



Red Hat Enterprise Linux 8

Configuring and managing logical volumes

A guide to the configuration and management of LVM logical volumes

Red Hat Enterprise Linux 8 Configuring and managing logical volumes

A guide to the configuration and management of LVM logical volumes

Legal Notice

Copyright © 2019 Red Hat, Inc.

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, OpenShift, 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.

Node.js[®] is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This documentation collection provides instructions on how to manage LVM logical volumes on Red Hat Enterprise Linux 8.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION	5
CHAPTER 1. LOGICAL VOLUMES	6
1.1. LVM ARCHITECTURE OVERVIEW	6
1.2. PHYSICAL VOLUMES	7
1.2.1. LVM physical volume layout	7
1.2.2. Multiple partitions on a disk	8
1.3. VOLUME GROUPS	8
1.4. LVM LOGICAL VOLUMES	9
1.4.1. Linear Volumes	9
1.4.2. Striped Logical Volumes	11
1.4.3. RAID logical volumes	12
1.4.4. Thinly-provisioned logical volumes (thin volumes)	13
1.4.5. Snapshot Volumes	13
1.4.6. Thinly-provisioned snapshot volumes	14
1.4.7. Cache Volumes	15
CHAPTER 2. CONFIGURING LVM LOGICAL VOLUMES	16
2.1. CREATING AN LVM LOGICAL VOLUME ON THREE DISKS	16
2.2. CREATING A RAID0 (STRIPED) LOGICAL VOLUME	17
2.3. SPLITTING A VOLUME GROUP	19
2.4. REMOVING A DISK FROM A LOGICAL VOLUME	20
2.4.1. Moving extents to existing physical volumes	20
2.4.2. Moving Extents to a New Disk	21
2.5. COMBINING VOLUME GROUPS	22
2.6. MOVING A VOLUME GROUP TO ANOTHER SYSTEM	23
2.7. CONFIGURING PERSISTENT DEVICE NUMBERS	23
2.8. SPECIFYING LVM EXTENT SIZE	24
CHAPTER 3. MODIFYING THE SIZE OF A LOGICAL VOLUME	25
3.1. GROWING LOGICAL VOLUMES	25
3.2. GROWING A FILE SYSTEM ON A LOGICAL VOLUME	25
3.3. SHRINKING LOGICAL VOLUMES	26
3.4. EXTENDING A STRIPED LOGICAL VOLUME	27
CHAPTER 4. MANAGING LVM PHYSICAL VOLUMES	29
4.1. SCANNING FOR BLOCK DEVICES TO USE AS PHYSICAL VOLUMES	29
4.2. SETTING THE PARTITION TYPE FOR A PHYSICAL VOLUME	29
4.3. RESIZING AN LVM PHYSICAL VOLUME	30
4.4. REMOVING PHYSICAL VOLUMES	30
4.5. ADDING PHYSICAL VOLUMES TO A VOLUME GROUP	30
4.6. REMOVING PHYSICAL VOLUMES FROM A VOLUME GROUP	30
CHAPTER 5. DISPLAYING LVM COMPONENTS	32
5.1. DISPLAYING LVM INFORMATION WITH THE LVM COMMAND	32
5.2. DISPLAYING PHYSICAL VOLUMES	32
5.3. DISPLAYING VOLUME GROUPS	33
5.4. DISPLAYING LOGICAL VOLUMES	34
CHAPTER 6. CUSTOMIZED REPORTING FOR LVM	35
6.1. CONTROLLING THE FORMAT OF THE LVM DISPLAY	35
6.2. LVM OBJECT DISPLAY FIELDS	36
6.3. SORTING LVM REPORTS	45

6.4. SPECIFYING THE UNITS FOR AN LVM REPORT DISPLAY	45
6.5. DISPLAYING LVM COMMAND OUTPUT IN JSON FORMAT	46
6.6. DISPLAYING THE LVM COMMAND LOG	47
CHAPTER 7. CONFIGURING RAID LOGICAL VOLUMES	49
7.1. CREATING RAID LOGICAL VOLUMES	50
7.2. CREATING A RAID0 (STRIPED) LOGICAL VOLUME	51
7.3. CONTROLLING THE RATE AT WHICH RAID VOLUMES ARE INITIALIZED	53
7.4. CONVERTING A LINEAR DEVICE TO A RAID DEVICE	53
7.5. CONVERTING AN LVM RAID1 LOGICAL VOLUME TO AN LVM LINEAR LOGICAL VOLUME	54
7.6. CONVERTING A MIRRORED LVM DEVICE TO A RAID1 DEVICE	55
7.7. RESIZING A RAID LOGICAL VOLUME	55
7.8. CHANGING THE NUMBER OF IMAGES IN AN EXISTING RAID1 DEVICE	56
7.9. SPLITTING OFF A RAID IMAGE AS A SEPARATE LOGICAL VOLUME	58
7.10. SPLITTING AND MERGING A RAID IMAGE	59
7.11. SETTING A RAID FAULT POLICY	61
7.11.1. The allocate RAID Fault Policy	61
7.11.2. The warn RAID Fault Policy	62
7.12. REPLACING A RAID DEVICE IN A LOGICAL VOLUME	62
7.12.1. Replacing a RAID device that has not failed	62
7.12.2. Replacing a failed RAID device in a logical volume	64
7.13. CHECKING DATA COHERENCY IN A RAID LOGICAL VOLUME (RAID SCRUBBING)	65
7.14. CONVERTING A RAID LEVEL (RAID TAKEOVER)	67
7.15. CHANGING ATTRIBUTES OF A RAID VOLUME (RAID RESHAPE)	67
7.16. CONTROLLING I/O OPERATIONS ON A RAID1 LOGICAL VOLUME	67
7.17. CHANGING THE REGION SIZE ON A RAID LOGICAL VOLUME	67
CHAPTER 8. SNAPSHOT LOGICAL VOLUMES	69
8.1. SNAPSHOT VOLUMES	69
8.2. CREATING SNAPSHOT VOLUMES	70
8.3. MERGING SNAPSHOT VOLUMES	72
CHAPTER 9. CREATING AND MANAGING THINLY-PROVISIONED LOGICAL VOLUMES (THIN VOLUMES)	73
9.1. THINLY-PROVISIONED LOGICAL VOLUMES (THIN VOLUMES)	73
9.2. CREATING THINLY-PROVISIONED LOGICAL VOLUMES	73
9.3. THINLY-PROVISIONED SNAPSHOT VOLUMES	75
9.4. CREATING THINLY-PROVISIONED SNAPSHOT VOLUMES	76
9.5. TRACKING AND DISPLAYING THIN SNAPSHOT VOLUMES THAT HAVE BEEN REMOVED	78
CHAPTER 10. LVM CACHE LOGICAL VOLUMES	82
10.1. CACHE VOLUME TYPES	82
10.2. CREATING AN LVM CACHE LOGICAL VOLUME	82
CHAPTER 11. LOGICAL VOLUME ACTIVATION	84
11.1. CONTROLLING AUTOACTIVATION OF LOGICAL VOLUMES	84
11.2. CONTROLLING LOGICAL VOLUME ACTIVATION	85
11.3. ACTIVATING SHARED LOGICAL VOLUMES	85
11.4. ACTIVATING A LOGICAL VOLUME WITH MISSING DEVICES	86
CHAPTER 12. CONTROLLING LVM DEVICE SCANNING	87
12.1. CONFIGURING FILTERS TO CONTROL DEVICE SCANNING	87
12.2. CONTROLLING WHETHER LVM COMMANDS SCAN LOGICAL VOLUMES	87
CHAPTER 13. CONTROLLING LVM ALLOCATION	89

13.1. LVM ALLOCATION POLICIES	89
13.2. PREVENTING ALLOCATION ON A PHYSICAL VOLUME	90
13.3. EXTENDING A LOGICAL VOLUME WITH THE CLING ALLOCATION POLICY	90

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better. To do so:

- For simple comments on specific passages, make sure you are viewing the documentation in the Multi-page HTML format. Highlight the part of text that you want to comment on. Then, click the **Add Feedback** pop-up that appears below the highlighted text, and follow the displayed instructions.
- For submitting more complex feedback, create a Bugzilla ticket:
 1. Go to the [Bugzilla](#) website.
 2. As the Component, use **Documentation**.
 3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
 4. Click **Submit Bug**.

CHAPTER 1. 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. 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.
- **Resizable 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 and custom named groups.
- **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.
- **Thin volumes**
Logical volumes can be thinly provisioned. This allows you to create logical volumes that are larger than the available extents.
- **Cache volumes**
A cache logical volume uses a small logical volume consisting of fast block devices (such as SSD drives) to improve the performance of a larger and slower logical volume by storing the frequently used blocks on the smaller, faster logical volume.

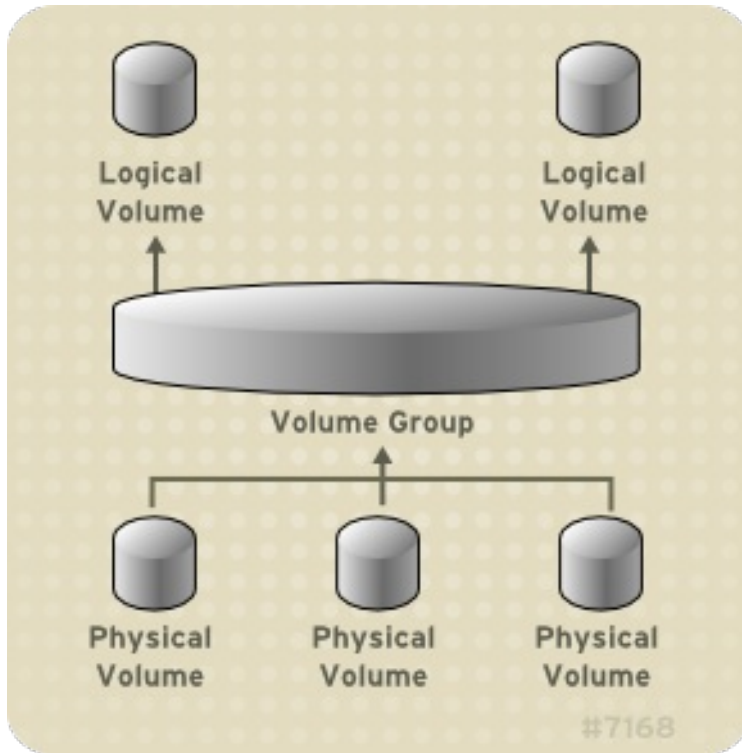
1.1. LVM ARCHITECTURE OVERVIEW

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



1.2. 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 when you create the physical volume. 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.

1.2.1. LVM physical volume layout

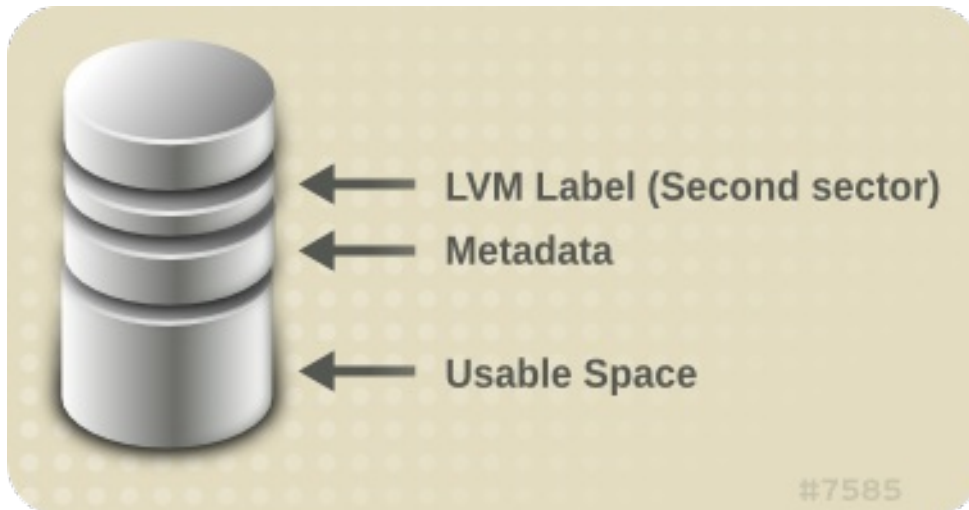
Figure 1.2, “Physical volume layout” shows the layout of an LVM physical volume. The LVM label is on the second sector, followed by the metadata area, followed by the usable space on the device.



NOTE

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

Figure 1.2. Physical volume layout



1.2.2. Multiple partitions on a disk

LVM allows you to create physical volumes out of disk partitions. Red Hat recommends 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.
- **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.

1.3. 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.

1.4. LVM LOGICAL VOLUMES

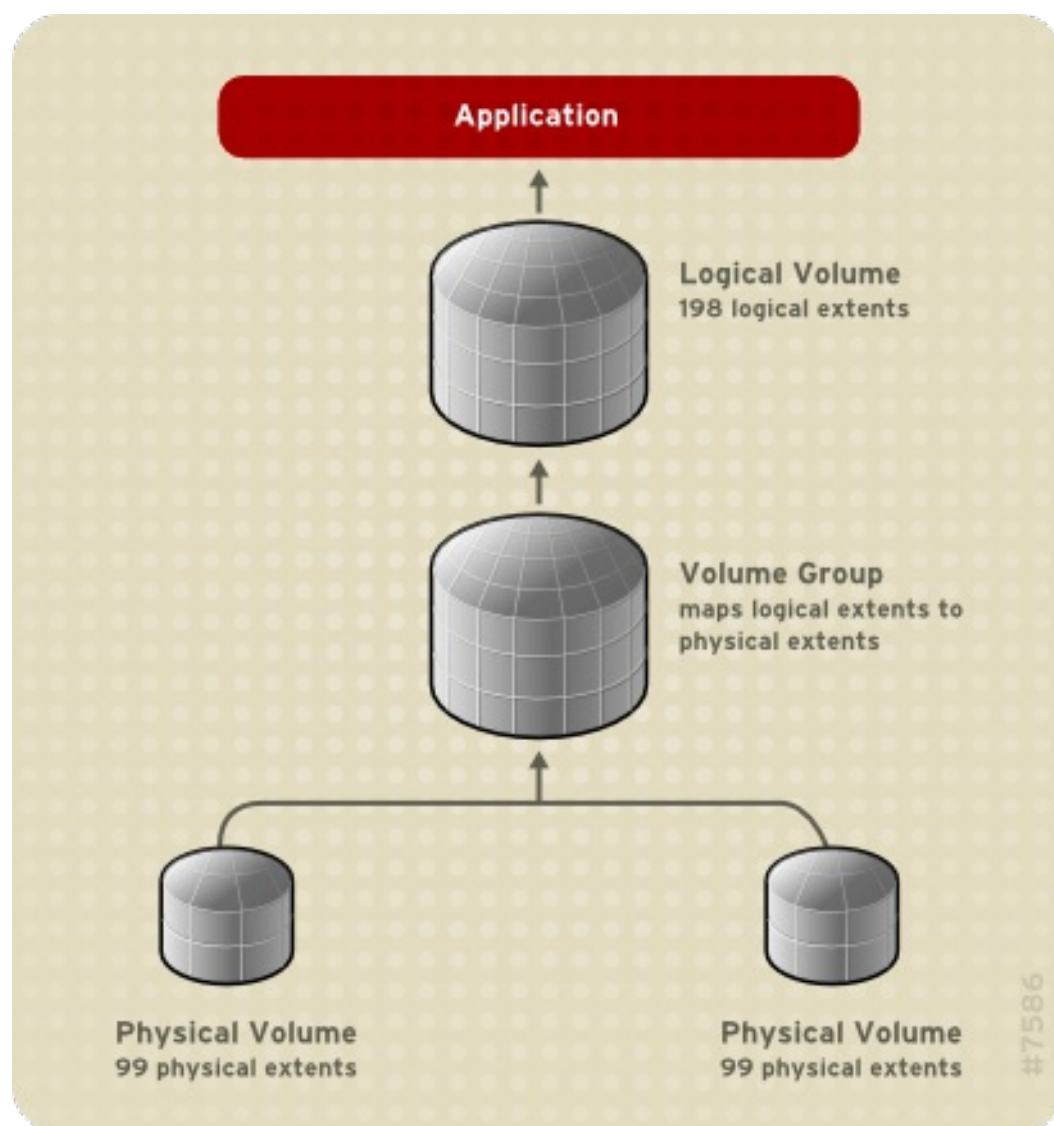
In LVM, a volume group is divided up into logical volumes. The following sections describe the different types of logical volumes.

1.4.1. Linear Volumes

A linear volume aggregates space from one or more 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.

