display is sorted by **`pv_name`** and **`pvseg_size`** within the physical volume.

```
# pvs --segments
  PV         VG         Fmt  Attr PSize  PFree  Start SSize
  /dev/hda2  VolGroup00 lvm2 a-   37.16G 32.00M     0  1172
  /dev/hda2  VolGroup00 lvm2 a-   37.16G 32.00M  1172    16
  /dev/hda2  VolGroup00 lvm2 a-   37.16G 32.00M  1188     1
  /dev/sda1  vg         lvm2 a-   17.14G 16.75G     0    26
  /dev/sda1  vg         lvm2 a-   17.14G 16.75G    26    24
  /dev/sda1  vg         lvm2 a-   17.14G 16.75G    50    26
  /dev/sda1  vg         lvm2 a-   17.14G 16.75G    76    24
  /dev/sda1  vg         lvm2 a-   17.14G 16.75G   100    26
  /dev/sda1  vg         lvm2 a-   17.14G 16.75G   126    24
  /dev/sda1  vg         lvm2 a-   17.14G 16.75G   150    22
  /dev/sda1  vg         lvm2 a-   17.14G 16.75G   172  4217
  /dev/sdb1  vg         lvm2 a-   17.14G 17.14G     0  4389
  /dev/sdc1  vg         lvm2 a-   17.14G 17.14G     0  4389
  /dev/sdd1  vg         lvm2 a-   17.14G 17.14G     0  4389
  /dev/sde1  vg         lvm2 a-   17.14G 17.14G     0  4389
  /dev/sdf1  vg         lvm2 a-   17.14G 17.14G     0  4389
  /dev/sdg1  vg         lvm2 a-   17.14G 17.14G     0  4389
```

You can use the **pvs -a** command to see devices detected by LVM that have not been initialized as LVM physical volumes.

```
# pvs -a
  PV                             VG     Fmt  Attr PSize  PFree
  /dev/VolGroup00/LogVol01                   --       0      0
  /dev/new_vg/lvol0                          --       0      0
  /dev/ram                                   --       0      0
  /dev/ram0                                  --       0      0
  /dev/ram2                                  --       0      0
  /dev/ram3                                  --       0      0
  /dev/ram4                                  --       0      0
  /dev/ram5                                  --       0      0
  /dev/ram6                                  --       0      0
  /dev/root                                  --       0      0
  /dev/sda                                   --       0      0
  /dev/sdb                                   --       0      0
  /dev/sdb1                      new_vg lvm2 a-   17.14G 17.14G
  /dev/sdc                                   --       0      0
  /dev/sdc1                      new_vg lvm2 a-   17.14G 17.09G
  /dev/sdd                                   --       0      0
  /dev/sdd1                      new_vg lvm2 a-   17.14G 17.14G
```

The vgs Command

Table 4.2, “vgs Display Fields” lists the display arguments of the **vgs** command, along with the field name as it appears in the header display and a description of the field.

Table 4.2. vgs Display Fields

Argument	Header	Description
lv_count	#LV	The number of logical volumes the volume group contains
max_lv	MaxLV	The maximum number of logical volumes allowed in the volume group (0 if unlimited)
max_pv	MaxPV	The maximum number of physical volumes allowed in the volume group (0 if unlimited)
pv_count	#PV	The number of physical volumes that define the volume group
snap_count	#SN	The number of snapshots the volume group contains
vg_attr	Attr	Status of the volume group: (w)riteable, (r)eadonly, resi(z)eable, e(x)ported, (p)artial and (c)lustered.
vg_extent_count	#Ext	The number of physical extents in the volume group
vg_extent_size	Ext	The size of the physical extents in the volume group
vg_fmt	Fmt	The metadata format of the volume group (**lvm2** or **lvm1**)
vg_free	VFree	Size of the free space remaining in the volume group

Argument	Header	Description
`vg_free_count`	Free	Number of free physical extents in the volume group
`vg_name`	VG	The volume group name
`vg_seqno`	Seq	Number representing the revision of the volume group
`vg_size`	VSize	The size of the volume group
`vg_sysid`	SYS ID	LVM1 System ID
`vg_tags`	VG Tags	LVM tags attached to the volume group
`vg_uuid`	VG UUID	The UUID of the volume group

The **`vgs`** command displays the following fields by default: **`vg_name`**, **`pv_count`**, **`lv_count`**, **`snap_count`**, **`vg_attr`**, **`vg_size`**, **`vg_free`**. The display is sorted by **`vg_name`**.

```
# vgs
  VG     #PV #LV #SN Attr   VSize  VFree
  new_vg   3   1   1 wz--n- 51.42G 51.36G
```

Using the **`-v`** argument with the **`vgs`** command adds the following fields to the default display: **`vg_extent_size`**, **`vg_uuid`**.

```
# vgs -v
    Finding all volume groups
    Finding volume group "new_vg"
  VG     Attr   Ext   #PV #LV #SN VSize  VFree  VG UUID
  new_vg wz--n- 4.00M   3   1   1 51.42G 51.36G jxQJ0a-ZKk0-OpMO-0118-nlwO-wwqd-fD5D32
```

The lvs Command

Table 4.3, “lvs Display Fields” lists the display arguments of the **`lvs`** command, along with the field name as it appears in the header display and a description of the field.

Table 4.3. lvs Display Fields

Argument	Header	Description
`chunksize` **`chunk_size`**	Chunk	Unit size in a snapshot volume
`copy_percent`	Copy%	The synchronization percentage of a mirrored logical volume; also used when physical extents are being moved with the **`pv_move`** command
`devices`	Devices	The underlying devices that make up the logical volume: the physical volumes, logical volumes, and start physical extents and logical extents
`lv_attr`	Attr	The status of the logical volume. The logical volume attribute bits are as follows: Bit 1: Volume type: (m)irrored, (M)irrored without initial sync, (o)rigin, (p)vmove, (s)napshot, invalid (S)napshot, (v)irtual Bit 2: Permissions: (w)riteable, (r)ead-only

Argument	Header	Description
		Bit 3: Allocation policy: (c)ontiguous, (n)ormal, (a)nywhere, (i)nherited. This is capitalized if the volume is currently locked against allocation changes, for example while executing the **`pvmove`** command. Bit 4: fixed (m)inor Bit 5: State: (a)ctive, (s)uspended, (I)nvalid snapshot, invalid (S)uspended snapshot, mapped (d)evice present without tables, mapped device present with (i)nactive table Bit 6: device (o)pen
`lv_kernel_major`	KMaj	Actual major device number of the logical volume (-1 if inactive)
`lv_kernel_minor`	KMIN	Actual minor device number of the logical volume (-1 if inactive)
`lv_major`	Maj	The persistent major device number of the logical volume (-1 if not specified)
`lv_minor`	Min	The persistent minor device number of the logical volume (-1 if not specified)
`lv_name`	LV	The name of the logical volume
`lv_size`	LSize	The size of the logical volume
`lv_tags`	LV Tags	LVM tags attached to the logical volume
`lv_uuid`	LV UUID	The UUID of the logical volume.
`mirror_log`	Log	Device on which the mirror log resides
`modules`	Modules	Corresponding kernel device-mapper target necessary to use this logical volume
`move_pv`	Move	Source physical volume of a temporary logical volume created with the **`pvmove`** command
`origin`	Origin	The origin device of a snapshot volume
`regionsize` **`region_size`**	Region	The unit size of a mirrored logical volume
`seg_count`	#Seg	The number of segments in the logical volume
`seg_size`	SSize	The size of the segments in the logical volume
`seg_start`	Start	Offset of the segment in the logical volume
`seg_tags`	Seg Tags	LVM tags attached to the segments of the logical volume
`segtype`	Type	The segment type of a logical volume (for example: mirror, striped, linear)
`snap_percent`	Snap%	Current percentage of a snapshot volume that is in use
`stripes`	#Str	Number of stripes or mirrors in a logical volume
`stripesize` **`stripe_size`**	Stripe	Unit size of the stripe in a striped logical volume

The **`lvs`** command displays the following fields by default: **`lv_name`**, **`vg_name`**, **`lv_attr`**, **`lv_size`**, **`origin`**, **`snap_percent`**, **`move_pv`**, **`mirror_log`**, **`copy_percent`**, **`convert_lv`**. The default display is sorted by **`vg_name`** and **`lv_name`** within the volume group.

```
# lvs
  LV         VG     Attr   LSize  Origin Snap%  Move Log Copy%  Convert
  lvol0      new_vg owi-a- 52.00M
  newvgsnap1 new_vg swi-a-  8.00M lvol0    0.20
```

Using the **`-v`** argument with the **`lvs`** command adds the following fields to the default display: **`seg_count`**, **`lv_major`**, **`lv_minor`**, **`lv_kernel_major`**, **`lv_kernel_minor`**, **`lv_uuid`**.

```
# lvs -v
    Finding all logical volumes
  LV         VG     #Seg Attr   LSize  Maj Min KMaj KMin Origin Snap%  Move Copy%  Log
 Convert LV UUID
  lvol0      new_vg    1 owi-a- 52.00M  -1  -1 253  3
  LBy1Tz-sr23-OjsI-LT03-nHLC-y8XW-EhCl78
  newvgsnap1 new_vg    1 swi-a-  8.00M  -1  -1 253  5    lvol0    0.20
  1ye1OU-1cIu-o79k-20h2-ZGF0-qCJm-CfbsIx
```

You can use the **`--segments`** argument of the **`lvs`** command to display information with default columns that emphasize the segment information. When you use the **`segments`** argument, the **`seg`** prefix is optional. The **`lvs --segments`** command displays the following fields by default: **`lv_name`**, **`vg_name`**, **`lv_attr`**, **`stripes`**, **`segtype`**, **`seg_size`**. The default display is sorted by **`vg_name`**, **`lv_name`** within the volume group, and **`seg_start`** within the logical volume. If the logical volumes were fragmented, the output from this command would show that.

```
# lvs --segments
  LV       VG         Attr   #Str Type   SSize
  LogVol00 VolGroup00 -wi-ao    1 linear  36.62G
  LogVol01 VolGroup00 -wi-ao    1 linear 512.00M
  lv       vg         -wi-a-    1 linear 104.00M
  lv       vg         -wi-a-    1 linear 104.00M
  lv       vg         -wi-a-    1 linear 104.00M
  lv       vg         -wi-a-    1 linear  88.00M
```

Using the **`-v`** argument with the **`lvs --segments`** command adds the following fields to the default display: **`seg_start`**, **`stripesize`**, **`chunksize`**.

```
# lvs -v --segments
    Finding all logical volumes
  LV         VG     Attr   Start SSize  #Str Type   Stripe Chunk
  lvol0      new_vg owi-a-    0  52.00M    1 linear     0     0
  newvgsnap1 new_vg swi-a-    0   8.00M    1 linear     0  8.00K
```

The following example shows the default output of the **`lvs`** command on a system with one logical volume configured, followed by the default output of the **`lvs`** command with the **`segments`** argument specified.

```
# lvs
  LV    VG     Attr   LSize  Origin Snap%  Move Log Copy%
  lvol0 new_vg -wi-a- 52.00M
# lvs --segments
  LV    VG     Attr   #Str Type   SSize
  lvol0 new_vg -wi-a-    1 linear 52.00M
```

4.8.3. Sorting LVM Reports

Normally the entire output of the **`lvs`**, **`vgs`**, or **`pvs`** command has to be generated and stored internally before it can be sorted and columns aligned correctly. You can specify the **`--unbuffered`** argument to display unsorted output as soon as it is generated.

To specify an alternative ordered list of columns to sort on, use the **`-O`** argument of any of the reporting commands. It is not necessary to include these fields within the output itself.

The following example shows the output of the **`pvs`** command that displays the physical volume name, size, and free space.

```
# pvs -o pv_name,pv_size,pv_free
  PV         PSize  PFree
  /dev/sdb1  17.14G 17.14G
  /dev/sdc1  17.14G 17.09G
  /dev/sdd1  17.14G 17.14G
```

The following example shows the same output, sorted by the free space field.

```
# pvs -o pv_name,pv_size,pv_free -O pv_free
  PV         PSize  PFree
  /dev/sdc1  17.14G 17.09G
  /dev/sdd1  17.14G 17.14G
  /dev/sdb1  17.14G 17.14G
```

The following example shows that you do not need to display the field on which you are sorting.

```
# pvs -o pv_name,pv_size -O pv_free
  PV         PSize
  /dev/sdc1  17.14G
  /dev/sdd1  17.14G
  /dev/sdb1  17.14G
```

To display a reverse sort, precede a field you specify after the **`-O`** argument with the **`-`** character.

```
# pvs -o pv_name,pv_size,pv_free -O -pv_free
  PV         PSize  PFree
  /dev/sdd1  17.14G 17.14G
  /dev/sdb1  17.14G 17.14G
  /dev/sdc1  17.14G 17.09G
```

4.8.4. Specifying Units

To specify the unit for the LVM report display, use the **`--units`** argument of the report command. You can specify (b)ytes, (k)ilobytes, (m)egabytes, (g)igabytes, (t)erabytes, (e)xabytes, (p)etabytes, and (h)uman-readable. The default display is human-readable. You can override the default by setting the **`units`** parameter in the **`global`** section of the **`lvm.conf`** file.

The following example specifies the output of the **`pvs`** command in megabytes rather than the default gigabytes.

```
# pvs --units m
  PV         VG     Fmt  Attr PSize     PFree
  /dev/sda1         lvm2 --   17555.40M 17555.40M
  /dev/sdb1  new_vg lvm2 a-   17552.00M 17552.00M
  /dev/sdc1  new_vg lvm2 a-   17552.00M 17500.00M
  /dev/sdd1  new_vg lvm2 a-   17552.00M 17552.00M
```

By default, units are displayed in powers of 2 (multiples of 1024). You can specify that units be displayed in multiples of 1000 by capitalizing the unit specification (B, K, M, G, T, H).

The following command displays the output as a multiple of 1024, the default behavior.

```
# pvs
  PV         VG     Fmt  Attr PSize  PFree
  /dev/sdb1  new_vg lvm2 a-   17.14G 17.14G
  /dev/sdc1  new_vg lvm2 a-   17.14G 17.09G
  /dev/sdd1  new_vg lvm2 a-   17.14G 17.14G
```

The following command displays the output as a multiple of 1000.

```
#  pvs --units G
  PV         VG     Fmt  Attr PSize  PFree
  /dev/sdb1  new_vg lvm2 a-   18.40G 18.40G
  /dev/sdc1  new_vg lvm2 a-   18.40G 18.35G
  /dev/sdd1  new_vg lvm2 a-   18.40G 18.40G
```

You can also specify (s)ectors (defined as 512 bytes) or custom units.

The following example displays the output of the **pvs** command as a number of sectors.

```
# pvs --units s
  PV         VG     Fmt  Attr PSize     PFree
  /dev/sdb1  new_vg lvm2 a-   35946496S 35946496S
  /dev/sdc1  new_vg lvm2 a-   35946496S 35840000S
  /dev/sdd1  new_vg lvm2 a-   35946496S 35946496S
```

The following example displays the output of the **pvs** command in units of 4 MB.

```
# pvs --units 4m
  PV         VG     Fmt  Attr PSize    PFree
  /dev/sdb1  new_vg lvm2 a-   4388.00U 4388.00U
  /dev/sdc1  new_vg lvm2 a-   4388.00U 4375.00U
  /dev/sdd1  new_vg lvm2 a-   4388.00U 4388.00U
```

LVM Configuration Examples

This chapter provides some basic LVM configuration examples.

5.1. Creating an LVM Logical Volume on Three Disks

This example creates an LVM logical volume called **`new_logical_volume`** that consists of the disks at **`/dev/sda1`**, **`/dev/sdb1`**, and **`/dev/sdc1`**.

5.1.1. Creating the Physical Volumes

To use disks in a volume group, you label them as LVM physical volumes.

