



# **Red Hat Enterprise Linux 7**

## 逻辑卷管理器管理

---

LVM 管理员指南



## LVM 管理员指南

## 法律通告

Copyright © 2015 Red Hat, Inc. and others.

This document is licensed by Red Hat under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

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.

## 摘要

本书旨在介绍 LVM 逻辑卷管理器 (logical volume manager) 信息，其中包括在集群环境中运行 LVM 的信息。

## 目录

<b>第 1 章 LVM 逻辑卷管理器</b>	3
1.1. Red Hat Enterprise Linux 7.1 中新的和更改的功能	3
1.2. 逻辑卷	3
1.3. LVM 构架概述	4
1.4. 集群逻辑卷管理器 (CLVM)	5
1.5. 文档概述	6
<b>第 2 章 LVM 组件</b>	7
2.1. 物理卷	7
2.2. 卷组	8
2.3. LVM 逻辑卷	8
<b>第 3 章 LVM 管理概述</b>	15
3.1. 在集群中创建 LVM 卷	15
3.2. 创建逻辑卷概述	16
3.3. 在逻辑卷中增大文件系统	16
3.4. 逻辑卷备份	16
3.5. 日志	17
3.6. 元数据守护进程 (lvmmetad)	17
3.7. 使用 lvm 命令显示 LVM 信息	18
<b>第 4 章 使用 CLI 命令管理 LVM</b>	19
4.1. 使用 CLI 命令	19
4.2. 物理卷管理	20
4.3. 卷组管理	23
4.4. 逻辑卷管理	29
4.5. 使用过滤器控制 LVM 设备扫描	61
4.6. 在线数据重新定位	62
4.7. 在集群的独立节点中激活逻辑卷	63
4.8. LVM 的自定义报告	63
<b>第 5 章 LVM 配置示例</b>	74
5.1. 在三个磁盘中创建 LVM 逻辑卷	74
5.2. 创建条带逻辑卷	75
5.3. 分割卷组	76
5.4. 从逻辑卷中删除磁盘	78
5.5. 在集群中创建镜像 LVM 逻辑卷	80
<b>第 6 章 LVM 故障排除</b>	84
6.1. 故障排除诊断	84
6.2. 在失败的设备中显示信息。	84
6.3. 恢复 LVM 镜像错误	85
6.4. 恢复物理卷元数据	88
6.5. 替换丢失的物理卷	89
6.6. 从卷组中删除丢失的物理卷。	89
6.7. 逻辑卷没有足够的可用扩展	90
<b>附录 A. 设备映射器 (Device Mapper)</b>	91
A.1. 设备表映射	91
A.2. dmsetup 命令	101
A.3. Device Mapper 支持 udev 设备管理器	104
<b>附录 B. LVM 配置文件</b>	107
B.1. LVM 配置文件	107

---

B.1. LVM 命令 .....	107
B.2. lvm dumpconfig 命令	107
B.3. LVM 配置文件	108
B.4. lvm.conf 文件示例	108
<b>附录 C. LVM 对象标签 .....</b>	<b>132</b>
C.1. 添加和删除对象标签	132
C.2. 主机标签	132
C.3. 使用标签控制激活	133
<b>附录 D. LVM 卷组元数据 .....</b>	<b>134</b>
D.1. 物理卷标签	134
D.2. 元数据内容	134
D.3. 元数据示例	135
<b>附录 E. 修订历史 .....</b>	<b>137</b>
<b>索引 .....</b>	<b>138</b>

# 第 1 章 LVM 逻辑卷管理器

本章概要介绍发布 Red Hat Enterprise Linux 7 后 LVM 逻辑卷管理器的新功能，同时还提供逻辑卷管理器组件的高级概述。

## 1.1. Red Hat Enterprise Linux 7.1 中新的和更改的功能

Red Hat Enterprise Linux 7.1 包括以下文档及功能更新和更改。

- » 已明确用于精简配置卷及精简配置快照的文档。有关 LVM 精简配置的附加信息，请参考 `lvmthin(7)` man page。有关精简配置卷的常规信息，请参考 [第 2.3.4 节“精简配置逻辑卷（精简卷）”](#)。有关精简配置快照卷的信息，请参考 [第 2.3.6 节“精简配置快照卷”](#)。
- » 本手册在 [第 B.2 节“lvm dumpconfig 命令”](#) 中论述 `lvm dumpconfig` 命令。
- » 本手册在 [第 B.3 节“LVM 配置文件”](#) 中论述 LVM 配置文件。
- » 本手册在 [第 3.7 节“使用 lvm 命令显示 LVM 信息”](#) 中论述 `lvm` 命令。
- » 在 Red Hat Enterprise Linux 7.1 发行本中，可以使用 `lvcreate` 和 `lvchange` 命令的 -k 和 -K 选项控制精简池快照的激活，如 [第 4.4.17 节“控制逻辑卷激活”](#) 所述。
- » 这个手册论述了 `vgimport` 命令的 `--force` 参数。可使用该参数导入缺少物理卷的卷组，并随后运行 `vgreduce --removemissing` 命令。有关 `vgimport` 命令的详情，请查看 [第 4.3.15 节“将卷组移动到其他系统”](#)。
- » 这个手册论述了 `vgreduce` 命令的 `--mirrornoonly` 参数。可使用该参数只删除从已失败物理卷镜像映象的逻辑卷。有关使用此选项的详情，请查看 [第 4.3.15 节“将卷组移动到其他系统”](#)。

另外对整篇文档进行小的技术修正及说明。

## 1.2. 逻辑卷

逻辑卷管理会根据物理存储生成提取层，以便创建逻辑存储卷。这样就比直接使用物理存储在很多方面提供了更大的灵活性。使用逻辑卷时不会受物理磁盘大小限制。另外，软件无法看到硬件存储配置，因此可在不停止应用程序或者卸载文件系统的情况下，重新定义大小并进行移动。这样可降低操作成本。

使用逻辑卷比直接使用物理存储时有以下优势：

- » 灵活的容量

当使用逻辑卷时，可在多个磁盘间扩展文件系统，因为可以将磁盘和分区集合成一个逻辑卷。

- » 可重新设定大小的存储池

可以使用简单软件命令增加或者减少逻辑卷大小，而无需对所在磁盘设备重新格式化或者重新分区。

- » 在线数据重新定位

要部署更新、更快或者更有弹性的存储子系统，以便您可以在系统活跃时移动数据。在磁盘处于使用状态时重新分配磁盘。例如，可以在删除热插拔磁盘前将其清空。

- » 方便的设备命名

可使用用户定义及自定义命名组管理逻辑存储卷。

- » 磁盘条带化

可以创建一个可在两个或者更多磁盘间条状分布数据的逻辑卷。这可大幅度提高吞吐量。

- » 镜像卷

逻辑卷为您提供了方便配置数据镜像的方法。

- » 卷快照

可使用逻辑卷提取设备快照，这样可在持续备份或者在不影响真实数据的情况下测试修改效果。

本文档的以下内容对在 LVM 中使用这些功能进行了论述。

## 1.3. LVM 构架概述

在 Linux 操作系统的 Red Hat Enterprise Linux 4 中，LVM2 替换了原来的 LVM1 逻辑卷管理器，LVM2 比 LVM1 具有更通用的内核架构。相对 LVM1，它有如下改进：

- » 灵活的容量
- » 更有效的元数据存储
- » 更好的修复格式
- » 新的 ASCII 元数据格式
- » 元数据的原子变化
- » 元数据冗余副本

LVM2 可向下兼容 LVM1，但不支持快照和集群。您可以使用 **vgconvert** 命令将卷组从 LVM1 格式转换成 LVM2 格式。有关转换 LVM 元数据格式的详情请参考 **vgconvert(8)** man page。

LVM 逻辑卷的基本物理存储单元是块设备，比如一个分区或者整张磁盘。将这个设备初始化为 LVM 物理卷 (PV)。

要创建一个 LVM 逻辑卷，就要将物理卷合并到卷组 (VG) 中。这就生成了磁盘空间池，用它可分配 LVM 逻辑卷 (LV)。这个过程和将磁盘分区的过程类似。逻辑卷由文件系统和应用程序（比如数据库）使用。

[图 1.1 “LVM 逻辑卷组成”](#) 演示一个简单 LVM 逻辑卷的组成：

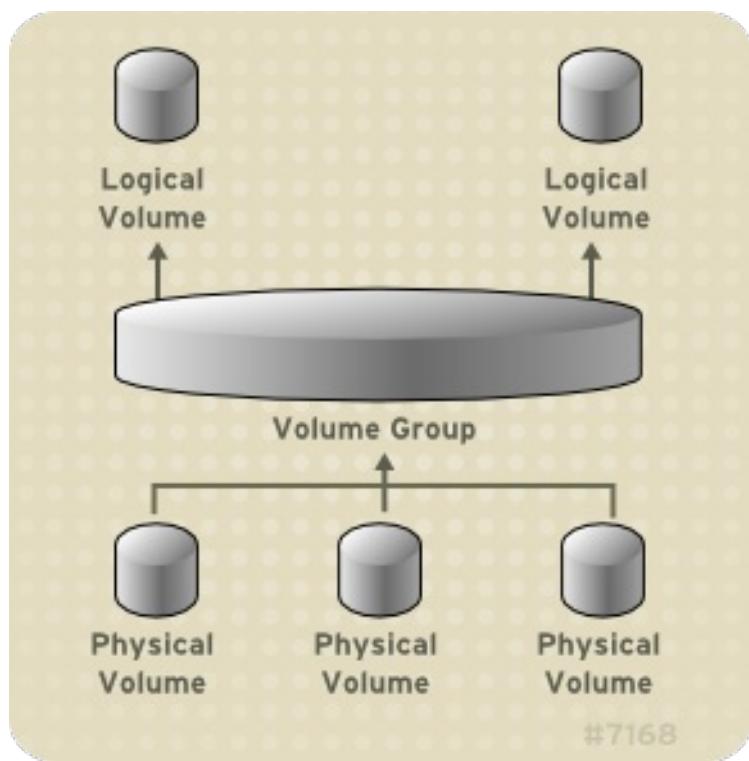


图 1.1. LVM 逻辑卷组成

有关 LVM 逻辑卷组成的详情请参考 [第 2 章 LVM 组件](#)。

## 1.4. 集群逻辑卷管理器 (CLVM)

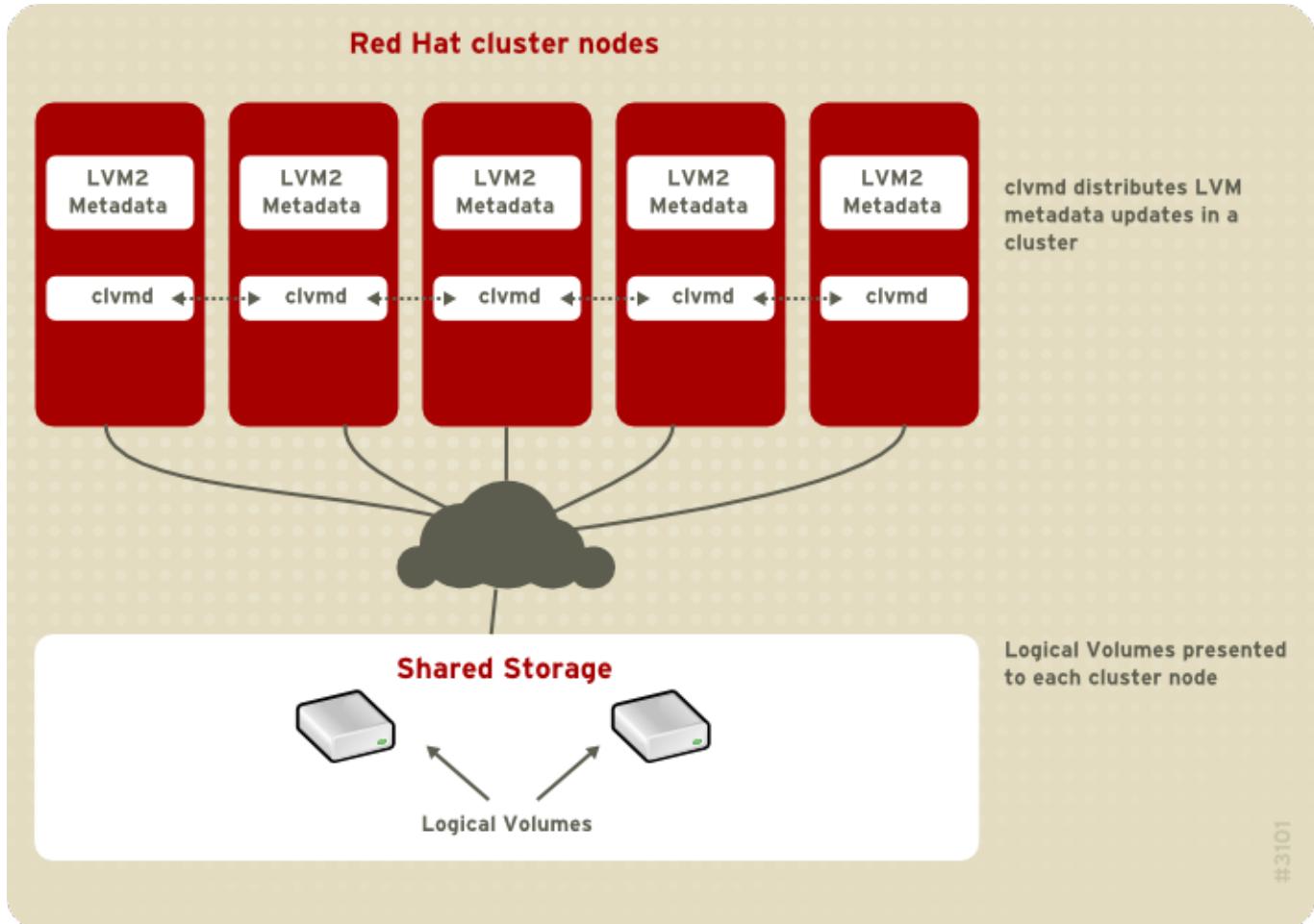
集群的逻辑卷管理器 (CLVM) 是 LVM 的一组集群扩展。这些扩展允许计算机集群使用 LVM 管理共享存储 (例如：在 SAN 中)。CLVM 是弹性存储附加组件的一部分。