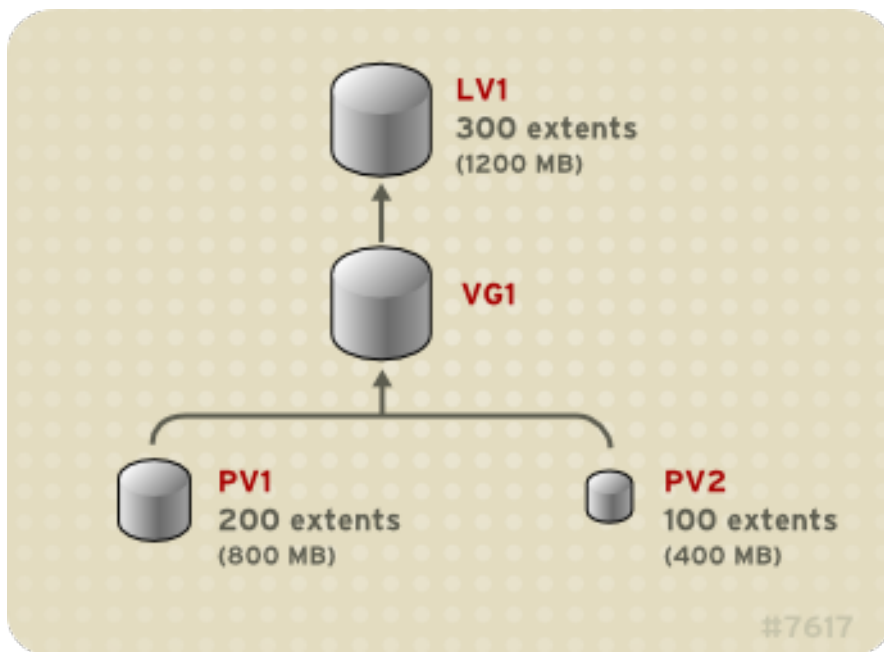
Creating a linear volume assigns a range of physical extents to an area of a logical volume in order. For example, as shown in [Figure 1.3, “Extent Mapping”](#) logical extents 1 to 99 could map to one physical volume and logical extents 100 to 198 could map to a second physical volume. From the point of view of the application, there is one device that is 198 extents in size.

Figure 1.3. Extent Mapping



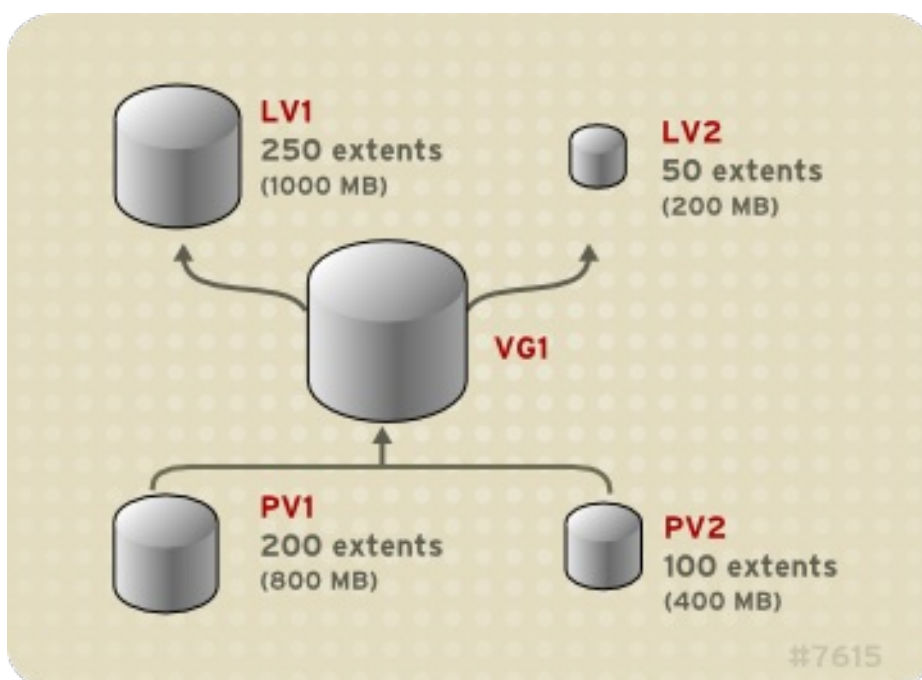
The physical volumes that make up a logical volume do not have to be the same size. [Figure 1.4, “Linear volume with unequal physical volumes”](#) shows volume group **VG1** with a physical extent size of 4MB. This volume group includes 2 physical volumes named **PV1** and **PV2**. The physical volumes are divided into 4MB units, since that is the extent size. In this example, **PV1** is 200 extents in size (800MB) and **PV2** is 100 extents in size (400MB). 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 1.4. 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 1.5, “Multiple logical volumes” shows the same volume group as in Figure 1.4, “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 1.5. Multiple logical volumes



1.4.2. 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.

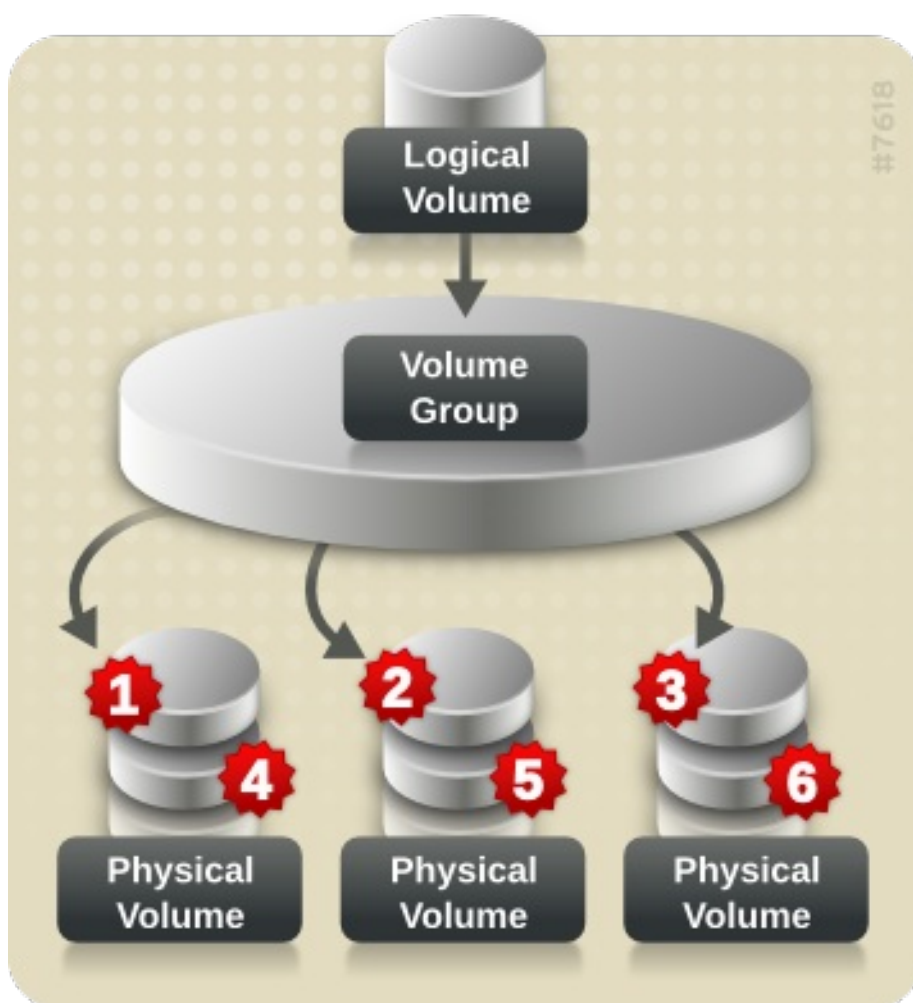
Striping enhances performance by writing data to a predetermined number of physical volumes in round-robin fashion. With striping, I/O can be done in parallel. In some situations, this can result in near-linear performance gain for each additional physical volume in the stripe.

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

- the first stripe of data is written to the first physical volume
- the second stripe of data is written to the second physical volume
- the third stripe of data is written to the third physical volume
- the fourth stripe of data is written to the first physical volume

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

Figure 1.6. 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 to extend a striped logical volume, however, there must be enough free space on the set of 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.

1.4.3. RAID logical volumes

LVM supports RAID0/1/4/5/6/10. An LVM RAID volume has the following characteristics:

- RAID logical volumes created and managed by means of LVM leverage the MD kernel drivers.
- RAID1 images can be temporarily split from the array and merged back into the array later.
- LVM RAID volumes support snapshots.



NOTE

RAID logical volumes are not cluster-aware. While RAID logical volumes can be created and activated exclusively on one machine, they cannot be activated simultaneously on more than one machine.

1.4.4. Thinly-provisioned logical volumes (thin volumes)

Logical volumes can be thinly provisioned. This allows you to create logical volumes that are larger than the available extents. Using thin provisioning, you can manage a storage pool of free space, known as a thin pool, which can be allocated to an arbitrary number of devices when needed by applications. You can then create devices that can be bound to the thin pool for later allocation when an application actually writes to the logical volume. The thin pool can be expanded dynamically when needed for cost-effective allocation of storage space.



NOTE

Thin volumes are not supported across the nodes in a cluster. The thin pool and all its thin volumes must be exclusively activated on only one cluster node.

By using thin provisioning, a storage administrator can overcommit the physical storage, often avoiding the need to purchase additional storage. For example, if ten users each request a 100GB file system for their application, the storage administrator can create what appears to be a 100GB file system for each user but which is backed by less actual storage that is used only when needed. When using thin provisioning, it is important that the storage administrator monitor the storage pool and add more capacity if it starts to become full.

To make sure that all available space can be used, LVM supports data discard. This allows for re-use of the space that was formerly used by a discarded file or other block range.

Thin volumes provide support for a new implementation of copy-on-write (COW) snapshot logical volumes, which allow many virtual devices to share the same data in the thin pool.

1.4.5. Snapshot Volumes

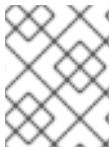
The LVM snapshot feature provides the ability to create virtual images of a device at a particular instant without causing a service interruption. When a change is made to the original device (the origin) after a snapshot is taken, the snapshot feature makes a copy of the changed data area as it was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



NOTE

LVM supports thinly-provisioned snapshots.

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 an 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 resizable, 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.

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.

1.4.6. Thinly-provisioned snapshot volumes

Red Hat Enterprise Linux provides support for thinly-provisioned snapshot volumes. Thin snapshot volumes allow many virtual devices to be stored on the same data volume. This simplifies administration and allows for the sharing of data between snapshot volumes.

As for all LVM snapshot volumes, as well as all thin volumes, thin snapshot volumes are not supported across the nodes in a cluster. The snapshot volume must be exclusively activated on only one cluster node.

Thin snapshot volumes provide the following benefits:

- A thin snapshot volume can reduce disk usage when there are multiple snapshots of the same origin volume.

- If there are multiple snapshots of the same origin, then a write to the origin will cause one COW operation to preserve the data. Increasing the number of snapshots of the origin should yield no major slowdown.
- Thin snapshot volumes can be used as a logical volume origin for another snapshot. This allows for an arbitrary depth of recursive snapshots (snapshots of snapshots of snapshots...).
- A snapshot of a thin logical volume also creates a thin logical volume. This consumes no data space until a COW operation is required, or until the snapshot itself is written.
- A thin snapshot volume does not need to be activated with its origin, so a user may have only the origin active while there are many inactive snapshot volumes of the origin.
- When you delete the origin of a thinly-provisioned snapshot volume, each snapshot of that origin volume becomes an independent thinly-provisioned volume. This means that instead of merging a snapshot with its origin volume, you may choose to delete the origin volume and then create a new thinly-provisioned snapshot using that independent volume as the origin volume for the new snapshot.

Although there are many advantages to using thin snapshot volumes, there are some use cases for which the older LVM snapshot volume feature may be more appropriate to your needs:

- You cannot change the chunk size of a thin pool. If the thin pool has a large chunk size (for example, 1MB) and you require a short-living snapshot for which a chunk size that large is not efficient, you may elect to use the older snapshot feature.
- You cannot limit the size of a thin snapshot volume; the snapshot will use all of the space in the thin pool, if necessary. This may not be appropriate for your needs.

In general, you should consider the specific requirements of your site when deciding which snapshot format to use.

1.4.7. Cache Volumes

LVM supports the use of fast block devices (such as SSD drives) as write-back or write-through caches for larger slower block devices. Users can create cache logical volumes to improve the performance of their existing logical volumes or create new cache logical volumes composed of a small and fast device coupled with a large and slow device.

CHAPTER 2. CONFIGURING LVM LOGICAL VOLUMES

The following procedures provide examples of basic LVM administration tasks.

2.1. CREATING AN LVM LOGICAL VOLUME ON THREE DISKS

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

1. To use disks in a volume group, label them as LVM physical volumes with the **pvcreate** command.



WARNING

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

```
# 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
```

2. Create the a volume group that consists of the LVM physical volumes you have created. The following command creates the volume group **myvg**.

```
# vgcreate myvg /dev/sda1 /dev/sdb1 /dev/sdc1
Volume group "myvg" successfully created
```

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

```
# vgs
VG  #PV #LV #SN Attr  VSize VFree
myvg 3  0  0 wz--n- 51.45G 51.45G
```

3. Create the logical volume from the volume group you have created. The following command creates the logical volume **mylv** from the volume group **myvg**. This example creates a logical volume that uses 2 gigabytes of the volume group.

```
# lvcreate -L 2G -n mylv myvg
Logical volume "mylv" created
```

4. Create a file system on the logical volume. The following command creates an **ext4** file system on the logical volume.

```
# mkfs.ext4 /dev/myvg/mylv
mke2fs 1.44.3 (10-July-2018)
Creating filesystem with 524288 4k blocks and 131072 inodes
Filesystem UUID: 616da032-8a48-4cd7-8705-bd94b7a1c8c4
Superblock backups stored on blocks:
```

```
32768, 98304, 163840, 229376, 294912
```

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

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

```
# mount /dev/myvg/mylv /mnt
# df
Filesystem      1K-blocks  Used Available Use% Mounted on
/dev/mapper/myvg-mylv 1998672  6144  1871288  1% /mnt
```

2.2. CREATING A RAID0 (STRIPED) LOGICAL VOLUME

A RAID0 logical volume spreads logical volume data across multiple data subvolumes in units of stripe size.

The format for the command to create a RAID0 volume is as follows.

```
lvcreate --type raid0[_meta] --stripes Stripes --stripesize StripeSize VolumeGroup
[PhysicalVolumePath ...]
```

Table 2.1. RAID0 Command Creation parameters

Parameter	Description
--type raid0[_meta]	Specifying raid0 creates a RAID0 volume without metadata volumes. Specifying raid0_meta creates a RAID0 volume with metadata volumes. Because RAID0 is non-resilient, it does not have to store any mirrored data blocks as RAID1/10 or calculate and store any parity blocks as RAID4/5/6 do. Hence, it does not need metadata volumes to keep state about resynchronization progress of mirrored or parity blocks. Metadata volumes become mandatory on a conversion from RAID0 to RAID4/5/6/10, however, and specifying raid0_meta preallocates those metadata volumes to prevent a respective allocation failure.
--stripes <i>Stripes</i>	Specifies the number of devices to spread the logical volume across.
--stripesize <i>StripeSize</i>	Specifies the size of each stripe in kilobytes. This is the amount of data that is written to one device before moving to the next device.
<i>VolumeGroup</i>	Specifies the volume group to use.

Parameter	Description
<i>PhysicalVolumePath ...</i>	Specifies the devices to use. If this is not specified, LVM will choose the number of devices specified by the <i>Stripes</i> option, one for each stripe.

This example procedure creates an LVM RAID0 logical volume called **mylv** that stripes data across the disks at **/dev/sda1**, **/dev/sdb1**, and **/dev/sdc1**.

1. Label the disks you will use in the volume group as LVM physical volumes with the **pvcreate** command.



WARNING

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

```
# 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
```

2. Create the volume group **myvg**. The following command creates the volume group **myvg**.

```
# vgcreate myvg /dev/sda1 /dev/sdb1 /dev/sdc1
Volume group "myvg" successfully created
```

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

```
# vgs
VG  #PV #LV #SN Attr  VSize VFree
myvg 3  0  0 wz--n- 51.45G 51.45G
```

3. Create a RAID0 logical volume from the volume group you have created. The following command creates the RAID0 volume **mylv** from the volume group **myvg**. This example creates a logical volume that is 2 gigabytes in size, with three stripes and a stripe size of 4 kilobytes.

```
# lvcreate --type raid0 -L 2G --stripes 3 --stripesize 4 -n mylv myvg
Rounding size 2.00 GiB (512 extents) up to stripe boundary size 2.00 GiB(513 extents).
Logical volume "mylv" created.
```

4. Create a file system on the RAID0 logical volume. The following command creates an **ext4** file system on the logical volume.

```
# mkfs.ext4 /dev/myvg/mylv
```

```
mke2fs 1.44.3 (10-July-2018)
Creating filesystem with 525312 4k blocks and 131376 inodes
Filesystem UUID: 9d4c0704-6028-450a-8b0a-8875358c0511
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912

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

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

```
# mount /dev/myvg/mylv /mnt
# df
Filesystem          1K-blocks    Used Available Use% Mounted on
/dev/mapper/myvg-mylv 2002684    6168  1875072   1% /mnt
```

2.3. SPLITTING A VOLUME GROUP

In this example procedure, 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 **myvg**, 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**.

1. Use the **pvscan** command to determine how much free space is currently available in the volume group.

```
# 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 ]
```

2. 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.

In a cluster, the **pvmove** command can move only logical volume that are active exclusively on a single node.

```
# 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.

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 ]
```

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

The following command splits the volume group **yourvg** from the volume group **myvg**, moving the physical volume **/dev/sdc1** into the new volume group **yourvg**.

lvchange -a n /dev/myvg/mylv**# 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.

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
```

4. After creating the new volume group, create the new logical volume **yourlv**.

lvcreate -L 5G -n yourlv yourvg

```
Logical volume "yourlv" created
```

5. Create a file system on the new logical volume and mount it.

mkfs.ext4 /dev/yourvg/yourlv

```
mke2fs 1.44.3 (10-July-2018)
Creating filesystem with 524288 4k blocks and 131072 inodes
Filesystem UUID: 616da032-8a48-4cd7-8705-bd94b7a1c8c4
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912
```

```
Allocating group tables: done
```

```
Writing inode tables: done
```

```
Creating journal (16384 blocks): done
```

```
Writing superblocks and filesystem accounting information: done
```

mount /dev/yourvg/yourlv /mnt

2.4. REMOVING A DISK FROM A LOGICAL VOLUME

These example procedures show 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.