Warning

This command destroys any data on **`/dev/sda1`**, **`/dev/sdb1`**, and **`/dev/sdc1`**.

```
[root@tng3-1 ~]# pvcreate /dev/sda1 /dev/sdb1 /dev/sdc1
  Physical volume "/dev/sda1" successfully created
  Physical volume "/dev/sdb1" successfully created
  Physical volume "/dev/sdc1" successfully created
```

5.1.2. Creating the Volume Group

The following command creates the volume group **`new_vol_group`**.

```
[root@tng3-1 ~]# vgcreate new_vol_group /dev/sda1 /dev/sdb1 /dev/sdc1
  Volume group "new_vol_group" successfully created
```

You can use the **`vgs`** command to display the attributes of the new volume group.

```
[root@tng3-1 ~]# vgs
  VG            #PV #LV #SN Attr   VSize  VFree
  new_vol_group   3   0   0 wz--n- 51.45G 51.45G
```

5.1.3. Creating the Logical Volume

The following command creates the logical volume **`new_logical_volume`** from the volume group **`new_vol_group`**. This example creates a logical volume that uses 2GB of the volume group.

```
[root@tng3-1 ~]# lvcreate -L2G -n new_logical_volume new_vol_group
  Logical volume "new_logical_volume" created
```

5.1.4. Creating the File System

The following command creates a GFS2 file system on the logical volume.

```
[root@tng3-1 ~]# mkfs.gfs2 -plock_nolock -j 1 /dev/new_vol_group/new_logical_volume
This will destroy any data on /dev/new_vol_group/new_logical_volume.

Are you sure you want to proceed? [y/n] y

Device:                    /dev/new_vol_group/new_logical_volume
Blocksize:                 4096
Filesystem Size:           491460
Journals:                  1
Resource Groups:           8
Locking Protocol:          lock_nolock
Lock Table:

Syncing...
All Done
```

The following commands mount the logical volume and report the file system disk space usage.

```
[root@tng3-1 ~]# mount /dev/new_vol_group/new_logical_volume /mnt
[root@tng3-1 ~]# df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/new_vol_group/new_logical_volume
                       1965840        20   1965820   1% /mnt
```

5.2. Creating a Striped Logical Volume

This example creates an LVM striped logical volume called **`striped_logical_volume`** that stripes data across the disks at **`/dev/sda1`**, **`/dev/sdb1`**, and **`/dev/sdc1`**.

5.2.1. Creating the Physical Volumes

Label the disks you will use in the volume groups as LVM physical volumes.

Warning

This command destroys any data on **`/dev/sda1`**, **`/dev/sdb1`**, and **`/dev/sdc1`**.

```
[root@tng3-1 ~]# pvcreate /dev/sda1 /dev/sdb1 /dev/sdc1
  Physical volume "/dev/sda1" successfully created
  Physical volume "/dev/sdb1" successfully created
  Physical volume "/dev/sdc1" successfully created
```

5.2.2. Creating the Volume Group

The following command creates the volume group **`volgroup01`**.

```
[root@tng3-1 ~]# vgcreate volgroup01 /dev/sda1 /dev/sdb1 /dev/sdc1
  Volume group "volgroup01" successfully created
```

You can use the **`vgs`** command to display the attributes of the new volume group.

```
[root@tng3-1 ~]# vgs
  VG                #PV #LV #SN Attr   VSize  VFree
  volgroup01          3   0   0 wz--n- 51.45G 51.45G
```

5.2.3. Creating the Logical Volume

The following command creates the striped logical volume **`striped_logical_volume`** from the volume group **`volgroup01`**. This example creates a logical volume that is 2 gigabytes in size, with three stripes and a stripe size of 4 kilobytes.

```
[root@tng3-1 ~]# lvcreate -i3 -I4 -L2G -nstriped_logical_volume volgroup01
  Rounding size (512 extents) up to stripe boundary size (513 extents)
  Logical volume "striped_logical_volume" created
```

5.2.4. Creating the File System

The following command creates a GFS2 file system on the logical volume.

```
[root@tng3-1 ~]# mkfs.gfs2 -plock_nolock -j 1 /dev/volgroup01/striped_logical_volume
This will destroy any data on /dev/volgroup01/striped_logical_volume.

Are you sure you want to proceed? [y/n] y

Device:                    /dev/volgroup01/striped_logical_volume
Blocksize:                 4096
Filesystem Size:           492484
Journals:                  1
Resource Groups:           8
Locking Protocol:          lock_nolock
Lock Table:

Syncing...
All Done
```

The following commands mount the logical volume and report the file system disk space usage.

```
[root@tng3-1 ~]# mount /dev/volgroup01/striped_logical_volume /mnt
[root@tng3-1 ~]# df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
                      13902624   1656776  11528232  13% /
/dev/hda1               101086     10787     85080  12% /boot
tmpfs                   127880         0    127880   0% /dev/shm
/dev/volgroup01/striped_logical_volume
                       1969936        20   1969916   1% /mnt
```

5.3. Splitting a Volume Group

In this example, an existing volume group consists of three physical volumes. If there is enough unused space on the physical volumes, a new volume group can be created without adding new disks.

In the initial set up, the logical volume **mylv** is carved from the volume group **myvol**, which in turn consists of the three physical volumes, **/dev/sda1**, **/dev/sdb1**, and **/dev/sdc1**.

After completing this procedure, the volume group **myvg** will consist of **/dev/sda1** and **/dev/sdb1**. A second volume group, **yourvg**, will consist of **/dev/sdc1**.

5.3.1. Determining Free Space

You can use the **pvscan** command to determine how much free space is currently available in the volume group.

```
[root@tng3-1 ~]# pvscan
  PV /dev/sda1  VG myvg   lvm2 [17.15 GB / 0    free]
  PV /dev/sdb1  VG myvg   lvm2 [17.15 GB / 12.15 GB free]
  PV /dev/sdc1  VG myvg   lvm2 [17.15 GB / 15.80 GB free]
  Total: 3 [51.45 GB] / in use: 3 [51.45 GB] / in no VG: 0 [0   ]
```

5.3.2. Moving the Data

You can move all the used physical extents in **/dev/sdc1** to **/dev/sdb1** with the **pvmove** command. The **pvmove** command can take a long time to execute.

```
[root@tng3-1 ~]# pvmove /dev/sdc1 /dev/sdb1
  /dev/sdc1: Moved: 14.7%
  /dev/sdc1: Moved: 30.3%
  /dev/sdc1: Moved: 45.7%
  /dev/sdc1: Moved: 61.0%
  /dev/sdc1: Moved: 76.6%
  /dev/sdc1: Moved: 92.2%
  /dev/sdc1: Moved: 100.0%
```

After moving the data, you can see that all of the space on **/dev/sdc1** is free.

```
[root@tng3-1 ~]# pvscan
  PV /dev/sda1   VG myvg   lvm2 [17.15 GB / 0    free]
  PV /dev/sdb1   VG myvg   lvm2 [17.15 GB / 10.80 GB free]
  PV /dev/sdc1   VG myvg   lvm2 [17.15 GB / 17.15 GB free]
  Total: 3 [51.45 GB] / in use: 3 [51.45 GB] / in no VG: 0 [0   ]
```

5.3.3. Splitting the Volume Group

To create the new volume group **yourvg**, use the **vgsplit** command to split the volume group **myvg**.

Before you can split the volume group, the logical volume must be inactive. If the file system is mounted, you must unmount the file system before deactivating the logical volume.

You can deactivate the logical volumes with the **lvchange** command or the **vgchange** command. The following command deactivates the logical volume **mylv** and then splits the volume group **yourvg** from the volume group **myvg**, moving the physical volume **/dev/sdc1** into the new volume group **yourvg**.

```
[root@tng3-1 ~]# lvchange -a n /dev/myvg/mylv
[root@tng3-1 ~]# vgsplit myvg yourvg /dev/sdc1
```

```
  Volume group "yourvg" successfully split from "myvg"
```

You can use the **vgs** command to see the attributes of the two volume groups.

```

[root@tng3-1 ~]# vgs
  VG     #PV #LV #SN Attr   VSize  VFree
  myvg     2   1   0 wz--n- 34.30G 10.80G
  yourvg   1   0   0 wz--n- 17.15G 17.15G
```

5.3.4. Creating the New Logical Volume

After creating the new volume group, you can create the new logical volume **`yourlv`**.

```

[root@tng3-1 ~]# lvcreate -L5G -n yourlv yourvg
  Logical volume "yourlv" created
```

5.3.5. Making a File System and Mounting the New Logical Volume

You can make a file system on the new logical volume and mount it.

```

[root@tng3-1 ~]#  mkfs.gfs2 -plock_nolock -j 1 /dev/yourvg/yourlv
This will destroy any data on /dev/yourvg/yourlv.

Are you sure you want to proceed? [y/n] y

Device:                    /dev/yourvg/yourlv
Blocksize:                 4096
Filesystem Size:           1277816
Journals:                  1
Resource Groups:           20
Locking Protocol:          lock_nolock
Lock Table:

Syncing...
All Done

[root@tng3-1 ~]# mount /dev/yourvg/yourlv /mnt
```

5.3.6. Activating and Mounting the Original Logical Volume

Since you had to deactivate the logical volume **`mylv`**, you need to activate it again before you can mount it.

```

root@tng3-1 ~]# lvchange -a y mylv

[root@tng3-1 ~]# mount /dev/myvg/mylv /mnt
[root@tng3-1 ~]# df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/yourvg/yourlv    24507776        32  24507744   1% /mnt
/dev/myvg/mylv        24507776        32  24507744   1% /mnt
```

5.4. Removing a Disk from a Logical Volume

This example shows how you can remove a disk from an existing logical volume, either to replace the disk or to use the disk as part of a different volume. In order to remove a disk, you must first move the extents on the LVM physical volume to a different disk or set of disks.

5.4.1. Moving Extents to Existing Physical Volumes

In this example, the logical volume is distributed across four physical volumes in the volume group **`myvg`**.

```
[root@tng3-1]# pvs -o+pv_used
  PV         VG   Fmt  Attr PSize  PFree  Used
  /dev/sda1  myvg lvm2 a-   17.15G 12.15G  5.00G
  /dev/sdb1  myvg lvm2 a-   17.15G 12.15G  5.00G
  /dev/sdc1  myvg lvm2 a-   17.15G 12.15G  5.00G
  /dev/sdd1  myvg lvm2 a-   17.15G  2.15G 15.00G
```

We want to move the extents off of **`/dev/sdb1`** so that we can remove it from the volume group.

If there are enough free extents on the other physical volumes in the volume group, you can execute the **`pvmove`** command on the device you want to remove with no other options and the extents will be distributed to the other devices.

```
[root@tng3-1 ~]# pvmove /dev/sdb1
  /dev/sdb1: Moved: 2.0%
 ...
  /dev/sdb1: Moved: 79.2%
 ...
  /dev/sdb1: Moved: 100.0%
```

After the **`pvmove`** command has finished executing, the distribution of extents is as follows:

```
[root@tng3-1]# pvs -o+pv_used
  PV         VG   Fmt  Attr PSize  PFree  Used
  /dev/sda1  myvg lvm2 a-   17.15G  7.15G 10.00G
  /dev/sdb1  myvg lvm2 a-   17.15G 17.15G     0
  /dev/sdc1  myvg lvm2 a-   17.15G 12.15G  5.00G
  /dev/sdd1  myvg lvm2 a-   17.15G  2.15G 15.00G
```

Use the **`vgreduce`** command to remove the physical volume **`/dev/sdb1`** from the volume group.

```
[root@tng3-1 ~]# vgreduce myvg /dev/sdb1
  Removed "/dev/sdb1" from volume group "myvg"
[root@tng3-1 ~]# pvs
  PV         VG   Fmt  Attr PSize  PFree
  /dev/sda1  myvg lvm2 a-   17.15G  7.15G
  /dev/sdb1       lvm2 --   17.15G 17.15G
  /dev/sdc1  myvg lvm2 a-   17.15G 12.15G
  /dev/sdd1  myvg lvm2 a-   17.15G  2.15G
```

The disk can now be physically removed or allocated to other users.

5.4.2. Moving Extents to a New Disk

In this example, the logical volume is distributed across three physical volumes in the volume group **`myvg`** as follows:

```
[root@tng3-1]# pvs -o+pv_used
  PV         VG   Fmt  Attr PSize  PFree  Used
  /dev/sda1  myvg lvm2 a-   17.15G  7.15G 10.00G
  /dev/sdb1  myvg lvm2 a-   17.15G 15.15G  2.00G
  /dev/sdc1  myvg lvm2 a-   17.15G 15.15G  2.00G
```

We want to move the extents of **`/dev/sdb1`** to a new device, **`/dev/sdd1`**.

5.4.2.1. Creating the New Physical Volume

Create a new physical volume from **`/dev/sdd1`**.

```
[root@tng3-1 ~]# pvcreate /dev/sdd1
  Physical volume "/dev/sdd1" successfully created
```

5.4.2.2. Adding the New Physical Volume to the Volume Group

Add **`/dev/sdd1`** to the existing volume group **`myvg`**.

```
[root@tng3-1 ~]# vgextend myvg /dev/sdd1
  Volume group "myvg" successfully extended
[root@tng3-1]# pvs -o+pv_used
  PV         VG   Fmt  Attr PSize  PFree  Used
  /dev/sda1   myvg lvm2 a-   17.15G  7.15G 10.00G
  /dev/sdb1   myvg lvm2 a-   17.15G 15.15G  2.00G
  /dev/sdc1   myvg lvm2 a-   17.15G 15.15G  2.00G
  /dev/sdd1   myvg lvm2 a-   17.15G 17.15G     0
```

5.4.2.3. Moving the Data

Use the **`pvmove`** command to move the data from **`/dev/sdb1`** to **`/dev/sdd1`**.

```
[root@tng3-1 ~]# pvmove /dev/sdb1 /dev/sdd1
  /dev/sdb1: Moved: 10.0%
...
  /dev/sdb1: Moved: 79.7%
...
  /dev/sdb1: Moved: 100.0%

[root@tng3-1]# pvs -o+pv_used
  PV          VG   Fmt  Attr PSize  PFree  Used
  /dev/sda1   myvg lvm2 a-   17.15G  7.15G 10.00G
  /dev/sdb1   myvg lvm2 a-   17.15G 17.15G     0
  /dev/sdc1   myvg lvm2 a-   17.15G 15.15G  2.00G
  /dev/sdd1   myvg lvm2 a-   17.15G 15.15G  2.00G
```

5.4.2.4. Removing the Old Physical Volume from the Volume Group

After you have moved the data off **`/dev/sdb1`**, you can remove it from the volume group.

```
[root@tng3-1 ~]# vgreduce myvg /dev/sdb1
  Removed "/dev/sdb1" from volume group "myvg"
```

You can now reallocate the disk to another volume group or remove the disk from the system.

5.5. Creating a Mirrored LVM Logical Volume in a Cluster

Creating a mirrored LVM logical volume in a cluster requires the same commands and procedures as creating a mirrored LVM logical volume on a single node. However, in order to create a mirrored LVM volume in a cluster the cluster and cluster mirror infrastructure must be running, the cluster must be quorate, and the locking type in the **`lvm.conf`** file must be set correctly to enable cluster locking, either directly or by means of the **`lvmconf`** command as described in *Section 3.1, "Creating LVM Volumes in a Cluster"*.

The following procedure creates a mirrored LVM volume in a cluster. First the procedure checks to see whether the cluster services are installed and running, then the procedure creates the mirrored volume.