是否应该使用 CLVM 取决于系统需要：

- » 如果系统中只有一个节点需要访问配置为逻辑卷的存储，则可以使用没有 CLVM 扩展的 LVM，同时那个节点创建的逻辑卷对那个节点来说均为本地卷。另外，如果要为只有访问该存储的单一节点永远处于活跃状态的故障转移使用集群的系统，也可以使用没有 CLVM 扩展的 LVM。在不需要 CLVM 扩展的集群中配置逻辑卷时，可使用 **LVM 高可用资源代理配置系统**。有关在集群中配置资源的详情，请查看《高可用附加组件参考》。
- » 如果集群中有一个以上节点需要访问由活跃节点共享的存储，则必须使用 CLVM。CLVM 可允许用户在共享的存储中配置逻辑卷，方法是在配置逻辑卷时锁定对物理存储的访问。LVM 使用集群的锁定服务管理共享存储。在需要 CLVM 扩展的集群中配置逻辑卷时，可使用 **c1vmd 资源代理配置您的系统**。有关在集群中配置资源的详情，请查看《高可用附加组件参考》。

要使用 CLVM，则必须让 High Availability Add-On 和 Resilient Storage Add-On 软件处于运行状态，包括 **c1vmd** 守护进程。**c1vmd** 守护进程是 LVM 的主要集群扩展。**c1vmd** 守护进程在每台集群计算机中运行，并在集群中分布 LVM 元数据更新，为每台集群计算机提供该逻辑卷的相同视图。

图 1.2 “[CLVM 概述](#)” 演示集群中的 CLVM 概述。



**图 1.2. CLVM 概述**

在 Red Hat Enterprise Linux 7 中是通过 Pacemaker 管理集群。只有与 Pacemaker 集群联合使用，且必须将其配置为集群资源时方支持集群的 LVM 逻辑卷。有关在集群中配额配置 LVM 卷的详情，请查看 [第 3.1 节“在集群中创建 LVM 卷”](#)。

## 1.5. 文档概述

本文档的剩余部分包括以下各章：

- » [第 2 章 LVM 组件](#) 论述组成 LVM 逻辑卷的内容。
- » [第 3 章 LVM 管理概述](#) 提供执行配置 LVM 逻辑卷的基本步骤概述。
- » [第 4 章 使用 CLI 命令管理 LVM](#) 总结可与 CLI 命令合用，创建和维护逻辑卷的独立管理任务。
- » [第 5 章 LVM 配置示例](#) 提供各种 LVM 配置示例。
- » [第 6 章 LVM 故障排除](#) 提供解决各种 LVM 问题的说明。
- » [附录 A, 设备映射器 \(Device Mapper\)](#) 论述 LVM 用来映射逻辑卷和物理卷的 Device Mapper。
- » [附录 B, LVM 配置文件](#) 论述 LVM 配置文件。
- » [附录 C, LVM 对象标签](#) 论述 LVM 对象标签和主机标签。
- » [附录 D, LVM 卷组元数据](#) 论述了 LVM 卷组元数据，其中包括复制 LVM 卷组元数据的示例。

# 第 2 章 LVM 组件

本章论述了 LVM 逻辑卷组件。

## 2.1. 物理卷

LVM 逻辑卷的底层物理存储单元是一个块设备，比如一个分区或整个磁盘。要在 LVM 逻辑卷中使用该设备，则必须将该设备初始化为物理卷 (PV)。将块设备初始化为物理卷会在该设备的起始扇区附近放置一个标签。

默认情况下，LVM 标签是放在第二个 512 字节扇区。可以将标签放在最开始的四个扇区之一来覆盖这个默认设置。这样就允许在必要时 LVM 卷可与其他使用这些扇区的用户共同存在。

LVM 标签可为物理设备提供正确的识别和设备排序，因为在引导系统时，设备可以任何顺序出现。LVM 标签在重新引导和整个集群中保持不变。

LVM 标签可将该设备识别为 LVM 物理卷。它包含物理卷的随机唯一识别符 (UUID)。它还以字节为单位记录块设备的大小，并记录 LVM 元数据在设备中的存储位置。

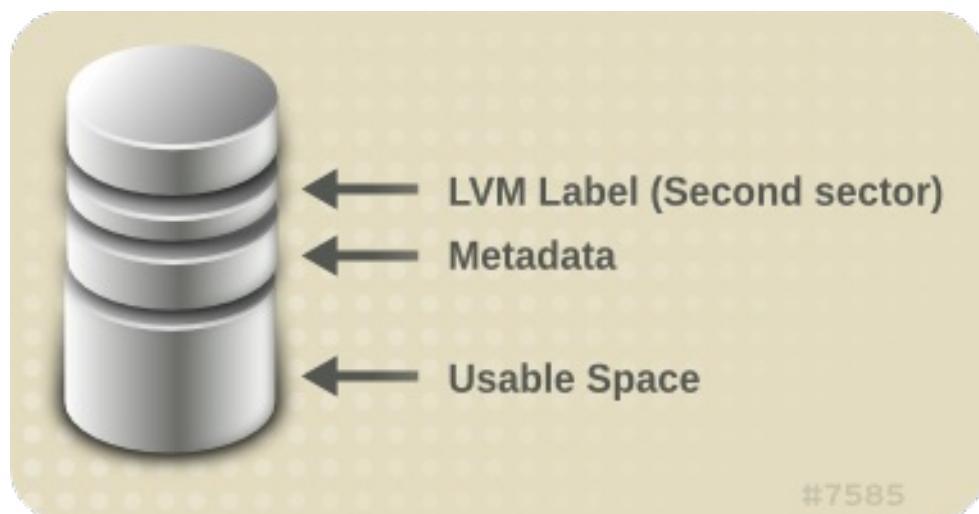
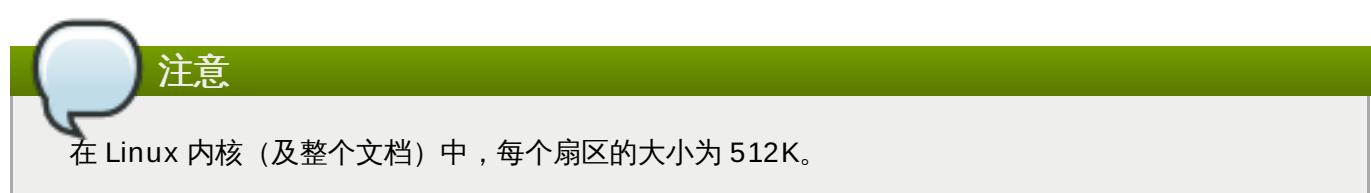
LVM 元数据包含系统中 LVM 卷组的配置详情。默认情况下，卷组中的每个物理卷中都会在其元数据区域保留一个一样的元数据副本。LVM 元数据很小，并以 ASCII 格式保存。

现在，LVM 允许在每个物理卷中保存 0、1 或者 2 个元数据副本。默认是保存一个副本。一旦设置了在物理卷中保存的元数据备份数目之后就无法再更改。第一个副本保存在设备的起始位置，紧挨着标签。如果有第二个副本，会将其放在设备的末尾。如果不小心写入了不同于想要写入的磁盘，从而覆盖了磁盘的起始部分，那么可以使用在设备末尾的元数据第二个副本恢复元数据。

有关 LVM 元数据和更改元数据参数的详情请参考 [附录 D, LVM 卷组元数据](#)。

### 2.1.1. LVM 物理卷布局

[图 2.1 “物理卷布局”](#) 显示 LVM 物理卷的布局。LVM 标签在第二个扇区，接下来是元数据区，之后是设备的可用空间。



## 图 2.1. 物理卷布局

### 2.1.2. 一个磁盘中有多个分区

LVM 允许在磁盘分区以外创建物理卷。Red Hat 通常建议创建可覆盖整张磁盘的单一分区，并将其标记为 LVM 物理卷，理由如下：

- » 方便管理

如果每个真实磁盘只出现一次，那么在系统中追踪硬件就比较容易，这在磁盘失败时尤为突出。另外，单一磁盘中有多个物理卷可导致内核在引导时发出未知分区类型警告。

- » 条带化性能

LVM 无法知道两个物理卷是否在同一物理磁盘中。如果要在两个物理卷处于同一物理磁盘中时创建条带逻辑卷，则条带可能位于同一磁盘的不同分区中。这样会降低性能而不是提升性能。

虽然不建议这样做，但可能会在某种情况下需要将磁盘分为不同的 LVM 物理卷。例如：如果一个系统中有几张磁盘，在将现有系统迁移至 LVM 卷时，可能需要在分区间移动数据。另外，如果有一个非常大的磁盘，并想要有一个以上的卷组以便管理，则需要将该磁盘分区。如果磁盘中有一个以上的分区，同时那些分区处于同一卷组中，则在创建条带卷指定逻辑卷中所包含分区时要特别小心。

## 2.2. 卷组

物理卷合并为卷组 (VG)。这样就创建了磁盘空间池，并可使用它分配逻辑卷。

在卷组中，可用来分配的磁盘空间被分为固定大小的单元，我们称之为扩展。扩展是可进行分配的最小空间单元。在物理卷中，扩展指的是物理扩展。

逻辑卷会被分配成与物理卷扩展相同大小的逻辑扩展。因此卷组中逻辑卷的扩展大小都是一样的。卷组将逻辑扩展与物理扩展匹配。

## 2.3. LVM 逻辑卷

在 LVM 中是将卷组分为逻辑卷。以下小节论述了逻辑卷的不同类型。

### 2.3.1. 线性卷

线性卷是将一个或者多个物理卷整合为一个逻辑卷。例如：如果有两个 60GB 的磁盘，则可以创建一个 120GB 的逻辑卷。其物理存储是连续的。

创建线性卷可按顺序为逻辑卷的区域分配物理扩展范围。例如：如 [图 2.2 “扩展映射”](#) 所述，逻辑扩展 1 到 99 可与一个物理卷对映，逻辑扩展 99 到 198 可与第二个物理卷对映。从应用程序的角度来看，就是有一个大小为 198 个扩展的设备。