2.4.1. Moving extents to existing physical volumes

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

■


```
# 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
```

This examples moves the extents off of **/dev/sdb1** so that it can be removed from the volume group.

1. 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.
In a cluster, the **pvmove** command can move only logical volume that are active exclusively on a single node.

```
# 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:

```
# 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
```

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

```
# vgreduce myvg /dev/sdb1
Removed "/dev/sdb1" from volume group "myvg"
# 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.

2.4.2. 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:

```
# 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
```

This example procedure moves the extents of **/dev/sdb1** to a new device, **/dev/sdd1**.

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

```
# pvcreate /dev/sdd1
Physical volume "/dev/sdd1" successfully created
```

2. Add the new physical volume **/dev/sdd1** to the existing volume group **myvg**.

```
# vgextend myvg /dev/sdd1
Volume group "myvg" successfully extended
# 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
```

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

```
# pvmove /dev/sdb1 /dev/sdd1
/dev/sdb1: Moved: 10.0%
...
/dev/sdb1: Moved: 79.7%
...
/dev/sdb1: Moved: 100.0%

# 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
```

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

```
# 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.

2.5. 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
```

2.6. 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.



NOTE

You can use the **--force** argument of the **vgimport** command. This allows you to import volume groups that are missing physical volumes and subsequently run the **vgreduce --removemissing** command.

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 from 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.

```
# 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.

2.7. CONFIGURING PERSISTENT DEVICE NUMBERS

Major and minor device numbers are allocated dynamically at module load. Some applications work best if the block device is always 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.

2.8. SPECIFYING LVM EXTENT SIZE

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.

CHAPTER 3. MODIFYING THE SIZE OF A LOGICAL VOLUME

After you have created a logical volume, you can modify the size of the volume.

3.1. 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**.

```
# 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.

3.2. GROWING A FILE SYSTEM ON A LOGICAL VOLUME

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

1. Determine whether there is sufficient unallocated space in the existing volume group to extend the logical volume. If not, perform the following procedure:
 - a. Create a new physical volume with the **pvcreeate** command.
 - b. Use the **vgextend** command to extend the volume group that contains the logical volume with the file system you are growing to include the new physical volume.

2. Once the volume group is large enough to include the larger file system, extend the logical volume with the **lvresize** command.
3. Resize the file system on the logical volume.

Note that you can use the **-r** option of the **lvresize** command to extend the logical volume and resize the underlying file system with a single command

3.3. SHRINKING LOGICAL VOLUMES

You can reduce the size of a logical volume with the **lvreduce** command.



NOTE

Shrinking is not supported on a GFS2 or XFS file system, so you cannot reduce the size of a logical volume that contains a GFS2 or XFS file system.

If the logical volume you are reducing contains a file system, to prevent data loss you must ensure that the file system is not using the space in the logical volume that is being reduced. For this reason, it is recommended that you use the **--resizefs** option of the **lvreduce** command when the logical volume contains a file system. When you use this option, the **lvreduce** command attempts to reduce the file system before shrinking the logical volume. If shrinking the file system fails, as can occur if the file system is full or the file system does not support shrinking, then the **lvreduce** command will fail and not attempt to shrink the logical volume.



WARNING

In most cases, the **lvreduce** command warns about possible data loss and asks for a confirmation. However, you should not rely on these confirmation prompts to prevent data loss because in some cases you will not see these prompts, such as when the logical volume is inactive or the **--resizefs** option is not used.

Note that using the **--test** option of the **lvreduce** command does not indicate where the operation is safe, as this option does not check the file system or test the file system resize.

The following command shrinks the logical volume **lv01** in volume group **vg00** to be 64 megabytes. In this example, **lv01** contains a file system, which this command resizes together with the logical volume. This example shows the output to the command.

```
# lvreduce --resizefs -L 64M vg00/lv01
fsck from util-linux 2.23.2
/dev/mapper/vg00-lv01: clean, 11/25688 files, 8896/102400 blocks
resize2fs 1.42.9 (28-Dec-2013)
Resizing the filesystem on /dev/mapper/vg00-lv01 to 65536 (1k) blocks.
The filesystem on /dev/mapper/vg00-lv01 is now 65536 blocks long.
```

```
Size of logical volume vg00/lv01 changed from 100.00 MiB (25 extents) to 64.00 MiB (16 extents).
Logical volume vg00/lv01 successfully resized.
```

Specifying the - sign before the resize value indicates that the value will be subtracted from the logical volume's actual size. The following example shows the command you would use if, instead of shrinking a logical volume to an absolute size of 64 megabytes, you wanted to shrink the volume by a value 64 megabytes.

```
# lvreduce --resizefs -L -64M vg00/lvol1
```

3.4. EXTENDING A STRIPED LOGICAL 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 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 135 gigabytes 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
```


CHAPTER 4. MANAGING LVM PHYSICAL VOLUMES

There are a variety of commands and procedures you can use to manage LVM physical volumes.

4.1. SCANNING FOR BLOCK DEVICES TO USE AS PHYSICAL VOLUMES

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. SETTING THE PARTITION TYPE FOR A PHYSICAL VOLUME

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
```

4.3. RESIZING AN LVM 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.

4.4. 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.

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

4.5. 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
```

4.6. 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 **pvdisk** command.

```
# pvdisk /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
```

If a logical volume contains a physical volume that fails, you cannot use that logical volume. To remove missing physical volumes from a volume group, you can use the **--removemissing** parameter of the **vgreduce** command, if there are no logical volumes that are allocated on the missing physical volumes.

If the physical volume that fails contains a mirror image of a logical volume of a **mirror** segment type, you can remove that image from the mirror with the **vgreduce --removemissing --mirroronly --force** command. This removes only the logical volumes that are mirror images from the physical volume.

CHAPTER 5. DISPLAYING LVM COMPONENTS

LVM provides a variety of ways to display the LVM components, as well as to customize the display. This section summarizes the usage of the basic LVM display commands.

5.1. DISPLAYING LVM INFORMATION WITH THE LVM COMMAND

The **lvm** command provides several built-in options that you can use to display information about LVM support and configuration.

- **lvm devtypes**
Displays the recognized built-in block device types
- **lvm formats**
Displays recognized metadata formats.
- **lvm help**
Displays LVM help text.
- **lvm segtypes**
Displays recognized logical volume segment types.
- **lvm tags**
Displays any tags defined on this host.
- **lvm version**
Displays the current version information.

5.2. DISPLAYING PHYSICAL VOLUMES

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

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.

The **pvdisk** command provides a verbose multi-line output for each physical volume. It displays physical properties (size, extents, volume group, and so on) in a fixed format.

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

```
# pvdisk
--- 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-ldwM-01S9-XO8M-mcpsVe
```

The **pvsan** 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** file so that this command will avoid scanning specific physical volumes.

5.3. 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 supported LVM block devices in the system for volume groups, can also be used to display the existing volume groups.

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.

The **vgdisplay** command displays volume group properties (such as size, extents, number of physical volumes, and so on) in a fixed form. The following example shows the output of the **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
```

The following example shows the output of the **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
```

5.4. 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.

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
```

CHAPTER 6. CUSTOMIZED REPORTING FOR LVM

LVM provides a wide range of configuration and command line options to produce customized reports and to filter the report's output. For a full description of LVM reporting features and capabilities, see the **lvmreport(7)** man page.

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.

You can report information about physical volumes, volume groups, logical volumes, physical volume segments, and logical volume segments all at once with the **lvm fullreport** command. For information on this command and its capabilities, see the **lvm-fullreport(8)** man page.

LVM supports log reports, which contain a log of operations, messages, and per-object status with complete object identification collected during LVM command execution. For further information about the LVM log report, see the **lvmreport(7)** man page.

6.1. CONTROLLING THE FORMAT OF THE LVM DISPLAY

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 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-ldwM-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-ldwM-01S9-XO8M-
```

```
mcpsVe
/dev/sdd1 new_vg lvm2 a- 17.14G 17.14G 17.14G yfvZK-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 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
```

6.2. LVM OBJECT DISPLAY FIELDS

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
```



NOTE

The number of characters in the attribute fields in **pvs**, **vgs**, and **lvs** output may increase in later releases. The existing character fields will not change position, but new fields may be added to the end. You should take this into account when writing scripts that search for particular attribute characters, searching for the character based on its relative position to the beginning of the field, but not for its relative position to the end of the field. For example, to search for the character **p** in the ninth bit of the **lv_attr** field, you could search for the string `"^/.....p/"`, but you should not search for the string `"/*p$/"`.

Table 6.1, “The **pvs** Command 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 6.1. The **pvs** Command 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
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

Argument	Header	Description
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-ldwM-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

```

Table 6.2, “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 6.2. 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.

Argument	Header	Description
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
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
```

Table 6.3, “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.



NOTE

In later releases of Red Hat Enterprise Linux, the output of the **lvs** command may differ, with additional fields in the output. The order of the fields, however, will remain the same and any additional fields will appear at the end of the display.

Table 6.3. 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_ancestors	Ancestors	For thin pool snapshots, the ancestors of the logical volume
lv_descendants	Descendants	For thin pool snapshots, the descendants of the logical volume
lv_attr	Attr	<p>The status of the logical volume. The logical volume attribute bits are as follows:</p> <ul style="list-style-type: none"> * Bit 1: Volume type: (m)irrored, (M)irrored without initial sync, (o)rigin, (O)rigin with merging snapshot, (r)aid, ®aid without initial sync, (s)napshot, merging (S)napshot, (p)vmove, (v)irtual, mirror or raid (i)mage, mirror or raid (l)mage out-of-sync, mirror (l)og device, under (c)onversion, thin (V)olume, (t)hin pool, (T)hin pool data, raid or thin pool m(e)tadata or pool metadata spare, * Bit 2: Permissions: (w)riteable, (r)ead-only, ®ead-only activation of non-read-only volume * Bit 3: Allocation policy: (a)nywhere, (c)ontiguous, (i)nherited, c(l)ing, (n)ormal. 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, (l)invalid snapshot, invalid (S)uspended snapshot, snapshot (m)erge failed, suspended snapshot (M)erge failed, mapped (d)evice present without tables, mapped device present with (i)nactive table * Bit 6: device (o)pen * Bit 7: Target type: (m)irror, (r)aid, (s)napshot, (t)hin, (u)nknown, (v)irtual. This groups logical volumes related to the same kernel target together. So, for example, mirror images, mirror logs as well as mirrors themselves appear as (m) if they use the original device-mapper mirror kernel

Argument	Header	Description
		<p>driver, whereas the raid equivalents using the md raid kernel driver appear as (r). Snapshots using the original device-mapper driver appear as (s), whereas snapshots of thin volumes using the thin provisioning driver appear as (t).</p> <p>* Bit 8: Newly-allocated data blocks are overwritten with blocks of zeroes before use.</p> <p>* Bit 9: Volume Health: (p)artial, (r)efresh needed, (m)ismatches exist, (w)ritemostly. (p)artial signifies that one or more of the Physical Volumes this Logical Volume uses is missing from the system. (r)efresh signifies that one or more of the Physical Volumes this RAID Logical Volume uses had suffered a write error. The write error could be due to a temporary failure of that Physical Volume or an indication that it is failing. The device should be refreshed or replaced. (m)ismatches signifies that the RAID logical volume has portions of the array that are not coherent. Inconsistencies are discovered by initiating a check operation on a RAID logical volume. (The scrubbing operations, check and repair, can be performed on a RAID Logical Volume by means of the lvchange command.) (w)ritemostly signifies the devices in a RAID 1 logical volume that have been marked write-mostly.</p> <p>* Bit 10: s(k)ip activation: this volume is flagged to be skipped during activation.</p>
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

Argument	Header	Description
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 provides the following display by default. The default display is sorted by **vg_name** and **lv_name** within the volume group.

```
# lvs
LV   VG      Attr   LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
origin VG      owi-a-s--- 1.00g
snap  VG      swi-a-s--- 100.00m   origin 0.00
```

A common use of the **lvs** command is to append **devices** to the command to display the underlying devices that make up the logical volume. This example also specifies the **-a** option to display the internal volumes that are components of the logical volumes, such as RAID mirrors, enclosed in brackets. This example includes a RAID volume, a striped volume, and a thinly-pooled volume.

```
# lvs -a -o +devices
LV           VG      Attr   LSize   Pool  Origin Data%  Meta%  Move Log Cpy%Sync Convert
Devices
raid1        VG      rwi-a-r--- 1.00g                                100.00
raid1_rimage_0(0),raid1_rimage_1(0)
[raid1_rimage_0] VG      iwi-aor--- 1.00g                                /dev/sde1(7041)
```

```

[raid1_rimage_1] VG      iwi-aor--- 1.00g                /dev/sdf1(7041)
[raid1_rmeta_0] VG      ewi-aor--- 4.00m                /dev/sde1(7040)
[raid1_rmeta_1] VG      ewi-aor--- 4.00m                /dev/sdf1(7040)
stripe1      VG      -wi-a----- 99.95g            /dev/sde1(0),/dev/sdf1(0)
stripe1      VG      -wi-a----- 99.95g            /dev/sdd1(0)
stripe1      VG      -wi-a----- 99.95g            /dev/sdc1(0)
[lvol0_pmspare] rhel_host-083 ewi----- 4.00m                /dev/vda2(0)
pool00       rhel_host-083 twi-aotz-- <4.79g            72.90 54.69
pool00_tdata(0)
[pool00_tdata] rhel_host-083 Twi-ao---- <4.79g                /dev/vda2(1)
[pool00_tmeta] rhel_host-083 ewi-ao---- 4.00m                /dev/vda2(1226)
root         rhel_host-083 Vwi-aotz-- <4.79g pool00      72.90
swap         rhel_host-083 -wi-ao---- 820.00m            /dev/vda2(1227)

```

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-Ojsl-LT03-
nHLC-y8XW-EhCI78
newvgsnap1 new_vg    1 swi-a- 8.00M -1 -1 253 5  lvol0   0.20          1ye1OU-1clu-
o79k-20h2-ZGF0-qCJm-Cfbslx

```

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
```

6.3. 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
```

6.4. SPECIFYING THE UNITS FOR AN LVM REPORT DISPLAY

To specify the units 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 **/etc/lvm/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
```

6.5. DISPLAYING LVM COMMAND OUTPUT IN JSON FORMAT

You can use the **--reportformat** option of the LVM display commands to display the output in JSON format.

The following example shows the output of the **lvs** in standard default format.

```
# lvs
LV      VG      Attr      LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
my_raid my_vg      Rwi-a-r--- 12.00m                               100.00
root    rhel_host-075 -wi-ao---- 6.67g
swap    rhel_host-075 -wi-ao---- 820.00m
```

The following command shows the output of the same LVM configuration when you specify JSON format.

```
# lvs --reportformat json
{
  "report": [
    {
      "lv": [
        {
          "lv_name": "my_raid", "vg_name": "my_vg", "lv_attr": "Rwi-a-r---", "lv_size": "12.00m",
          "pool_lv": "", "origin": "", "data_percent": "", "metadata_percent": "", "move_pv": "", "mirror_log": "",
          "copy_percent": "100.00", "convert_lv": ""
        },
        {
          "lv_name": "root", "vg_name": "rhel_host-075", "lv_attr": "-wi-ao----", "lv_size": "6.67g",
          "pool_lv": "", "origin": "", "data_percent": "", "metadata_percent": "", "move_pv": "", "mirror_log": "",
          "copy_percent": "", "convert_lv": ""
        },
        {
          "lv_name": "swap", "vg_name": "rhel_host-075", "lv_attr": "-wi-ao----", "lv_size": "820.00m",
          "pool_lv": "", "origin": "", "data_percent": "", "metadata_percent": "", "move_pv": "", "mirror_log": "",
          "copy_percent": "", "convert_lv": ""
        }
      ]
    }
  ]
}
```

You can also set the report format as a configuration option in the **/etc/lvm/lvm.conf** file, using the **output_format** setting. The **--reportformat** setting of the command line, however, takes precedence over this setting.

6.6. DISPLAYING THE LVM COMMAND LOG

Both report-oriented and processing-oriented LVM commands can report the command log if this is enabled with the **log/report_command_log** configuration setting. You can determine the set of fields to display and to sort by for this report.

The following examples configures LVM to generate a complete log report for LVM commands. In this example, you can see that both logical volumes **lvol0** and **lvol1** were successfully processed, as was the volume group **VG** that contains the volumes.

```
# lvmconfig --type full log/command_log_selection
command_log_selection="all"

# lvs
Logical Volume
=====
LV      LSize Cpy%Sync
lvol1 4.00m 100.00
```

```
lvof 4.00m
```

Command Log

```
=====
```

Seq	LogType	Context	ObjType	ObjName	ObjGrp	Msg	Errno	RetCode
1	status	processing	lv	lvof	vg	success	0	1
2	status	processing	lv	lvof	vg	success	0	1
3	status	processing	vg	vg		success	0	1

lvchange -an vg/lvof

Command Log

```
=====
```

Seq	LogType	Context	ObjType	ObjName	ObjGrp	Msg	Errno	RetCode
1	status	processing	lv	lvof	vg	success	0	1
2	status	processing	vg	vg		success	0	1

For further information on configuring LVM reports and command logs, see the **lvreport** man page.