1. In order to create a mirrored logical volume that is shared by all of the nodes in a cluster, the locking type must be set correctly in the **`lvm.conf`** file in every node of the cluster. By default, the locking type is set to local. To change this, execute the following command in each node of the cluster to enable clustered locking:

   ```
   # /sbin/lvmconf --enable-cluster
   ```

2. To create a clustered logical volume, the cluster infrastructure must be up and running on every node in the cluster. The following example verifies that the **`clvmd`** daemon is running on the node from which it was issued:

   ```
   [root@doc-07 ~]# ps auxw | grep clvmd
   root     17642  0.0  0.1 32164 1072 ?        Ssl  Apr06   0:00 clvmd -T20 -t 90
   ```

   The following command shows the local view of the cluster status:

   ```
   [root@example-01 ~]# cman_tool services
   fence domain
   member count  3
   victim count  0
   victim now    0
   master nodeid 2
   wait state    none
   members       1 2 3

   dlm lockspaces
   name          clvmd
   id            0x4104eefa
   flags         0x00000000
   change        member 3 joined 1 remove 0 failed 0 seq 1,1
   members       1 2 3
   ```

3. Ensure that the **`cmirror`** package is installed.

4. Start the **`cmirrord`** service.

   ```
   [root@hexample-01 ~]# service cmirrord start
   Starting cmirrord:                                         [  OK  ]
   ```

5. Create the mirror. The first step is creating the physical volumes. The following commands create three physical volumes. Two of the physical volumes will be used for the legs of the mirror, and the third physical volume will contain the mirror log.

```
[root@doc-07 ~]# pvcreate /dev/xvdb1
  Physical volume "/dev/xvdb1" successfully created
[root@doc-07 ~]# pvcreate /dev/xvdb2
  Physical volume "/dev/xvdb2" successfully created
[root@doc-07 ~]# pvcreate /dev/xvdc1
  Physical volume "/dev/xvdc1" successfully created
```

6. Create the volume group. This example creates a volume group **`vg001`** that consists of the three physical volumes that were created in the previous step.

```
[root@doc-07 ~]# vgcreate vg001 /dev/xvdb1 /dev/xvdb2 /dev/xvdc1
  Clustered volume group "vg001" successfully created
```

Note that the output of the **`vgcreate`** command indicates that the volume group is clustered. You can verify that a volume group is clustered with the **`vgs`** command, which will show the volume group's attributes. If a volume group is clustered, it will show a c attribute.

```
[root@doc-07 ~]# vgs vg001
  VG       #PV #LV #SN Attr   VSize  VFree
  vg001      3   0   0 wz--nc 68.97G 68.97G
```

7. Create the mirrored logical volume. This example creates the logical volume **`mirrorlv`** from the volume group **`vg001`**. This volume has one mirror leg. This example specifies which extents of the physical volume will be used for the logical volume.

```
[root@doc-07 ~]# lvcreate -l 1000 -m1 vg001 -n mirrorlv /dev/xvdb1:1-1000 /dev/
xvdb2:1-1000 /dev/xvdc1:0
  Logical volume "mirrorlv" created
```

You can use the **`lvs`** command to display the progress of the mirror creation. The following example shows that the mirror is 47% synced, then 91% synced, then 100% synced when the mirror is complete.

```
[root@doc-07 log]# lvs vg001/mirrorlv
  LV       VG       Attr   LSize Origin Snap%  Move Log           Copy%  Convert
  mirrorlv vg001    mwi-a- 3.91G                    vg001_mlog     47.00
[root@doc-07 log]# lvs vg001/mirrorlv
  LV       VG       Attr   LSize Origin Snap%  Move Log           Copy%  Convert
  mirrorlv vg001    mwi-a- 3.91G                    vg001_mlog     91.00
[root@doc-07 ~]#  lvs vg001/mirrorlv
  LV       VG       Attr   LSize Origin Snap%  Move Log           Copy%  Convert
  mirrorlv vg001    mwi-a- 3.91G                    vg001_mlog    100.00
```

The completion of the mirror is noted in the system log:

```
May 10 14:52:52 doc-07 [19402]: Monitoring mirror device vg001-mirrorlv for events
May 10 14:55:00 doc-07 lvm[19402]: vg001-mirrorlv is now in-sync
```

8. You can use the **`lvs`** with the **`-o +devices`** options to display the configuration of the mirror, including which devices make up the mirror legs. You can see that the logical volume in this example is composed of two linear images and one log.

```
[root@doc-07 ~]# lvs -a -o +devices
  LV                  VG         Attr   LSize  Origin Snap%  Move Log           Copy%
 Convert Devices
  mirrorlv            vg001      mwi-a-  3.91G                    mirrorlv_mlog 100.00
      mirrorlv_mimage_0(0),mirrorlv_mimage_1(0)
  [mirrorlv_mimage_0] vg001      iwi-ao  3.91G
      /dev/xvdb1(1)
  [mirrorlv_mimage_1] vg001      iwi-ao  3.91G
      /dev/xvdb2(1)
  [mirrorlv_mlog]     vg001      lwi-ao  4.00M
      /dev/xvdc1(0)
```

You can use the **`seg_pe_ranges`** option of the **`lvs`** to display the data layout. You can use this option to verify that your layout is properly redundant. The output of this command displays PE ranges in the same format that the **`lvcreate`** and **`lvresize`** commands take as input.

```
[root@doc-07 ~]# lvs -a -o +seg_pe_ranges --segments
  PE Ranges
  mirrorlv_mimage_0:0-999 mirrorlv_mimage_1:0-999
  /dev/xvdb1:1-1000
  /dev/xvdb2:1-1000
  /dev/xvdc1:0-0
```

Note

For information on recovering from the failure of one of the legs of an LVM mirrored volume, see *Section 6.3, “Recovering from LVM Mirror Failure”*.

Chapter 6.

LVM Troubleshooting

This chapter provide instructions for troubleshooting a variety of LVM issues.

6.1. Troubleshooting Diagnostics

If a command is not working as expected, you can gather diagnostics in the following ways:

- Use the **`-v`**, **`-vv`**, **`-vvv`**, or **`-vvvv`** argument of any command for increasingly verbose levels of output.
- If the problem is related to the logical volume activation, set 'activation = 1' in the 'log' section of the configuration file and run the command with the **`-vvvv`** argument. After you have finished examining this output be sure to reset this parameter to 0, to avoid possible problems with the machine locking during low memory situations.
- Run the **`lvmdump`** command, which provides an information dump for diagnostic purposes. For information, see the **`lvmdump`**(8) man page.
- Execute the **`lvs -v`**, **`pvs -a`** or **`dmsetup info -c`** command for additional system information.
- Examine the last backup of the metadata in the **`/etc/lvm/backup`** file and archived versions in the **`/etc/lvm/archive`** file.
- Check the current configuration information by running the **`lvm dumpconfig`** command.
- Check the **`.cache`** file in the **`/etc/lvm`** directory for a record of which devices have physical volumes on them.

6.2. Displaying Information on Failed Devices

You can use the **`-P`** argument of the **`lvs`** or **`vgs`** command to display information about a failed volume that would otherwise not appear in the output. This argument permits some operations even though the metadata is not completely consistent internally. For example, if one of the devices that made up the volume group **`vg`** failed, the **`vgs`** command might show the following output.

```
[root@link-07 tmp]# vgs -o +devices
  Volume group "vg" not found
```

If you specify the **`-P`** argument of the **`vgs`** command, the volume group is still unusable but you can see more information about the failed device.

```
[root@link-07 tmp]# vgs -P -o +devices
  Partial mode. Incomplete volume groups will be activated read-only.
  VG   #PV #LV #SN Attr   VSize VFree Devices
  vg     9   2   0 rz-pn- 2.11T 2.07T unknown device(0)
  vg     9   2   0 rz-pn- 2.11T 2.07T unknown device(5120),/dev/sda1(0)
```

In this example, the failed device caused both a linear and a striped logical volume in the volume group to fail. The **`lvs`** command without the **`-P`** argument shows the following output.

```
[root@link-07 tmp]# lvs -a -o +devices
  Volume group "vg" not found
```

Using the **`-P`** argument shows the logical volumes that have failed.

```
[root@link-07 tmp]# lvs -P -a -o +devices
  Partial mode. Incomplete volume groups will be activated read-only.
  LV     VG   Attr   LSize  Origin Snap%  Move Log Copy%  Devices
  linear vg   -wi-a- 20.00G                               unknown device(0)
  stripe vg   -wi-a- 20.00G                               unknown device(5120),/dev/sda1(0)
```

The following examples show the output of the **`pvs`** and **`lvs`** commands with the **`-P`** argument specified when a leg of a mirrored logical volume has failed.

```
root@link-08 ~]#  vgs -a -o +devices -P
  Partial mode. Incomplete volume groups will be activated read-only.
  VG    #PV #LV #SN Attr   VSize VFree Devices
  corey   4   4   0 rz-pnc 1.58T 1.34T my_mirror_mimage_0(0),my_mirror_mimage_1(0)
  corey   4   4   0 rz-pnc 1.58T 1.34T /dev/sdd1(0)
  corey   4   4   0 rz-pnc 1.58T 1.34T unknown device(0)
  corey   4   4   0 rz-pnc 1.58T 1.34T /dev/sdb1(0)
```

```
[root@link-08 ~]# lvs -a -o +devices -P
  Partial mode. Incomplete volume groups will be activated read-only.
  LV                   VG    Attr   LSize   Origin Snap%  Move Log            Copy%  Devices
  my_mirror            corey mwi-a- 120.00G                    my_mirror_mlog   1.95
 my_mirror_mimage_0(0),my_mirror_mimage_1(0)
  [my_mirror_mimage_0] corey iwi-ao 120.00G                                           unknown
 device(0)
  [my_mirror_mimage_1] corey iwi-ao 120.00G                                           /dev/
sdb1(0)
  [my_mirror_mlog]     corey lwi-ao   4.00M                                           /dev/
sdd1(0)
```

6.3. Recovering from LVM Mirror Failure

This section provides an example of recovering from a situation where one leg of an LVM mirrored volume fails because the underlying device for a physical volume goes down and the **`mirror_log_fault_policy`** parameter is set to **`remove`**, requiring that you manually rebuild the mirror. For information on setting the **`mirror_log_fault_policy`** parameter, refer to *Section 6.3, “Recovering from LVM Mirror Failure”*.

When a mirror leg fails, LVM converts the mirrored volume into a linear volume, which continues to operate as before but without the mirrored redundancy. At that point, you can add a new disk device to the system to use as a replacement physical device and rebuild the mirror.

The following command creates the physical volumes which will be used for the mirror.

```
[root@link-08 ~]# pvcreate /dev/sd[abcdefgh][12]
  Physical volume "/dev/sda1" successfully created
  Physical volume "/dev/sda2" successfully created
  Physical volume "/dev/sdb1" successfully created
  Physical volume "/dev/sdb2" successfully created
  Physical volume "/dev/sdc1" successfully created
  Physical volume "/dev/sdc2" successfully created
  Physical volume "/dev/sdd1" successfully created
  Physical volume "/dev/sdd2" successfully created
  Physical volume "/dev/sde1" successfully created
  Physical volume "/dev/sde2" successfully created
  Physical volume "/dev/sdf1" successfully created
  Physical volume "/dev/sdf2" successfully created
  Physical volume "/dev/sdg1" successfully created
  Physical volume "/dev/sdg2" successfully created
  Physical volume "/dev/sdh1" successfully created
  Physical volume "/dev/sdh2" successfully created
```

The following commands creates the volume group **`vg`** and the mirrored volume **`groupfs`**.

```
[root@link-08 ~]# vgcreate vg /dev/sd[abcdefgh][12]
  Volume group "vg" successfully created
[root@link-08 ~]# lvcreate -L 750M -n groupfs -m 1 vg /dev/sda1 /dev/sdb1 /dev/sdc1
  Rounding up size to full physical extent 752.00 MB
  Logical volume "groupfs" created
```

You can use the **`lvs`** command to verify the layout of the mirrored volume and the underlying devices for the mirror leg and the mirror log. Note that in the first example the mirror is not yet completely synced; you should wait until the **`Copy%`** field displays 100.00 before continuing.

```
[root@link-08 ~]# lvs -a -o +devices
  LV                 VG   Attr   LSize   Origin Snap%  Move Log          Copy% Devices
  groupfs            vg   mwi-a- 752.00M                    groupfs_mlog 21.28
 groupfs_mimage_0(0),groupfs_mimage_1(0)
  [groupfs_mimage_0] vg   iwi-ao 752.00M                                       /dev/sda1(0)
  [groupfs_mimage_1] vg   iwi-ao 752.00M                                       /dev/sdb1(0)
  [groupfs_mlog]     vg   lwi-ao   4.00M                                       /dev/sdc1(0)

[root@link-08 ~]# lvs -a -o +devices
  LV                 VG   Attr   LSize   Origin Snap%  Move Log          Copy%  Devices
  groupfs            vg   mwi-a- 752.00M                    groupfs_mlog 100.00
 groupfs_mimage_0(0),groupfs_mimage_1(0)
  [groupfs_mimage_0] vg   iwi-ao 752.00M                                        /dev/sda1(0)
  [groupfs_mimage_1] vg   iwi-ao 752.00M                                        /dev/sdb1(0)
  [groupfs_mlog]     vg   lwi-ao   4.00M     i                                  /dev/sdc1(0)
```

In this example, the primary leg of the mirror **`/dev/sda1`** fails. Any write activity to the mirrored volume causes LVM to detect the failed mirror. When this occurs, LVM converts the mirror into a single linear volume. In this case, to trigger the conversion, we execute a **dd** command

```
[root@link-08 ~]# dd if=/dev/zero of=/dev/vg/groupfs count=10
10+0 records in
10+0 records out
```

You can use the **`lvs`** command to verify that the device is now a linear device. Because of the failed disk, I/O errors occur.

```
[root@link-08 ~]# lvs -a -o +devices
  /dev/sda1: read failed after 0 of 2048 at 0: Input/output error
  /dev/sda2: read failed after 0 of 2048 at 0: Input/output error
  LV      VG   Attr   LSize   Origin Snap%  Move Log Copy%  Devices
  groupfs vg   -wi-a- 752.00M                               /dev/sdb1(0)
```

At this point you should still be able to use the logical volume, but there will be no mirror redundancy.

To rebuild the mirrored volume, you replace the broken drive and recreate the physical volume. If you use the same disk rather than replacing it with a new one, you will see "inconsistent" warnings when you run the **`pvcreate`** command. You can prevent that warning from appearing by executing the **`vgreduce --removemissing`** command.

```
[root@link-08 ~]# pvcreate /dev/sdi[12]
  Physical volume "/dev/sdi1" successfully created
  Physical volume "/dev/sdi2" successfully created

[root@link-08 ~]# pvscan
  PV /dev/sdb1   VG vg   lvm2 [67.83 GB / 67.10 GB free]
  PV /dev/sdb2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdc1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdc2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdd1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdd2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sde1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sde2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdf1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdf2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdg1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdg2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdh1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdh2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdi1           lvm2 [603.94 GB]
  PV /dev/sdi2           lvm2 [603.94 GB]
  Total: 16 [2.11 TB] / in use: 14 [949.65 GB] / in no VG: 2 [1.18 TB]
```

Next you extend the original volume group with the new physical volume.

```
[root@link-08 ~]# vgextend vg /dev/sdi[12]
  Volume group "vg" successfully extended

[root@link-08 ~]# pvscan
  PV /dev/sdb1   VG vg   lvm2 [67.83 GB / 67.10 GB free]
  PV /dev/sdb2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdc1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdc2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdd1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdd2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sde1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sde2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdf1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdf2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdg1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdg2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdh1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdh2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdi1   VG vg   lvm2 [603.93 GB / 603.93 GB free]
  PV /dev/sdi2   VG vg   lvm2 [603.93 GB / 603.93 GB free]
  Total: 16 [2.11 TB] / in use: 16 [2.11 TB] / in no VG: 0 [0   ]
```

Convert the linear volume back to its original mirrored state.

```
[root@link-08 ~]# lvconvert -m 1 /dev/vg/groupfs /dev/sdi1 /dev/sdb1 /dev/sdc1
```

```
  Logical volume mirror converted.
```

You can use the **`lvs`** command to verify that the mirror is restored.

```
[root@link-08 ~]# lvs -a -o +devices
  LV                 VG   Attr   LSize   Origin Snap%  Move Log          Copy% Devices
  groupfs            vg   mwi-a- 752.00M                    groupfs_mlog 68.62
 groupfs_mimage_0(0),groupfs_mimage_1(0)
  [groupfs_mimage_0] vg   iwi-ao 752.00M                                       /dev/sdb1(0)
  [groupfs_mimage_1] vg   iwi-ao 752.00M                                       /dev/sdi1(0)
  [groupfs_mlog]     vg   lwi-ao   4.00M                                       /dev/sdc1(0)
```

6.4. Recovering Physical Volume Metadata

If the volume group metadata area of a physical volume is accidentally overwritten or otherwise destroyed, you will get an error message indicating that the metadata area is incorrect, or that the system was unable to find a physical volume with a particular UUID. You may be able to recover the data the physical volume by writing a new metadata area on the physical volume specifying the same UUID as the lost metadata.

Warning

You should not attempt this procedure with a working LVM logical volume. You will lose your data if you specify the incorrect UUID.

The following example shows the sort of output you may see if the metadata area is missing or corrupted.

```
[root@link-07 backup]# lvs -a -o +devices
  Couldn't find device with uuid 'FmGRh3-zhok-iVI8-7qTD-S5BI-MAEN-NYM5Sk'.
  Couldn't find all physical volumes for volume group VG.
  Couldn't find device with uuid 'FmGRh3-zhok-iVI8-7qTD-S5BI-MAEN-NYM5Sk'.
  Couldn't find all physical volumes for volume group VG.
  ...
```

You may be able to find the UUID for the physical volume that was overwritten by looking in the **`/etc/lvm/archive`** directory. Look in the file ***`VolumeGroupName_xxxx`*`.vg`** for the last known valid archived LVM metadata for that volume group.