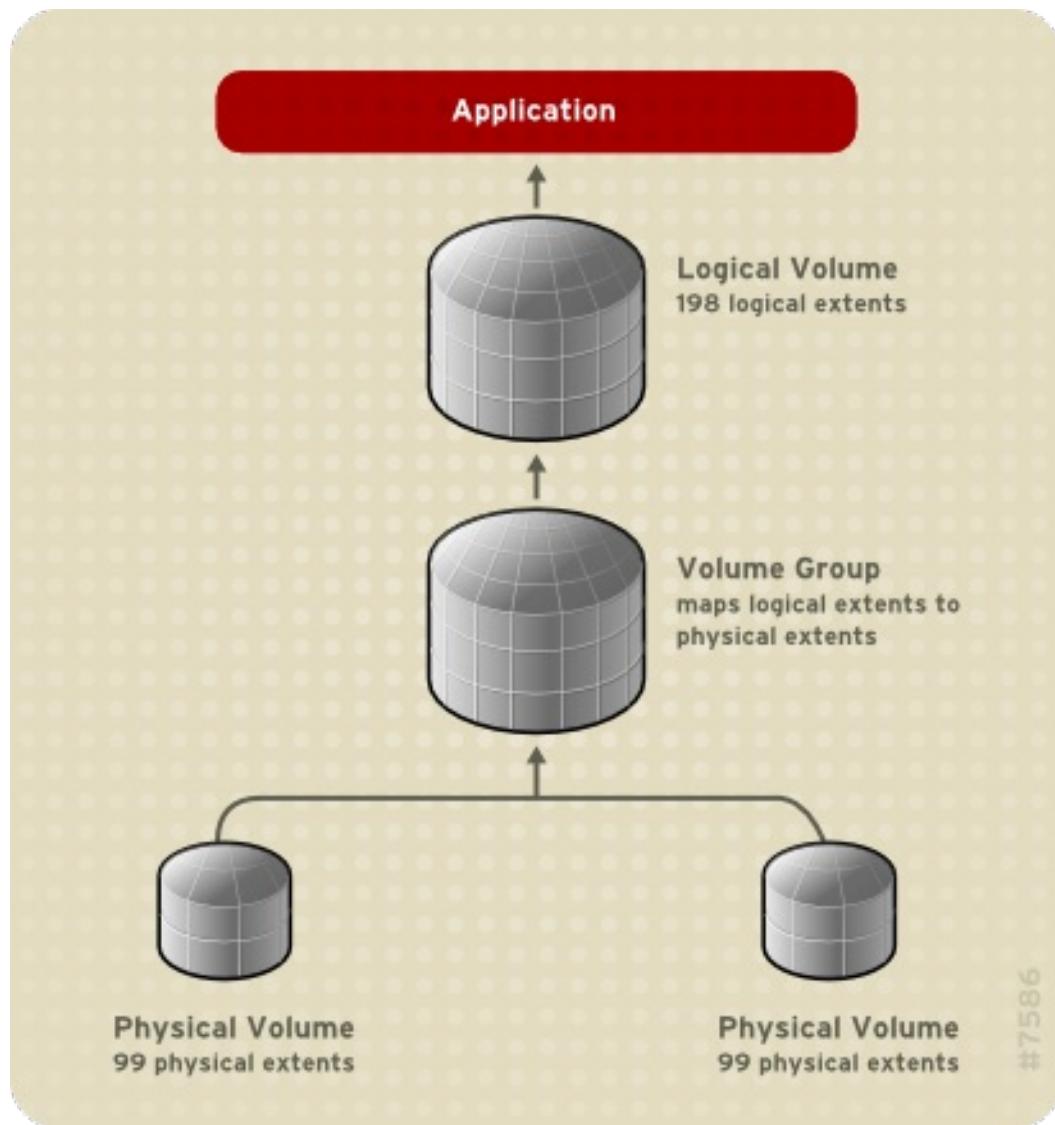


图 2.2. 扩展映射

组成逻辑卷的物理卷不一定要一样大小。[图 2.3 “物理卷大小不同的线性卷”](#) 演示了物理扩展为 4MB 的卷组 **VG1**。这个卷组包括两个物理卷，分别名为 **PV1** 和 **PV2**。该物理卷可分为 4MB 的单元，因为这是扩展的大小。在这个示例中，**PV1** 大小为 200 个扩展 (800MB)，**PV2** 大小为 100 个扩展 (400MB)。您可以创建大小在 1 到 300 个扩展 (4MB 到 1200MB) 之间的任意线性卷。在这个示例中，该线性卷名为 **LV1**，大小为 300 个扩展。

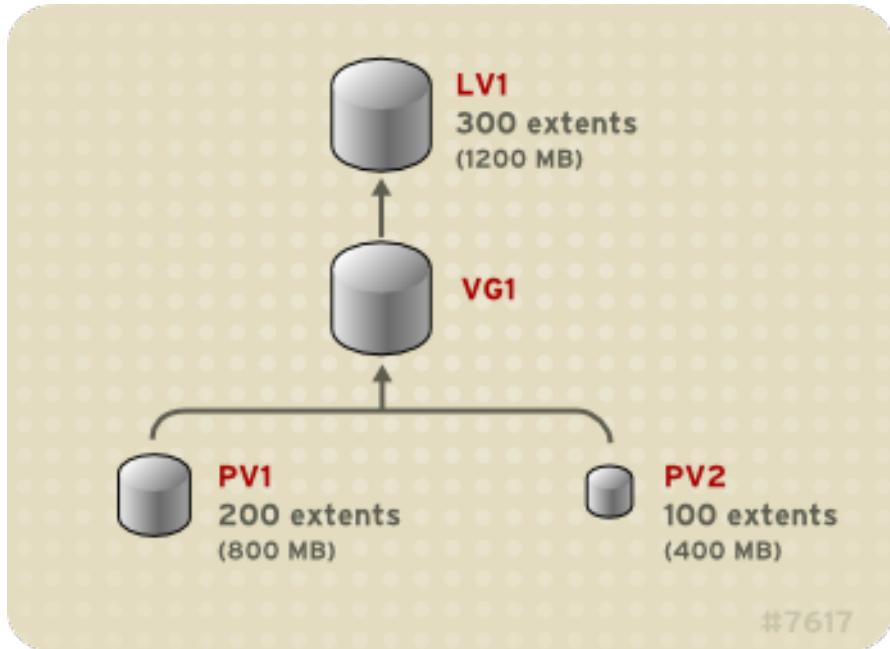


图 2.3. 物理卷大小不同的线性卷

可以使用物理扩展池创建一个以上任意大小的线性逻辑卷。[图 2.4 “多逻辑卷”](#) 演示与 [图 2.3 “物理卷大小不同的线性卷”](#) 中相同的卷组，但在这个示例中是使用卷组创建两个逻辑卷：即大小为 250 个扩展 (1000MB) 的 **LV1** 和大小为 50 个扩展 (200MB) 的 **LV2**。

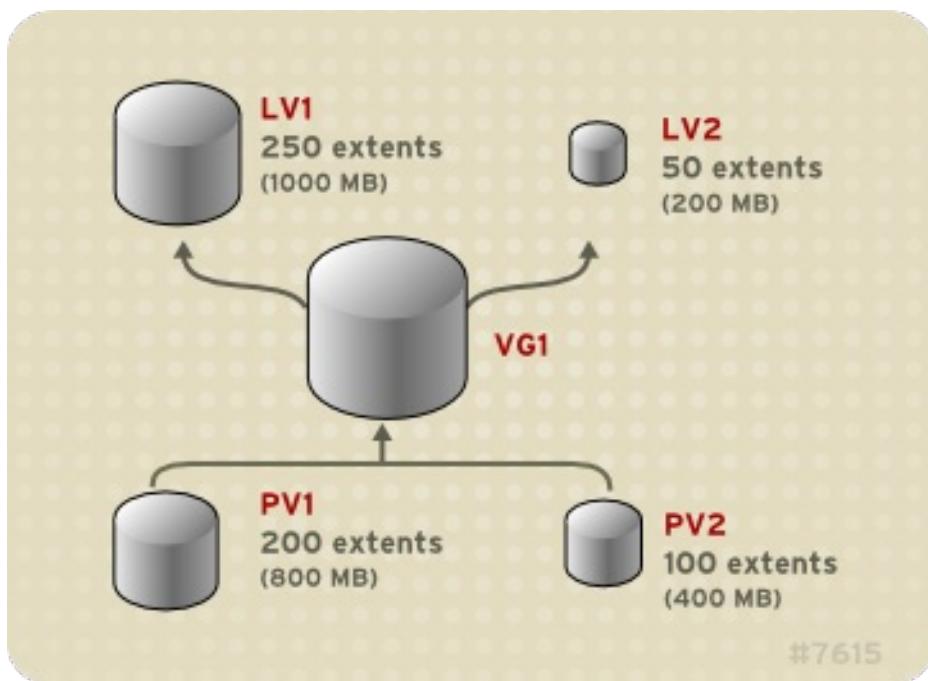


图 2.4. 多逻辑卷

### 2.3.2. 条带逻辑卷

向 LVM 逻辑卷写入数据时，文件系统在基本物理卷之间部署数据。可以通过创建条带逻辑卷控制数据向物理卷写入的方法。对于大批量的读取和写入，这样可以提高数据输入/输出的效率。

条带化数据可通过以 round-robin 模式向预定数目的物理卷写入数据来提高性能。使用条带模式，I/O 可以平行执行。在有些情况下，这样可以使条带中每个附加的物理卷获得类似线性卷的性能。

以下示例显示数据在三个物理卷之间进行条带分布。在这个图表中：

- » 在第一个物理卷中写入第一个数据条带
- » 在第二个物理卷中写入第二个数据条带
- » 在第三个物理卷中写入第三个数据条带
- » 在第四个物理卷中写入第四个数据条带

在条带逻辑卷中，条带大小不能超过扩展的大小。

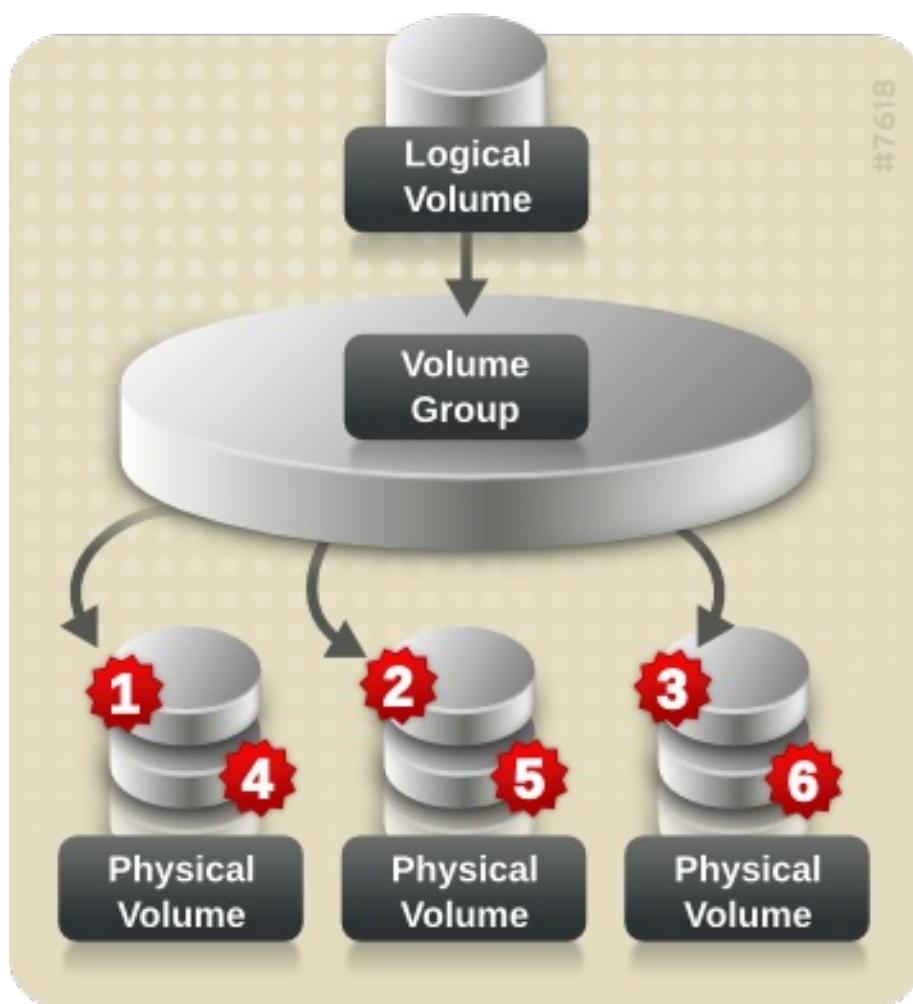


图 2.5. 跨三个物理卷的条带数据

条带逻辑卷可通过在第一组设备的末尾连接另一组设备来扩大容量。要扩展条带逻辑卷，就必须在基本物理卷中有足够的可用空间组成卷组来支持条带卷。例如：如果有一个双向条带使用了整个卷组，那么向卷组中添加单一物理卷不会允许您扩展该条带。反之，必须在卷组中添加至少两个物理卷。有关扩展条带卷的详情请参考 [第 4.4.15.1 节“扩展条带卷”](#)。