CHAPTER 7. CONFIGURING RAID LOGICAL VOLUMES

LVM supports RAID0/1/4/5/6/10.



NOTE

RAID logical volumes are not cluster-aware. While RAID logical volumes can be created and activated exclusively on one machine, they cannot be activated simultaneously on more than one machine.

To create a RAID logical volume, you specify a raid type as the **--type** argument of the **lvcreate** command. [Table 7.1, “RAID Segment Types”](#) describes the possible RAID segment types. After you have created a RAID logical volume with LVM, you can activate, change, remove, display, and use the volume just as you would any other LVM logical volume.

Table 7.1. RAID Segment Types

Segment type	Description
raid1	RAID1 mirroring. This is the default value for the --type argument of the lvcreate command when you specify the -m but you do not specify striping.
raid4	RAID4 dedicated parity disk
raid5	Same as raid5_ls
raid5_la	<ul style="list-style-type: none"> * RAID5 left asymmetric. * Rotating parity 0 with data continuation
raid5_ra	<ul style="list-style-type: none"> * RAID5 right asymmetric. * Rotating parity N with data continuation
raid5_ls	<ul style="list-style-type: none"> * RAID5 left symmetric. * Rotating parity 0 with data restart
raid5_rs	<ul style="list-style-type: none"> * RAID5 right symmetric. * Rotating parity N with data restart
raid6	Same as raid6_zr
raid6_zr	<ul style="list-style-type: none"> * RAID6 zero restart * Rotating parity zero (left-to-right) with data restart
raid6_nr	<ul style="list-style-type: none"> * RAID6 N restart * Rotating parity N (left-to-right) with data restart

Segment type	Description
raid6_nc	<ul style="list-style-type: none"> * RAID6 N continue * Rotating parity N (left-to-right) with data continuation
raid10	<ul style="list-style-type: none"> * Striped mirrors. This is the default value for the --type argument of the lvcreate command if you specify the -m and you specify a number of stripes that is greater than 1. * Striping of mirror sets
raid0/raid0_meta	Striping. RAID0 spreads logical volume data across multiple data subvolumes in units of stripe size. This is used to increase performance. Logical volume data will be lost if any of the data subvolumes fail.

For most users, specifying one of the five available primary types (**raid1**, **raid4**, **raid5**, **raid6**, **raid10**) should be sufficient. For more information on the different algorithms used by RAID 5/6, see chapter four of the *Common RAID Disk Data Format Specification* at http://www.snia.org/sites/default/files/SNIA_DDF_Technical_Position_v2.0.pdf.

When you create a RAID logical volume, LVM creates a metadata subvolume that is one extent in size for every data or parity subvolume in the array. For example, creating a 2-way RAID1 array results in two metadata subvolumes (**lv_rmeta_0** and **lv_rmeta_1**) and two data subvolumes (**lv_rimage_0** and **lv_rimage_1**). Similarly, creating a 3-way stripe (plus 1 implicit parity device) RAID4 results in 4 metadata subvolumes (**lv_rmeta_0**, **lv_rmeta_1**, **lv_rmeta_2**, and **lv_rmeta_3**) and 4 data subvolumes (**lv_rimage_0**, **lv_rimage_1**, **lv_rimage_2**, and **lv_rimage_3**).



NOTE

You can generate commands to create logical volumes on RAID storage with the LVM RAID Calculator application. This application uses the information you input about your current or planned storage to generate these commands. The LVM RAID Calculator application can be found at <https://access.redhat.com/labs/lvmraidcalculator/>.

7.1. CREATING RAID LOGICAL VOLUMES

This section provides example commands that create different types of RAID logical volume.

You can create RAID1 arrays with different numbers of copies according to the value you specify for the **-m** argument. Similarly, you specify the number of stripes for a RAID 4/5/6 logical volume with the **-i** argument. You can also specify the stripe size with the **-l** argument.

The following command creates a 2-way RAID1 array named **my_lv** in the volume group **my_vg** that is one gigabyte in size.

```
# lvcreate --type raid1 -m 1 -L 1G -n my_lv my_vg
```

The following command creates a RAID5 array (3 stripes + 1 implicit parity drive) named **my_lv** in the volume group **my_vg** that is one gigabyte in size. Note that you specify the number of stripes just as you do for an LVM striped volume; the correct number of parity drives is added automatically.

```
# lvcreate --type raid5 -i 3 -L 1G -n my_lv my_vg
```

The following command creates a RAID6 array (3 stripes + 2 implicit parity drives) named **my_lv** in the volume group **my_vg** that is one gigabyte in size.

```
# lvcreate --type raid6 -i 3 -L 1G -n my_lv my_vg
```

7.2. CREATING A RAID0 (STRIPED) LOGICAL VOLUME

A RAID0 logical volume spreads logical volume data across multiple data subvolumes in units of stripe size.

The format for the command to create a RAID0 volume is as follows.

```
lvcreate --type raid0[_meta] --stripes Stripes --stripesize StripeSize VolumeGroup
[PhysicalVolumePath ...]
```

Table 7.2. RAID0 Command Creation parameters

Parameter	Description
--type raid0[_meta]	Specifying raid0 creates a RAID0 volume without metadata volumes. Specifying raid0_meta creates a RAID0 volume with metadata volumes. Because RAID0 is non-resilient, it does not have to store any mirrored data blocks as RAID1/10 or calculate and store any parity blocks as RAID4/5/6 do. Hence, it does not need metadata volumes to keep state about resynchronization progress of mirrored or parity blocks. Metadata volumes become mandatory on a conversion from RAID0 to RAID4/5/6/10, however, and specifying raid0_meta preallocates those metadata volumes to prevent a respective allocation failure.
--stripes <i>Stripes</i>	Specifies the number of devices to spread the logical volume across.
--stripesize <i>StripeSize</i>	Specifies the size of each stripe in kilobytes. This is the amount of data that is written to one device before moving to the next device.
<i>VolumeGroup</i>	Specifies the volume group to use.

Parameter	Description
<i>PhysicalVolumePath ...</i>	Specifies the devices to use. If this is not specified, LVM will choose the number of devices specified by the <i>Stripes</i> option, one for each stripe.

This example procedure creates an LVM RAID0 logical volume called **mylv** that stripes data across the disks at **/dev/sda1**, **/dev/sdb1**, and **/dev/sdc1**.

1. Label the disks you will use in the volume group as LVM physical volumes with the **pvcreate** command.



WARNING

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

```
# 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
```

2. Create the volume group **myvg**. The following command creates the volume group **myvg**.

```
# vgcreate myvg /dev/sda1 /dev/sdb1 /dev/sdc1
Volume group "myvg" successfully created
```

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

```
# vgs
VG #PV #LV #SN Attr VSize VFree
myvg 3 0 0 wz--n- 51.45G 51.45G
```

3. Create a RAID0 logical volume from the volume group you have created. The following command creates the RAID0 volume **mylv** from the volume group **myvg**. This example creates a logical volume that is 2 gigabytes in size, with three stripes and a stripe size of 4 kilobytes.

```
# lvcreate --type raid0 -L 2G --stripes 3 --stripesize 4 -n mylv myvg
Rounding size 2.00 GiB (512 extents) up to stripe boundary size 2.00 GiB(513 extents).
Logical volume "mylv" created.
```

4. Create a file system on the RAID0 logical volume. The following command creates an **ext4** file system on the logical volume.

```
# mkfs.ext4 /dev/myvg/mylv
```



```
mke2fs 1.44.3 (10-July-2018)
Creating filesystem with 525312 4k blocks and 131376 inodes
Filesystem UUID: 9d4c0704-6028-450a-8b0a-8875358c0511
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912
```

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

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

```
# mount /dev/myvg/mylv /mnt
# df
Filesystem          1K-blocks    Used Available Use% Mounted on
/dev/mapper/myvg-mylv 2002684    6168  1875072   1% /mnt
```

7.3. CONTROLLING THE RATE AT WHICH RAID VOLUMES ARE INITIALIZED

When you create RAID10 logical volumes, the background I/O required to initialize the logical volumes with a **sync** operation can crowd out other I/O operations to LVM devices, such as updates to volume group metadata, particularly when you are creating many RAID logical volumes. This can cause the other LVM operations to slow down.

You can control the rate at which a RAID logical volume is initialized by implementing recovery throttling. You control the rate at which **sync** operations are performed by setting the minimum and maximum I/O rate for those operations with the **--minrecoveryrate** and **--maxrecoveryrate** options of the **lvcreate** command. You specify these options as follows.

- **--maxrecoveryrate *Rate*[bBsSkKmMgG]**
Sets the maximum recovery rate for a RAID logical volume so that it will not crowd out nominal I/O operations. The *Rate* is specified as an amount per second for each device in the array. If no suffix is given, then kiB/sec/device is assumed. Setting the recovery rate to 0 means it will be unbounded.
- **--minrecoveryrate *Rate*[bBsSkKmMgG]**
Sets the minimum recovery rate for a RAID logical volume to ensure that I/O for **sync** operations achieves a minimum throughput, even when heavy nominal I/O is present. The *Rate* is specified as an amount per second for each device in the array. If no suffix is given, then kiB/sec/device is assumed.

The following command creates a 2-way RAID10 array with 3 stripes that is 10 gigabytes in size with a maximum recovery rate of 128 kiB/sec/device. The array is named **my_lv** and is in the volume group **my_vg**.

```
# lvcreate --type raid10 -i 2 -m 1 -L 10G --maxrecoveryrate 128 -n my_lv my_vg
```

You can also specify minimum and maximum recovery rates for a RAID scrubbing operation.

7.4. CONVERTING A LINEAR DEVICE TO A RAID DEVICE

You can convert an existing linear logical volume to a RAID device by using the **--type** argument of the **lvconvert** command.

The following command converts the linear logical volume **my_lv** in volume group **my_vg** to a 2-way RAID1 array.

```
# lvconvert --type raid1 -m 1 my_vg/my_lv
```

Since RAID logical volumes are composed of metadata and data subvolume pairs, when you convert a linear device to a RAID1 array, a new metadata subvolume is created and associated with the original logical volume on (one of) the same physical volumes that the linear volume is on. The additional images are added in metadata/data subvolume pairs. For example, if the original device is as follows:

```
# lvs -a -o name,copy_percent,devices my_vg
LV      Copy%  Devices
my_lv    /dev/sde1(0)
```

After conversion to a 2-way RAID1 array the device contains the following data and metadata subvolume pairs:

```
# lvconvert --type raid1 -m 1 my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
LV      Copy%  Devices
my_lv    6.25  my_lv_rimage_0(0),my_lv_rimage_1(0)
[my_lv_rimage_0]  /dev/sde1(0)
[my_lv_rimage_1]  /dev/sdf1(1)
[my_lv_rmeta_0]   /dev/sde1(256)
[my_lv_rmeta_1]   /dev/sdf1(0)
```

If the metadata image that pairs with the original logical volume cannot be placed on the same physical volume, the **lvconvert** will fail.

7.5. CONVERTING AN LVM RAID1 LOGICAL VOLUME TO AN LVM LINEAR LOGICAL VOLUME

You can convert an existing RAID1 LVM logical volume to an LVM linear logical volume with the **lvconvert** command by specifying the **-m0** argument. This removes all the RAID data subvolumes and all the RAID metadata subvolumes that make up the RAID array, leaving the top-level RAID1 image as the linear logical volume.

The following example displays an existing LVM RAID1 logical volume.

```
# lvs -a -o name,copy_percent,devices my_vg
LV      Copy%  Devices
my_lv    100.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
[my_lv_rimage_0]  /dev/sde1(1)
[my_lv_rimage_1]  /dev/sdf1(1)
[my_lv_rmeta_0]   /dev/sde1(0)
[my_lv_rmeta_1]   /dev/sdf1(0)
```

The following command converts the LVM RAID1 logical volume **my_vg/my_lv** to an LVM linear device.

```
# lvconvert -m0 my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
```

```
LV      Copy%  Devices
my_lv    100.00 /dev/sde1(1)
```

When you convert an LVM RAID1 logical volume to an LVM linear volume, you can specify which physical volumes to remove. The following example shows the layout of an LVM RAID1 logical volume made up of two images: **/dev/sda1** and **/dev/sdb1**. In this example, the **lvconvert** command specifies that you want to remove **/dev/sda1**, leaving **/dev/sdb1** as the physical volume that makes up the linear device.

```
# lvs -a -o name,copy_percent,devices my_vg
LV      Copy%  Devices
my_lv    100.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
[my_lv_rimage_0]    /dev/sda1(1)
[my_lv_rimage_1]    /dev/sdb1(1)
[my_lv_rmeta_0]     /dev/sda1(0)
[my_lv_rmeta_1]     /dev/sdb1(0)
# lvconvert -m0 my_vg/my_lv /dev/sda1
# lvs -a -o name,copy_percent,devices my_vg
LV      Copy%  Devices
my_lv    100.00 /dev/sdb1(1)
```

7.6. CONVERTING A MIRRORED LVM DEVICE TO A RAID1 DEVICE

You can convert an existing mirrored LVM device with a segment type of **mirror** to a RAID1 LVM device with the **lvconvert** command by specifying the **--type raid1** argument. This renames the mirror subvolumes (**mimage**) to RAID subvolumes (**rimage**). In addition, the mirror log is removed and metadata subvolumes (**rmeta**) are created for the data subvolumes on the same physical volumes as the corresponding data subvolumes.

The following example shows the layout of a mirrored logical volume **my_vg/my_lv**.

```
# lvs -a -o name,copy_percent,devices my_vg
LV      Copy%  Devices
my_lv    15.20 my_lv_mimage_0(0),my_lv_mimage_1(0)
[my_lv_mimage_0]    /dev/sde1(0)
[my_lv_mimage_1]    /dev/sdf1(0)
[my_lv_mlog]        /dev/sdd1(0)
```

The following command converts the mirrored logical volume **my_vg/my_lv** to a RAID1 logical volume.

```
# lvconvert --type raid1 my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
LV      Copy%  Devices
my_lv    100.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
[my_lv_rimage_0]    /dev/sde1(0)
[my_lv_rimage_1]    /dev/sdf1(0)
[my_lv_rmeta_0]     /dev/sde1(125)
[my_lv_rmeta_1]     /dev/sdf1(125)
```

7.7. RESIZING A RAID LOGICAL VOLUME

You can resize a RAID logical volume in the following ways;

- You can increase the size of a RAID logical volume of any type with the **lvresize** or **lvextend** command. This does not change the number of RAID images. For striped RAID logical volumes the same stripe rounding constraints apply as when you create a striped RAID logical volume.
- You can reduce the size of a RAID logical volume of any type with the **lvresize** or **lvreduce** command. This does not change the number of RAID images. As with the **lvextend** command, the same stripe rounding constraints apply as when you create a striped RAID logical volume.
- You can change the number of stripes on a striped RAID logical volume (**raid4/5/6/10**) with the **-stripes N** parameter of the **lvconvert** command. This increases or reduces the size of the RAID logical volume by the capacity of the stripes added or removed. Note that **raid10** volumes are capable only of adding stripes. This capability is part of the RAID *reshaping* feature that allows you to change attributes of a RAID logical volume while keeping the same RAID level. For information on RAID reshaping and examples of using the **lvconvert** command to reshape a RAID logical volume, see the **lvraid(7)** man page.

7.8. CHANGING THE NUMBER OF IMAGES IN AN EXISTING RAID1 DEVICE

You can change the number of images in an existing RAID1 array just as you can change the number of images in the earlier implementation of LVM mirroring. Use the **lvconvert** command to specify the number of additional metadata/data subvolume pairs to add or remove.

When you add images to a RAID1 device with the **lvconvert** command, you can specify the total number of images for the resulting device, or you can specify how many images to add to the device. You can also optionally specify on which physical volumes the new metadata/data image pairs will reside.

Metadata subvolumes (named **rmeta**) always exist on the same physical devices as their data subvolume counterparts **rimage**). The metadata/data subvolume pairs will not be created on the same physical volumes as those from another metadata/data subvolume pair in the RAID array (unless you specify **--alloc anywhere**).