Alternately, you may find that deactivating the volume and setting the **`partial`** (**`-P`**) argument will enable you to find the UUID of the missing corrupted physical volume.

```
[root@link-07 backup]# vgchange -an --partial
  Partial mode. Incomplete volume groups will be activated read-only.
  Couldn't find device with uuid 'FmGRh3-zhok-iVI8-7qTD-S5BI-MAEN-NYM5Sk'.
  Couldn't find device with uuid 'FmGRh3-zhok-iVI8-7qTD-S5BI-MAEN-NYM5Sk'.
  ...
```

Use the **`--uuid`** and **`--restorefile`** arguments of the **`pvcreate`** command to restore the physical volume. The following example labels the **`/dev/sdh1`** device as a physical volume with the UUID indicated above, **`FmGRh3-zhok-iVI8-7qTD-S5BI-MAEN-NYM5Sk`**. This command restores the physical volume label with the metadata information contained in **`VG_00050.vg`**, the most recent good archived metadata for the volume group. The **`restorefile`** argument instructs the **`pvcreate`** command to make the new physical volume compatible with the old one on the volume group, ensuring that the new metadata will not be placed where the old physical volume contained data (which could happen, for example, if the original **`pvcreate`** command had used the command line arguments that control metadata placement, or if the physical volume was originally created using a different version of the software that used different defaults). The **`pvcreate`** command overwrites only the LVM metadata areas and does not affect the existing data areas.

```
[root@link-07 backup]# pvcreate --uuid "FmGRh3-zhok-iVI8-7qTD-S5BI-MAEN-NYM5Sk" --
restorefile /etc/lvm/archive/VG_00050.vg /dev/sdh1
  Physical volume "/dev/sdh1" successfully created
```

You can then use the **`vgcfgrestore`** command to restore the volume group's metadata.

```
[root@link-07 backup]# vgcfgrestore VG
  Restored volume group VG
```

You can now display the logical volumes.

```
[root@link-07 backup]# lvs -a -o +devices
  LV     VG   Attr   LSize   Origin Snap%  Move Log Copy%  Devices
  stripe VG   -wi--- 300.00G                               /dev/sdh1 (0),/dev/sda1(0)
  stripe VG   -wi--- 300.00G                               /dev/sdh1 (34728),/dev/sdb1(0)
```

The following commands activate the volumes and display the active volumes.

```
[root@link-07 backup]# lvchange -ay /dev/VG/stripe
[root@link-07 backup]# lvs -a -o +devices
  LV     VG   Attr   LSize   Origin Snap%  Move Log Copy%  Devices
  stripe VG   -wi-a- 300.00G                               /dev/sdh1 (0),/dev/sda1(0)
  stripe VG   -wi-a- 300.00G                               /dev/sdh1 (34728),/dev/sdb1(0)
```

If the on-disk LVM metadata takes as least as much space as what overrode it, this command can recover the physical volume. If what overrode the metadata went past the metadata area, the data on the volume may have been affected. You might be able to use the **`fsck`** command to recover that data.

6.5. Replacing a Missing Physical Volume

If a physical volume fails or otherwise needs to be replaced, you can label a new physical volume to replace the one that has been lost in the existing volume group by following the same procedure as you would for recovering physical volume metadata, described in *Section 6.4, “Recovering Physical Volume Metadata”*. You can use the **`--partial`** and **`--verbose`** arguments of the **`vgdisplay`** command to display the UUIDs and sizes of any physical volumes that are no longer present. If you wish to substitute another physical volume of the same size, you can use the **`pvcreate`** command with the **`--restorefile`** and **`--uuid`** arguments to initialize a new device with the same UUID as the missing physical volume. You can then use the **`vgcfgrestore`** command to restore the volume group's metadata.

6.6. Removing Lost Physical Volumes from a Volume Group

If you lose a physical volume, you can activate the remaining physical volumes in the volume group with the **`--partial`** argument of the **`vgchange`** command. You can remove all the logical volumes that used that physical volume from the volume group with the **`--removemissing`** argument of the **`vgreduce`** command.

It is recommended that you run the **`vgreduce`** command with the **`--test`** argument to verify what you will be destroying.

Like most LVM operations, the **`vgreduce`** command is reversible in a sense if you immediately use the **`vgcfgrestore`** command to restore the volume group metadata to its previous state. For example, if you used the **`--removemissing`** argument of the **`vgreduce`** command without the **`--test`** argument and find you have removed logical volumes you wanted to keep, you can still replace the physical volume and use another **`vgcfgrestore`** command to return the volume group to its previous state.

6.7. Insufficient Free Extents for a Logical Volume

You may get the error message "Insufficient free extents" when creating a logical volume when you think you have enough extents based on the output of the **`vgdisplay`** or **`vgs`** commands. This is because these commands round figures to 2 decimal places to provide human-readable output. To specify exact size, use free physical extent count instead of some multiple of bytes to determine the size of the logical volume.

The **`vgdisplay`** command, by default, includes this line of output that indicates the free physical extents.

```
# vgdisplay
  --- Volume group ---
  ...
  Free  PE / Size       8780 / 34.30 GB
```

Alternately, you can use the **`vg_free_count`** and **`vg_extent_count`** arguments of the **`vgs`** command to display the free extents and the total number of extents.

```
[root@tng3-1 ~]# vgs -o +vg_free_count,vg_extent_count
  VG     #PV #LV #SN Attr   VSize  VFree  Free #Ext
  testvg   2   0   0 wz--n- 34.30G 34.30G 8780 8780
```

With 8780 free physical extents, you can run the following command, using the lower-case l argument to use extents instead of bytes:

```
# lvcreate -l8780 -n testlv testvg
```

This uses all the free extents in the volume group.

```
# vgs -o +vg_free_count,vg_extent_count
  VG     #PV #LV #SN Attr   VSize  VFree Free #Ext
  testvg   2   1   0 wz--n- 34.30G     0     0 8780
```

Alternately, you can extend the logical volume to use a percentage of the remaining free space in the volume group by using the **`-l`** argument of the **`lvcreate`** command. For information, see *Section 4.4.1, “Creating Linear Logical Volumes”*.

Chapter 7.

LVM Administration with the LVM GUI

In addition to the Command Line Interface (CLI), LVM provides a Graphical User Interface (GUI) which you can use to configure LVM logical volumes. You can bring up this utility by typing **`system-config-lvm`**. The LVM chapter of the *Storage Administration Guide* provides step-by-step instructions for configuring an LVM logical volume using this utility.

Appendix A. The Device Mapper

The Device Mapper is a kernel driver that provides a framework for volume management. It provides a generic way of creating mapped devices, which may be used as logical volumes. It does not specifically know about volume groups or metadata formats.

The Device Mapper provides the foundation for a number of higher-level technologies. In addition to LVM, Device-Mapper multipath and the **`dmraid`** command use the Device Mapper. The application interface to the Device Mapper is the **`ioctl`** system call. The user interface is the **`dmsetup`** command.

LVM logical volumes are activated using the Device Mapper. Each logical volume is translated into a mapped device. Each segment translates into a line in the mapping table that describes the device. The Device Mapper supports a variety of mapping targets, including linear mapping, striped mapping, and error mapping. So, for example, two disks may be concatenated into one logical volume with a pair of linear mappings, one for each disk. When LVM creates a volume, it creates an underlying device-mapper device that can be queried with the **`dmsetup`** command. For information about the format of devices in a mapping table, see *Section A.1, “Device Table Mappings”*. For information about using the **`dmsetup`** command to query a device, see *Section A.2, “The dmsetup Command”*.

A.1. Device Table Mappings

A mapped device is defined by a table that specifies how to map each range of logical sectors of the device using a supported Device Table mapping. The table for a mapped device is constructed from a list of lines of the form:

```
start length mapping [mapping_parameters...]
```

In the first line of a Device Mapper table, the *`start`* parameter must equal 0. The *`start`* + *`length`* parameters on one line must equal the *`start`* on the next line. Which mapping parameters are specified in a line of the mapping table depends on which *`mapping`* type is specified on the line.

Sizes in the Device Mapper are always specified in sectors (512 bytes).

When a device is specified as a mapping parameter in the Device Mapper, it can be referenced by the device name in the filesystem (for example, **`/dev/hda`**) or by the major and minor numbers in the format *`major`*:*`minor`*. The major:minor format is preferred because it avoids pathname lookups.

The following shows a sample mapping table for a device. In this table there are four linear targets:

```
0 35258368 linear 8:48 65920
35258368 35258368 linear 8:32 65920
70516736 17694720 linear 8:16 17694976
88211456 17694720 linear 8:16 256
```

The first 2 parameters of each line are the segment starting block and the length of the segment. The next keyword is the mapping target, which in all of the cases in this example is **`linear`**. The rest of the line consists of the parameters for a **`linear`** target.

The following subsections describe the format of the following mappings:

- linear
- striped

- mirror
- snapshot and snapshot-origin
- error
- zero
- multipath
- crypt

A.1.1. The linear Mapping Target

A linear mapping target maps a continuous range of blocks onto another block device. The format of a linear target is as follows:

```
start length linear device offset
```

`start`
: starting block in virtual device

`length`
: length of this segment

`device`
: block device, referenced by the device name in the filesystem or by the major and minor numbers in the format *`major`*:*`minor`*

`offset`
: starting offset of the mapping on the device

The following example shows a linear target with a starting block in the virtual device of 0, a segment length of 1638400, a major:minor number pair of 8:2, and a starting offset for the device of 41146992.

```
0 16384000 linear 8:2 41156992
```

The following example shows a linear target with the device parameter specified as the device **`/dev/hda`**.

```
0 20971520 linear /dev/hda 384
```

A.1.2. The striped Mapping Target

The striped mapping target supports striping across physical devices. It takes as arguments the number of stripes and the striping chunk size followed by a list of pairs of device name and sector. The format of a striped target is as follows:

```
start length striped #stripes chunk_size device1 offset1 ... deviceN offsetN
```

There is one set of *`device`* and *`offset`* parameters for each stripe.

`start`
: starting block in virtual device

`length`
: length of this segment

`#stripes`
: number of stripes for the virtual device

`chunk_size`
: number of sectors written to each stripe before switching to the next; must be power of 2 at least as big as the kernel page size

`device`
: block device, referenced by the device name in the filesystem or by the major and minor numbers in the format *`major`*:*`minor`*.

`offset`
: starting offset of the mapping on the device

The following example shows a striped target with three stripes and a chunk size of 128:

```
0 73728 striped 3 128 8:9 384 8:8 384 8:7 9789824
```

0
: starting block in virtual device

73728
: length of this segment

striped 3 128
: stripe across three devices with chunk size of 128 blocks

8:9
: major:minor numbers of first device

384
: starting offset of the mapping on the first device

8:8
: major:minor numbers of second device

384
: starting offset of the mapping on the second device

8:7
: major:minor numbers of third device

9789824
: starting offset of the mapping on the third device

The following example shows a striped target for 2 stripes with 256 KiB chunks, with the device parameters specified by the device names in the file system rather than by the major and minor numbers.

```
0 65536 striped 2 512 /dev/hda 0 /dev/hdb 0
```

A.1.3. The mirror Mapping Target

The mirror mapping target supports the mapping of a mirrored logical device. The format of a mirrored target is as follows:

```
start length mirror log_type #logargs logarg1 ... logargN #devs device1 offset1 ... deviceN
 offsetN
```

`start`
: starting block in virtual device

`length`
: length of this segment

`log_type`
: The possible log types and their arguments are as follows:

  `core`
  : The mirror is local and the mirror log is kept in core memory. This log type takes 1 - 3 arguments:

    `regionsize` [[**no**]**sync**] [**`block_on_error`**]

  `disk`
  : The mirror is local and the mirror log is kept on disk. This log type takes 2 - 4 arguments:

    `logdevice regionsize` [[**no**]**sync**] [**`block_on_error`**]

  `clustered_core`
  : The mirror is clustered and the mirror log is kept in core memory. This log type takes 2 - 4 arguments:

    `regionsize UUID` [[**no**]**sync**] [**`block_on_error`**]

  `clustered_disk`
  : The mirror is clustered and the mirror log is kept on disk. This log type takes 3 - 5 arguments:

    `logdevice regionsize UUID` [[**no**]**sync**] [**`block_on_error`**]

  LVM maintains a small log which it uses to keep track of which regions are in sync with the mirror or mirrors. The *`regionsize`* argument specifies the size of these regions.

  In a clustered environment, the *`UUID`* argument is a unique identifier associated with the mirror log device so that the log state can be maintained throughout the cluster.

  The optional **`[no]sync`** argument can be used to specify the mirror as "in-sync" or "out-of-sync". The **`block_on_error`** argument is used to tell the mirror to respond to errors rather than ignoring them.

`#log_args`
: number of log arguments that will be specified in the mapping

`logargs`
: the log arguments for the mirror; the number of log arguments provided is specified by the *`#log-args`* parameter and the valid log arguments are determined by the *`log_type`* parameter.

`#devs`
: the number of legs in the mirror; a device and an offset is specified for each leg

`device`
: block device for each mirror leg, referenced by the device name in the filesystem or by the major and minor numbers in the format *`major`*:*`minor`*. A block device and offset is specified for each mirror leg, as indicated by the *`#devs`* parameter.

`offset`
: starting offset of the mapping on the device. A block device and offset is specified for each mirror leg, as indicated by the *`#devs`* parameter.

The following example shows a mirror mapping target for a clustered mirror with a mirror log kept on disk.

```
0 52428800 mirror clustered_disk 4 253:2 1024 UUID block_on_error 3 253:3 0 253:4 0 253:5 0
```

0
: starting block in virtual device

52428800
: length of this segment

mirror clustered_disk
: mirror target with a log type specifying that mirror is clustered and the mirror log is maintained on disk

4
: 4 mirror log arguments will follow

253:2
: major:minor numbers of log device

1024
: region size the mirror log uses to keep track of what is in sync

`UUID`
: UUID of mirror log device to maintain log information throughout a cluster

`block_on_error`
: mirror should respond to errors

3
: number of legs in mirror

253:3 0 253:4 0 253:5 0
: major:minor numbers and offset for devices constituting each leg of mirror

A.1.4. The snapshot and snapshot-origin Mapping Targets

When you create the first LVM snapshot of a volume, four Device Mapper devices are used:

1. A device with a **`linear`** mapping containing the original mapping table of the source volume.
2. A device with a **`linear`** mapping used as the copy-on-write (COW) device for the source volume; for each write, the original data is saved in the COW device of each snapshot to keep its visible content unchanged (until the COW device fills up).
3. A device with a **`snapshot`** mapping combining #1 and #2, which is the visible snapshot volume.
4. The "original" volume (which uses the device number used by the original source volume), whose table is replaced by a "snapshot-origin" mapping from device #1.

A fixed naming scheme is used to create these devices, For example, you might use the following commands to create an LVM volume named **`base`** and a snapshot volume named **`snap`** based on that volume.

```
# lvcreate -L 1G -n base volumeGroup
# lvcreate -L 100M --snapshot -n snap volumeGroup/base
```

This yields four devices, which you can view with the following commands:

```
# dmsetup table|grep volumeGroup
volumeGroup-base-real: 0 2097152 linear 8:19 384
volumeGroup-snap-cow: 0 204800 linear 8:19 2097536
volumeGroup-snap: 0 2097152 snapshot 254:11 254:12 P 16
volumeGroup-base: 0 2097152 snapshot-origin 254:11

# ls -lL /dev/mapper/volumeGroup-*
brw-------  1 root root 254, 11 29 ago 18:15 /dev/mapper/volumeGroup-base-real
brw-------  1 root root 254, 12 29 ago 18:15 /dev/mapper/volumeGroup-snap-cow
brw-------  1 root root 254, 13 29 ago 18:15 /dev/mapper/volumeGroup-snap
brw-------  1 root root 254, 10 29 ago 18:14 /dev/mapper/volumeGroup-base
```

The format for the **`snapshot-origin`** target is as follows:

```
start length snapshot-origin origin
```

`start`
: starting block in virtual device

`length`
: length of this segment

`origin`
: base volume of snapshot

The **`snapshot-origin`** will normally have one or more snapshots based on it. Reads will be mapped directly to the backing device. For each write, the original data will be saved in the COW device of each snapshot to keep its visible content unchanged until the COW device fills up.

The format for the **`snapshot`** target is as follows:

```
start length snapshot origin COW-device P|N chunksize
```

`start`
: starting block in virtual device

`length`
: length of this segment

`origin`
: base volume of snapshot

`COW-device`
: Device on which changed chunks of data are stored

P|N
: P (Persistent) or N (Not persistent); indicates whether snapshot will survive after reboot. For transient snapshots (N) less metadata must be saved on disk; they can be kept in memory by the kernel.

`chunksize`
: Size in sectors of changed chunks of data that will be stored on the COW device

The following example shows a **`snapshot-origin`** target with an origin device of 254:11.

```
0 2097152 snapshot-origin 254:11
```

The following example shows a **`snapshot`** target with an origin device of 254:11 and a COW device of 254:12. This snapshot device is persistent across reboots and the chunk size for the data stored on the COW device is 16 sectors.

```
0 2097152 snapshot 254:11 254:12 P 16
```

A.1.5. The error Mapping Target

With an error mapping target, any I/O operation to the mapped sector fails.

An error mapping target can be used for testing. To test how a device behaves in failure, you can create a device mapping with a bad sector in the middle of a device, or you can swap out the leg of a mirror and replace the leg with an error target.