### 2.3.3. RAID 逻辑卷

LVM 支持 RAID 1/4/5/6/10。LVM RAID 卷有以下特征：

- » 通过利用 MD 内核驱动程序的 LVM 创建和管理 RAID 逻辑卷。
- » RAID 1 映象可临时从该阵列中分离，且稍后可合并回该阵列中。

- » LVM RAID 卷支持快照。

有关创建 RAID 逻辑卷的详情，请查看 [第 4.4.3 节 “RAID 逻辑卷”](#)。

### 注意

集群无法识别 RAID 逻辑卷。由于 RAID 逻辑卷可单独在机器中创建并激活，不可以同时在一台以上机器中激活。如果需要非互斥镜像卷，则必须使用 `mirror` 片段类型创建卷，如 [第 4.4.4 节 “创建镜像卷”](#) 所示。

## 2.3.4. 精简配置逻辑卷（精简卷）

逻辑卷可进行简化配置。这样可创建超出可用扩展的逻辑卷。使用精简配置，可以管理剩余空间的存储池，也称精简池。需要时应用程序可将精简池分配给任意数量的设备。然后在之后应用程序实际写入该逻辑卷时创建设备，将其绑定到精简池中。需要时可动态扩展该精简池，以便进行有效的存储空间分配。

### 注意

集群中的节点间不支持精简卷。该精简池及其所有精简卷只能以独占方式在一个集群节点中激活。

存储管理员可使用精简配置过度使用物理存储，一般是为避免购买额外的存储。例如：如果十位用户每人为其应用程序申请 100GB 文件系统，则存储管理员可为每位用户创建看似为 100GB 的文件系统，支持该文件系统的实际存储小于 100GB，且只在需要时使用。使用精简配置时，关键是存储管理员要监控存储池，并在其开始变满时开始添加更多的容量。

为确定所有可用空间均可以使用，LVM 支持数据放弃。这样可重复使用之前被已放弃文件或其他块范围使用的空间。

有关创建精简卷的详情，请查看 [第 4.4.5 节 “创建精简配置逻辑卷”](#)。

精简卷支持写时复制快照逻辑卷的新实施，该实施可允许虚拟设备共享精简池中的同一数据。有关精简快照卷的详情，请查看 [第 2.3.6 节 “精简配置快照卷”](#)。

## 2.3.5. 快照卷

LVM 的快照功能为您提供了一个特定时刻，在不导致服务中断的情况下创建设备的虚拟映射功能。在提取快照后，当对原始设备进行修改时，快照功能可生成有变化的数据区域的副本，以便重建该设备的状态。

### 注意

LVM 支持精简配置的快照。有关精简配置的快照卷的详情，请查看 [第 2.3.6 节 “精简配置快照卷”](#)。

### 注意

在集群中不支持跨节点的 LVM 快照。不能在集群的卷组中创建快照卷。

因为快照只复制在生成快照之后修改的数据区域，扩展特性需要的存储空间较小。例如，对于很少更新的原始数据，原始容量的 3-5 % 就足以进行快照维护。

## 注意

文件系统的快照副本是虚拟副本，不是文件系统的真实介质备份。快照不能替代备份过程。

预留用于保存原始卷更改的空间大小取决于快照的大小。例如：如果要生成一个快照，且要完全覆盖原始卷，则快照应该至少与原始卷同样大小方可保存更改。需要根据预期的更改程度设置快照大小。例如：对于一个多数用于读取卷的短期快照，比如 `/usr`，它所需空间要比经常会有写入操作的卷，比如 `/home` 的长期快照要小。

如果快照已满，则快照就会变得无效，因为它已经无法跟踪原始卷的更改。您应该常规监控快照的大小。快照是可以重新设定大小的，因此如果您有额外的存储容量，您可以增大快照卷容量以避免漏掉快照。相反，如果您发现快照卷超过您的需要，您可以减小它来为其它逻辑卷最大限度释放空间。

创建文件系统的快照时，仍可对原始系统有完全的读和写访问。如果修改了快照中的一个块，就会标记出那个块，并再不从原始卷中复制这个块。

快照功能有几个用途：

- » 最典型的就是，当需要在不影响那个不断更新数据系统的情况下在逻辑卷中执行备份，可以提取一个快照。
- » 可以在快照文件系统中执行 `fsck` 命令检查文件系统的完整性，并决定原始文件系统是否需要修复。
- » 因为快照是可读/写的，因此可以根据产品数据测试应用程序，方法是提取一个快照并根据快照运行测试而不接触真实数据。
- » 可创建用于 Red Hat Virtualization 的 LVM 卷。LVM 快照可用于创建虚拟机映象的快照。这些快照可提供修改现有虚拟机或使用最小附加存储创建新虚拟机的简便方式。有关使用 Red Hat Virtualization 创建基于 LVM 存储池的详情，请查看《[虚拟化管理指南](#)》。

有关创建和修改快照卷的详情请参考 [第 4.4.6 节“创建快照卷”](#)。

可使用 `lvconvert` 命令的 `--merge` 选项将快照合并到其原始卷中。如果丢失数据或文件，或需要将系统恢复到之前的状态，则可使用这个功能执行系统回退。合并快照卷后，得到的逻辑卷会使用原始卷的名称，次要版本号及合并快照的 UUID 都会被删除。有关使用此选项的详情，请查看 [第 4.4.8 节“合并快照卷”](#)。

### 2.3.6. 精简配置快照卷

Red Hat Enterprise Linux 提供精简配置的快照卷支持。精简快照卷可将很多虚拟设备储存在同一数据卷中。这样可简化管理，并允许在快照卷之间共享数据。

所有 LVM 快照卷以及所有精简快照卷均无法在集群中跨节点支持。该快照卷必须只能以独占方式在一个集群节点中激活。

精简快照卷有以下特点：

- » 精简快照卷可在有多个来自同一原始卷的快照时减少磁盘用量。
- » 如果同一源卷有多个快照卷，那么写入源卷则会造成一个 COW 操作保留数据。增加源卷的快照数应该不会造成主要速度降低。
- » 精简快照卷可作为另一个快照的逻辑卷源使用。这样就允许递归快照的任意深度（即快照的快照的快照……）。

- » 精简逻辑卷的快照还可以创建精简逻辑卷。这样在需要 COW 操作前，或者快照本身写入前不会消耗数据空间。
- » 精简快照卷不需要使用其源卷激活，因此用户只要激活源卷，则可以有很多未激活的快照卷。
- » 从精简配置的快照卷中删除源卷时，源卷的每个快照都会成为独立的精简配置卷。就是说不会将快照与其源卷合并，而是可以选择删除源卷，然后创建一个新的精简配置快照，使用那个独立卷作为源卷生成新的快照。

虽然使用精简快照卷有很多优点，但在一些情况下使用老的 LVM 快照卷功能可能更适合您的需要：

- » 无法更改精简池的区块大小。如果精简池有一个大区块（例如：1MB），且您需要一个短时快照，而那么大的区块无法有效使用，则可能会选择使用老的快照功能。
- » 无法限制精简快照卷的大小。必要时，快照会使用精简池中的所有空间。这样不太适合您的需要。

通常，在决定使用哪种快照格式时应考虑您网站的具体要求。

有关创建精简快照卷的详情，请查看 [第 4.4.7 节“创建精简配置快照卷”](#)。

### 2.3.7. 缓存卷

从 Red Hat Enterprise Linux 7.1 发行本开始，LVM 支持使用快速块设备（比如 SSD 驱动器）作为较大慢速块设备的回写或写入缓存。用户可创建缓存逻辑卷以改进其现有逻辑卷的性能，或创建由一个小且快速设备与一个大慢速设备组成的新缓存逻辑卷。

有关 LVM 缓存的详情，包括卷创建示例，请查看 **`lvmcache(7)`** man page。

# 第 3 章 LVM 管理概述

本章提供了用来配置 LVM 逻辑卷的管理流程概述，旨在提供对其所包含步骤的一般了解。有关常用 LVM 配置流程的具体步骤示例，请参考 [第 5 章 LVM 配置示例](#)。

有关用来执行 LVM 管理的 CLI 命令，请参考 [第 4 章 使用 CLI 命令管理 LVM](#)。

## 3.1. 在集群中创建 LVM 卷

可使用一组 LVM 集群扩展，即集群的逻辑卷管理器（CLVM）在集群环境中创建逻辑卷。这些扩展允许计算机集群使用 LVM 管理共享存储（例如：在 SAN 中）。

在 Red Hat Enterprise Linux 7 中使用 Pacemaker 管理集群。只有与 Pacemaker 集群联合使用方可支持集群的 LVM 逻辑卷，且必须将其配置为集群资源。

以下为您提供将集群 LVM 卷配置为集群资源时所需步骤概述。

1. 安装集群软件和 LVM 软件包，启动集群软件，并创建集群。必须为该集群配置 fencing。文档《高可用附加组件管理》中提供了创建集群，并为该集群中的每个节点配置 fencing 的示例步骤。文档《高可用附加组件管理》提供了有关集群组件配置的详情。
2. CLVM 要求为每个节点的 `/etc/lvm/lvm.conf` 文件启用集群锁定。可以作为 root 用户使用 `lvmconf --enable-cluster` 命令启用集群锁定。执行这个命令更改锁定类型并禁用 `lvmtd` 守护进程。有关 `lvmtd` 守护进程的详情，请查看 [第 3.6 节“元数据守护进程（lvmtd）”](#)。  
有关手动配置 `lvm.conf` 文件支持集群锁定的详情，请查看 `lvm.conf` 文件本身的内容。有关 `lvm.conf` 文件的详情，请查看 [附录 B, LVM 配置文件](#)。
3. 为集群设定 `d1m` 资源。可将该资源作为克隆资源创建，以便其可以在该集群的每个节点中运行。

```
# pcs resource create d1m ocf:pacemaker:controld op monitor
interval=30s on-fail=fence clone interleave=true ordered=true
```

4. 将 `clvmd` 配置为集群资源。与 `d1m` 资源一样，可将其配置为克隆的资源，以便在集群的所有节点中运行。

```
# pcs resource create clvmd ocf:heartbeat:clvm op monitor
interval=30s on-fail=fence clone interleave=true ordered=true
```

5. 设定 `clvmd` 和 `d1m` 的相依性及启动顺序。`clvmd` 必须在 `d1m` 之后启动，且必须与 `d1m` 在同一节点中运行。

```
# pcs constraint order start d1m-clone then clvmd-clone
# pcs constraint colocation add clvmd-clone with d1m-clone
```

6. 创建集群的逻辑卷。在集群环境中创建 LVM 逻辑卷和在单一节点创建 LVM 逻辑卷是一样的。LVM 命令本身没有什么不同。要启用您在集群中创建的 LVM 卷，集群构架必须正在运行且集群必须可以仲裁。

默认情况下，在所有可访问共享存储的计算机中都可看到在共享存储中使用 CLVM 创建的逻辑卷。但也可能只有从集群中的某一个节点才可看到存储设备中创建逻辑卷。还可将逻辑卷状态从本地卷改为集群卷。有关详情请参考 [第 4.3.3 节“在集群中创建卷组”](#) 以及 [第 4.3.8 节“更改卷组参数”](#)。



## 警告

使用 CLVM 在共享存储中创建卷组时，必须确定该集群中的所有节点都可访问组成该卷组的物理卷。不对称的集群配置可导致有些节点可访问该存储而有些则不能。

在集群中创建镜像逻辑卷的示例请参考 [第 5.5 节 “在集群中创建镜像 LVM 逻辑卷”](#)。

## 3.2. 创建逻辑卷概述

以下总结了创建 LVM 逻辑卷的步骤。

1. 将要用作 LVM 卷的分区初始化为物理卷（这样可标记它们）。
2. 创建卷组。
3. 创建逻辑卷。

创建逻辑卷后，可以生成并挂载该文件系统。本文档示例使用的是 GFS2 文件系统。

1. 在逻辑卷中用 **gfs\_mkfs2** 创建 GFS2 文件系统。
2. 使用 **mkdir** 命令创建一个新的挂载点。在集群的系统中，在集群的所有节点中创建挂载点。
3. 挂载文件系统。需要在 **fstab** 为系统中的每个节点添加一行。

另外，可以使用 LVM GUI 创建并挂载 GFS2 文件系统。



## 注意

虽然可在独立系统中部署 GFS2 文件系统，也可将其作为集群配置的一部分，但在 Red Hat Enterprise Linux 7 中不支持将 GFS2 作为独立文件系统使用。Red Hat 将继续为集群文件系统挂载快照支持单一节点 GFS2 文件系统（例如：用于备份）。