The format for the command to add images to a RAID1 volume is as follows:

```
lvconvert -m new_absolute_count vg/lv [removable_PVs]
lvconvert -m +num_additional_images vg/lv [removable_PVs]
```

For example, the following command displays the LVM device **my_vg/my_lv**, which is a 2-way RAID1 array:

```
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       6.25   my_lv_rimage_0(0),my_lv_rimage_1(0)
[my_lv_rimage_0]  /dev/sde1(0)
[my_lv_rimage_1]  /dev/sdf1(1)
[my_lv_rmeta_0]   /dev/sde1(256)
[my_lv_rmeta_1]   /dev/sdf1(0)
```

The following command converts the 2-way RAID1 device **my_vg/my_lv** to a 3-way RAID1 device:

```
# lvconvert -m 2 my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       6.25   my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]  /dev/sde1(0)
```

```
[my_lv_rimage_1]    /dev/sdf1(1)
[my_lv_rimage_2]    /dev/sdg1(1)
[my_lv_rmeta_0]     /dev/sde1(256)
[my_lv_rmeta_1]     /dev/sdf1(0)
[my_lv_rmeta_2]     /dev/sdg1(0)
```

When you add an image to a RAID1 array, you can specify which physical volumes to use for the image. The following command converts the 2-way RAID1 device **my_vg/my_lv** to a 3-way RAID1 device, specifying that the physical volume **/dev/sdd1** be used for the array:

```
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       56.00  my_lv_rimage_0(0),my_lv_rimage_1(0)
[my_lv_rimage_0]    /dev/sda1(1)
[my_lv_rimage_1]    /dev/sdb1(1)
[my_lv_rmeta_0]     /dev/sda1(0)
[my_lv_rmeta_1]     /dev/sdb1(0)
# lvconvert -m 2 my_vg/my_lv /dev/sdd1
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       28.00  my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]    /dev/sda1(1)
[my_lv_rimage_1]    /dev/sdb1(1)
[my_lv_rimage_2]    /dev/sdd1(1)
[my_lv_rmeta_0]     /dev/sda1(0)
[my_lv_rmeta_1]     /dev/sdb1(0)
[my_lv_rmeta_2]     /dev/sdd1(0)
```

To remove images from a RAID1 array, use the following command. When you remove images from a RAID1 device with the **lvconvert** command, you can specify the total number of images for the resulting device, or you can specify how many images to remove from the device. You can also optionally specify the physical volumes from which to remove the device.

```
lvconvert -m new_absolute_count vg/lv [removable_PVs]
lvconvert -m -num_fewer_images vg/lv [removable_PVs]
```

Additionally, when an image and its associated metadata subvolume volume are removed, any higher-numbered images will be shifted down to fill the slot. If you remove **lv_rimage_1** from a 3-way RAID1 array that consists of **lv_rimage_0**, **lv_rimage_1**, and **lv_rimage_2**, this results in a RAID1 array that consists of **lv_rimage_0** and **lv_rimage_1**. The subvolume **lv_rimage_2** will be renamed and take over the empty slot, becoming **lv_rimage_1**.

The following example shows the layout of a 3-way RAID1 logical volume **my_vg/my_lv**.

```
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]    /dev/sde1(1)
[my_lv_rimage_1]    /dev/sdf1(1)
[my_lv_rimage_2]    /dev/sdg1(1)
[my_lv_rmeta_0]     /dev/sde1(0)
[my_lv_rmeta_1]     /dev/sdf1(0)
[my_lv_rmeta_2]     /dev/sdg1(0)
```

The following command converts the 3-way RAID1 logical volume into a 2-way RAID1 logical volume.

```
# lvconvert -m1 my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       100.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
[my_lv_rimage_0]  /dev/sde1(1)
[my_lv_rimage_1]  /dev/sdf1(1)
[my_lv_rmeta_0]   /dev/sde1(0)
[my_lv_rmeta_1]   /dev/sdf1(0)
```

The following command converts the 3-way RAID1 logical volume into a 2-way RAID1 logical volume, specifying the physical volume that contains the image to remove as **/dev/sde1**.

```
# lvconvert -m1 my_vg/my_lv /dev/sde1
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       100.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
[my_lv_rimage_0]  /dev/sdf1(1)
[my_lv_rimage_1]  /dev/sdg1(1)
[my_lv_rmeta_0]   /dev/sdf1(0)
[my_lv_rmeta_1]   /dev/sdg1(0)
```

7.9. SPLITTING OFF A RAID IMAGE AS A SEPARATE LOGICAL VOLUME

You can split off an image of a RAID logical volume to form a new logical volume.

The format of the command to split off a RAID image is as follows:

```
lvconvert --splitmirrors count -n splitname vg/lv [removable_PVs]
```

Just as when you are removing a RAID image from an existing RAID1 logical volume, when you remove a RAID data subvolume (and its associated metadata subvolume) from the middle of the device any higher numbered images will be shifted down to fill the slot. The index numbers on the logical volumes that make up a RAID array will thus be an unbroken sequence of integers.



NOTE

You cannot split off a RAID image if the RAID1 array is not yet in sync.

The following example splits a 2-way RAID1 logical volume, **my_lv**, into two linear logical volumes, **my_lv** and **new**.

```
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       12.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
[my_lv_rimage_0]  /dev/sde1(1)
[my_lv_rimage_1]  /dev/sdf1(1)
[my_lv_rmeta_0]   /dev/sde1(0)
[my_lv_rmeta_1]   /dev/sdf1(0)
# lvconvert --splitmirror 1 -n new my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
```

```

LV      Copy%  Devices
my_lv   100.00 /dev/sde1(1)
new     100.00 /dev/sdf1(1)

```

The following example splits a 3-way RAID1 logical volume, **my_lv**, into a 2-way RAID1 logical volume, **my_lv**, and a linear logical volume, **new**

```

# lvs -a -o name,copy_percent,devices my_vg
LV      Copy%  Devices
my_lv   100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]  /dev/sde1(1)
[my_lv_rimage_1]  /dev/sdf1(1)
[my_lv_rimage_2]  /dev/sdg1(1)
[my_lv_rmeta_0]   /dev/sde1(0)
[my_lv_rmeta_1]   /dev/sdf1(0)
[my_lv_rmeta_2]   /dev/sdg1(0)
# lvconvert --splitmirror 1 -n new my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
LV      Copy%  Devices
my_lv   100.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
[my_lv_rimage_0]  /dev/sde1(1)
[my_lv_rimage_1]  /dev/sdf1(1)
[my_lv_rmeta_0]   /dev/sde1(0)
[my_lv_rmeta_1]   /dev/sdf1(0)
new     100.00 /dev/sdg1(1)

```

7.10. SPLITTING AND MERGING A RAID IMAGE

You can temporarily split off an image of a RAID1 array for read-only use while keeping track of any changes by using the **--trackchanges** argument in conjunction with the **--splitmirrors** argument of the **lvconvert** command. This allows you to merge the image back into the array at a later time while resyncing only those portions of the array that have changed since the image was split.

The format for the **lvconvert** command to split off a RAID image is as follows.

```
lvconvert --splitmirrors count --trackchanges vg/lv [removable_PVs]
```

When you split off a RAID image with the **--trackchanges** argument, you can specify which image to split but you cannot change the name of the volume being split. In addition, the resulting volumes have the following constraints.

- The new volume you create is read-only.
- You cannot resize the new volume.
- You cannot rename the remaining array.
- You cannot resize the remaining array.
- You can activate the new volume and the remaining array independently.

You can merge an image that was split off with the **--trackchanges** argument specified by executing a subsequent **lvconvert** command with the **--merge** argument. When you merge the image, only the portions of the array that have changed since the image was split are resynced.

The format for the **lvconvert** command to merge a RAID image is as follows.

```
lvconvert --merge raid_image
```

The following example creates a RAID1 logical volume and then splits off an image from that volume while tracking changes to the remaining array.

```
# lvcreate --type raid1 -m 2 -L 1G -n my_lv my_vg
Logical volume "my_lv" created
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]    /dev/sdb1(1)
[my_lv_rimage_1]    /dev/sdc1(1)
[my_lv_rimage_2]    /dev/sdd1(1)
[my_lv_rmeta_0]     /dev/sdb1(0)
[my_lv_rmeta_1]     /dev/sdc1(0)
[my_lv_rmeta_2]     /dev/sdd1(0)
# lvconvert --splitmirrors 1 --trackchanges my_vg/my_lv
my_lv_rimage_2 split from my_lv for read-only purposes.
Use 'lvconvert --merge my_vg/my_lv_rimage_2' to merge back into my_lv
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]    /dev/sdb1(1)
[my_lv_rimage_1]    /dev/sdc1(1)
my_lv_rimage_2      /dev/sdd1(1)
[my_lv_rmeta_0]     /dev/sdb1(0)
[my_lv_rmeta_1]     /dev/sdc1(0)
[my_lv_rmeta_2]     /dev/sdd1(0)
```

The following example splits off an image from a RAID1 volume while tracking changes to the remaining array, then merges the volume back into the array.

```
# lvconvert --splitmirrors 1 --trackchanges my_vg/my_lv
lv_rimage_1 split from my_lv for read-only purposes.
Use 'lvconvert --merge my_vg/my_lv_rimage_1' to merge back into my_lv
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       100.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
[my_lv_rimage_0]    /dev/sdc1(1)
my_lv_rimage_1      /dev/sdd1(1)
[my_lv_rmeta_0]     /dev/sdc1(0)
[my_lv_rmeta_1]     /dev/sdd1(0)
# lvconvert --merge my_vg/my_lv_rimage_1
my_vg/my_lv_rimage_1 successfully merged back into my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       100.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
[my_lv_rimage_0]    /dev/sdc1(1)
[my_lv_rimage_1]    /dev/sdd1(1)
[my_lv_rmeta_0]     /dev/sdc1(0)
[my_lv_rmeta_1]     /dev/sdd1(0)
```


7.11. SETTING A RAID FAULT POLICY

LVM RAID handles device failures in an automatic fashion based on the preferences defined by the **raid_fault_policy** field in the **lvm.conf** file.

- If the **raid_fault_policy** field is set to **allocate**, the system will attempt to replace the failed device with a spare device from the volume group. If there is no available spare device, this will be reported to the system log.
- If the **raid_fault_policy** field is set to **warn**, the system will produce a warning and the log will indicate that a device has failed. This allows the user to determine the course of action to take.

As long as there are enough devices remaining to support usability, the RAID logical volume will continue to operate.

7.11.1. The allocate RAID Fault Policy

In the following example, the **raid_fault_policy** field has been set to **allocate** in the **lvm.conf** file. The RAID logical volume is laid out as follows.

```
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]  /dev/sde1(1)
[my_lv_rimage_1]  /dev/sdf1(1)
[my_lv_rimage_2]  /dev/sdg1(1)
[my_lv_rmeta_0]   /dev/sde1(0)
[my_lv_rmeta_1]   /dev/sdf1(0)
[my_lv_rmeta_2]   /dev/sdg1(0)
```

If the **/dev/sde** device fails, the system log will display error messages.

```
# grep lvm /var/log/messages
Jan 17 15:57:18 bp-01 lvm[8599]: Device #0 of raid1 array, my_vg-my_lv, has failed.
Jan 17 15:57:18 bp-01 lvm[8599]: /dev/sde1: read failed after 0 of 2048 at
250994294784: Input/output error
Jan 17 15:57:18 bp-01 lvm[8599]: /dev/sde1: read failed after 0 of 2048 at
250994376704: Input/output error
Jan 17 15:57:18 bp-01 lvm[8599]: /dev/sde1: read failed after 0 of 2048 at 0:
Input/output error
Jan 17 15:57:18 bp-01 lvm[8599]: /dev/sde1: read failed after 0 of 2048 at
4096: Input/output error
Jan 17 15:57:19 bp-01 lvm[8599]: Couldn't find device with uuid
3lugiV-3eSP-AFAR-sdrP-H20O-wM2M-qdMANy.
Jan 17 15:57:27 bp-01 lvm[8599]: raid1 array, my_vg-my_lv, is not in-sync.
Jan 17 15:57:36 bp-01 lvm[8599]: raid1 array, my_vg-my_lv, is now in-sync.
```

Since the **raid_fault_policy** field has been set to **allocate**, the failed device is replaced with a new device from the volume group.

```
# lvs -a -o name,copy_percent,devices vg
Couldn't find device with uuid 3lugiV-3eSP-AFAR-sdrP-H20O-wM2M-qdMANy.
LV          Copy%  Devices
lv          100.00 lv_rimage_0(0),lv_rimage_1(0),lv_rimage_2(0)
[lv_rimage_0]  /dev/sdh1(1)
```

```
[lv_rimage_1]    /dev/sdf1(1)
[lv_rimage_2]    /dev/sdg1(1)
[lv_rmeta_0]     /dev/sdh1(0)
[lv_rmeta_1]     /dev/sdf1(0)
[lv_rmeta_2]     /dev/sdg1(0)
```

Note that even though the failed device has been replaced, the display still indicates that LVM could not find the failed device. This is because, although the failed device has been removed from the RAID logical volume, the failed device has not yet been removed from the volume group. To remove the failed device from the volume group, you can execute **vgreduce --removemissing VG**.

If the **raid_fault_policy** has been set to **allocate** but there are no spare devices, the allocation will fail, leaving the logical volume as it is. If the allocation fails, you have the option of fixing the drive, then initiating recovery of the failed device with the **--refresh** option of the **lvchange** command. Alternately, you can replace the failed device.

7.11.2. The warn RAID Fault Policy

In the following example, the **raid_fault_policy** field has been set to **warn** in the **lvm.conf** file. The RAID logical volume is laid out as follows.

```
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]    /dev/sdh1(1)
[my_lv_rimage_1]    /dev/sdf1(1)
[my_lv_rimage_2]    /dev/sdg1(1)
[my_lv_rmeta_0]     /dev/sdh1(0)
[my_lv_rmeta_1]     /dev/sdf1(0)
[my_lv_rmeta_2]     /dev/sdg1(0)
```

If the **/dev/sdh** device fails, the system log will display error messages. In this case, however, LVM will not automatically attempt to repair the RAID device by replacing one of the images. Instead, if the device has failed you can replace the device with the **--repair** argument of the **lvconvert** command.

7.12. REPLACING A RAID DEVICE IN A LOGICAL VOLUME

You can replace a RAID device in a logical volume with the **lvconvert** command.

- If there has been no failure on a RAID device, use the **--replace** argument of the **lvconvert** command to replace the device.
- If the RAID device has failed, use the **--repair** argument of the **lvconvert** command to replace the failed device.

7.12.1. Replacing a RAID device that has not failed

To replace a RAID device in a logical volume, use the **--replace** argument of the **lvconvert** command. Note that this command will not work if the RAID device has failed.

The format for the **lvconvert --replace** command is as follows.

```
lvconvert --replace dev_to_remove vg/lv [possible_replacements]
```

The following example creates a RAID1 logical volume and then replaces a device in that volume.

```
# lvcreate --type raid1 -m 2 -L 1G -n my_lv my_vg
Logical volume "my_lv" created
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       100.00  my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]  /dev/sdb1(1)
[my_lv_rimage_1]  /dev/sdb2(1)
[my_lv_rimage_2]  /dev/sdc1(1)
[my_lv_rmeta_0]   /dev/sdb1(0)
[my_lv_rmeta_1]   /dev/sdb2(0)
[my_lv_rmeta_2]   /dev/sdc1(0)
# lvconvert --replace /dev/sdb2 my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       37.50  my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]  /dev/sdb1(1)
[my_lv_rimage_1]  /dev/sdc2(1)
[my_lv_rimage_2]  /dev/sdc1(1)
[my_lv_rmeta_0]   /dev/sdb1(0)
[my_lv_rmeta_1]   /dev/sdc2(0)
[my_lv_rmeta_2]   /dev/sdc1(0)
```

The following example creates a RAID1 logical volume and then replaces a device in that volume, specifying which physical volume to use for the replacement.

```
# lvcreate --type raid1 -m 1 -L 100 -n my_lv my_vg
Logical volume "my_lv" created
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       100.00  my_lv_rimage_0(0),my_lv_rimage_1(0)
[my_lv_rimage_0]  /dev/sda1(1)
[my_lv_rimage_1]  /dev/sdb1(1)
[my_lv_rmeta_0]   /dev/sda1(0)
[my_lv_rmeta_1]   /dev/sdb1(0)
# pvs
PV          VG      Fmt Attr PSize  PFree
/dev/sda1   my_vg   lvm2 a-- 1020.00m 916.00m
/dev/sdb1   my_vg   lvm2 a-- 1020.00m 916.00m
/dev/sdc1   my_vg   lvm2 a-- 1020.00m 1020.00m
/dev/sdd1   my_vg   lvm2 a-- 1020.00m 1020.00m
# lvconvert --replace /dev/sdb1 my_vg/my_lv /dev/sdd1
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       28.00  my_lv_rimage_0(0),my_lv_rimage_1(0)
[my_lv_rimage_0]  /dev/sda1(1)
[my_lv_rimage_1]  /dev/sdd1(1)
[my_lv_rmeta_0]   /dev/sda1(0)
[my_lv_rmeta_1]   /dev/sdd1(0)
```

You can replace more than one RAID device at a time by specifying multiple **replace** arguments, as in the following example.

```
# lvcreate --type raid1 -m 2 -L 100 -n my_lv my_vg
```

```

Logical volume "my_lv" created
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]    /dev/sda1(1)
[my_lv_rimage_1]    /dev/sdb1(1)
[my_lv_rimage_2]    /dev/sdc1(1)
[my_lv_rmeta_0]     /dev/sda1(0)
[my_lv_rmeta_1]     /dev/sdb1(0)
[my_lv_rmeta_2]     /dev/sdc1(0)
# lvconvert --replace /dev/sdb1 --replace /dev/sdc1 my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       60.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]    /dev/sda1(1)
[my_lv_rimage_1]    /dev/sdd1(1)
[my_lv_rimage_2]    /dev/sde1(1)
[my_lv_rmeta_0]     /dev/sda1(0)
[my_lv_rmeta_1]     /dev/sdd1(0)
[my_lv_rmeta_2]     /dev/sde1(0)

```

7.12.2. Replacing a failed RAID device in a logical volume

RAID is not like traditional LVM mirroring. LVM mirroring required failed devices to be removed or the mirrored logical volume would hang. RAID arrays can keep on running with failed devices. In fact, for RAID types other than RAID1, removing a device would mean converting to a lower level RAID (for example, from RAID6 to RAID5, or from RAID4 or RAID5 to RAID0). Therefore, rather than removing a failed device unconditionally and potentially allocating a replacement, LVM allows you to replace a failed device in a RAID volume in a one-step solution by using the **--repair** argument of the **lvconvert** command.