An error target can be used in place of a failing device, as a way of avoiding timeouts and retries on the actual device. It can serve as an intermediate target while you rearrange LVM metadata during failures.

The **`error`** mapping target takes no additional parameters besides the *`start`* and *`length`* parameters.

The following example shows an **`error`** target.

```
0 65536 error
```

A.1.6. The zero Mapping Target

The **`zero`** mapping target is a block device equivalent of **`/dev/zero`**. A read operation to this mapping returns blocks of zeros. Data written to this mapping is discarded, but the write succeeds.

The **`zero`** mapping target takes no additional parameters besides the *`start`* and *`length`* parameters.

The following example shows a **`zero`** target for a 16Tb Device.

```
0 65536 zero
```

A.1.7. The multipath Mapping Target

The multipath mapping target supports the mapping of a multipathed device. The format for the **`multipath`** target is as follows:

```
start length  multipath  #features [feature1 ... featureN] #handlerargs [handlerarg1 ...
 handlerargN] #pathgroups pathgroup pathgroupargs1 ... pathgroupargsN
```

There is one set of *`pathgroupargs`* parameters for each path group.

`start`
: starting block in virtual device

`length`
: length of this segment

`#features`
: The number of multipath features, followed by those features. If this parameter is zero, then there is no *`feature`* parameter and the next device mapping parameter is *`#handlerargs`*. Currently there is one supported feature that can be set with the **`features`** attribute in the **`multipath.conf`** file, **`queue_if_no_path`**. This indicates that this multipathed device is currently set to queue I/O operations if there is no path available.

In the following example, the **`no_path_retry`** attribute in the **`multipath.conf`** file has been set to queue I/O operations only until all paths have been marked as failed after a set number of attempts have been made to use the paths. In this case, the mapping appears as follows until all the path checkers have failed the specified number of checks.

```
0 71014400 multipath 1 queue_if_no_path 0 2 1 round-robin 0 2 1 66:128 \
1000 65:64 1000 round-robin 0 2 1 8:0 1000 67:192 1000
```

After all the path checkers have failed the specified number of checks, the mapping would appear as follows.

```
0 71014400 multipath 0 0 2 1 round-robin 0 2 1 66:128 1000 65:64 1000 \
round-robin 0 2 1 8:0 1000 67:192 1000
```

`#handlerargs`
: The number of hardware handler arguments, followed by those arguments. A hardware handler specifies a module that will be used to perform hardware-specific actions when switching path groups or handling I/O errors. If this is set to 0, then the next parameter is *`#pathgroups`*.

`#pathgroups`
: The number of path groups. A path group is the set of paths over which a multipathed device will load balance. There is one set of *`pathgroupargs`* parameters for each path group.

pathgroup

The next path group to try.

pathgroupsargs

Each path group consists of the following arguments:

```
pathselector #selectorargs #paths #pathargs device1 ioreqs1 ... deviceN ioreqsN
```

There is one set of path arguments for each path in the path group.

pathselector

Specifies the algorithm in use to determine what path in this path group to use for the next I/O operation.

#selectorargs

The number of path selector arguments which follow this argument in the multipath mapping. Currently, the value of this argument is always 0.

#paths

The number of paths in this path group.

#pathargs

The number of path arguments specified for each path in this group. Currently this number is always 1, the *ioreqs* argument.

device

The block device number of the path, referenced by the major and minor numbers in the format *major*:*minor*

ioreqs

The number of I/O requests to route to this path before switching to the next path in the current group.

Figure A.1, “Multipath Mapping Target” shows the format of a multipath target with two path groups.

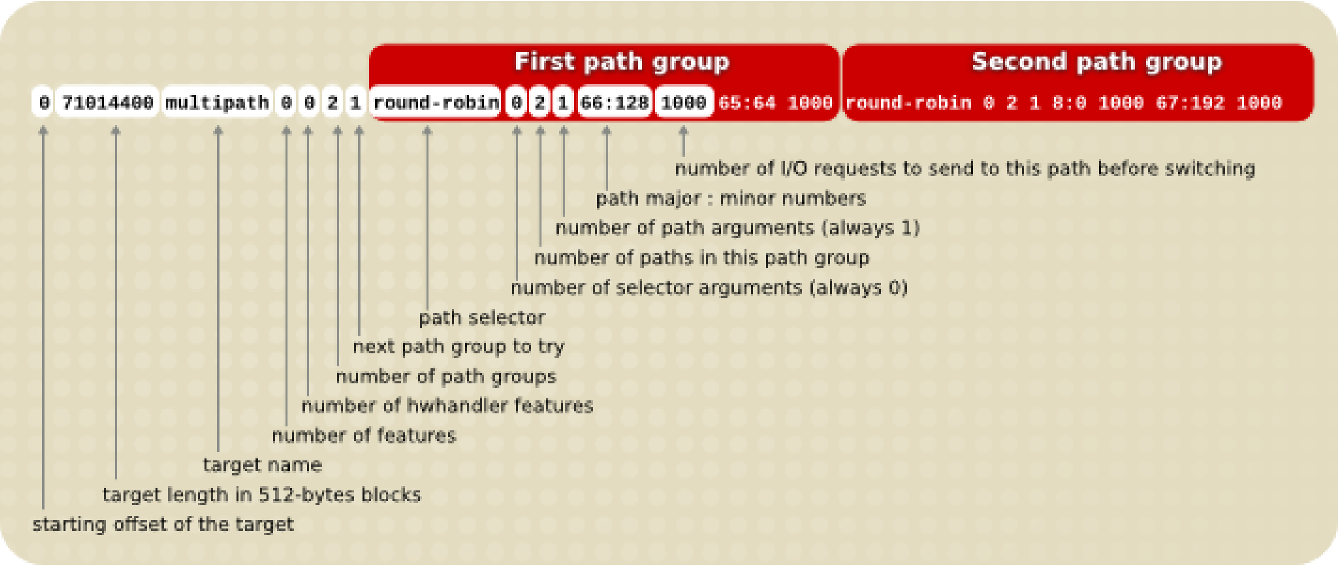
First path group
Second path group
0 71014400 multipath 0 0 2 1 round-robin 0 2 1 66:128 1000 65:64 1000 round-robin 0 2 1 8:0 1000 67:192 1000
number of I/O requests to send to this path before switching
path major : minor numbers
number of path arguments (always 1)
number of paths in this path group
number of selector arguments (always 0)
path selector
next path group to try
number of path groups
number of hwhandler features
number of features
target name
target length in 512-bytes blocks
starting offset of the target

Figure A.1. Multipath Mapping Target

The following example shows a pure failover target definition for the same multipath device. In this target there are four path groups, with only one open path per path group so that the multipathed device will use only one path at a time.

```
0 71014400 multipath 0 0 4 1 round-robin 0 1 1 66:112 1000 \
round-robin 0 1 1 67:176 1000 round-robin 0 1 1 68:240 1000 \
round-robin 0 1 1 65:48 1000
```

The following example shows a full spread (multibus) target definition for the same multipathed device. In this target there is only one path group, which includes all of the paths. In this setup, multipath spreads the load evenly out to all of the paths.

```
0 71014400 multipath 0 0 1 1 round-robin 0 4 1 66:112 1000 \
 67:176 1000 68:240 1000 65:48 1000
```

For further information about multipathing, see the *Using Device Mapper Multipath* document.

A.1.8. The crypt Mapping Target

The **`crypt`** target encrypts the data passing through the specified device. It uses the kernel Crypto API.

The format for the **`crypt`** target is as follows:

```
start length crypt cipher key IV-offset device offset
```

`start`
  starting block in virtual device

`length`
  length of this segment

`cipher`
  Cipher consists of *`cipher[-chainmode]-ivmode[:iv options]`*.

  `cipher`
    Ciphers available are listed in **`/proc/crypto`** (for example, **`aes`**).

  `chainmode`
    Always use **`cbc`**. Do not use **`ebc`**; it does not use an initial vector (IV).

  `ivmode[:iv options]`
    IV is an initial vector used to vary the encryption. The IV mode is **`plain`** or **`essiv:hash`**. An *`ivmode`* of **`-plain`** uses the sector number (plus IV offset) as the IV. An *`ivmode`* of **`-essiv`** is an enhancement avoiding a watermark weakness.

`key`
  Encryption key, supplied in hex

`IV-offset`
  Initial Vector (IV) offset

`device`

block device, referenced by the device name in the filesystem or by the major and minor numbers in the format *`major`*:*`minor`*

`offset`

starting offset of the mapping on the device

The following is an example of a **`crypt`** target.

```
0 2097152 crypt aes-plain 0123456789abcdef0123456789abcdef 0 /dev/hda 0
```

A.2. The dmsetup Command

The **`dmsetup`** command is a command line wrapper for communication with the Device Mapper. For general system information about LVM devices, you may find the **`info`**, **`ls`**, **`status`**, and **`deps`** options of the **`dmsetup`** command to be useful, as described in the following subsections.

For information about additional options and capabilities of the **`dmsetup`** command, see the **`dmsetup`**(8) man page.

A.2.1. The dmsetup info Command

The **`dmsetup info`** ***`device`*** command provides summary information about Device Mapper devices. If you do not specify a device name, the output is information about all of the currently configured Device Mapper devices. If you specify a device, then this command yields information for that device only.

The **`dmsetup info`** command provides information in the following categories:

`Name`

The name of the device. An LVM device is expressed as the volume group name and the logical volume name separated by a hyphen. A hyphen in the original name is translated to two hyphens.

`State`

Possible device states are **`SUSPENDED`**, **`ACTIVE`**, and **`READ-ONLY`**. The **`dmsetup suspend`** command sets a device state to **`SUSPENDED`**. When a device is suspended, all I/O operations to that device stop. The **`dmsetup resume`** command restores a device state to **`ACTIVE`**.

`Read Ahead`

The number of data blocks that the system reads ahead for any open file on which read operations are ongoing. By default, the kernel chooses a suitable value automatically. You can change this value with the **`--readahead`** option of the **`dmsetup`** command.

`Tables present`

Possible states for this category are **`LIVE`** and **`INACTIVE`**. An **`INACTIVE`** state indicates that a table has been loaded which will be swapped in when a **`dmsetup resume`** command restores a device state to **`ACTIVE`**, at which point the table's state becomes **`LIVE`**. For information, see the **`dmsetup`** man page.

`Open count`

The open reference count indicates how many times the device is opened. A **`mount`** command opens a device.

`Event number`
: The current number of events received. Issuing a **`dmsetup wait`** ***`n`*** command allows the user to wait for the n'th event, blocking the call until it is received.

`Major, minor`
: Major and minor device number

`Number of targets`
: The number of fragments that make up a device. For example, a linear device spanning 3 disks would have 3 targets. A linear device composed of the beginning and end of a disk, but not the middle would have 2 targets.

`UUID`
: UUID of the device.

The following example shows partial output for the **`dmsetup info`** command.

```
[root@ask-07 ~]# dmsetup info
Name:              testgfsvg-testgfslv1
State:             ACTIVE
Read Ahead:        256
Tables present:    LIVE
Open count:        0
Event number:      0
Major, minor:      253, 2
Number of targets: 2
UUID: LVM-K528WUGQgPadNXYcFrrf9LnPlUMswgkCkpgPIgYzSvigM7SfeWCypddNSWtNzc2N
...
Name:              VolGroup00-LogVol00
State:             ACTIVE
Read Ahead:        256
Tables present:    LIVE
Open count:        1
Event number:      0
Major, minor:      253, 0
Number of targets: 1
UUID: LVM-tOcS1kqFV9drb0X1Vr8sxeYP0tqcrpdegyqj5lZxe45JMGlmvtqLmbLpBcenh2L3
```

A.2.2. The dmsetup ls Command

You can list the device names of mapped devices with the **`dmsetup ls`** command. You can list devices that have at least one target of a specified type with the **`dmsetup ls --target`** ***`target_type`*** command. For other options of the **`dmsetup ls`**, see the **`dmsetup`** man page.

The following example shows the command to list the device names of currently configured mapped devices.

```
[root@ask-07 ~]# dmsetup ls
testgfsvg-testgfslv3    (253, 4)
testgfsvg-testgfslv2    (253, 3)
testgfsvg-testgfslv1    (253, 2)
VolGroup00-LogVol01     (253, 1)
VolGroup00-LogVol00     (253, 0)
```

The following example shows the command to list the devices names of currently configured mirror mappings.

```
[root@grant-01 ~]# dmsetup ls --target mirror
lock_stress-grant--02.1722      (253, 34)
lock_stress-grant--01.1720      (253, 18)
lock_stress-grant--03.1718      (253, 52)
lock_stress-grant--02.1716      (253, 40)
lock_stress-grant--03.1713      (253, 47)
lock_stress-grant--02.1709      (253, 23)
lock_stress-grant--01.1707      (253, 8)
lock_stress-grant--01.1724      (253, 14)
lock_stress-grant--03.1711      (253, 27)
```

LVM configurations that are stacked on multipath or other device mapper devices can be complex to sort out. The **`dmsetup ls`** command provides a **`--tree`** option that displays dependencies between devices as a tree, as in the following example.

```
# dmsetup ls --tree
vgtest-lvmir (253:13)
 ├─vgtest-lvmir_mimage_1 (253:12)
 │  └─mpathep1 (253:8)
 │     └─mpathe (253:5)
 │        ├─ (8:112)
 │        └─ (8:64)
 ├─vgtest-lvmir_mimage_0 (253:11)
 │  └─mpathcp1 (253:3)
 │     └─mpathc (253:2)
 │        ├─ (8:32)
 │        └─ (8:16)
 └─vgtest-lvmir_mlog (253:4)
    └─mpathfp1 (253:10)
       └─mpathf (253:6)
          ├─ (8:128)
          └─ (8:80)
```

A.2.3. The dmsetup status Command

The **`dmsetup status`** ***`device`*** command provides status information for each target in a specified device. If you do not specify a device name, the output is information about all of the currently configured Device Mapper devices. You can list the status only of devices that have at least one target of a specified type with the **`dmsetup status --target`** ***`target_type`*** command.

The following example shows the command to list the status of the targets in all currently configured mapped devices.

```
[root@ask-07 ~]# dmsetup status
testgfsvg-testgfslv3: 0 312352768 linear
testgfsvg-testgfslv2: 0 312352768 linear
testgfsvg-testgfslv1: 0 312352768 linear
testgfsvg-testgfslv1: 312352768 50331648 linear
VolGroup00-LogVol01: 0 4063232 linear
VolGroup00-LogVol00: 0 151912448 linear
```

A.2.4. The dmsetup deps Command

The **`dmsetup deps`** ***`device`*** command provides a list of (major, minor) pairs for devices referenced by the mapping table for the specified device. If you do not specify a device name, the output is information about all of the currently configured Device Mapper devices.

The following example shows the command to list the dependencies of all currently configured mapped devices.

```
[root@ask-07 ~]# dmsetup deps
testgfsvg-testgfslv3: 1 dependencies    : (8, 16)
testgfsvg-testgfslv2: 1 dependencies    : (8, 16)
testgfsvg-testgfslv1: 1 dependencies    : (8, 16)
VolGroup00-LogVol01: 1 dependencies     : (8, 2)
VolGroup00-LogVol00: 1 dependencies     : (8, 2)
```

The following example shows the command to list the dependencies only of the device **`lock_stress-grant--02.1722`**:

```
[root@grant-01 ~]# dmsetup deps lock_stress-grant--02.1722
3 dependencies  : (253, 33) (253, 32) (253, 31)
```

A.3. Device Mapper Support for the udev Device Manager

The primary role of the **udev** device manager is to provide a dynamic way of setting up nodes in the **`/dev`** directory. The creation of these nodes is directed by the application of **udev** rules in userspace. These rules are processed on **udev** events sent from the kernel directly as a result of adding, removing or changing particular devices. This provides a convenient and central mechanism for hotplugging support.

Besides creating the actual nodes, the **udev** device manager is able to create symbolic links which the user can name. This provides users the freedom to choose their own customized naming and directory structure in the**`/dev`** directory, if needed.

Each **udev** event contains basic information about the device being processed, such as its name, the subsystem it belongs to, the device's type, its major and minor number used, and the type of the event. Given that, and having the possibility of accessing all the information found in the **`/sys`** directory that is also accessible within **udev** rules, the users are able to utilize simple filters based on this information and run the rules conditionally based on this information.

The **udev** device manager also provides a centralized way of setting up the nodes' permissions. A user can easily add a customized set of rules to define the permissions for any device specified by any bit of information that is available while processing the event.

It is also possible to add program hooks in **udev** rules directly. The **udev** device manager can call these programs to provide further processing that is needed to handle the event. Also, the program can export environment variables as a result of this processing. Any results given can be used further in the rules as a supplementary source of information.