创建 LVM 卷在每台机器上都是不同的，因为保存 LVM 设置信息的区域是在物理卷中，而不是在创建该卷的机器中。使用存储的服务器有本地副本，但可使用物理卷中的内容重新生成。如果 LVM 版本兼容，则可以将物理卷附加到不同服务器中。

## 3.3. 在逻辑卷中增大文件系统

要在逻辑卷中增大文件系统，请按以下步骤执行：

1. 创建新物理卷。
2. 扩展带有您想要增大的文件系统逻辑卷的卷组，使其包含新的物理卷。
3. 扩展逻辑卷使其包含新的物理卷。
4. 增大文件系统。

如果卷组中有足够的未分配空间，则可以使用那些空间来扩展逻辑卷，而不是执行步骤 1 和 2。

## 3.4. 逻辑卷备份

元数据备份和归档会在每次修改卷组和逻辑卷配置时自动进行，除非在 **lvm.conf** 文件中禁用了此功能。默认情况下，元数据备份保存在 **/etc/lvm/backup** 文件中，元数据归档保存在 **/etc/lvm/archive** 文件中。元数据归档在 **/etc/lvm/archive** 文件中保存的时间和多少取决于您在 **lvm.conf** 文件中设定的参数。日常系统备份应该在备份中包含 **/etc/lvm** 目录的内容。

注意：元数据备份并不包含逻辑卷中的用户和系统数据。

可以手动使用 **vgcfgbackup** 命令将元数据备份到 **/etc/lvm/backup** 文件中。可以使用 **vgcfgrestore** 恢复元数据。有关 **vgcfgbackup** 和 **vgcfgrestore** 命令的论述请参考 [第4.3.13节“备份卷组元数据”](#)。

## 3.5. 日志

所有信息输出都是通过日志模块传递，日志模式根据日志级别有不同的选择：

- » 标准输出/错误
- » 系统日志
- » 日志文件
- » 外部日志功能

在 **/etc/lvm/lvm.conf** 中设定日志级别，有关详情请参考 [附录B, LVM配置文件](#)。

## 3.6. 元数据守护进程 (**lvmetad**)

LVM 可选择性使用中央元数据缓存，通过守护进程 (**lvmetad**) 和 **udev** 规则实施。该元数据守护进程有两个主要目的：提高 LVM 命令性能，同时允许 **udev** 自动激活逻辑卷或整个卷组使其在该系统中可用。

将 LVM 配置为通过在 **lvm.conf** 配置文件中将 **global/use\_lvmetad** 变量设定为 1 使用该守护进程。这是默认值。有关 **lvm.conf** 配置文件的详情请参考 [附录B, LVM配置文件](#)。

### 注意

目前还不跨集群节点支持 **lvmetad** 守护进程，同时要求锁定类型为本地基于文件的锁定。使用 **lvmconf --enable-cluster/--disable-cluster** 命令时，会正确配置 **lvm.conf** 文件，包括 **use\_lvmetad** 设置 (**locking\_type=3** 应为 0)。

如果将 **use\_lvmetad** 的值从 1 改为 0，则必须重启或使用以下命令手动停止 **lvmetad** 服务：

```
# systemctl stop lvm2-lvmetad.service
```

通常每个 LVM 命令执行一次磁盘扫描查找所有相关物理卷，并读取卷组元数据。但如果元数据守护进程正在运行且已启动，则可跳过这个耗时的扫描。同时 **lvmetad** 守护进程在其可用时，会根据 **udev** 规则对每个设备只扫描一次。这样可节省大量 I/O，同时减少完成 LVM 操作所需时间，尤其是对有很多磁盘的系统。

当在运行时有新卷组可用时（例如通过热插拔或者 iSCSI），则不必激活其逻辑卷方可使用。启用 **lvmetad** 守护进程后，可使用 **lvm.conf** 配置文件中的 **activation/auto\_activation\_volume\_list** 选项配置一系列可自动激活的卷组和（/或者）逻辑卷。如果没有 **lvmetad** 守护进程，则需要手动进行该操作。



## 注意

运行 `lvmtd` 守护进程时，如果执行 `pvscan --cache device` 命令，则不会应用 `/etc/lvm/lvm.conf` 文件中的 `filter =` 设置。要过滤设备，则需要使用 `global_filter =` 设置。LVM 不会打开无法通过全局过滤的设备，也永远不会对其进行扫描。您可能需要使用全局过滤器，例如：当中 VM 中使用 LVM 设备，但不想要物理主机扫描 VM 中的设备内容。

## 3.7. 使用 `lvm` 命令显示 LVM 信息

`lvm` 命令提供几个可用来显示 LVM 支持和配置的内置选项。

- » **`lvm dumpconfig`**

显示载入 `/etc/lvm/lvm.conf` 文件及其他任何配置文件后的 LVM 配置信息。有关 LVM 配置文件的详情，请查看 [附录 B, LVM 配置文件](#)。

- » **`lvm devtypes`**

显示可识别的内建块设备类型（Red Hat Enterprise Linux 发行本 6.6 及之后的产品）。

- » **`lvm formats`**

显示可识别元数据格式。

- » **`lvm help`**

显示 LVM 帮助信息。

- » **`lvm segtypes`**

显示可识别逻辑卷片段类型。

- » **`lvm tags`**

显示这个主机中定义的标签。有关 LVM 对象标签的详情，请查看 [附录 C, LVM 对象标签](#)。

- » **`lvm version`**

显示当前版本信息。

# 第 4 章 使用 CLI 命令管理 LVM

本章总结了可使用 LVM 命令行界面 (CLI) 创建和维护逻辑卷的独立管理任务。

## 注意

如果要为集群环境生成或者修改 LVM 卷，则必须确定正在运行 **c1vmd** 守护进程。有关详情请参考[第 3.1 节“在集群中创建 LVM 卷”](#)。

除 LVM 命令行界面 (CLI) 外，还可以使用系统存储管理器 (System Storage Manager , SSM) 配置 LVM 逻辑卷。有关将 SSM 与 LVM 合用的详情，请查看《[存储管理指南](#)》。

## 4.1. 使用 CLI 命令

LVM CLI 命令有一些常规功能。

命令行参数中需要指定大小时，可以明确指定单位。如果不指定单位，那么就使用默认的 KB 或者 MB。LVM CLI 不接受分数。

LVM 指定单位时要无需区分大小写，比如 M 或者 m 的效果是一样的，例如 2 的乘方（乘 1024）。但是，在某个命令中指定 **--units** 参数时，小写表示该单位乘 1024，而大写表示该单位乘 1000。

在使用卷组或者逻辑卷名称作为命令参数时，完整路径名称是可选项。可将卷组 **vg0** 中的逻辑卷 **lvol0** 指定为 **vg0/lvol0**。如果要求列出卷组列表，但却为空白，则会使用所有卷组列表替换。如果需要列出逻辑卷列表，但给出的是卷组，则会使用该卷组中的所有逻辑卷列表替换。例如：**lvdisplay vg0** 命令将显示卷组 **vg0** 中的所有逻辑卷。

所有 LVM 命令都接受 **-v** 参数，多输入几次 v 可提高输出的详细程度。例如：以下示例显示的是 **lvcreate** 命令的默认输出。

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

下面是使用 **-v** 参数的 **lvcreate** 命令输出结果。

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

还可以使用 **-vv**、**-vvv** 或者 **-vvvv** 参数提高命令执行的详细程度。**-vvvv** 参数可以提供最多的信息。以下是 **lvcreate** 命令使用 **-vvvv** 参数时输出结果的前几行。

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

可以用命令的 **--help** 参数来显示任意 LVM CLI 命令的帮助信息。

```
# commandname --help
```

要显示某个命令的 man page，请执行 **man** 命令：

```
# man commandname
```

**man lvm** 命令提供有关 LVM 的常规在线信息。

所有 LVM 对象均使用创建该对象时分配的 UUID 作为内部参考。这在删除作为卷组一部分的名为 **/dev/sdf** 的物理卷时很有用，因为将其插回后，它的名称会变为 **/dev/sdk**。LVM 仍可以找到该物理卷，因为它是根据其 UUID 而不是其设备名称识别。有关创建物理卷时指定物理卷 UUID 详情的信息，请查看 [第 6.4 节“恢复物理卷元数据”](#)。

## 4.2. 物理卷管理

本小节论述了管理物理卷不同方面的命令。

### 4.2.1. 创建物理卷

以下小节描述了创建物理卷时使用的命令。

#### 4.2.1.1. 设定分区类型

如果您的物理卷使用整张磁盘设备，该磁盘不得有任何分区表。如果使用 DOS 磁盘分区，则应使用 **fdisk** 或者 **cfdisk** 命令或对等的命令将分区 id 设定为 0x8e。如果使用整个磁盘设备，则只需要删除分区表，即彻底破坏该磁盘中的数据即可。可使用以下命令将第一扇区归零，从而删除现有分区：

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

#### 4.2.1.2. 初始化物理卷

使用 **pvccreate** 命令初始化要作为物理卷使用的块设备。初始化是模拟格式化文件系统。

以下命令将 **/dev/sdd**、**/dev/sde** 和 **/dev/sdf** 作为 LVM 物理卷初始化，以供之后成为 LVM 逻辑卷的一部分使用。

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

若只是初始化分区而不是整张磁盘，则应在该分区中运行 **pvcreate** 命令。以下示例将分区 **/dev/hdb1** 作为 LVM 物理卷初始化，以供之后成为 LVM 逻辑卷的一部分使用。

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

#### 4.2.1.3. 扫描块设备

可使用 **lvmdiskscan** 命令扫描要作为物理卷使用的块设备，如下所示。

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

#### 4.2.2. 显示物理卷

可使用以下三个命令显示 LVM 物理卷属性：**pvs**、**pvdisplay** 和 **pvscan**。

**pvs** 命令以可配置的格式提供物理卷信息，每行显示一个物理卷。**pvs** 命令提供大量格式控制，供脚本使用。有关使用 **pvs** 命令自定义输出结果的详情，请查看 [第 4.8 节“LVM 的自定义报告”](#)。

**pvdisplay** 命令为每个物理卷提供详细的多行输出结果。它以固定格式显示物理属性（大小、扩展、卷组等）。

以下是在单一物理卷中使用 **pvdisplay** 命令显示的输出结果示例。

```
# pvdisplay
--- Physical volume ---
PV Name              /dev/sdc1
VG Name              new_vg
PV Size              17.14 GB / not usable 3.40 MB
Allocatable          yes
PE Size (KByte)     4096
Total PE             4388
Free PE              4375
Allocated PE         13
PV UUID              Joqlch-yWSj-kuEn-IDwM-01S9-X08M-mcpsVe
```

**pvscan** 命令在系统中为物理卷扫描所有支持的 LVM 块设备。

下面的命令显示所有找到的物理设备：

```
# 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    VG vg0    lvm2 [964.84 MB]
Total: 3 [2.83 GB] / in use: 2 [1.88 GB] / in no VG: 1 [964.84 MB]
```

可在 **lvm.conf** 文件中定义过滤器，以便这个命令可以避免扫描特定物理卷。有关使用过滤器控制要扫描设备的详情，请查看 [第 4.5 节“使用过滤器控制 LVM 设备扫描”](#)。

### 4.2.3. 防止在物理卷中分配

可使用 **pvchange** 命令防止在一个或多个物理卷的剩余空间中分配物理扩展。如果有磁盘错误或要删除该物理卷时需要这个操作。

以下命令不允许在 **/dev/sd1** 中分配物理扩展。

```
# pvchange -x n /dev/sd1
```

还可使用 **pvchange** 命令的 **-xy** 参数允许分配之间已禁止分配的扩展。

### 4.2.4. 重新定义物理卷大小

若出于任何原因需要更改底层块设备的大小，请使用 **pvresize** 命令使用新大小更新 LVM。可在 LVM 使用物理卷的同时执行这个命令。

### 4.2.5. 删除物理卷

如果 LVM 不再需要使用某个设备，则可使用 **pvremove** 命令删除 LVM 标签。执行 **pvremove** 命令将空白物理卷中的 LVM 元数据归零。

如果要删除的物理卷目前是某个卷组的一部分，则必须使用 **vgreduce** 命令将其从卷组中删除，如 [第 4.3.7 节“从卷组中删除物理卷”](#) 所述。