In the following example, a RAID logical volume is laid out as follows.

```

# lvs -a -o name,copy_percent,devices my_vg
LV          Cpy%Sync Devices
my_lv       100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]    /dev/sde1(1)
[my_lv_rimage_1]    /dev/sdc1(1)
[my_lv_rimage_2]    /dev/sdd1(1)
[my_lv_rmeta_0]     /dev/sde1(0)
[my_lv_rmeta_1]     /dev/sdc1(0)
[my_lv_rmeta_2]     /dev/sdd1(0)

```

If the **/dev/sdc** device fails, the output of the **lvs** command is as follows.

```

# lvs -a -o name,copy_percent,devices my_vg
/dev/sdc: open failed: No such device or address
Couldn't find device with uuid A4kRI2-vIzA-uyCb-cci7-bOod-H5tX-lzH4Ee.
WARNING: Couldn't find all devices for LV my_vg/my_lv_rimage_1 while checking used and
assumed devices.
WARNING: Couldn't find all devices for LV my_vg/my_lv_rmeta_1 while checking used and
assumed devices.
LV          Cpy%Sync Devices
my_lv       100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]    /dev/sde1(1)

```

```
[my_lv_rimage_1]    [unknown](1)
[my_lv_rimage_2]    /dev/sdd1(1)
[my_lv_rmeta_0]     /dev/sde1(0)
[my_lv_rmeta_1]     [unknown](0)
[my_lv_rmeta_2]     /dev/sdd1(0)
```

Use the following commands to replace the failed device and display the logical volume.

```
# lvconvert --repair my_vg/my_lv
/dev/sdc: open failed: No such device or address
Couldn't find device with uuid A4kRI2-vIzA-uyCb-cci7-bOod-H5tX-lzH4Ee.
WARNING: Couldn't find all devices for LV my_vg/my_lv_rimage_1 while checking used and
assumed devices.
WARNING: Couldn't find all devices for LV my_vg/my_lv_rmeta_1 while checking used and
assumed devices.
Attempt to replace failed RAID images (requires full device resync)? [y/n]: y
Faulty devices in my_vg/my_lv successfully replaced.
# lvs -a -o name,copy_percent,devices my_vg
/dev/sdc: open failed: No such device or address
/dev/sdc1: open failed: No such device or address
Couldn't find device with uuid A4kRI2-vIzA-uyCb-cci7-bOod-H5tX-lzH4Ee.
LV          Cpy%Sync Devices
my_lv       43.79  my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]    /dev/sde1(1)
[my_lv_rimage_1]    /dev/sdb1(1)
[my_lv_rimage_2]    /dev/sdd1(1)
[my_lv_rmeta_0]     /dev/sde1(0)
[my_lv_rmeta_1]     /dev/sdb1(0)
[my_lv_rmeta_2]     /dev/sdd1(0)
```

Note that even though the failed device has been replaced, the display still indicates that LVM could not find the failed device. This is because, although the failed device has been removed from the RAID logical volume, the failed device has not yet been removed from the volume group. To remove the failed device from the volume group, you can execute **vgreduce --removemissing VG**.

If the device failure is a transient failure or you are able to repair the device that failed, you can initiate recovery of the failed device with the **--refresh** option of the **lvchange** command.

The following command refreshes a logical volume.

```
# lvchange --refresh my_vg/my_lv
```

7.13. CHECKING DATA COHERENCY IN A RAID LOGICAL VOLUME (RAID SCRUBBING)

LVM provides scrubbing support for RAID logical volumes. RAID scrubbing is the process of reading all the data and parity blocks in an array and checking to see whether they are coherent.

You initiate a RAID scrubbing operation with the **--syncaction** option of the **lvchange** command. You specify either a **check** or **repair** operation. A **check** operation goes over the array and records the number of discrepancies in the array but does not repair them. A **repair** operation corrects the discrepancies as it finds them.

The format of the command to scrub a RAID logical volume is as follows:

```
lvchange --syncaction {check|repair} vg/raid_lv
```



NOTE

The **lvchange --syncaction repair vg/raid_lv** operation does not perform the same function as the **lvconvert --repair vg/raid_lv** operation. The **lvchange --syncaction repair** operation initiates a background synchronization operation on the array, while the **lvconvert --repair** operation is designed to repair/replace failed devices in a mirror or RAID logical volume.

In support of the RAID scrubbing operation, the **lvs** command supports two new printable fields: **raid_sync_action** and **raid_mismatch_count**. These fields are not printed by default. To display these fields you specify them with the **-o** parameter of the **lvs**, as follows.

```
lvs -o +raid_sync_action,raid_mismatch_count vg/lv
```

The **raid_sync_action** field displays the current synchronization operation that the raid volume is performing. It can be one of the following values:

- **idle**: All sync operations complete (doing nothing)
- **resync**: Initializing an array or recovering after a machine failure
- **recover**: Replacing a device in the array
- **check**: Looking for array inconsistencies
- **repair**: Looking for and repairing inconsistencies

The **raid_mismatch_count** field displays the number of discrepancies found during a **check** operation.

The **Cpy%Sync** field of the **lvs** command now prints the progress of any of the **raid_sync_action** operations, including **check** and **repair**.

The **lv_attr** field of the **lvs** command output now provides additional indicators in support of the RAID scrubbing operation. Bit 9 of this field displays the health of the logical volume, and it now supports the following indicators.

- **(m)ismatches** indicates that there are discrepancies in a RAID logical volume. This character is shown after a scrubbing operation has detected that portions of the RAID are not coherent.
- **(r)efresh** indicates that a device in a RAID array has suffered a failure and the kernel regards it as failed, even though LVM can read the device label and considers the device to be operational. The logical volume should be **(r)efreshed** to notify the kernel that the device is now available, or the device should be **(r)eplaced** if it is suspected of having failed.

When you perform a RAID scrubbing operation, the background I/O required by the **sync** operations can crowd out other I/O operations to LVM devices, such as updates to volume group metadata. This can cause the other LVM operations to slow down. You can control the rate at which the RAID logical volume is scrubbed by implementing recovery throttling.

You control the rate at which **sync** operations are performed by setting the minimum and maximum I/O rate for those operations with the **--minrecoverrate** and **--maxrecoverrate** options of the **lvchange** command. You specify these options as follows.

- **--maxrecoveryrate *Rate*[bBsSkKmMgG]**

Sets the maximum recovery rate for a RAID logical volume so that it will not crowd out nominal I/O operations. The *Rate* is specified as an amount per second for each device in the array. If no suffix is given, then kiB/sec/device is assumed. Setting the recovery rate to 0 means it will be unbounded.

- **--minrecoveryrate *Rate*[bBsSkKmMgG]**

Sets the minimum recovery rate for a RAID logical volume to ensure that I/O for **sync** operations achieves a minimum throughput, even when heavy nominal I/O is present. The *Rate* is specified as an amount per second for each device in the array. If no suffix is given, then kiB/sec/device is assumed.

7.14. CONVERTING A RAID LEVEL (RAID TAKEOVER)

LVM supports *Raid takeover*, which means converting a RAID logical volume from one RAID level to another (such as from RAID 5 to RAID 6). Changing the RAID level is usually done to increase or decrease resilience to device failures or to restripe logical volumes. You use the **lvconvert** for RAID takeover. For information on RAID takeover and for examples of using the **lvconvert** to convert a RAID logical volume, see the **lvraid**(7) man page.

7.15. CHANGING ATTRIBUTES OF A RAID VOLUME (RAID RESHAPE)

RAID *reshaping* means changing attributes of a RAID logical volume while keeping the same RAID level. Some attributes you can change include RAID layout, stripe size, and number of stripes. For information on RAID reshaping and examples of using the **lvconvert** command to reshape a RAID logical volume, see the **lvraid**(7) man page.

7.16. CONTROLLING I/O OPERATIONS ON A RAID1 LOGICAL VOLUME

You can control the I/O operations for a device in a RAID1 logical volume by using the **--writemostly** and **--writebehind** parameters of the **lvchange** command. The format for using these parameters is as follows.

- **--[raid]writemostly *PhysicalVolume*[:{t|y|n}]**

Marks a device in a RAID1 logical volume as **write-mostly**. All reads to these drives will be avoided unless necessary. Setting this parameter keeps the number of I/O operations to the drive to a minimum. By default, the **write-mostly** attribute is set to yes for the specified physical volume in the logical volume. It is possible to remove the **write-mostly** flag by appending **:n** to the physical volume or to toggle the value by specifying **:t**. The **--writemostly** argument can be specified more than one time in a single command, making it possible to toggle the write-mostly attributes for all the physical volumes in a logical volume at once.

- **--[raid]writebehind *IOCount***

Specifies the maximum number of outstanding writes that are allowed to devices in a RAID1 logical volume that are marked as **write-mostly**. Once this value is exceeded, writes become synchronous, causing all writes to the constituent devices to complete before the array signals the write has completed. Setting the value to zero clears the preference and allows the system to choose the value arbitrarily.

7.17. CHANGING THE REGION SIZE ON A RAID LOGICAL VOLUME

When you create a RAID logical volume, the region size for the logical volume will be the value of the **raid_region_size** parameter in the **/etc/lvm/lvm.conf** file. You can override this default value with the **-R** option of the **lvcreate** command.

After you have created a RAID logical volume, you can change the region size of the volume with the **-R** option of the **lvconvert** command. The following example changes the region size of logical volume **vg/raidlv** to 4096K. The RAID volume must be synced in order to change the region size.

```
# lvconvert -R 4096K vg/raid1
```

```
Do you really want to change the region_size 512.00 KiB of LV vg/raid1 to 4.00 MiB? [y/n]: y
Changed region size on RAID LV vg/raid1 to 4.00 MiB.
```


CHAPTER 8. SNAPSHOT LOGICAL VOLUMES

The LVM snapshot feature provides the ability to create virtual images of a device at a particular instant without causing a service interruption.

8.1. 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 supports thinly-provisioned snapshots.

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 an 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 resizable, 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.

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.

8.2. 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 shared volume group. However, if you need to create a consistent backup of data on a shared logical volume you can activate the volume exclusively and then create the snapshot.



NOTE

Snapshots are supported for RAID logical volumes.

LVM does not allow you to create a snapshot volume that is larger than the size of the origin volume plus needed metadata for the volume. If you specify a snapshot volume that is larger than this, the system will create a snapshot volume that is only as large as will be needed for the size of the origin.

By default, a snapshot volume is skipped during normal activation commands.

The following procedure creates an origin logical volume named **origin** and a snapshot volume of the original volume named **snap**.

1. Create a logical volume named **origin** from the volume group **VG**.

```
# lvcreate -L 1G -n origin VG
Logical volume "origin" created.
```

2. Create a snapshot logical volume of **/dev/VG/origin** that is 100 MB in size named **snap**. 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/VG/origin
Logical volume "snap" created.
```

3. Display the status of logical volume **/dev/VG/origin**, showing all snapshot logical volumes and their status (active or inactive).

```
# lvsdisplay /dev/VG/origin
--- Logical volume ---
LV Path                /dev/VG/origin
```

```

LV Name          origin
VG Name          VG
LV UUID          EsFoBp-CB9H-Epl5-pUO4-Yevi-EdFS-xtFnaF
LV Write Access   read/write
LV Creation host, time host-083.virt.lab.msp.redhat.com, 2019-04-11 14:45:06 -0500
LV snapshot status source of
                  snap [active]
LV Status         available
# open           0
LV Size          1.00 GiB
Current LE       256
Segments        1
Allocation       inherit
Read ahead sectors auto
- currently set to 8192
Block device     253:6

```

4. The **lvs** command, by default, displays the origin volume and the current percentage of the snapshot volume being used. The following example shows the default output for the **lvs** command after you have created the snapshot volume, with a display that includes the devices that constitute the logical volumes.

```
# lvs -a -o +devices
```

LV	VG	Attr	LSize	Pool	Origin	Data%	Meta%	Move	Log	Cpy%	Sync	Convert
Devices												
origin	VG	owi-a-s---	1.00g									/dev/sde1(0)
snap	VG	swi-a-s---	100.00m		origin	0.00						/dev/sde1(256)



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.

In addition to the snapshot itself being invalidated when full, any mounted file systems on that snapshot device are forcibly unmounted, avoiding the inevitable file system errors upon access to the mount point. In addition, you can specify the **snapshot_autoextend_threshold** option in the **lvm.conf** file. This option allows automatic extension of a snapshot whenever the remaining snapshot space drops below the threshold you set. This feature requires that there be unallocated space in the volume group.

LVM does not allow you to create a snapshot volume that is larger than the size of the origin volume plus needed metadata for the volume. Similarly, automatic extension of a snapshot will not increase the size of a snapshot volume beyond the maximum calculated size that is necessary for the snapshot. Once a snapshot has grown large enough to cover the origin, it is no longer monitored for automatic extension.

Information on setting **snapshot_autoextend_threshold** and **snapshot_autoextend_percent** is provided in the **/etc/lvm/lvm.conf** file itself.

8.3. MERGING SNAPSHOT VOLUMES

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 further information on the **lvconvert --merge** command, see the **lvconvert(8)** man page.

CHAPTER 9. CREATING AND MANAGING THINLY-PROVISIONED LOGICAL VOLUMES (THIN VOLUMES)

Logical volumes can be thinly provisioned. This allows you to create logical volumes that are larger than the available extents.

9.1. THINLY-PROVISIONED LOGICAL VOLUMES (THIN VOLUMES)

Logical volumes can be thinly provisioned. This allows you to create logical volumes that are larger than the available extents. Using thin provisioning, you can manage a storage pool of free space, known as a thin pool, which can be allocated to an arbitrary number of devices when needed by applications. You can then create devices that can be bound to the thin pool for later allocation when an application actually writes to the logical volume. The thin pool can be expanded dynamically when needed for cost-effective allocation of storage space.



NOTE

Thin volumes are not supported across the nodes in a cluster. The thin pool and all its thin volumes must be exclusively activated on only one cluster node.

By using thin provisioning, a storage administrator can overcommit the physical storage, often avoiding the need to purchase additional storage. For example, if ten users each request a 100GB file system for their application, the storage administrator can create what appears to be a 100GB file system for each user but which is backed by less actual storage that is used only when needed. When using thin provisioning, it is important that the storage administrator monitor the storage pool and add more capacity if it starts to become full.

To make sure that all available space can be used, LVM supports data discard. This allows for re-use of the space that was formerly used by a discarded file or other block range.

Thin volumes provide support for a new implementation of copy-on-write (COW) snapshot logical volumes, which allow many virtual devices to share the same data in the thin pool.

9.2. CREATING THINLY-PROVISIONED LOGICAL VOLUMES

This procedure provides an overview of the basic commands you use to create and grow thinly-provisioned logical volumes. For detailed information on LVM thin provisioning as well as information on using the LVM commands and utilities with thinly-provisioned logical volumes, see the **lvthin(7)** man page.

To create a thin volume, perform the following tasks:

1. Create a volume group with the **vgcreate** command.
2. Create a thin pool with the **lvcreate** command.
3. Create a thin volume in the thin pool with the **lvcreate** command.

You can use the **-T** (or **--thin**) option of the **lvcreate** command to create either a thin pool or a thin volume. You can also use **-T** option of the **lvcreate** command to create both a thin pool and a thin volume in that pool at the same time with a single command.

The following command uses the **-T** option of the **lvcreate** command to create a thin pool named **mythinpool** in the volume group **vg001** and that is 100M in size. Note that since you are creating a pool

of physical space, you must specify the size of the pool. The **-T** option of the **lvcreate** command does not take an argument; it deduces what type of device is to be created from the other options the command specifies.

```
# lvcreate -L 100M -T vg001/mythinpool
```

Thin pool volume with chunk size 64.00 KiB can address at most 15.81 TiB of data.

Logical volume "mythinpool" created.

```
# lvs
```

LV	VG	Attr	LSize	Pool	Origin	Data%	Meta%	Move	Log	Cpy%	Sync	Convert
mythinpool	vg001	twi-a-tz--	100.00m			0.00	10.84					

The following command uses the **-T** option of the **lvcreate** command to create a thin volume named **thinvolume** in the thin pool **vg001/mythinpool**. Note that in this case you are specifying virtual size, and that you are specifying a virtual size for the volume that is greater than the pool that contains it.

```
# lvcreate -V 1G -T vg001/mythinpool -n thinvolume
```

WARNING: Sum of all thin volume sizes (1.00 GiB) exceeds the size of thin pool vg001/mythinpool (100.00 MiB).

WARNING: You have not turned on protection against thin pools running out of space.

WARNING: Set activation/thin_pool_autoextend_threshold below 100 to trigger automatic extension of thin pools before they get full.

Logical volume "thinvolume" created.

```
# lvs
```

LV	VG	Attr	LSize	Pool	Origin	Data%	Move	Log	Copy%	Convert
mythinpool	vg001	twi-a-tz	100.00m			0.00				
thinvolume	vg001	Vwi-a-tz	1.00g	mythinpool		0.00				

The following command uses the **-T** option of the **lvcreate** command to create a thin pool and a thin volume in that pool by specifying both a size and a virtual size argument for the **lvcreate** command. This command creates a thin pool named **mythinpool** in the volume group **vg001** and it also creates a thin volume named **thinvolume** in that pool.