Any software using the **udev** library is able to receive and process **udev** events with all the information that is available, so the processing is not bound to the **udev** daemon only.

A.3.1. udev Integration with the Device Mapper

In Red Hat Enterprise Linux 6, the Device Mapper provides direct support for **udev** integration. This synchronizes the Device Mapper with all **udev** processing related to Device Mapper devices, including LVM devices. The synchronization is needed since the rule application in the **udev** daemon is a form of parallel processing with the program that is the source of the device's changes (such as **`dmsetup`** and LVM). Without this support, it was a common problem for a user to try to remove a device that was still open and processed by **udev** rules as a result of a previous change event; this was particularly common when there was a very short time between changes for that device.

The Red Hat Enterprise Linux 6 release provides officially supported **udev** rules for Device Mapper devices in general and for LVM as well. *Table A.1, “udev Rules for Device-Mapper Devices”* summarizes these rules, which are installed in **`/lib/udev/rules.d`**.

Table A.1. udev Rules for Device-Mapper Devices

Filename	Description
`10-dm.rules`	Contains basic/general Device Mapper rules and creates the symlinks in **`/dev/mapper`** with a **`/dev/dm-N`** target where N is a number assigned dynamically to a device by the kernel (**`/dev/dm-N`** is a node) NOTE: **`/dev/dm-N`** nodes should *never* be used in scripts to access the device since the N number is assigned dynamically and changes with the sequence of how devices are activated. Therefore, true names in the **`/dev/mapper`** directory should be used. This layout is to support **udev** requirements of how nodes/symlinks should be created.
`11-dm-lvm.rules`	Contains rules applied for LVM devices and creates the symlinks for the volume group's logical volumes. The symlinks are created in the **`/dev/vgname`** directory with a **`/dev/dm-N`** target. NOTE: To be consistent with the standard for naming all future rules for Device Mapper subsystems, udev rules should follow the format **`11-dm-subsystem_name.rules`**. Any **`libdevmapper`** users providing **udev** rules as well should follow this standard.
`13-dm-disk.rules`	Contains rules to be applied for all Device Mapper devices in general and creates symlinks in the **`/dev/disk/by-id`**, **`/dev/disk/by-uuid`** and the **`/dev/disk/by-uuid`** directories.
`95-dm-notify.rules`	Contains the rule to notify the waiting process using **`libdevmapper`** (just like LVM and **dmsetup**). The notification is done after all previous rules are applied, to ensure any **udev** processing is complete. Notified process is then resumed.

You can add additional customized permission rules by means of the **`12-dm-permissions.rules`** file. This file is *not* installed in the **`/lib/udev/rules`** directory; it is found in the **`/usr/share/doc/device-mapper-version`** directory. The **`12-dm-permissions.rules`** file is a template containing hints for how to set the permissions, based on some matching rules given as an example; the file contains examples for some common situations. You can edit this file and place it manually in the **`/etc/udev/rules.d`** directory where it will survive updates, so the settings will remain.

These rules set all basic variables that could be used by any other rules while processing the events.

The following variables are set in 10-dm.rules:

- **`DM_NAME`**: Device Mapper device name
- **`DM_UUID`**: Device Mapper device UUID

- **`DM_SUSPENDED`**: the suspended state of Device Mapper device
- **`DM_UDEV_RULES_VSN`**: **`udev`** rules version (this is primarily for all other rules to check that previously mentioned variables are set directly by official Device Mapper rules)

The following variables are set in **`11-dm-lvm.rules`**:

- **`DM_LV_NAME`**: logical volume name
- **`DM_VG_NAME`**: volume group name
- **`DM_LV_LAYER`**: LVM layer name

All these variables can be used in the **`12-dm-permissions.rules`** file to define a permission for specific Device Mapper devices, as documented in the **`12-dm-permissions.rules`** file.

A.3.2. Commands and Interfaces that Support udev

Table A.2, “dmsetup Commands to Support udev” summarizes the **`dmsetup`** commands that support **`udev`** integration.

Table A.2. dmsetup Commands to Support udev

Command	Description
`dmsetup udevcomplete`	Used to notify that udev has completed processing the rules and unlocks waiting process (called from within **`udev`** rules in **`95-dm-notify.rules`**).
`dmsetup udevcomplete_all`	Used for debugging purposes to manually unlock all waiting processes.
`dmsetup udevcookies`	Used for debugging purposes, to show all existing cookies (system-wide semaphores).
`dmsetup udevcreatecookie`	Used to create a cookie (semaphore) manually. This is useful to run more processes under one synchronization resource.
`dmsetup udevreleasecookie`	Used to wait for all **`udev`** processing related to all processes put under that one synchronization cookie.

The **`dmsetup`** options that support **`udev`** integration are as follows.

`--udevcookie`

Needs to be defined for all dmsetup processes we would like to add into a udev transaction. It is used in conjunction with **`udevcreatecookie`** and **`udevreleasecookie`**:

```
COOKIE=$(dmsetup udevcreatecookie)
  dmsetup command --udevcookie $COOKIE ....
  dmsetup command --udevcookie $COOKIE ....
  ....
  dmsetup command --udevcookie $COOKIE ....
dmsetup udevreleasecookie --udevcookie $COOKIE
```

Besides using the **`--udevcookie`** option, you can just export the variable into an environment of the process:

```
export DM_UDEV_COOKIE=$(dmsetup udevcreatecookie)
  dmsetup command ...
  dmsetup command ...
  ...
  dmsetup command ...
```

`--noudevrules`

Disables udev rules. Nodes/symlinks will be created by **`libdevmapper`** itself (the old way). This option is for debugging purposes, if **`udev`** does not work correctly.

`--noudevsync`

Disables **`udev`** synchronization. This is also for debugging purposes.

For more information on the **`dmsetup`** and its options, see the **`dmsetup`**(8) man page.

The LVM commands support the following options that support **`udev`** integration:

- **`--noudevrules`**: as for the **`dmsetup`** command, disables **`udev`** rules.
- **`--noudevsync`**: as for the **`dmsetup`** command, disables **`udev`** synchronization.

The **`lvm.conf`** file includes the following options that support **`udev`** integration:

- **`udev_rules`**: enables/disables **`udev_rules`** for all LVM2 commands globally.
- **`udev_sync`**: enables/disables **`udev`** synchronization for all LVM commands globally.

For more information on the **`lvm.conf`** file options, see the inline comments in the **`lvm.conf`** file.

Appendix B. The LVM Configuration Files

LVM supports multiple configuration files. At system startup, the **`lvm.conf`** configuration file is loaded from the directory specified by the environment variable **`LVM_SYSTEM_DIR`**, which is set to **`/etc/lvm`** by default.

The **`lvm.conf`** file can specify additional configuration files to load. Settings in later files override settings from earlier ones. To display the settings in use after loading all the configuration files, execute the **`lvm dumpconfig`** command.

For information on loading additional configuration files, see *Section C.2, “Host Tags”*.

B.1. The LVM Configuration Files

The following files are used for LVM configuration:

/etc/lvm/lvm.conf

Central configuration file read by the tools.

etc/lvm/lvm_*`hosttag`*.conf

For each host tag, an extra configuration file is read if it exists: **`lvm_`*`hosttag`*`.conf`**. If that file defines new tags, then further configuration files will be appended to the list of tiles to read in. For information on host tags, see *Section C.2, “Host Tags”*.

In addition to the LVM configuration files, a system running LVM includes the following files that affect LVM system setup:

/etc/lvm/cache/.cache

Device name filter cache file (configurable).

/etc/lvm/backup/

Directory for automatic volume group metadata backups (configurable).

/etc/lvm/archive/

Directory for automatic volume group metadata archives (configurable with regard to directory path and archive history depth).

/var/lock/lvm/

In single-host configuration, lock files to prevent parallel tool runs from corrupting the metadata; in a cluster, cluster-wide DLM is used.