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

## 4.3. 卷组管理

本小节论述了执行各种卷组管理的命令。

### 4.3.1. 创建卷组

请使用 **vgcreate** 命令为一个或多个物理卷创建卷组。**vgcreate** 命令根据名称新建卷组，并至少在其中添加一个物理卷。

以下命令创建名为 **vg1**，包含物理卷 **/dev/sdd1** 和 **/dev/sde1** 的卷组。

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

使用物理卷创建卷组时，默认将其磁盘空间分成大小为 4MB 的扩展。这个扩展是逻辑卷增大或减小的最小的量。大的扩展数目不会影响逻辑卷的 I/O 性能。

如果默认扩展大小不适合您，可使用 **vgcreate** 命令的 **-s** 选项指定扩展大小。可以使用 **vgcreate** 命令的 **-p** 或 **-l** 参数设定物理卷或逻辑卷数量限制。

默认情况下，卷组会根据常识规则分配物理扩展，比如不要在同一物理卷中放置平行条带。这是 **normal** 分配策略。可以使用 **vgcreate** 命令的 **--alloc** 参数指定 **contiguous**、**anywhere** 或者 **cling** 策略。通常只在特殊情况下才会需要 **normal** 以外的分配策略，比如或指定非常规或非标准扩展。有关 LVM 如何分配物理扩展的详情，请查看 [第 4.3.2 节“LVM 分配”](#)。

LVM 卷组及底层逻辑卷均包含在 **/dev** 目录的设备特殊文件目录树中，其布局为：

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

例如：如果创建了两个卷组，即 **myvg1** 和 **myvg2**，每个均包含名为 **lv01**、**lv02** 和 **lv03** 的三个逻辑卷，则会生成六个设备特殊文件，即：

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

如果对应逻辑卷目前未激活，则不会显示该设备的特殊文件。

64 位 CPU 中的最大 LVM 的设备大小为 8 EB。

### 4.3.2. LVM 分配

当 LVM 操作需要为一个或多个逻辑卷分配物理扩展时，分配的步骤如下：

- » 在卷组中生成一组完整的未分配物理扩展供使用。如果在命令后结尾处提供任何物理扩展范围，则只会使用指定物理卷中符合那些范围的未分配物理扩展。
- » 将依次尝试每个分配策略，从最严格的策略（即 **contiguous**）开始，到使用 **--alloc** 选项指定的分配策略结束，也可以将具体逻辑卷或卷组设定为默认选项。在每个策略中，从需要填充的空白逻辑卷空间数字最小的逻辑扩展开始，该空间填充后，则根据分配策略限制依次填充。如果需要更多空间，则 LVM 会移至下一个策略。

分配策略限制如下：

- » **contiguous** 分配策略要求任意逻辑扩展（不是逻辑卷的第一个逻辑扩展）的物理位置紧邻它前面一个逻辑扩展的物理位置。

逻辑卷为条带或镜像时，**contiguous** 分配限制可独立应用于每个需要空间的条带或镜像（leg）。

- » **cling** 分配策略要求将要添加到现有逻辑卷中的任意逻辑扩展使用的物理卷条件到任意逻辑扩展使用的物理卷条件到之前已经由那个逻辑卷中的一个逻辑扩展使用的现有逻辑卷中。如果已定义配置参数 **allocation/ cling\_tag\_list**，则在两个物理卷中出现任何一个列出的标签时，就将这两个物理卷视为匹配。这样就可以分配为目的，标记有相似属性（比如其物理位置）的物理卷，并将其视为对等。有关与 LVM 标签一同使用 **cling** 策略指定扩展 LVM 卷时要附加物理卷的详情，请查看 [第 4.4.15.3 节“使用 cling 分配策略扩展逻辑卷”](#)。

逻辑卷为条带或镜像时，**cling** 分配限制可独立应用于每个需要空间的条带或镜像（leg）。

- » **normal** 分配策略不会在平行逻辑卷中使用相同的偏移选择与分配给该平行逻辑卷（即不同的条带或镜像映象/分支）的逻辑扩展共享同一物理卷的物理扩展。

分配镜像日志的同时分配逻辑卷以保留镜像数据时，**normal** 分配策略会首先尝试为该日志和数据选择不同类型的物理卷。如果无法实现，且将 **allocation/mirror\_logs\_require\_separate\_pvs** 配置参数设定为 0，它就会允许将部分数据记录到共享物理卷中。

同样，分配精简池元数据时，**normal** 分配策略会与考虑镜像日志分配一样根据 **allocation/thin\_pool\_metadata\_require\_separate\_pvs** 配置参数做决定。

- » 如果有足够的剩余扩展满足分配要求，但 **normal** 分配策略不会使用它们，则 **anywhere** 会使用，即使出现将两个条带放到同一物理卷，从而降低性能的情况。

可使用 **vgchange** 命令更改分配策略。

### 注意

如果根据定义的分配策略需要使用本小节外的任意布局行为，则应注意在以后的版本中可能有所变化。例如：如果在命令行中提供两个有相同可用剩余物理扩展数供分配的空物理卷，LVM 目前考虑使用每个卷以便将其列出；不保证今后的发行本仍保留这个属性。如果获取某个具体逻辑卷的特定布局很重要，则应使用 **lvcreate** 和 **lvconvert** 步骤顺序构建，以便在每个步骤中应用分配策略，让 LVM 不会在布局上自由裁量。

要查看目前在任意具体示例中的分配进度方法，可查看 debug 日志输出结果，例如在命令中添加 **-vvvv** 选项。

### 4.3.3. 在集群中创建卷组

可使用 **vgcreate** 命令在集群环境中创建卷组，这与在单一节点中创建卷组是一样的。

默认情况下，所有可访问共享存储的计算机都可以看到使用 CLVM 在共享存储中创建的卷组。但也可以 使用 **vgcreate** 命令的 **-c n** 选项创建只能在该集群的一个节点中本地看到的卷组。

在集群环境中执行以下命令可在执行该命令节点中创建本地卷组。该命令创建名为 **vg1**，包含物理卷 **/dev/sdd1** 和 **/dev/sde1** 的卷组。

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

可使用 **vgchange** 命令的 **-c** 选项指定现有卷组是本地卷组还是集群卷组，如 [第 4.3.8 节“更改卷组参数”](#) 所述。

可使用 **vgs** 命令查看现有卷组是否为集群的卷组，如果是集群的卷组，则会显示 **c** 属性。下面的命令显示卷组 **VolGroup00** 和 **testvg1** 属性。在这个示例中，**VolGroup00** 不是集群的卷组，而 **testvg1** 是集群的卷组，如 **Attr** 标题下的 **c** 属性所示。

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

有关 **vgs** 命令的详情，请查看 [第 4.3.5 节“显示卷组”](#) [第 4.8 节“LVM 的自定义报告”](#) 以及 **vgs** man page。

#### 4.3.4. 在卷组中添加物理卷

请使用 **vgextend** 命令在现有卷组中添加额外的物理卷。**vgextend** 命令通过添加一个或多个可用物理卷增大卷组容量。

下面的命令在卷组 **vg1** 中添加物理卷 **/dev/sdf1**。

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

#### 4.3.5. 显示卷组

有两个命令可显示 LVM 卷组的属性：即 **vgs** 和 **vgdisplay**。

**vgscan** 命令扫描卷组的所有磁盘，并重新构建 LVM 缓存文件，同时显示卷组。有关 **vgscan** 命令的详情，请查看 [第 4.3.6 节“为卷组扫描磁盘以构建缓存文件”](#)。

**vgs** 命令使用可配置格式提供卷组信息，每行表示一个卷组。**vgs** 命令提供大量的格式控制，有助于脚本编写。有关使用 **vgs** 命令自定义输出结果的详情，请查看 [第 4.8 节“LVM 的自定义报告”](#)。

**vgdisplay** 命令采用固定格式显示卷组属性（比如大小、扩展、物理卷数等等）。下面的示例显示使用 **vgdisplay** 命令显示卷组 **new\_vg** 的输出结果。如果无法指定卷组，则会显示当前所有卷组。

```
# 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
Cur LV
Open LV
Max PV
Cur PV
Act PV
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-ZKk0-0pM0-0118-nlw0-wwqd-fD5D32
```

### 4.3.6. 为卷组扫描磁盘以构建缓存文件

**vgscan** 命令扫描系统中的所有支持的磁盘设备以查找 LVM 物理卷和卷组。这样可在 `/etc/lvm/cache/.cache` 文件中构建缓存，以保持当前 LVM 设备列表。

LVM 会在系统启动及执行 LVM 操作时自动运行 **vgscan** 命令，比如当您执行 **vgcreate** 命令或 LVM 探测到非一致性时。

#### 注意

更改硬件配置以及在某个节点中添加或删除设备时，可能需要手动运行 **vgscan** 命令，以便系统可以识别系统启动时不存在的新设备。这是必需的，例如当在 SAN 中的系统中添加新磁盘，或热插拔标记为物理卷的新磁盘时。

可在 `lvm.conf` 文件中定义过滤器进行限制以避免扫描具体设备。有关使用过滤器限制要扫描设备的详情，请查看 [第 4.5 节“使用过滤器控制 LVM 设备扫描”](#)。

以下是 **vgscan** 命令的输出结果示例。

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

### 4.3.7. 从卷组中删除物理卷

使用 **vgreduce** 命令从卷组中删除未使用的物理卷。**vgreduce** 命令通过删除一个或多个空物理卷来缩小卷组容量。这样可释放那些不同卷组中要使用的物理卷，或者将其从该系统中删除。

从卷组中删除物理卷前，可使用 **pvdisplay** 命令确定任何逻辑卷都没有使用该物理卷。

```
# pvdisplay /dev/hda1
-- Physical volume --
PV Name          /dev/hda1
VG Name          myvg
PV Size          1.95 GB / NOT usable 4 MB [LVM: 122 KB]
PV#              1
PV Status        available
Allocatable      yes (but full)
Cur LV           1
PE Size (KByte) 4096
Total PE         499
Free PE          0
Allocated PE     499
PV UUID          Sd44tK-9IRw-SrMC-M0kn-76iP-ifz-0VSe7
```

如果该物理卷仍被使用，则必须使用 **pvmove** 命令将该数据迁移到另一个物理卷中。然后使用 **vgreduce** 命令删除该物理卷。

下面的命令从卷组 `my_volume_group` 中删除物理卷 `/dev/hda1`。

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

如果逻辑卷包含失败的物理卷，则不能使用那个逻辑卷。要从卷组中删除物理卷，如果没有在缺少的物理卷中分配逻辑卷，则可以使用 **vgreduce** 命令的 **--removemissing** 参数。

如果失败的物理卷包含 **mirror** 片段类型的逻辑卷镜像映像，则可以使用 **vgreduce --removemissing --mirrorsonly --force** 命令从镜像中删除该映像。这样只会删除该物理卷映像镜像的逻辑卷。

有关从 LVM 镜像失败中进行恢复的详情，请查看 [第 6.3 节“恢复 LVM 镜像错误”](#)。有关从卷组中删除丢失的物理卷的详情，请查看 [第 6.6 节“从卷组中删除丢失的物理卷”](#)。

### 4.3.8. 更改卷组参数

**vgchange** 命令是用来停用和激活卷组，如 [第 4.3.9 节“激活和停用卷组”](#) 所述。还可以使用这个命令为现有卷组更改几个卷组参数。

下面的命令将卷组 **vg00** 的最大逻辑卷数改为 128。

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

有关可使用 **vgchange** 命令更改卷组参数的论述，请查看 **vgchange(8)** man page。

### 4.3.9. 激活和停用卷组

创建卷组时，默认为激活状态。就是说可访问该组中的逻辑卷，并可进行更改。

在有些情况下需要让卷组处于不活跃状态，并因此无法被内核所识别。要停用或激活卷组，则请使用 **vgchange** 命令的 **-a** (**--available**) 参数。

下面的示例停用卷组 **my\_volume\_group**。

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

如果启用集群的锁定，添加 ‘e’ 只在一个节点中激活或停用卷组；而添加 ‘l’ 则只可在本地节点中激活或停用卷组。使用单主机快照的逻辑卷总是独占激活，因为一次只能在一个节点中使用它们。

可使用 **lvchange** 命令停用独立逻辑卷，如 [第 4.4.11 节“更改逻辑卷组的参数”](#) 所述。有关在集群内的独立节点中激活逻辑卷的详情，请查看 [第 4.7 节“在集群的独立节点中激活逻辑卷”](#)。

### 4.3.10. 删除卷组

请使用 **vgremove** 命令删除不包含逻辑卷的卷组。

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

### 4.3.11. 分割卷组

请使用 **vgsplit** 命令拆分卷组的物理卷，并创建新卷组。

不能在卷组间拆分逻辑卷。每个现有逻辑卷必须全部位于物理卷中，构成旧的或新的卷组。但必要时，也可使用 **pvmove** 命令强制拆分。

下面的示例从卷组 **bigvg** 中拆分新卷组 **smallvg**。

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

### 4.3.12. 组合卷组

可使用 **vgmerge** 命令将两个卷组组合为一个卷组。如果卷的物理扩展大小相等，且两个卷组的物理卷和逻辑卷概述均在目的卷组的限制范围内，则可将不活跃“源”卷与活跃或不活跃“目的”卷合并。