```
# lvcreate -L 100M -T vg001/mythinpool -V 1G -n thinvolume
```

Thin pool volume with chunk size 64.00 KiB can address at most 15.81 TiB of data.

WARNING: Sum of all thin volume sizes (1.00 GiB) exceeds the size of thin pool vg001/mythinpool (100.00 MiB).

WARNING: You have not turned on protection against thin pools running out of space.

WARNING: Set activation/thin_pool_autoextend_threshold below 100 to trigger automatic extension of thin pools before they get full.

Logical volume "thinvolume" created.

```
# lvs
```

LV	VG	Attr	LSize	Pool	Origin	Data%	Meta%	Move	Log	Cpy%	Sync	Convert
mythinpool	vg001	twi-aotz--	100.00m			0.00	10.94					
thinvolume	vg001	Vwi-a-tz--	1.00g	mythinpool		0.00						

You can also create a thin pool by specifying the **--thinpool** parameter of the **lvcreate** command. Unlike the **-T** option, the **--thinpool** parameter requires an argument, which is the name of the thin pool logical volume that you are creating. The following example specifies the **--thinpool** parameter of the **lvcreate** command to create a thin pool named **mythinpool** in the volume group **vg001** and that is 100M in size:

```
# lvcreate -L 100M --thinpool mythinpool vg001
```

Thin pool volume with chunk size 64.00 KiB can address at most 15.81 TiB of data.

Logical volume "mythinpool" created.

```
# lvs
```

LV	VG	Attr	LSize	Pool	Origin	Data%	Meta%	Move	Log	Cpy%	Sync	Convert
mythinpool	vg001	twi-a-tz--	100.00m			0.00	10.84					

Striping is supported for pool creation. The following command creates a 100M thin pool named **pool** in volume group **vg001** with two 64 kB stripes and a chunk size of 256 kB. It also creates a 1T thin volume, **vg00/thin_lv**.

```
# lvcreate -i 2 -l 64 -c 256 -L 100M -T vg00/pool -V 1T --name thin_lv
```

You can extend the size of a thin volume with the **lvextend** command. You cannot, however, reduce the size of a thin pool.

The following command resizes an existing thin pool that is 100M in size by extending it another 100M.

```
# lvextend -L+100M vg001/mythinpool
```

```
Extending logical volume mythinpool to 200.00 MiB
Logical volume mythinpool successfully resized
```

```
# lvs
```

LV	VG	Attr	LSize	Pool	Origin	Data%	Meta%	Move	Log	Copy%	Convert
mythinpool	vg001	twi-a-tz	200.00m			0.00					
thinvolume	vg001	Vwi-a-tz	1.00g	mythinpool		0.00					

As with other types of logical volumes, you can rename the volume with the **lvrename**, you can remove the volume with the **lvremove**, and you can display information about the volume with the **lvs** and **lvdisplay** commands.

By default, the **lvcreate** command sets the size of the thin pool's metadata logical volume according to the formula $(\text{Pool_LV_size} / \text{Pool_LV_chunk_size} * 64)$. If you will have large numbers of snapshots or if you have small chunk sizes for your thin pool and thus expect significant growth of the size of the thin pool at a later time, you may need to increase the default value of the thin pool's metadata volume with the **--poolmetadatasize** parameter of the **lvcreate** command. The supported value for the thin pool's metadata logical volume is in the range between 2MiB and 16GiB.

You can use the **--thinpool** parameter of the **lvconvert** command to convert an existing logical volume to a thin pool volume. When you convert an existing logical volume to a thin pool volume, you must use the **--poolmetadata** parameter in conjunction with the **--thinpool** parameter of the **lvconvert** to convert an existing logical volume to the thin pool volume's metadata volume.



NOTE

Converting a logical volume to a thin pool volume or a thin pool metadata volume destroys the content of the logical volume, since in this case the **lvconvert** does not preserve the content of the devices but instead overwrites the content.

The following example converts the existing logical volume **lv1** in volume group **vg001** to a thin pool volume and converts the existing logical volume **lv2** in volume group **vg001** to the metadata volume for that thin pool volume.

```
# lvconvert --thinpool vg001/lv1 --poolmetadata vg001/lv2
```

```
Converted vg001/lv1 to thin pool.
```

9.3. THINLY-PROVISIONED SNAPSHOT VOLUMES

Red Hat Enterprise Linux provides support for thinly-provisioned snapshot volumes. Thin snapshot volumes allow many virtual devices to be stored on the same data volume. This simplifies administration and allows for the sharing of data between snapshot volumes.

As for all LVM snapshot volumes, as well as all thin volumes, thin snapshot volumes are not supported across the nodes in a cluster. The snapshot volume must be exclusively activated on only one cluster node.

Thin snapshot volumes provide the following benefits:

- A thin snapshot volume can reduce disk usage when there are multiple snapshots of the same origin volume.
- If there are multiple snapshots of the same origin, then a write to the origin will cause one COW operation to preserve the data. Increasing the number of snapshots of the origin should yield no major slowdown.
- Thin snapshot volumes can be used as a logical volume origin for another snapshot. This allows for an arbitrary depth of recursive snapshots (snapshots of snapshots of snapshots...).
- A snapshot of a thin logical volume also creates a thin logical volume. This consumes no data space until a COW operation is required, or until the snapshot itself is written.
- A thin snapshot volume does not need to be activated with its origin, so a user may have only the origin active while there are many inactive snapshot volumes of the origin.
- When you delete the origin of a thinly-provisioned snapshot volume, each snapshot of that origin volume becomes an independent thinly-provisioned volume. This means that instead of merging a snapshot with its origin volume, you may choose to delete the origin volume and then create a new thinly-provisioned snapshot using that independent volume as the origin volume for the new snapshot.

Although there are many advantages to using thin snapshot volumes, there are some use cases for which the older LVM snapshot volume feature may be more appropriate to your needs:

- You cannot change the chunk size of a thin pool. If the thin pool has a large chunk size (for example, 1MB) and you require a short-living snapshot for which a chunk size that large is not efficient, you may elect to use the older snapshot feature.
- You cannot limit the size of a thin snapshot volume; the snapshot will use all of the space in the thin pool, if necessary. This may not be appropriate for your needs.

In general, you should consider the specific requirements of your site when deciding which snapshot format to use.

9.4. CREATING THINLY-PROVISIONED SNAPSHOT VOLUMES

Red Hat Enterprise Linux provides support for thinly-provisioned snapshot volumes.



NOTE

This section provides an overview of the basic commands you use to create and grow thinly-provisioned snapshot volumes. For detailed information on LVM thin provisioning as well as information on using the LVM commands and utilities with thinly-provisioned logical volumes, see the **lvminthin(7)** man page.



IMPORTANT

When creating a thin snapshot volume, you do not specify the size of the volume. If you specify a size parameter, the snapshot that will be created will not be a thin snapshot volume and will not use the thin pool for storing data. For example, the command **lvcreate -s vg/thinvolume -L10M** will not create a thin snapshot, even though the origin volume is a thin volume.

Thin snapshots can be created for thinly-provisioned origin volumes, or for origin volumes that are not thinly-provisioned.

You can specify a name for the snapshot volume with the **--name** option of the **lvcreate** command. The following command creates a thinly-provisioned snapshot volume of the thinly-provisioned logical volume **vg001/thinvolume** that is named **mynsnapshot1**.

```
# lvcreate -s --name mynsnapshot1 vg001/thinvolume
Logical volume "mynsnapshot1" created
# lvs
LV      VG      Attr   LSize   Pool    Origin   Data%  Move Log Copy%  Convert
mynsnapshot1 vg001   Vwi-a-tz 1.00g mythinpool thinvolume 0.00
mythinpool  vg001   twi-a-tz 100.00m                0.00
thinvolume  vg001   Vwi-a-tz 1.00g mythinpool                0.00
```

A thin snapshot volume has the same characteristics as any other thin volume. You can independently activate the volume, extend the volume, rename the volume, remove the volume, and even snapshot the volume.

By default, a snapshot volume is skipped during normal activation commands. For information on controlling the activation of a logical volume, see [Logical volume activation](#).

You can also create a thinly-provisioned snapshot of a non-thinly-provisioned logical volume. Since the non-thinly-provisioned logical volume is not contained within a thin pool, it is referred to as an *external origin*. External origin volumes can be used and shared by many thinly-provisioned snapshot volumes, even from different thin pools. The external origin must be inactive and read-only at the time the thinly-provisioned snapshot is created.

To create a thinly-provisioned snapshot of an external origin, you must specify the **--thinpool** option. The following command creates a thin snapshot volume of the read-only inactive volume **origin_volume**. The thin snapshot volume is named **mythinsnap**. The logical volume **origin_volume** then becomes the thin external origin for the thin snapshot volume **mythinsnap** in volume group **vg001** that will use the existing thin pool **vg001/pool**. Because the origin volume must be in the same volume group as the snapshot volume, you do not need to specify the volume group when specifying the origin logical volume.

```
# lvcreate -s --thinpool vg001/pool origin_volume --name mythinsnap
```

You can create a second thinly-provisioned snapshot volume of the first snapshot volume, as in the following command.

```
# lvcreate -s vg001/mythinsnap --name my2ndthinsnap
```

You can display a list of all ancestors and descendants of a thin snapshot logical volume by specifying the **lv_ancestors** and **lv_descendants** reporting fields of the **lvs** command.

In the following example:

- **stack1** is an origin volume in volume group **vg001**.
- **stack2** is a snapshot of **stack1**
- **stack3** is a snapshot of **stack2**
- **stack4** is a snapshot of **stack3**

Additionally:

- **stack5** is also a snapshot of **stack2**
- **stack6** is a snapshot of **stack5**

```
$ lvs -o name,lv_ancestors,lv_descendants vg001
LV      Ancestors      Descendants
stack1
stack2  stack1          stack2,stack3,stack4,stack5,stack6
stack3  stack2,stack1      stack3,stack4,stack5,stack6
stack4  stack2,stack1      stack4
stack5  stack3,stack2,stack1
stack6  stack3,stack2,stack1
stack6  stack5,stack2,stack1
pool
```

NOTE

The **lv_ancestors** and **lv_descendants** fields display existing dependencies but do not track removed entries which can break a dependency chain if the entry was removed from the middle of the chain. For example, if you remove the logical volume **stack3** from this sample configuration, the display is as follows.

```
$ lvs -o name,lv_ancestors,lv_descendants vg001
LV      Ancestors      Descendants
stack1
stack2  stack1          stack2,stack5,stack6
stack4
stack5  stack2,stack1      stack6
stack6  stack5,stack2,stack1
pool
```

You can configure your system to track and display logical volumes that have been removed, and you can display the full dependency chain that includes those volumes by specifying the **lv_ancestors_full** and **lv_descendants_full** fields.

9.5. TRACKING AND DISPLAYING THIN SNAPSHOT VOLUMES THAT HAVE BEEN REMOVED

You can configure your system to track thin snapshot and thin logical volumes that have been removed by enabling the **record_lvs_history** metadata option in the **lvm.conf** configuration file. This allows you to display a full thin snapshot dependency chain that includes logical volumes that have been removed from the original dependency chain and have become *historical* logical volumes.

You can configure your system to retain historical volumes for a defined period of time by specifying the retention time, in seconds, with the **lvs_history_retention_time** metadata option in the **lvm.conf** configuration file.

A historical logical volume retains a simplified representation of the logical volume that has been removed, including the following reporting fields for the volume:

- **lv_time_removed**: the removal time of the logical volume
- **lv_time**: the creation time of the logical volume
- **lv_name**: the name of the logical volume
- **lv_uuid**: the UUID of the logical volume
- **vg_name**: the volume group that contains the logical volume.

When a volume is removed, the historical logical volume name acquires a hyphen as a prefix. For example, when you remove the logical volume **lvol1**, the name of the historical volume is **-lvol1**. A historical logical volume cannot be reactivated.

Even when the **record_lvs_history** metadata option enabled, you can prevent the retention of historical logical volumes on an individual basis when you remove a logical volume by specifying the **--nohistory** option of the **lvremove** command.

To include historical logical volumes in volume display, you specify the **-H|--history** option of an LVM display command. You can display a full thin snapshot dependency chain that includes historical volumes by specifying the **lv_full_ancestors** and **lv_full_descendants** reporting fields along with the **-H** option.

The following series of commands provides examples of how you can display and manage historical logical volumes.

1. Ensure that historical logical volumes are retained by setting **record_lvs_history=1** in the **lvm.conf** file. This metadata option is not enabled by default.
2. Enter the following command to display a thin provisioned snapshot chain.
In this example:

- **lvol1** is an origin volume, the first volume in the chain.
- **lvol2** is a snapshot of **lvol1**.
- **lvol3** is a snapshot of **lvol2**.
- **lvol4** is a snapshot of **lvol3**.
- **lvol5** is also a snapshot of **lvol3**.

Note that even though the example **lvs** display command includes the **-H** option, no thin snapshot volume has yet been removed and there are no historical logical volumes to display.

```
# lvs -H -o name,full_ancestors,full_descendants
LV   FAncestors      FDescendants
lvol1                lvol2,lvol3,lvol4,lvol5
lvol2 lvol1          lvol3,lvol4,lvol5
lvol3 lvol2,lvol1     lvol4,lvol5
```

```
lvol4 lvol3,lvol2,lvol1
lvol5 lvol3,lvol2,lvol1
pool
```

3. Remove logical volume **lvvol3** from the snapshot chain, then run the following **lvs** command again to see how historical logical volumes are displayed, along with their ancestors and descendants.

```
# lvremove -f vg/lvol3
Logical volume "lvol3" successfully removed
# lvs -H -o name,full_ancestors,full_descendants
LV   FAncestors      FDescendants
lvol1          lvol2,-lvol3,lvol4,lvol5
lvol2 lvol1        -lvol3,lvol4,lvol5
-lvol3 lvol2,lvol1  lvol4,lvol5
lvol4 -lvol3,lvol2,lvol1
lvol5 -lvol3,lvol2,lvol1
pool
```

4. You can use the **lv_time_removed** reporting field to display the time a historical volume was removed.

```
# lvs -H -o name,full_ancestors,full_descendants,time_removed
LV   FAncestors      FDescendants      RTime
lvol1          lvol2,-lvol3,lvol4,lvol5
lvol2 lvol1        -lvol3,lvol4,lvol5
-lvol3 lvol2,lvol1  lvol4,lvol5      2016-03-14 14:14:32 +0100
lvol4 -lvol3,lvol2,lvol1
lvol5 -lvol3,lvol2,lvol1
pool
```

5. You can reference historical logical volumes individually in a display command by specifying the *vgname/lvname* format, as in the following example. Note that the fifth bit in the **lv_attr** field is set to **h** to indicate the volume is a historical volume.

```
# lvs -H vg/-lvol3
LV VG Attr  LSize
-lvol3 vg ----h----- 0
```

6. LVM does not keep historical logical volumes if the volume has no live descendant. This means that if you remove a logical volume at the end of a snapshot chain, the logical volume is not retained as a historical logical volume.

```
# lvremove -f vg/lvol5
Automatically removing historical logical volume vg/-lvol5.
Logical volume "lvol5" successfully removed
# lvs -H -o name,full_ancestors,full_descendants
LV   FAncestors      FDescendants
lvol1          lvol2,-lvol3,lvol4
lvol2 lvol1        -lvol3,lvol4
-lvol3 lvol2,lvol1  lvol4
lvol4 -lvol3,lvol2,lvol1
pool
```

- Run the following commands to remove the volume **lv1** and **lv2** and to see how the **lvs** command displays the volumes once they have been removed.

```
# lvremove -f vg/lv1 vg/lv2
Logical volume "lv1" successfully removed
Logical volume "lv2" successfully removed
# lvs -H -o name,full_ancestors,full_descendants
LV   FAncestors      FDescendants
-lv1          -lv2,-lv3,lv4
-lv2 -lv1          -lv3,lv4
-lv3 -lv2,-lv1     lv4
lv4  -lv3,-lv2,-lv1
pool
```

- To remove a historical logical volume completely, you can run the **lvremove** command again, specifying the name of the historical volume that now includes the hyphen, as in the following example.

```
# lvremove -f vg/-lv3
Historical logical volume "lv3" successfully removed
# lvs -H -o name,full_ancestors,full_descendants
LV   FAncestors      FDescendants
-lv1          -lv2,lv4
-lv2 -lv1          lv4
lv4  -lv2,-lv1
pool
```

- A historical logical volumes is retained as long as there is a chain that includes live volumes in its descendants. This means that removing a historical logical volume also removes all of the logical volumes in the chain if no existing descendant is linked to them, as shown in the following example.

```
# lvremove -f vg/lv4
Automatically removing historical logical volume vg/-lv1.
Automatically removing historical logical volume vg/-lv2.
Automatically removing historical logical volume vg/-lv4.
Logical volume "lv4" successfully removed
```

CHAPTER 10. LVM CACHE LOGICAL VOLUMES

LVM provides full support for LVM cache logical volumes. A cache logical volume uses a small logical volume consisting of fast block devices (such as SSD drives) to improve the performance of a larger and slower logical volume by storing the frequently used blocks on the smaller, faster logical volume.

10.1. CACHE VOLUME TYPES

LVM caching uses the following LVM logical volume types. All of these associated logical volumes must be in the same volume group.

- Origin logical volume – the large, slow logical volume
- Cache pool logical volume – the small, fast logical volume, which is composed of two devices: the cache data logical volume, and the cache metadata logical volume
- Cache data logical volume – the logical volume containing the data blocks for the cache pool logical volume
- Cache metadata logical volume – the logical volume containing the metadata for the cache pool logical volume, which holds the accounting information that specifies where data blocks are stored (for example, on the origin logical volume or the cache data logical volume).
- Cache logical volume – the logical volume containing the origin logical volume and the cache pool logical volume. This is the resultant usable device which encapsulates the various cache volume components.

10.2. CREATING AN LVM CACHE LOGICAL VOLUME

The following procedure creates an LVM cache logical volume.

1. Create a volume group that contains a slow physical volume and a fast physical volume. In this example, **/dev/sde1** is a slow device and **/dev/sdf1** is a fast device and both devices are contained in volume group **VG**.