B.2. Sample lvm.conf File

The following is a sample **`lvm.conf`** configuration file. Your configuration file may differ slightly from this one.

```

# This is an example configuration file for the LVM2 system.
# It contains the default settings that would be used if there was no
# /etc/lvm/lvm.conf file.
#
# Refer to 'man lvm.conf' for further information including the file layout.
#
# To put this file in a different directory and override /etc/lvm set
# the environment variable LVM_SYSTEM_DIR before running the tools.
#
# N.B. Take care that each setting only appears once if uncommenting
# example settings in this file.


# This section allows you to configure which block devices should
# be used by the LVM system.
devices {

    # Where do you want your volume groups to appear ?
    dir = "/dev"

    # An array of directories that contain the device nodes you wish
    # to use with LVM2.
    scan = [ "/dev" ]

    # If set, the cache of block device nodes with all associated symlinks
    # will be constructed out of the existing udev database content.
    # This avoids using and opening any inapplicable non-block devices or
    # subdirectories found in the device directory. This setting is applied
    # to udev-managed device directory only, other directories will be scanned
    # fully. LVM2 needs to be compiled with udev support for this setting to
    # take effect. N.B. Any device node or symlink not managed by udev in
    # udev directory will be ignored with this setting on.
    obtain_device_list_from_udev = 1

    # If several entries in the scanned directories correspond to the
    # same block device and the tools need to display a name for device,
    # all the pathnames are matched against each item in the following
    # list of regular expressions in turn and the first match is used.
    # preferred_names = [ ]

    # Try to avoid using undescriptive /dev/dm-N names, if present.
    preferred_names = [ "^/dev/mpath/", "^/dev/mapper/mpath", "^/dev/[hs]d" ]

    # A filter that tells LVM2 to only use a restricted set of devices.
    # The filter consists of an array of regular expressions.  These
    # expressions can be delimited by a character of your choice, and
    # prefixed with either an 'a' (for accept) or 'r' (for reject).
    # The first expression found to match a device name determines if
    # the device will be accepted or rejected (ignored).  Devices that
    # don't match any patterns are accepted.

    # Be careful if there are symbolic links or multiple filesystem
    # entries for the same device as each name is checked separately against
    # the list of patterns.  The effect is that if any name matches any 'a'
    # pattern, the device is accepted; otherwise if any name matches any 'r'
    # pattern it is rejected; otherwise it is accepted.

    # Don't have more than one filter line active at once: only one gets used.

    # Run vgscan after you change this parameter to ensure that
    # the cache file gets regenerated (see below).
    # If it doesn't do what you expect, check the output of 'vgscan -vvvv'.


    # By default we accept every block device:
    filter = [ "a/.*/" ]

    # Exclude the cdrom drive
    # filter = [ "r|/dev/cdrom|" ]

    # When testing I like to work with just loopback devices:
    # filter = [ "a/loop/", "r/.*/" ]

    # Or maybe all loops and ide drives except hdc:
```

```
    # filter =[ "a|loop|", "r|/dev/hdc|", "a|/dev/ide|", "r|.*|" ]

    # Use anchors if you want to be really specific
    # filter = [ "a|^/dev/hda8$|", "r/.*/" ]

    # The results of the filtering are cached on disk to avoid
    # rescanning dud devices (which can take a very long time).
    # By default this cache is stored in the /etc/lvm/cache directory
    # in a file called '.cache'.
    # It is safe to delete the contents: the tools regenerate it.
    # (The old setting 'cache' is still respected if neither of
    # these new ones is present.)
    cache_dir = "/etc/lvm/cache"
    cache_file_prefix = ""

    # You can turn off writing this cache file by setting this to 0.
    write_cache_state = 1

    # Advanced settings.

    # List of pairs of additional acceptable block device types found
    # in /proc/devices with maximum (non-zero) number of partitions.
    # types = [ "fd", 16 ]

    # If sysfs is mounted (2.6 kernels) restrict device scanning to
    # the block devices it believes are valid.
    # 1 enables; 0 disables.
    sysfs_scan = 1

    # By default, LVM2 will ignore devices used as components of
    # software RAID (md) devices by looking for md superblocks.
    # 1 enables; 0 disables.
    md_component_detection = 1

    # By default, if a PV is placed directly upon an md device, LVM2
    # will align its data blocks with the md device's stripe-width.
    # 1 enables; 0 disables.
    md_chunk_alignment = 1

    # Default alignment of the start of a data area in MB.  If set to 0,
    # a value of 64KB will be used.  Set to 1 for 1MiB, 2 for 2MiB, etc.
    # default_data_alignment = 1

    # By default, the start of a PV's data area will be a multiple of
    # the 'minimum_io_size' or 'optimal_io_size' exposed in sysfs.
    # - minimum_io_size - the smallest request the device can perform
    #   w/o incurring a read-modify-write penalty (e.g. MD's chunk size)
    # - optimal_io_size - the device's preferred unit of receiving I/O
    #   (e.g. MD's stripe width)
    # minimum_io_size is used if optimal_io_size is undefined (0).
    # If md_chunk_alignment is enabled, that detects the optimal_io_size.
    # This setting takes precedence over md_chunk_alignment.
    # 1 enables; 0 disables.
    data_alignment_detection = 1

    # Alignment (in KB) of start of data area when creating a new PV.
    # md_chunk_alignment and data_alignment_detection are disabled if set.
    # Set to 0 for the default alignment (see: data_alignment_default)
    # or page size, if larger.
    data_alignment = 0

    # By default, the start of the PV's aligned data area will be shifted by
    # the 'alignment_offset' exposed in sysfs.  This offset is often 0 but
    # may be non-zero; e.g.: certain 4KB sector drives that compensate for
    # windows partitioning will have an alignment_offset of 3584 bytes
    # (sector 7 is the lowest aligned logical block, the 4KB sectors start
    # at LBA -1, and consequently sector 63 is aligned on a 4KB boundary).
    # But note that pvcreate --dataalignmentoffset will skip this detection.
    # 1 enables; 0 disables.
    data_alignment_offset_detection = 1

    # If, while scanning the system for PVs, LVM2 encounters a device-mapper
    # device that has its I/O suspended, it waits for it to become accessible.
    # Set this to 1 to skip such devices.  This should only be needed
    # in recovery situations.
    ignore_suspended_devices = 0

    # During each LVM operation errors received from each device are counted.
    # If the counter of a particular device exceeds the limit set here, no
    # further I/O is sent to that device for the remainder of the respective
    # operation. Setting the parameter to 0 disables the counters altogether.
    disable_after_error_count = 0

    # Allow use of pvcreate --uuid without requiring --restorefile.
    require_restorefile_with_uuid = 1

    # Minimum size (in KB) of block devices which can be used as PVs.
    # In a clustered environment all nodes must use the same value.
    # Any value smaller than 512KB is ignored.

    # Ignore devices smaller than 2MB such as floppy drives.
    pv_min_size = 2048

    # The original built-in setting was 512 up to and including version 2.02.84.
    # pv_min_size = 512

    # Issue discards to a logical volumes's underlying physical volume(s) when
    # the logical volume is no longer using the physical volumes' space (e.g.
    # lvremove, lvreduce, etc).  Discards inform the storage that a region is
    # no longer in use.  Storage that supports discards advertise the protocol
    # specific way discards should be issued by the kernel (TRIM, UNMAP, or
    # WRITE SAME with UNMAP bit set).  Not all storage will support or benefit
    # from discards but SSDs and thinly provisioned LUNs generally do.  If set
    # to 1, discards will only be issued if both the storage and kernel provide
    # support.
    # 1 enables; 0 disables.
    issue_discards = 0
}

# This section allows you to configure the way in which LVM selects
# free space for its Logical Volumes.
#allocation {
#    When searching for free space to extend an LV, the "cling"
#    allocation policy will choose space on the same PVs as the last
#    segment of the existing LV.  If there is insufficient space and a
#    list of tags is defined here, it will check whether any of them are
#    attached to the PVs concerned and then seek to match those PV tags
#    between existing extents and new extents.
#    Use the special tag "@*" as a wildcard to match any PV tag.
#
#    Example: LVs are mirrored between two sites within a single VG.
#    PVs are tagged with either @site1 or @site2 to indicate where
#    they are situated.
#
#    cling_tag_list = [ "@site1", "@site2" ]
#    cling_tag_list = [ "@*" ]
#
#    Changes made in version 2.02.85 extended the reach of the 'cling'
#    policies to detect more situations where data can be grouped
#    onto the same disks.  Set this to 0 to revert to the previous
#    algorithm.
#
#    maximise_cling = 1
#
```

```
#    Set to 1 to guarantee that mirror logs will always be placed on
#    different PVs from the mirror images.  This was the default
#    until version 2.02.85.
#
#    mirror_logs_require_separate_pvs = 0
#}

# This section that allows you to configure the nature of the
# information that LVM2 reports.
log {

    # Controls the messages sent to stdout or stderr.
    # There are three levels of verbosity, 3 being the most verbose.
    verbose = 0

    # Should we send log messages through syslog?
    # 1 is yes; 0 is no.
    syslog = 1

    # Should we log error and debug messages to a file?
    # By default there is no log file.
    #file = "/var/log/lvm2.log"

    # Should we overwrite the log file each time the program is run?
    # By default we append.
    overwrite = 0

    # What level of log messages should we send to the log file and/or syslog?
    # There are 6 syslog-like log levels currently in use - 2 to 7 inclusive.
    # 7 is the most verbose (LOG_DEBUG).
    level = 0

    # Format of output messages
    # Whether or not (1 or 0) to indent messages according to their severity
    indent = 1

    # Whether or not (1 or 0) to display the command name on each line output
    command_names = 0

    # A prefix to use before the message text (but after the command name,
    # if selected).  Default is two spaces, so you can see/grep the severity
    # of each message.
    prefix = "  "

    # To make the messages look similar to the original LVM tools use:
    #   indent = 0
    #   command_names = 1
    #   prefix = " -- "

    # Set this if you want log messages during activation.
    # Don't use this in low memory situations (can deadlock).
    # activation = 0
}

# Configuration of metadata backups and archiving.  In LVM2 when we
# talk about a 'backup' we mean making a copy of the metadata for the
# *current* system.  The 'archive' contains old metadata configurations.
# Backups are stored in a human readeable text format.
backup {

    # Should we maintain a backup of the current metadata configuration ?
    # Use 1 for Yes; 0 for No.
    # Think very hard before turning this off!
    backup = 1

    # Where shall we keep it ?
    # Remember to back up this directory regularly!
    backup_dir = "/etc/lvm/backup"

    # Should we maintain an archive of old metadata configurations.
    # Use 1 for Yes; 0 for No.
    # On by default.  Think very hard before turning this off.
    archive = 1

    # Where should archived files go ?
    # Remember to back up this directory regularly!
    archive_dir = "/etc/lvm/archive"

    # What is the minimum number of archive files you wish to keep ?
    retain_min = 10

    # What is the minimum time you wish to keep an archive file for ?
    retain_days = 30
}

# Settings for the running LVM2 in shell (readline) mode.
shell {

    # Number of lines of history to store in ~/.lvm_history
    history_size = 100
}


# Miscellaneous global LVM2 settings
global {

    # The file creation mask for any files and directories created.
    # Interpreted as octal if the first digit is zero.
    umask = 077

    # Allow other users to read the files
    #umask = 022

    # Enabling test mode means that no changes to the on disk metadata
    # will be made.  Equivalent to having the -t option on every
    # command.  Defaults to off.
    test = 0

    # Default value for --units argument
    units = "h"

    # Since version 2.02.54, the tools distinguish between powers of
    # 1024 bytes (e.g. KiB, MiB, GiB) and powers of 1000 bytes (e.g.
    # KB, MB, GB).
    # If you have scripts that depend on the old behaviour, set this to 0
    # temporarily until you update them.
    si_unit_consistency = 1

    # Whether or not to communicate with the kernel device-mapper.
    # Set to 0 if you want to use the tools to manipulate LVM metadata
    # without activating any logical volumes.
    # If the device-mapper kernel driver is not present in your kernel
    # setting this to 0 should suppress the error messages.
    activation = 1

    # If we can't communicate with device-mapper, should we try running
    # the LVM1 tools?
    # This option only applies to 2.4 kernels and is provided to help you
    # switch between device-mapper kernels and LVM1 kernels.
    # The LVM1 tools need to be installed with .lvm1 suffices
    # e.g. vgscan.lvm1 and they will stop working after you start using
    # the new lvm2 on-disk metadata format.
    # The default value is set when the tools are built.
    # fallback_to_lvm1 = 0
    # The default metadata format that commands should use - "lvm1" or "lvm2".
    # The command line override is -M1 or -M2.
    # Defaults to "lvm2".
    # format = "lvm2"

    # Location of proc filesystem
    proc = "/proc"

    # Type of locking to use. Defaults to local file-based locking (1).
    # Turn locking off by setting to 0 (dangerous: risks metadata corruption
    # if LVM2 commands get run concurrently).
    # Type 2 uses the external shared library locking_library.
    # Type 3 uses built-in clustered locking.
    # Type 4 uses read-only locking which forbids any operations that might
    # change metadata.
    locking_type = 1

    # Set to 0 to fail when a lock request cannot be satisfied immediately.
    wait_for_locks = 1

    # If using external locking (type 2) and initialisation fails,
    # with this set to 1 an attempt will be made to use the built-in
    # clustered locking.
    # If you are using a customised locking_library you should set this to 0.
    fallback_to_clustered_locking = 1

    # If an attempt to initialise type 2 or type 3 locking failed, perhaps
    # because cluster components such as clvmd are not running, with this set
    # to 1 an attempt will be made to use local file-based locking (type 1).
    # If this succeeds, only commands against local volume groups will proceed.
    # Volume Groups marked as clustered will be ignored.
    fallback_to_local_locking = 1

    # Local non-LV directory that holds file-based locks while commands are
    # in progress.  A directory like /tmp that may get wiped on reboot is OK.
    locking_dir = "/var/lock/lvm"

    # Whenever there are competing read-only and read-write access requests for
    # a volume group's metadata, instead of always granting the read-only
    # requests immediately, delay them to allow the read-write requests to be
    # serviced.  Without this setting, write access may be stalled by a high
    # volume of read-only requests.
    # NB. This option only affects locking_type = 1 viz. local file-based
    # locking.
    prioritise_write_locks = 1

    # Other entries can go here to allow you to load shared libraries
    # e.g. if support for LVM1 metadata was compiled as a shared library use
    #   format_libraries = "liblvm2format1.so"
    # Full pathnames can be given.

    # Search this directory first for shared libraries.
    #   library_dir = "/lib"

    # The external locking library to load if locking_type is set to 2.
    #   locking_library = "liblvm2clusterlock.so"

    # Treat any internal errors as fatal errors, aborting the process that
    # encountered the internal error. Please only enable for debugging.
    abort_on_internal_errors = 0

    # Check whether CRC is matching when parsed VG is used multiple times.
    # This is useful to catch unexpected internal cached volume group
    # structure modification. Please only enable for debugging.
    detect_internal_vg_cache_corruption = 0
    # If set to 1, no operations that change on-disk metadata will be permitted.
    # Additionally, read-only commands that encounter metadata in need of repair
    # will still be allowed to proceed exactly as if the repair had been
    # performed (except for the unchanged vg_seqno).
    # Inappropriate use could mess up your system, so seek advice first!
    metadata_read_only = 0

    # 'mirror_segtype_default' defines which segtype will be used when the
    # shorthand '-m' option is used for mirroring.  The possible options are:
    #
    # "mirror" - The original RAID1 implementation provided by LVM2/DM.  It is
    #            characterized by a flexible log solution (core, disk, mirrored)
    #   and by the necessity to block I/O while reconfiguring in the
    #   event of a failure.  Snapshots of this type of RAID1 can be
    #   problematic.
    #
    # "raid1"  - This implementation leverages MD's RAID1 personality through
    #            device-mapper.  It is characterized by a lack of log options.
    #   (A log is always allocated for every device and they are placed
    #   on the same device as the image - no separate devices are
    #   required.)  This mirror implementation does not require I/O
    #   to be blocked in the kernel in the event of a failure.
    #
    # Specify the '--type <mirror|raid1>' option to override this default
    # setting.
    mirror_segtype_default = "mirror"
}

activation {
    # Set to 1 to perform internal checks on the operations issued to
    # libdevmapper.  Useful for debugging problems with activation.
    # Some of the checks may be expensive, so it's best to use this
    # only when there seems to be a problem.
    checks = 0

    # Set to 0 to disable udev synchronisation (if compiled into the binaries).
    # Processes will not wait for notification from udev.
    # They will continue irrespective of any possible udev processing
    # in the background.  You should only use this if udev is not running
    # or has rules that ignore the devices LVM2 creates.
    # The command line argument --nodevsync takes precedence over this setting.
    # If set to 1 when udev is not running, and there are LVM2 processes
    # waiting for udev, run 'dmsetup udevcomplete_all' manually to wake them up.
    udev_sync = 1

    # Set to 0 to disable the udev rules installed by LVM2 (if built with
    # --enable-udev_rules). LVM2 will then manage the /dev nodes and symlinks
    # for active logical volumes directly itself.
    # N.B. Manual intervention may be required if this setting is changed
    # while any logical volumes are active.
    udev_rules = 1

    # Set to 1 for LVM2 to verify operations performed by udev. This turns on
    # additional checks (and if necessary, repairs) on entries in the device
    # directory after udev has completed processing its events.
    # Useful for diagnosing problems with LVM2/udev interactions.
    verify_udev_operations = 0

    # How to fill in missing stripes if activating an incomplete volume.
    # Using "error" will make inaccessible parts of the device return
    # I/O errors on access.  You can instead use a device path, in which
    # case, that device will be used to in place of missing stripes.
    # But note that using anything other than "error" with mirrored
    # or snapshotted volumes is likely to result in data corruption.
    missing_stripe_filler = "error"

    # How much stack (in KB) to reserve for use while devices suspended
    reserved_stack = 256

    # How much memory (in KB) to reserve for use while devices suspended
    reserved_memory = 8192

    # Nice value used while devices suspended
    process_priority = -18

    # If volume_list is defined, each LV is only activated if there is a
    # match against the list.
    #   "vgname" and "vgname/lvname" are matched exactly.
    #   "@tag" matches any tag set in the LV or VG.
    #   "@*" matches if any tag defined on the host is also set in the LV or VG
    #
    # volume_list = [ "vg1", "vg2/lvol1", "@tag1", "@*" ]

    # Size (in KB) of each copy operation when mirroring
    mirror_region_size = 512

    # Setting to use when there is no readahead value stored in the metadata.
    #
    # "none" - Disable readahead.
    # "auto" - Use default value chosen by kernel.
    readahead = "auto"

    # 'mirror_image_fault_policy' and 'mirror_log_fault_policy' define
    # how a device failure affecting a mirror is handled.
    # A mirror is composed of mirror images (copies) and a log.
    # A disk log ensures that a mirror does not need to be re-synced
    # (all copies made the same) every time a machine reboots or crashes.
    #
    # In the event of a failure, the specified policy will be used to determine
    # what happens. This applies to automatic repairs (when the mirror is being
    # monitored by dmeventd) and to manual lvconvert --repair when
    # --use-policies is given.
    #
    # "remove" - Simply remove the faulty device and run without it.  If
    #            the log device fails, the mirror would convert to using
    #            an in-memory log.  This means the mirror will not
    #            remember its sync status across crashes/reboots and
    #            the entire mirror will be re-synced.  If a
    #            mirror image fails, the mirror will convert to a
    #            non-mirrored device if there is only one remaining good
    #            copy.
    #
    # "allocate" - Remove the faulty device and try to allocate space on
    #            a new device to be a replacement for the failed device.
    #            Using this policy for the log is fast and maintains the
    #            ability to remember sync state through crashes/reboots.
    #            Using this policy for a mirror device is slow, as it
    #            requires the mirror to resynchronize the devices, but it
    #            will preserve the mirror characteristic of the device.
    #            This policy acts like "remove" if no suitable device and
    #            space can be allocated for the replacement.
    #
    # "allocate_anywhere" - Not yet implemented. Useful to place the log device
    #            temporarily on same physical volume as one of the mirror
    #            images. This policy is not recommended for mirror devices
    #            since it would break the redundant nature of the mirror. This
    #            policy acts like "remove" if no suitable device and space can
    #            be allocated for the replacement.

    mirror_log_fault_policy = "allocate"
    mirror_image_fault_policy = "remove"

    # 'snapshot_autoextend_threshold' and 'snapshot_autoextend_percent' define
    # how to handle automatic snapshot extension. The former defines when the
    # snapshot should be extended: when its space usage exceeds this many
    # percent. The latter defines how much extra space should be allocated for
    # the snapshot, in percent of its current size.
    #
    # For example, if you set snapshot_autoextend_threshold to 70 and
    # snapshot_autoextend_percent to 20, whenever a snapshot exceeds 70% usage,
    # it will be extended by another 20%. For a 1G snapshot, using up 700M will
    # trigger a resize to 1.2G. When the usage exceeds 840M, the snapshot will
    # be extended to 1.44G, and so on.
    #
    # Setting snapshot_autoextend_threshold to 100 disables automatic
    # extensions. The minimum value is 50 (A setting below 50 will be treated
    # as 50).

    snapshot_autoextend_threshold = 100
    snapshot_autoextend_percent = 20

    # While activating devices, I/O to devices being (re)configured is
    # suspended, and as a precaution against deadlocks, LVM2 needs to pin
    # any memory it is using so it is not paged out.  Groups of pages that
    # are known not to be accessed during activation need not be pinned
    # into memory.  Each string listed in this setting is compared against
    # each line in /proc/self/maps, and the pages corresponding to any
    # lines that match are not pinned.  On some systems locale-archive was
    # found to make up over 80% of the memory used by the process.
    # mlock_filter = [ "locale/locale-archive", "gconv/gconv-modules.cache" ]

    # Set to 1 to revert to the default behaviour prior to version 2.02.62
    # which used mlockall() to pin the whole process's memory while activating
    # devices.
    use_mlockall = 0

    # Monitoring is enabled by default when activating logical volumes.
    # Set to 0 to disable monitoring or use the --ignoremonitoring option.
    monitoring = 1

    # When pvmove or lvconvert must wait for the kernel to finish
    # synchronising or merging data, they check and report progress
    # at intervals of this number of seconds.  The default is 15 seconds.
    # If this is set to 0 and there is only one thing to wait for, there
    # are no progress reports, but the process is awoken immediately the
    # operation is complete.
    polling_interval = 15
}


####################
# Advanced section #
####################

# Metadata settings
#
# metadata {
    # Default number of copies of metadata to hold on each PV.  0, 1 or 2.
    # You might want to override it from the command line with 0
    # when running pvcreate on new PVs which are to be added to large VGs.

    # pvmetadatacopies = 1

    # Default number of copies of metadata to maintain for each VG.
    # If set to a non-zero value, LVM automatically chooses which of
    # the available metadata areas to use to achieve the requested
    # number of copies of the VG metadata.  If you set a value larger
    # than the total number of metadata areas available then
    # metadata is stored in them all.
    # The default value of 0 ("unmanaged") disables this automatic
    # management and allows you to control which metadata areas
    # are used at the individual PV level using 'pvchange
    # --metadataignore y/n'.

    # vgmetadatacopies = 0

    # Approximate default size of on-disk metadata areas in sectors.
    # You should increase this if you have large volume groups or
    # you want to retain a large on-disk history of your metadata changes.

    # pvmetadatasize = 255

    # List of directories holding live copies of text format metadata.
    # These directories must not be on logical volumes!
    # It's possible to use LVM2 with a couple of directories here,
    # preferably on different (non-LV) filesystems, and with no other
    # on-disk metadata (pvmetadatacopies = 0). Or this can be in
    # addition to on-disk metadata areas.
    # The feature was originally added to simplify testing and is not
    # supported under low memory situations - the machine could lock up.
    #
    # Never edit any files in these directories by hand unless you
    # you are absolutely sure you know what you are doing! Use
    # the supplied toolset to make changes (e.g. vgcfgrestore).

    # dirs = [ "/etc/lvm/metadata", "/mnt/disk2/lvm/metadata2" ]
#}

# Event daemon
#
dmeventd {
    # mirror_library is the library used when monitoring a mirror device.
    #
    # "libdevmapper-event-lvm2mirror.so" attempts to recover from
    # failures.  It removes failed devices from a volume group and
    # reconfigures a mirror as necessary. If no mirror library is
    # provided, mirrors are not monitored through dmeventd.

    mirror_library = "libdevmapper-event-lvm2mirror.so"

    # snapshot_library is the library used when monitoring a snapshot device.
    #
    # "libdevmapper-event-lvm2snapshot.so" monitors the filling of
    # snapshots and emits a warning through syslog when the use of
    # the snapshot exceeds 80%. The warning is repeated when 85%, 90% and
    # 95% of the snapshot is filled.

    snapshot_library = "libdevmapper-event-lvm2snapshot.so"

    # Full path of the dmeventd binary.
    #
    # executable = "/sbin/dmeventd"
}
```

Appendix C. LVM Object Tags

An LVM tag is a word that can be used to group LVM2 objects of the same type together. Tags can be attached to objects such as physical volumes, volume groups, and logical volumes. Tags can be attached to hosts in a cluster configuration. Snapshots cannot be tagged.

Tags can be given on the command line in place of PV, VG or LV arguments. Tags should be prefixed with @ to avoid ambiguity. Each tag is expanded by replacing it with all objects possessing that tag which are of the type expected by its position on the command line.