下面的命令将不活跃卷组 **my\_vg** 与活跃或不活跃卷组 **databases** 合并，并给出详细运行时信息。

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

### 4.3.13. 备份卷组元数据

每次卷组和逻辑卷配置更改后都会自从生成元数据备份和归档，除非在 **lvm.conf** 文件中禁用了此功能。默认情况下是在 **/etc/lvm/backup** 文件中保存元数据备份，在 **/etc/lvm/archive** 文件中保存元数据归档。可使用 **vgcfgbackup** 命令手动将元数据备份到 **/etc/lvm/backup** 文件中。

**vgcfgrestore** 命令使用归档在所有物理卷中恢复卷组元数据。

有关使用 **vgcfgrestore** 命令恢复物理卷元数据的详情，请查看 [第 6.4 节“恢复物理卷元数据”](#)。

### 4.3.14. 重新命名卷组

使用 **vgrename** 命令重命名现有卷组。

使用下面的命令将现有卷组 **vg02** 重命名为 **my\_volume\_group**：

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

```
# vgrename vg02 my_volume_group
```

### 4.3.15. 将卷组移动到其他系统

可将 LVM 卷组移动到另一个系统。建议使用 **vgexport** 和 **vgimport** 命令进行操作。



#### 注意

可使用 **vgimport** 命令的 **--force** 参数。这样可导入缺少物理卷的卷组，并随后运行 **vgreduce --removemissing** 命令。

**vgexport** 可让不活跃的卷组服务访问该系统，以便拆离其物理卷。**vgimport** 命令可让机器在使用 **vgexport** 命令时期不活跃后重新可以访问该卷组。

请执行以下步骤将卷组从一个系统移动到另一个系统：

1. 确定没有用户访问卷组活跃卷中的文件，然后卸载该逻辑卷。
2. 使用 **vgchange** 命令的 **-a n** 参数将该卷组标记为不活跃，这样可防止卷组的进一步活动。
3. 使用 **vgexport** 命令导出卷组。这样可防止要从中删除卷组的系统访问该卷组。

导出卷组后，执行 **pvscan** 命令时，会在导出的卷组中显示该物理卷，如下面的示例所示。

```
# pvs
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]
...
```

下次关闭系统时，可拔出组成该卷组的磁盘，并将其连接到新系统中。

4. 将磁盘插入新系统后，使用 **vgimport** 命令导入卷组，以便新系统可以访问该卷组。
5. 使用 **vgchange** 命令的 **-a y** 参数激活卷组。
6. 挂载该文件系统使其可用。

#### 4.3.16. 重新创建卷组目录

请使用 **vgmknodes** 命令重新创建卷组和逻辑卷特殊文件。这个命令检查 **/dev** 目录中用来激活逻辑卷的 LVM2 特殊文件。它会创建所有缺少的特殊文件，并删除不使用的文件。

可在 **vgscan** 命令中指定 **mknodes** 参数将 **vgmknodes** 命令整合至 **vgscan** 命令。

### 4.4. 逻辑卷管理

本小节论述执行各方面逻辑卷管理的命令。

#### 4.4.1. 创建线性逻辑卷

请使用 **lvcreate** 命令创建逻辑卷。如果没有为该逻辑卷指定名称，则默认使用 **lvol#**，其中 # 是逻辑卷的内部编号。

创建逻辑卷后，则会从组成卷组之物理卷中的剩余扩展中分出逻辑卷。通常逻辑卷会用完底层物理卷中的所有可用空间。修改该逻辑卷可释放并重新分配物理卷中的空间。

下面的命令在卷组 **vg1** 中创建大小为 10GB 的逻辑卷。

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

下面的命令创建大小为 1500MB，名为 **testlv** 的线性逻辑卷，该卷位于卷组 **testvg** 中，创建块设备 **/dev/testvg/testlv**。

```
# lvcreate -L 1500 -n testlv testvg
```

下面的命令使用卷组 **vg0** 中的剩余扩展创建名为 **gfs1v**，大小为 50GB 的逻辑卷。

```
# lvcreate -L 50G -n gfs1v vg0
```

可使用 **lvcreate** 命令的 **-l** 参数，以范围为单位指定逻辑卷大小。还可以使用这个参数指定用于该逻辑卷的卷组百分比。下面的命令创建名为 **mylv**，使用卷组 **testvg** 总空间 60% 的逻辑卷。

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

还可以使用 **lvcreate** 命令的 **-l** 参数指定卷组中剩余可用空间的百分比作为逻辑卷的大小。下面的命令创建名为 **yourlv**，使用卷组 **testvg** 中所有未分配空间的逻辑卷。

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

可使用 **lvcreate** 命令的 **-l** 参数创建使用整个卷组的逻辑卷。另一个使用整个卷组创建逻辑卷的方法是使用 **vgdisplay** 命令找到 "Total PE" 大小，并在 **lvcreate** 命令中输入那些结果。

下面的命令创建名为 **mylv** 的逻辑卷，该卷充满名为 **testvg** 的卷组。

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

如果需要删除物理卷，则用来创建逻辑卷的基层物理卷会变得很重要，因此需要考虑创建该逻辑卷的可能性。有关从卷组中删除物理卷的详情，请查看 [第 4.3.7 节“从卷组中删除物理卷”](#)。

要创建使用卷组中具体物理卷分配的逻辑卷，请在 **lvcreate** 命令行的末端指定物理卷或多个物理卷。下面的命令在卷组 **testvg** 中创建名为 **testlv** 的逻辑卷，将其分配到物理卷 **/dev/sdg1** 中。

```
# lvcreate -L 1500 -n testlv testvg /dev/sdg1
```

可指定逻辑卷使用的物理卷扩展。下面的示例中使用卷组 **testvg** 中的物理卷 **/dev/sda1** 的扩展 0 到 24 和物理卷 **/dev/sdb1** 扩展 50 到 124 创建线性逻辑卷。

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

下面的示例使用物理卷 **/dev/sda1** 的扩展 0 到 25 创建线性逻辑卷，然后继续从扩展 100 开始布设逻辑卷。

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

如何扩展逻辑卷的默认策略是采用 **inherit** 分配，它在卷组中应用相同的策略。可使用 **lvchange** 命令更改这些策略。有关分配策略的详情，请查看 [第 4.3.1 节“创建卷组”](#)。

#### 4.4.2. 创建条带卷

如果有大量连续读、写操作，创建条带逻辑卷可提高数据 I/O 的效率。有关条带卷的常规信息，请查看 [第 2.3.2 节“条带逻辑卷”](#)。

创建条带逻辑卷时，可使用 **lvcreate** 命令的 **-i** 参数指定条带数。这样就决定了逻辑卷会在多少物理卷之间形成条带。条带数不能超过该卷组中的物理卷数（除非使用 **--alloc anywhere** 参数）。

如果组成逻辑卷的底层物理设备大小不同，则最大的条带卷是由最小的底层设备决定。例如：在有两个分支的条带中，条带卷大小不能超过较小设备的两倍。在有三个分支的条带中，条带卷大小不能超过最小设备的三倍。

下面的命令在两个物理卷之间创建条带逻辑卷，条带大小为 64kB。该逻辑卷为 50GB，名为 **gfs1v**，并从中创建卷组 **vg0**。

```
# lvcreate -L 50G -i 2 -I 64 -n gfs1v vg0
```

可使用线性卷指定用于该条带的物理卷。下面的命令创建大小为 100 扩展的条带卷，该条带跨两个物理卷，名为 **stripelv**，位于卷组 **testvg** 中。该条带使用 **/dev/sda1** 的扇区 0-49，以及 **/dev/sdb1** 的扇区 50-99。

```
# lvcreate -l 100 -i 2 -n stripelv testvg /dev/sda1:0-49 /dev/sdb1:50-99
Using default stripesize 64.00 KB
Logical volume "stripelv" created
```

#### 4.4.3. RAID 逻辑卷

LVM 支持 RAID 1/4/5/6/10。

##### 注意

集群无法识别 RAID 逻辑卷。由于只可在一台机器中创建并激活 RAID 逻辑卷，因此无法在一台以上的机器中同时激活它们。如果需要独占镜像卷，则必须使用 **mirror** 片段类型创建卷，如 [第 4.4.4 节“创建镜像卷”](#) 所述。

可使用 **lvcreate** 命令的 **--type** 参数指定 raid 类型，以便创建 RAID 逻辑卷。可能的 RAID 片段类型如表 4.1 “RAID 片段类型” 所述。

表 4.1. RAID 片段类型

片段类型	描述
<b>raid1</b>	RAID1 镜像。这是在指定 <b>-m</b> 但没有指定条带时， <b>lvcreate</b> 命令 <b>--type</b> 参数的默认值。
<b>raid4</b>	RAID2 专用奇偶磁盘
<b>raid5</b>	同 <b>raid5_ls</b>
<b>raid5_la</b>	RAID5 左侧不对称。 Rotating parity 0 with data continuation
<b>raid5_ra</b>	RAID5 右侧不对称。 Rotating parity N with data continuation
<b>raid5_ls</b>	RAID5 左侧不对称。 Rotating parity 0 with data restart
<b>raid5_rs</b>	RAID5 右侧不对称。 Rotating parity N with data restart
<b>raid6</b>	同 <b>raid6_zr</b>
<b>raid6_zr</b>	RAID6 零重启 Rotating parity zero (left-to-right) with data restart
<b>raid6_nr</b>	RAID6 N 重启 Rotating parity N (left-to-right) with data restart

片段类型	描述
<b>raid6_nc</b>	RAID6 N 延续 Rotating parity N (left-to-right) with data continuation
<b>raid10</b>	条带镜像。如果指定 <b>-m</b> 并制定大于 1 的条带数，这就是 <b>lvcreate</b> 中 <b>--type</b> 参数的默认值。 条带镜像集

对大多数用户来说，指定五个主要类型 (**raid1**、**raid4**、**raid5**、**raid6**、**raid10**) 之一应该就足够了。有关 RAID5/6 所使用的不同算法的详情，请查看《通用 RAID 磁盘数据格式规格》一书中的第四章，网址为：[http://www.snia.org/sites/default/files/SNIA\\_DDF\\_Technical\\_Position\\_v2.0.pdf](http://www.snia.org/sites/default/files/SNIA_DDF_Technical_Position_v2.0.pdf)。

创建 RAID 逻辑卷时，LVM 会为每个数据生成大小为 1 个扩展的元数据子卷，或在阵列中生成奇偶校验子卷。例如：生成双向 RAID1 阵列可得到两个元数据子卷（即 **lv\_rmeta\_0** 和 **lv\_rmeta\_1**）和两个数据子卷 (**lv\_rimage\_0** 和 **lv\_rimage\_1**)。同样，创建三向条带（加上一个隐式奇偶校验设备）RAID4 得到四个元数据子卷 (**lv\_rmeta\_0**、**lv\_rmeta\_1**、**lv\_rmeta\_2** 和 **lv\_rmeta\_3**) 及四个数据子卷 (**lv\_rimage\_0**、**lv\_rimage\_1**、**lv\_rimage\_2** 和 **lv\_rimage\_3**)。

下面的命令在 1GB 的卷组 **my\_vg** 中创建名为 **my\_lv** 的双向 RAID1 阵列。

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

可根据指定的 **-m** 参数值使用不同副本数创建 RAID1 阵列。同样，可使用 **-i argument** 选项为 RAID4/5/6 逻辑卷指定条带数。还可以使用 **-I** 参数指定条带大小。

下面的命令在 1GB 的卷组 **my\_vg** 中创建名为 **my\_lv** 的 RAID5 阵列（三个条带 + 一个隐式奇偶校验驱动器）。注：指定要在 LVM 条带卷中的条带数，会自动添加奇偶校验驱动器的正确数字。

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

下面的命令在 1GB 的卷组 **my\_vg** 中创建名为 **my\_lv** 的 RAID6 阵列（三个条带 + 两个隐式奇偶校验驱动器）。

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

使用 LVM 创建 RAID 逻辑卷后，可如同其他 LVM 逻辑卷一样激活、更改、删除、显示并使用该卷。

创建 RAID10 逻辑卷后，初始化附带 **sync** 操作逻辑卷所需的后台 I/O 会将其他 I/O 操作挤入 LVM 设备，比如卷组元数据更新，特别是在创建很多 RAID 逻辑卷时。这样会导致其他 LVM 操作变缓。