```
# pvcreate /dev/sde1
# pvcreate /dev/sdf1
# vgcreate VG /dev/sde1 /dev/sdf1
```

2. Create the origin volume. This example creates an origin volume named **lv** that is ten gigabytes in size and that consists of **/dev/sde1**, the slow physical volume.

```
# lvcreate -L 10G -n lv VG /dev/sde1
```

3. Create the cache pool logical volume. This example creates the cache pool logical volume named **cpool** on the fast device **/dev/sdf1**, which is part of the volume group **VG**. The cache pool logical volume this command creates consists of the hidden cache data logical volume **cpool_cdata** and the hidden cache metadata logical volume **cpool_cmeta**.

```
# lvcreate --type cache-pool -L 5G -n cpool VG /dev/sdf1
Using default stripesize 64.00 KiB.
Logical volume "cpool" created.
# lvs -a -o name,size,attr,devices VG
LV          LSize Attr   Devices
```

```

cpool      5.00g Cwi---C--- cpool_cdata(0)
[cpool_cdata] 5.00g Cwi----- /dev/sdf1(4)
[cpool_cmeta] 8.00m ewi----- /dev/sdf1(2)
lv         10.00g -wi-a----- /dev/sde1(0)
[lvol0_pmspare] 8.00m ewi----- /dev/sdf1(0)

```

For more complicated configurations you may need to create the cache data and the cache metadata logical volumes individually and then combine the volumes into a cache pool logical volume. For information on this procedure, see the **lvmcache(7)** man page.

4. Create the cache logical volume by linking the cache pool logical volume to the origin logical volume. The resulting user-accessible cache logical volume takes the name of the origin logical volume. The origin logical volume becomes a hidden logical volume with **_corig** appended to the original name. Note that this conversion can be done live, although you must ensure you have performed a backup first.

```

# lvconvert --type cache --cachepool cpool VG/lv
Logical volume cpool is now cached.
# lvs -a -o name,size,attr,devices VG
LV          LSize Attr   Devices
[cpool]      5.00g Cwi---C--- cpool_cdata(0)
[cpool_cdata] 5.00g Cwi-ao---- /dev/sdf1(4)
[cpool_cmeta] 8.00m ewi-ao---- /dev/sdf1(2)
lv          10.00g Cwi-a-C--- lv_corig(0)
[lv_corig]   10.00g owi-aoC--- /dev/sde1(0)
[lvol0_pmspare] 8.00m ewi----- /dev/sdf1(0)

```

5. Optionally, you can convert the cached logical volume to a thin pool logical volume. Note that any thin logical volumes created from the pool will share the cache. The following command uses the fast device, **/dev/sdf1**, for allocating the thin pool metadata (**lv_tmeta**). This is the same device that is used by the cache pool volume, which means that the thin pool metadata volume shares that device with both the cache data logical volume **cpool_cdata** and the cache metadata logical volume **cpool_cmeta**.

```

# lvconvert --type thin-pool VG/lv /dev/sdf1
WARNING: Converting logical volume VG/lv to thin pool's data volume with metadata
wiping.
THIS WILL DESTROY CONTENT OF LOGICAL VOLUME (filesystem etc.)
Do you really want to convert VG/lv? [y/n]: y
Converted VG/lv to thin pool.
# lvs -a -o name,size,attr,devices vg
LV          LSize Attr   Devices
[cpool]      5.00g Cwi---C--- cpool_cdata(0)
[cpool_cdata] 5.00g Cwi-ao---- /dev/sdf1(4)
[cpool_cmeta] 8.00m ewi-ao---- /dev/sdf1(2)
lv          10.00g twi-a-tz-- lv_tdata(0)
[lv_tdata]   10.00g Cwi-aoC--- lv_tdata_corig(0)
[lv_tdata_corig] 10.00g owi-aoC--- /dev/sde1(0)
[lv_tmeta]   12.00m ewi-ao---- /dev/sdf1(1284)
[lvol0_pmspare] 12.00m ewi----- /dev/sdf1(0)
[lvol0_pmspare] 12.00m ewi----- /dev/sdf1(1287)

```

For further information on LVM cache volumes, including additional administrative examples, see the **lvmcache(7)** man page.

CHAPTER 11. LOGICAL VOLUME ACTIVATION

A logical volume that is in an active state can be used through a block device. A logical volume that is activated is accessible and is subject to change. When you create a logical volume it is activated by default.

There are various circumstances for which you need to make an individual logical volume inactive and thus unknown to the kernel. You can activate or deactivate individual logical volume with the **-a** option of the **lvchange** command.

The format for the command to deactivate an individual logical volume is as follows.

```
lvchange -an vg/lv
```

The format for the command to activate an individual logical volume is as follows.

```
lvchange -ay vg/lv
```

You can activate or deactivate all of the logical volumes in a volume group with the **-a** option of the **vgchange** command. This is the equivalent of running the **lvchange -a** command on each individual logical volume in the volume group.

The format for the command to deactivate all of the logical volumes in a volume group is as follows.

```
vgchange -an vg
```

The format for the command to activate all of the logical volumes in a volume group is as follows.

```
vgchange -ay vg
```

11.1. CONTROLLING AUTOACTIVATION OF LOGICAL VOLUMES

Autoactivation of a logical volume refers to the event-based automatic activation of a logical volume during system startup. As devices become available on the system (device online events), **systemd/udev** runs the **lvm2-pvscan** service for each device. This service runs the **pvscan --cache -aay device** command, which reads the named device. If the device belongs to a volume group, the **pvscan** command will check if all of the physical volumes for that volume group are present on the system. If so, the command will activate logical volumes in that volume group.

You can use the following configuration options in the **/etc/lvm/lvm.conf** configuration file to control autoactivation of logical volumes.

- **global/event_activation**

When **event_activation** is disabled, **systemd/udev** will autoactivate logical volume only on whichever physical volumes are present during system startup. If all physical volumes have not appeared yet, then some logical volumes may not be autoactivated.

- **activation/auto_activation_volume_list**

Setting **auto_activation_volume_list** to an empty list disables autoactivation entirely. Setting **auto_activation_volume_list** to specific logical volumes and volume groups limits autoactivation to those logical volumes.

For information on setting these options, see the **/etc/lvm/lvm.conf** configuration file.

11.2. CONTROLLING LOGICAL VOLUME ACTIVATION

You can control the activation of logical volume in the following ways:

- Through the **activation/volume_list** setting in the **/etc/lvm/conf** file. This allows you to specify which logical volumes are activated. For information on using this option, see the **/etc/lvm/lvm.conf** configuration file.
- By means of the activation skip flag for a logical volume. When this flag is set for a logical volume, the volume is skipped during normal activation commands.

You can set the activation skip flag on a logical volume in the following ways.

- You can turn off the activation skip flag when creating a logical volume by specifying the **-kn** or **--setactivationskip n** option of the **lvcreate** command.
- You can turn off the activation skip flag for an existing logical volume by specifying the **-kn** or **--setactivationskip n** option of the **lvchange** command.
- You can turn on the activation skip flag on again for a volume where it has been turned off with the **-ky** or **--setactivationskip y** option of the **lvchange** command.

To determine whether the activation skip flag is set for a logical volume run the **lvs** command, which displays the **k** attribute as in the following example.

```
# lvs vg/thin1s1
LV      VG Attr   LSize Pool Origin
thin1s1  vg  Vwi---tz-k 1.00t pool0 thin1
```

You can activate a logical volume with the **k** attribute set by using the **-K** or **--ignoreactivationskip** option in addition to the standard **-ay** or **--activate y** option.

By default, thin snapshot volumes are flagged for activation skip when they are created. You can control the default activation skip setting on new thin snapshot volumes with the **auto_set_activation_skip** setting in the **/etc/lvm/lvm.conf** file.

The following command activates a thin snapshot logical volume that has the activation skip flag set.

```
# lvchange -ay -K VG/SnapLV
```

The following command creates a thin snapshot without the activation skip flag

```
# lvcreate --type thin -n SnapLV -kn -s ThinLV --thinpool VG/ThinPoolLV
```

The following command removes the activation skip flag from a snapshot logical volume.

```
# lvchange -kn VG/SnapLV
```

11.3. ACTIVATING SHARED LOGICAL VOLUMES

You can control logical volume activation of a shared logical volume with the **-a** option of the **lvchange** and **vgchange** commands, as follows.

Command	Activation
lvchange -ay e	Activate the shared logical volume in exclusive mode, allowing only a single host to activate the logical volume. If the activation fails, as would happen if the logical volume is active on another host, an error is reported.
lvchange -asy	Activate the shared logical volume in shared mode, allowing multiple hosts to activate the logical volume concurrently. If the activation fails, as would happen if the logical volume is active exclusively on another host, an error is reported. If the logical type prohibits shared access, such as a snapshot, the command will report an error and fail. Logical volume types that cannot be used concurrently from multiple hosts include thin, cache, raid, and snapshot.
lvchange -an	Deactivate the logical volume.

11.4. ACTIVATING A LOGICAL VOLUME WITH MISSING DEVICES

You can configure how a logical volume with missing devices is activated by setting the **activation_mode** parameter with the **lvchange** command to one of the following values.

Activation Mode	Meaning
complete	Allows only logical volumes with no missing physical volumes. to be activated. This is the most restrictive mode.
degraded	Allows RAID logical volumes with missing physical volumes to be activated.
partial	Allows any logical volume with missing physical volumes to be activated. This option should be used for recovery or repair only.

The default value of **activation_mode** is determined by the **activation_mode** setting in the **/etc/lvm/lvm.conf** file. For further information, see the **lvraid(7)** man page.

CHAPTER 12. CONTROLLING LVM DEVICE SCANNING

You can control LVM device scanning by configuring filters in the `/etc/lvm/lvm.conf` file. The filters in the `lvm.conf` file consist of a series of simple regular expressions that get applied to the device names in the `/dev` directory to decide whether to accept or reject each block device found.

12.1. CONFIGURING FILTERS TO CONTROL DEVICE SCANNING

The following examples show the use of filters to control which devices LVM scans. Note that some of these examples do not necessarily represent recommended 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/soloooperation/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/*/" ]
```

12.2. CONTROLLING WHETHER LVM COMMANDS SCAN LOGICAL VOLUMES

By default, LVM commands do not scan the logical volumes on your system. This default behavior provides the following advantages:

- If there are many active logical volumes on the system, every LVM command would require additional time, negatively impacting performance and causing unwanted delays or timeouts.
- If logical volumes contain physical volumes from a guest VM image, the host usually does not want to scan or use those layered physical volumes which belong to the guest. Note, however, that in the cases where a guest VM's physical volume exists directly on an SCSI device visible to the host, in order to prevent LVM on the host from accessing those physical volumes you will need to configure a filter, as described in [Configuring filters to control device scanning](#).

Scanning logical volumes may be necessary when layering physical volumes on top of logical volumes is intentional. This will allow the `pvccreate` command to be run on a logical volume. To configure LVM to scan all logical volumes, set the `scan_lvs` configuration option in the `/etc/lvm/lvm.conf` file to

scan_lvs=1. To restrict which logical volumes LVM commands scan, you can then set up device filters in the **/etc/lvm/lvm.conf** configuration file, as described in [Configuring filters to control device scanning](#).

CHAPTER 13. CONTROLLING LVM ALLOCATION

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

13.1. LVM ALLOCATION POLICIES

When an LVM operation needs to allocate physical extents for one or more logical volumes, the allocation proceeds as follows:

- The complete set of unallocated physical extents in the volume group is generated for consideration. If you supply any ranges of physical extents at the end of the command line, only unallocated physical extents within those ranges on the specified physical volumes are considered.
- Each allocation policy is tried in turn, starting with the strictest policy (**contiguous**) and ending with the allocation policy specified using the **--alloc** option or set as the default for the particular logical volume or volume group. For each policy, working from the lowest-numbered logical extent of the empty logical volume space that needs to be filled, as much space as possible is allocated, according to the restrictions imposed by the allocation policy. If more space is needed, LVM moves on to the next policy.

The allocation policy restrictions are as follows:

- An allocation policy of **contiguous** requires that the physical location of any logical extent that is not the first logical extent of a logical volume is adjacent to the physical location of the logical extent immediately preceding it.
When a logical volume is striped or mirrored, the **contiguous** allocation restriction is applied independently to each stripe or mirror image (leg) that needs space.
- An allocation policy of **cling** requires that the physical volume used for any logical extent be added to an existing logical volume that is already in use by at least one logical extent earlier in that logical volume. If the configuration parameter **allocation/cling_tag_list** is defined, then two physical volumes are considered to match if any of the listed tags is present on both physical volumes. This allows groups of physical volumes with similar properties (such as their physical location) to be tagged and treated as equivalent for allocation purposes.
When a Logical Volume is striped or mirrored, the **cling** allocation restriction is applied independently to each stripe or mirror image (leg) that needs space.
- An allocation policy of **normal** will not choose a physical extent that shares the same physical volume as a logical extent already allocated to a parallel logical volume (that is, a different stripe or mirror image/leg) at the same offset within that parallel logical volume.
When allocating a mirror log at the same time as logical volumes to hold the mirror data, an allocation policy of **normal** will first try to select different physical volumes for the log and the data. If that is not possible and the **allocation/mirror_logs_require_separate_pvs** configuration parameter is set to 0, it will then allow the log to share physical volume(s) with part of the data.

Similarly, when allocating thin pool metadata, an allocation policy of **normal** will follow the same considerations as for allocation of a mirror log, based on the value of the **allocation/thin_pool_metadata_require_separate_pvs** configuration parameter.

- 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 allocation policies can be changed using the **vgchange** command.



NOTE

If you rely upon any layout behavior beyond that documented in this section according to the defined allocation policies, you should note that this might change in future versions of the code. For example, if you supply on the command line two empty physical volumes that have an identical number of free physical extents available for allocation, LVM currently considers using each of them in the order they are listed; there is no guarantee that future releases will maintain that property. If it is important to obtain a specific layout for a particular Logical Volume, then you should build it up through a sequence of **lvcreate** and **lvconvert** steps such that the allocation policies applied to each step leave LVM no discretion over the layout.

To view the way the allocation process currently works in any specific case, you can read the debug logging output, for example by adding the **-vvvv** option to a command.

13.2. 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.

13.3. 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 **/etc/lvm/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. You can then specify the following line in the **lvm.conf** file:

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

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.

```
# pvs -a -o +pv_tags /dev/sd[bcdefgh]
PV      VG  Fmt Attr PSize PFree PV Tags
/dev/sdb1 taft lvm2 a-- 15.00g 15.00g A
/dev/sdc1 taft lvm2 a-- 15.00g 15.00g B
/dev/sdd1 taft lvm2 a-- 15.00g 15.00g B
/dev/sde1 taft lvm2 a-- 15.00g 15.00g C
/dev/sdf1 taft lvm2 a-- 15.00g 15.00g C
/dev/sdg1 taft lvm2 a-- 15.00g 15.00g A
/dev/sdh1 taft lvm2 a-- 15.00g 15.00g A
```

The following command creates a 10 gigabyte mirrored volume from the volume group **taft**.

```
# lvcreate --type raid1 -m 1 -n mirror --nosync -L 10G taft
WARNING: New raid1 won't be synchronised. Don't read what you didn't write!
Logical volume "mirror" created
```

The following command shows which devices are used for the mirror legs and RAID metadata subvolumes.

```
# lvs -a -o +devices
LV      VG  Attr      LSize Log Cpy%Sync Devices
mirror   taft Rwi-a-r--- 10.00g 100.00 mirror_rimage_0(0),mirror_rimage_1(0)
[mirror_rimage_0] taft iwi-aor--- 10.00g /dev/sdb1(1)
[mirror_rimage_1] taft iwi-aor--- 10.00g /dev/sdc1(1)
[mirror_rmeta_0] taft ewi-aor--- 4.00m /dev/sdb1(0)
[mirror_rmeta_1] taft ewi-aor--- 4.00m /dev/sdc1(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.

```
# lvextend --alloc cling -L +10G taft/mirror
Extending 2 mirror images.
Extending logical volume mirror to 20.00 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.

```
# lvs -a -o +devices
LV      VG  Attr      LSize Log Cpy%Sync Devices
mirror   taft Rwi-a-r--- 20.00g 100.00 mirror_rimage_0(0),mirror_rimage_1(0)
[mirror_rimage_0] taft iwi-aor--- 20.00g /dev/sdb1(1)
[mirror_rimage_0] taft iwi-aor--- 20.00g /dev/sdg1(0)
[mirror_rimage_1] taft iwi-aor--- 20.00g /dev/sdc1(1)
[mirror_rimage_1] taft iwi-aor--- 20.00g /dev/sdd1(0)
[mirror_rmeta_0] taft ewi-aor--- 4.00m /dev/sdb1(0)
[mirror_rmeta_1] taft ewi-aor--- 4.00m /dev/sdc1(0)
```