As of the Red Hat Enterprise Linux 6.1 release, LVM tags are strings of up to 1024 characters (for earlier releases the upper length limit was 128 characters). LVM tags cannot start with a hyphen.

A valid tag can consist of a limited range of characters only. For the Red Hat Enterprise Linux 6.0 release, the allowed characters are [A-Za-z0-9_+.-]. As of the Red Hat Enterprise Linux 6.1 release, the list of allowed characters has been extended, and tags can contain the "/", "=", "!", ":", "#", and "&" characters.

Only objects in a volume group can be tagged. Physical volumes lose their tags if they are removed from a volume group; this is because tags are stored as part of the volume group metadata and that is deleted when a physical volume is removed. Snapshots cannot be tagged.

The following command lists all the logical volumes with the **`database`** tag.

```
lvs @database
```

C.1. Adding and Removing Object Tags

To add or delete tags from physical volumes, use the **`--addtag`** or **`--deltag`** option of the **`pvchange`** command.

To add or delete tags from volume groups, use the **`--addtag`** or **`--deltag`** option of the **`vgchange`** or **`vgcreate`** commands.

To add or delete tags from logical volumes, use the **`--addtag`** or **`--deltag`** option of the **`lvchange`** or **`lvcreate`** commands.

As of the Red Hat Enterprise Linux 6.1 release, you can specify multiple **`--addtag`** and **`--deltag`** arguments within a single **`pvchange`**, **`vgchange`**, or **`lvchange`** command. For example, the following command deletes the tags **`T9`** and **`T10`** and adds the tags **`T13`** and **`T14`** to the volume group **`grant`**.

```
vgchange --deltag T9 --deltag T10 --addtag T13 --addtag T14 grant
```

C.2. Host Tags

In a cluster configuration, you can define host tags in the configuration files. If you set **`hosttags = 1`** in the **`tags`** section, a host tag is automatically defined using the machine's hostname. This allow you to use a common configuration file which can be replicated on all your machines so they hold identical copies of the file, but the behavior can differ between machines according to the hostname.

For information on the configuration files, see *Appendix B, The LVM Configuration Files*.

For each host tag, an extra configuration file is read if it exists: lvm_*`hosttag`*`.`conf. If that file defines new tags, then further configuration files will be appended to the list of files to read in.

For example, the following entry in the configuration file always defines **`tag1`**, and defines **`tag2`** if the hostname is **`host1`**.

```
tags { tag1 { }  tag2 { host_list = ["host1"] } }
```

C.3. Controlling Activation with Tags

You can specify in the configuration file that only certain logical volumes should be activated on that host. For example, the following entry acts as a filter for activation requests (such as **`vgchange -ay`**) and only activates **`vg1/lvol0`** and any logical volumes or volume groups with the **`database`** tag in the metadata on that host.

```
activation { volume_list = ["vg1/lvol0", "@database" ] }
```

There is a special match "@*" that causes a match only if any metadata tag matches any host tag on that machine.

As another example, consider a situation where every machine in the cluster has the following entry in the configuration file:

```
tags { hosttags = 1 }
```

If you want to activate **`vg1/lvol2`** only on host **`db2`**, do the following:

1. Run **`lvchange --addtag @db2 vg1/lvol2`** from any host in the cluster.
2. Run **`lvchange -ay vg1/lvol2`**.

This solution involves storing hostnames inside the volume group metadata.

Appendix D. LVM Volume Group Metadata

The configuration details of a volume group are referred to as the metadata. By default, an identical copy of the metadata is maintained in every metadata area in every physical volume within the volume group. LVM volume group metadata is stored as ASCII.

If a volume group contains many physical volumes, having many redundant copies of the metadata is inefficient. It is possible to create a physical volume without any metadata copies by using the **`--metadatacopies 0`** option of the **`pvcreate`** command. Once you have selected the number of metadata copies the physical volume will contain, you cannot change that at a later point. Selecting 0 copies can result in faster updates on configuration changes. Note, however, that at all times every volume group must contain at least one physical volume with a metadata area (unless you are using the advanced configuration settings that allow you to store volume group metadata in a file system). If you intend to split the volume group in the future, every volume group needs at least one metadata copy.

The core metadata is stored in ASCII. A metadata area is a circular buffer. New metadata is appended to the old metadata and then the pointer to the start of it is updated.

You can specify the size of metadata area with the **`--metadatasize`**. option of the **`pvcreate`** command. The default size may be too small for volume groups that contain physical volumes and logical volumes that number in the hundreds.

D.1. The Physical Volume Label

By default, the **`pvcreate`** command places the physical volume label in the 2nd 512-byte sector. This label can optionally be placed in any of the first four sectors, since the LVM tools that scan for a physical volume label check the first 4 sectors. The physical volume label begins with the string **`LABELONE`**.

The physical volume label Contains:

- Physical volume UUID
- Size of block device in bytes
- NULL-terminated list of data area locations
- NULL-terminated lists of metadata area locations

Metadata locations are stored as offset and size (in bytes). There is room in the label for about 15 locations, but the LVM tools currently use 3: a single data area plus up to two metadata areas.

D.2. Metadata Contents

The volume group metadata contains:

- Information about how and when it was created
- Information about the volume group:

The volume group information contains:

- Name and unique id

- A version number which is incremented whenever the metadata gets updated
- Any properties: Read/Write? Resizeable?
- Any administrative limit on the number of physical volumes and logical volumes it may contain
- The extent size (in units of sectors which are defined as 512 bytes)
- An unordered list of physical volumes making up the volume group, each with:
  - Its UUID, used to determine the block device containing it
  - Any properties, such as whether the physical volume is allocatable
  - The offset to the start of the first extent within the physical volume (in sectors)
  - The number of extents
- An unordered list of logical volumes. each consisting of
  - An ordered list of logical volume segments. For each segment the metadata includes a mapping applied to an ordered list of physical volume segments or logical volume segments

D.3. Sample Metadata

The following shows an example of LVM volume group metadata for a volume group called **`myvg`**.

```
# Generated by LVM2: Tue Jan 30 16:28:15 2007

contents = "Text Format Volume Group"
version = 1

description = "Created *before* executing 'lvextend -L+5G /dev/myvg/mylv /dev/sdc'"

creation_host = "tng3-1"        # Linux tng3-1 2.6.18-8.el5 #1 SMP Fri Jan 26 14:15:21 EST
 2007 i686
creation_time = 1170196095      # Tue Jan 30 16:28:15 2007

myvg {
        id = "0zd3UT-wbYT-lDHq-lMPs-EjoE-0o18-wL28X4"
        seqno = 3
        status = ["RESIZEABLE", "READ", "WRITE"]
        extent_size = 8192              # 4 Megabytes
        max_lv = 0
        max_pv = 0

        physical_volumes {

                pv0 {
                        id = "ZBW5qW-dXF2-0bGw-ZCad-2RlV-phwu-1c1RFt"
                        device = "/dev/sda"     # Hint only

                        status = ["ALLOCATABLE"]
                        dev_size = 35964301     # 17.1491 Gigabytes
                        pe_start = 384
                        pe_count = 4390 # 17.1484 Gigabytes
                }

                pv1 {
                        id = "ZHEZJW-MR64-D3QM-Rv7V-Hxsa-zU24-wztY19"
                        device = "/dev/sdb"     # Hint only
                        status = ["ALLOCATABLE"]
                        dev_size = 35964301     # 17.1491 Gigabytes
                        pe_start = 384
                        pe_count = 4390 # 17.1484 Gigabytes
                }

                pv2 {
                        id = "wCoG4p-55Ui-9tbp-VTEA-jO6s-RAVx-UREW0G"
                        device = "/dev/sdc"     # Hint only

                        status = ["ALLOCATABLE"]
                        dev_size = 35964301     # 17.1491 Gigabytes
                        pe_start = 384
                        pe_count = 4390 # 17.1484 Gigabytes
                }

                pv3 {
                        id = "hGlUwi-zsBg-39FF-do88-pHxY-8XA2-9WKIiA"
                        device = "/dev/sdd"     # Hint only

                        status = ["ALLOCATABLE"]
                        dev_size = 35964301     # 17.1491 Gigabytes
                        pe_start = 384
                        pe_count = 4390 # 17.1484 Gigabytes
                }
        }
        logical_volumes {

                mylv {
                        id = "GhUYSF-qVM3-rzQo-a6D2-o0aV-LQet-Ur9OF9"
                        status = ["READ", "WRITE", "VISIBLE"]
                        segment_count = 2

                        segment1 {
                                start_extent = 0
                                extent_count = 1280     # 5 Gigabytes

                                type = "striped"
                                stripe_count = 1        # linear

                                stripes = [
                                        "pv0", 0
                                ]
                        }
                        segment2 {
                                start_extent = 1280
                                extent_count = 1280     # 5 Gigabytes

                                type = "striped"
                                stripe_count = 1        # linear

                                stripes = [
                                        "pv1", 0
                                ]
                        }
                }
        }
}
```

Appendix E. Revision History

Revision 3.0-5 Thu Dec 1 2011 **Steven Levine** *slevine@redhat.com*

Release for GA of Red Hat Enterprise Linux 6.2

Revision 3.0-4 Mon Nov 21 2011 **Steven Levine** *slevine@redhat.com*

Resolves: #755371, #755373, #755374
Documentation QE review

Revision 3.0-3 Mon Nov 07 2011 **Steven Levine** *slevine@redhat.com*

Resolves: #749487
Clarifies pvcreate example for a whole disk.

Revision 3.0-2 Wed Oct 12 2011 **Steven Levine** *slevine@redhat.com*

Resolves: #744999
Fixes small typographical errors.

Revision 3.0-1 Mon Sep 19 2011 **Steven Levine** *slevine@redhat.com*

Initial revision for Red Hat Enterprise Linux 6.2 Beta release

Resolves: #730788
Documents support for issuing discards to a logical volume's underlying physical volume space when the physical volume space is no longer in use.

Resolves: #728361
Removes outdated reference to deprecated documentation.

Resolves: #714579
Fixes minor typographical errors.

Resolves: #664107
Corrects documented location of **`.cache`** file.

Revision 2.0-1 Thu May 19 2011 **Steven Levine** *slevine@redhat.com*

Initial release for Red Hat Enterprise Linux 6.1

Resolves: #694619
Documents new **`cling`** allocation policy when extending a logical volume.

Resolves: #682649
Adds warning about running multiple mirror creation commands in succession on clustered volumes.

Resolves: #674100
Adds example of output for **`dmsetup ls --tree`** command.

Resolves: #694607

Documents support for including multiple --addtag and --deltag arguments on a single command line.

Resolves: #694604
Documents support for expanded character list in tags.

Resolves: #694611
Documents support for mirrored stripes.

Resolves: #694616
Documents support for snapshots of mirrored volumes.

Resolves: #694618
Documents support for snapshots of exclusively-activated cluster volumes.

Resolves: #682648
Documents that when a mirror leg is re-allocated the mirror log may be moved as well.

Resolves: #661530
Updates example **`cluster.conf`** to one that documents current features.

Resolves: #642400
Adds note about cluster log management being maintained by cluster node with lowest cluster ID.

Resolves: #663462
Removes outdated references to Xen virtual machine monitor.

Revision 1.0-1 Wed Nov 10 2010 **Steven Levine** *`slevine@redhat.com`*

Initial release for Red Hat Enterprise Linux 6

Index

Symbols
/lib/udev/rules.d directory, 93

A
- activating logical volumes
  - individual nodes, 49
- activating volume groups, 30
  - individual nodes, 30
  - local node only, 30
- administrative procedures,
- allocation
  - policy, 26
  - preventing, 25
- archive file, 18, 31

B
- backup
  - file, 18
  - metadata, 18, 31
- backup file, 31
- block device
  - scanning, 23

C
- cache file
  - building, 28
- cluster environment, 4, 17
- CLVM
  - definition, 4
- clvmd daemon, 5
- command line units, 21
- configuration examples,
- creating
  - logical volume, 32
  - logical volume, example, 59
  - LVM volumes in a cluster, 17
  - physical volumes, 22
  - striped logical volume, example, 60
  - volume group, clustered, 26
  - volume groups, 25
- creating LVM volumes
  - overview, 17

D
- data relocation, online, 48
- deactivating volume groups, 30
  - exclusive on one node, 30
  - local node only, 30
- device numbers
  - major, 41
  - minor, 41
  - persistent, 41
- device path names, 21
- device scan filters, 47
- device size, maximum, 26
- device special file directory, 26
- display
  - sorting output, 57
- displaying
  - logical volumes, 43, 54
  - physical volumes, 24, 51
  - volume groups, 27, 53

E
- extent
  - allocation, 26
  - definition, 10, 26

F
- failed devices
  - displaying, 69
- features, new and changed, 1
- feedback
  - contact information for this manual, viii
- file system
  - growing on a logical volume, 18
- filters, 47

G
- growing file system
  - logical volume, 18

H
- help display, 22

I
- initializing
  - partitions, 23
  - physical volumes, 23
- Insufficient Free Extents message, 75

L
- linear logical volume
  - converting to mirrored, 39
  - creation, 32
  - definition, 11
- logging, 19
- logical volume
  - administration, general, 32
  - changing parameters, 42
  - creation, 32
  - creation example, 59
  - definition, 2, 11

  displaying, 43, 49, 54
  exclusive access, 49
  extending, 43
  growing, 43
  linear, 32
  local access, 49
  lvs display arguments, 54
  mirrored, 34
  reducing, 47
  removing, 42
  renaming, 42
  resizing, 42
  shrinking, 47
  snapshot, 39
  striped, 34
lvchange command, 42
lvconvert command, 39
lvcreate command, 32
lvdisplay command, 43
lvextend command, 43
LVM
  architecture overview, 3
  clustered, 4
  components, 4,
  custom report format, 49
  directory structure, 26
  help, 22
  history, 3
  label, 9
  logging, 19
  logical volume administration, 32
  physical volume administration, 22
  physical volume, definition, 9
  volume group, definition, 10
LVM1, 3
LVM2, 3
lvmdiskscan command, 23
lvreduce command, 42, 47
lvremove command, 42
lvrename command, 42
lvs command, 49, 54
  display arguments, 54
lvscan command, 43

M

man page display, 22
metadata
  backup, 18, 31
  recovery, 73
mirrored logical volume
  clustered, 66
  converting to linear, 39
  creation, 34
  definition, 13
  failure policy, 37
  failure recovery, 70
  reconfiguration, 39
mirror_image_fault_policy configuration parameter, 37
mirror_log_fault_policy configuration parameter, 37

O

online data relocation, 48
overview
  features, new and changed, 1

P

partition type, setting, 23
partitions
  multiple, 10
path names, 21
persistent device numbers, 41
physical extent
  preventing allocation, 25
physical volume
  adding to a volume group, 27
  administration, general, 22
  creating, 22
  definition, 9
  display, 51
  displaying, 24, 49
  illustration, 9
  initializing, 23
  layout, 9
  pvs display arguments, 51
  recovery, 74
  removing, 25
  removing from volume group, 29
  removing lost volume, 75
  resizing, 25
pvdisplay command, 24
pvmove command, 48
pvremove command, 25
pvresize command, 25
pvs command, 49
  display arguments, 51
pvscan command, 24

R

removing
  disk from a logical volume, 63
  logical volume, 42
  physical volumes, 25
renaming
  logical volume, 42
  volume group, 31

report format, LVM devices, 49
resizing
  logical volume, 42
  physical volume, 25
rules.d directory, 93

S

scanning
  block devices, 23
scanning devices, filters, 47
snapshot logical volume
  creation, 39
snapshot volume
  definition, 14
striped logical volume
  creation, 34
  creation example, 60
  definition, 12
  extending, 44
  growing, 44

T

troubleshooting,

U

udev device manager, 92
udev rules, 93
units, command line, 21

V

verbose output, 21
vgcfbackup command, 31
vgcfrestore command, 31
vgchange command, 29
vgcreate command, 25, 26
vgdisplay command, 28
vgexport command, 31
vgextend command, 27
vgimport command, 31
vgmerge command, 30
vgmknodes command, 32
vgreduce command, 29
vgrename command, 31
vgs command, 49
  display arguments, 53
vgscan command, 28
vgsplit command, 30
volume group
  activating, 30
  administration, general, 25
  changing parameters, 29
  combining, 30
  creating, 25
  creating in a cluster, 26
  deactivating, 30
  definition, 10
  displaying, 27, 49, 53
  extending, 27
  growing, 27
  merging, 30
  moving between systems, 31
  reducing, 29
  removing, 30
  renaming, 31
  shrinking, 29
  splitting, 30
    example procedure, 61
vgs display arguments, 53