可使用恢复限制控制 RAID 逻辑卷初始化的比例。通过使用 **lvcreate** 命令的 **--minrecoveryrate** 和 **--maxrecoveryrate** 选项设定那些操作的最小和最大 I/O 比例控制 **sync** 操作的执行比例。可按以下方法指定这些选项。

#### » **--maxrecoveryrate Rate[bBsSkMmGg]**

为 RAID 逻辑卷设定最大恢复比例，以便其不会排挤正常 I/O 操作。将 **Rate** 指定为该阵列中每个设备的每秒恢复数量。如果没有给出后缀，则假设使用 KiB/sec/device。将恢复比例设定为 0 的含义是不绑定。

#### » **--minrecoveryrate Rate[bBsSkMmGg]**

为 RAID 逻辑卷设定最小恢复比例，这样可以保证即使有很多正常 I/O 操作，**sync** 操作的 I/O 也可达到最小的吞吐量。将 **Rate** 指定为阵列中的每台设备的每秒恢复量。如果没有给出后缀，则假设为 KiB/sec/device。

下面的命令使用大小为 10GB 的三个条带创建一个双向 RAID10 阵列，其最大恢复比例为 128 kiB/sec/device。这个阵列名为 **my\_lv**，位于卷组 **my\_vg**。

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

还可为 RAID 擦除操作指定最小和最大恢复比例。有关 RAID 擦除的详情，请查看 [第 4.4.3.7.4 节“擦除 RAID 逻辑卷”](#)。

下面各小节论述了可在 LVM RAID 设备中执行的管理任务：

- » [第 4.4.3.1 节“将线性设备转换为 RAID 设备”](#)
- » [第 4.4.3.2 节“将 LVM RAID1 逻辑卷转换为 LVM 线性逻辑卷”](#)
- » [第 4.4.3.3 节“将镜像 LVM 卷转换为 RAID1 设备”](#)
- » [第 4.4.3.4 节“更改现有 RAID1 设备中的映像数”](#)
- » [第 4.4.3.5 节“将 RAID 映像拆分为独立的逻辑卷”](#)
- » [第 4.4.3.6 节“拆分及合并 RAID 映象”](#)
- » [第 4.4.3.7 节“设定 RAID 错误策略”](#)
- » [第 4.4.3.7.3 节“替换 RAID 设备”](#)
- » [第 4.4.3.7.4 节“擦除 RAID 逻辑卷”](#)
- » [第 4.4.3.7.5 节“控制 RAID 逻辑卷中的 I/O 操作”](#)

#### 4.4.3.1. 将线性设备转换为 RAID 设备

可使用 **lvconvert** 命令的 **--type** 参数将现有线性逻辑卷转换为 RAID 设备。

使用下面的命令可将卷组 **my\_vg** 中的线性逻辑卷 **my\_lv** 转换为双向 RAID1 阵列。

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

因为 RAID 逻辑卷由元数据及数据子卷对组成，当将线性设备转换为 RAID1 阵列时，会创建新的元数据子卷，并与该线性卷所在物理卷相同的原始逻辑卷（之一）关联。会将该附加映像添加到元数据/数组子卷对中。例如：如果原始设备如下：

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

转换为双向 RAID1 阵列后，该设备包含以下数据和元数据子卷对：

```
# 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)
```

如果与原始逻辑卷配对的元数据映像无法放到同一物理卷中，则 **lvconvert** 会失败。

#### 4.4.3.2. 将 LVM RAID1 逻辑卷转换为 LVM 线性逻辑卷

使用 **lvconvert** 命令的 **-m0** 参数可将现有 RAID2 LVM 逻辑卷转换为 LVM 现有逻辑卷。这样会删除所有 RAID 数据子卷，及组成 RAID 阵列的所有 RAID 元数据子卷，让顶层 RAID2 映像保留为线性逻辑卷。

下面的示例显示现有 LVM RAID1 逻辑卷。

```
# 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)
```

下面的命令将 LVM RAID1 逻辑卷 **my\_vg/my\_lv** 转换为 LVM 线性设备。

```
# 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)
```

将 LVM RAID1 逻辑卷转换为 LVM 线性卷时，可指定要删除的物理卷。下面的示例显示组成两个映像的 LVM RAID1 逻辑卷布局：**/dev/sda1** 和 **/dev/sda2**。在这个示例中，**lvconvert** 命令指定要删除 **/dev/sda1**，保留 **/dev/sdb1** 作为组成线性设备的物理卷。

```
# 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)
```

#### 4.4.3.3. 将镜像 LVM 卷转换为 RAID1 设备

可使用 **lvconvert** 命令的 **--type raid1** 参数将现有使用 **mirror** 片段类型的镜像 LVM 设备转换为 RAID1 LVM 设备。这样会将该镜像子卷 (**\*\_mimage\_\***) 重命名为 RAID 子卷 (**\*\_rimage\_\***)。此外，会删除该镜像日志，并在同一物理卷中为数据子卷创建元数据子卷 (**\*\_rmeta\_\***) 作为对应的数据子卷。

下面的示例显示镜像逻辑卷 **my\_vg/my\_lv** 的布局。

```
# 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)
```

下面的命令将镜像逻辑卷 `my_vg/my_lv` 转换为 RAID1 逻辑卷。

```
# 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)
```

#### 4.4.3.4. 更改现有 RAID1 设备中的映像数

可与更改之前的 LVM 镜像部署中的映像数一样更改现有 RAID1 阵列的映像数，方法是使用 `lvconvert` 命令指定要添加或删除的额外元数据/数据子卷对。有关在之前的 LVM 镜像实施中更改卷配置的详情，请查看[第 4.4.4.4 节“更改镜像卷配置”](#)。

使用 `lvconvert` 命令在 RAID1 设备中添加映像时，可为得到的设备指定映像总数，或指定要在该设备中添加多少映像。还可自选指定新元数据/数据映像对所在物理卷。

元数据子卷（名为 `*_rmeta_*`）总是位于其数据子卷副本（`*_rimage_*`）所在的同一物理设备中。不会在该 RAID 阵列中的另一个元数据/数据子卷对所在的同一物理卷中创建该元数据/数据子卷对（除非指定 `--alloc anywhere` 选项）。

在 RAID1 卷中添加映像命令的格式如下：

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

例如：下面内容中的 LVM 设备 `my_vg/my_lv` 是一个双向 RAID1 阵列：

```
# 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)
```

下面的命令将双向 RAID 1 设备 `my_vg/my_lv` 转换为三向 RAID1 设备：

```
# 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)
```

在 RAID1 阵列添加映像时，可指定使用该映像的物理卷。下面的命令将双向 RAID1 设备 `my_vg/my_lv` 转换为三向 RAID1 设备，指定该阵列使用物理卷 `/dev/sdd1`：

```
# 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)
```

可使用下面的命令从 RAID1 阵列删除映像。使用 **lvconvert** 命令从 RAID1 设备中删除映像时，可为得到的设备指定映像总数，或者可指定要从该设备中删除多少映像。还可以自选指定要从中删除该设备的物理卷。

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

另外，删除某个映像机器关联的元数据子卷后，数字较高的映像会降低以填入该插槽。如果要从一个由 **lv\_rimage\_0**、**lv\_rimage\_1** 和 **lv\_rimage\_2** 组成的三向 RAID1 阵列中删除 **lv\_rimage\_1**，得到的 RAID1 阵列由 **lv\_rimage\_0** 和 **lv\_rimage\_1** 组成。该子卷 **lv\_rimage\_2** 会重新命名，并接管空白的插槽，使其成为 **lv\_rimage\_1**。

下面的示例演示三向 RAID1 逻辑卷 **my\_vg/my\_lv** 的布局。

```
# 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)
```

下面的命令将三向 RAID1 逻辑卷转换为双向 RAID1 逻辑卷。

```
# 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)
```

下面的命令将三向 RAID1 逻辑卷转换为双向 RAID1 逻辑卷，指定包含删除为 **/dev/sde1** 映象的物理卷。

```
# 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)
```

#### 4.4.3.5. 将 RAID 映像拆分为独立的逻辑卷

可拆分 RAID 逻辑卷映像，以便组成新的逻辑卷。这个拆分 RAID 映像的过程与拆分镜像逻辑卷冗余映像的过程一样，如 [第 4.4.4.2 节“拆分镜像逻辑卷的冗余映象”](#) 所述。

拆分 RAID 映像的命令格式如下：

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

与从现有 RAID1 逻辑卷中删除 RAID 映像一样（如 [第 4.4.3.4 节“更改现有 RAID1 设备中的映像数”](#) 所述），从该设备中间删除 RAID 数据子卷（及其关联元数据子卷）时，所有较高编号的映像都会向下填入该槽。因此组成 RAID 阵列的逻辑卷索引号不会破坏其整数顺序。

#### 注意

如果该 RAID1 阵列未同步则无法拆分 RAID 映像。

下面的示例演示了如何将双向 RAID1 逻辑卷 **my\_lv** 拆分为两个线性逻辑卷 **my\_lv** 和 **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 /dev/sde1(1)
new /dev/sdf1(1)
```

下面的示例演示了如何将三向 RAID1 逻辑卷 **my\_lv** 拆分为一个双向逻辑卷 **my\_lv** 和一个线性逻辑卷 **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                  /dev/sdg1(1)
```

#### 4.4.3.6. 拆分及合并 RAID 映象

可与 **lvconvert** 命令的 **--splitmirrors** 参数联合使用 **--trackchanges** 参数临时拆分 RAID1 阵列，同时跟踪所有变更。这样可稍后将该映象合回到该阵列，同时只重新同步那些将该映象拆分后阵列有变化的部分。

**lvconvert** 命令拆分 RAID 映象的格式如下。

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

使用 **--trackchanges** 参数拆分 RAID 映象时，可指定要拆分的映象，但不能更改要拆分的卷名称。另外，得到的卷有以下限制。

- » 所创建新卷为只读。
- » 无法重新定义新卷大小。
- » 无法重新命名剩余阵列。
- » 无法重新定义剩余的阵列大小。
- » 可单独激活新卷及剩余的阵列。

可通过随后执行附带 **--merge** 参数的 **lvconvert** 命令指定 **--trackchanges** 参数，以合并拆分的映象。合并该映象时，只重新同步拆分该映象后更改的那部分阵列。

合并 RAID 映象的 **lvconvert** 命令格式如下。

```
lvconvert --merge raid_image
```

下面的示例创建了一个 RAID1 逻辑卷，然后从该卷中拆分出一个映象，同时跟踪剩余阵列的变化。

```
# lvcreate --type raid1 -m 2 -L 1G -n my_lv .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)
```

下面的示例演示了从 RAID1卷中拆分出一个映象，同时追踪剩余阵列的变化，然后将该卷合并回该阵列。

```
# 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)
```

从 RAID1 卷中拆分出映象后，再次运行 **lvconvert --splitmirrors** 命令，即重复最初用来拆分该映象的 **lvconvert** 命令，但不指定 **--trackchanges** 参数，即可永久获得该拆分。这样会破坏 **--trackchanges** 参数生成的链接。

使用 **--trackchanges** 参数拆分映象后，则无法在那个阵列中继续使用 **lvconvert --splitmirrors** 命令，除非要永久拆分所跟踪的映象。

下面的一系列命令可拆分映象，并跟踪该映象，然后永久拆分那个跟踪的映象。

```
# lvconvert --splitmirrors 1 --trackchanges my_vg/my_lv
my_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
# lvconvert --splitmirrors 1 -n new my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
LV  Copy%  Devices
my_lv      /dev/sdc1(1)
new        /dev/sdd1(1)
```

注：但以下命令系列会失败。

```
# lvconvert --splitmirrors 1 --trackchanges my_vg/my_lv
my_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
# lvconvert --splitmirrors 1 --trackchanges my_vg/my_lv
Cannot track more than one split image at a time
```

同样，下面的命令系列也会失败，因为拆分的映象不是跟踪的映象。

```
# lvconvert --splitmirrors 1 --trackchanges my_vg/my_lv
my_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 --splitmirrors 1 -n new my_vg/my_lv /dev/sdc1
Unable to split additional image from my_lv while tracking changes for
my_lv_rimage_1
```

#### 4.4.3.7. 设定 RAID 错误策略

LVM RAID 根据 `lvm.conf` 文件中 `raid_fault_policy` 字段所定义的属性，以自动方式处理设备失败。

- » 如果将 `raid_fault_policy` 字段设定为 `allocate`，则系统会尝试使用该卷组中的可用设备替换失败的设备。如果没有可用的剩余设备，则会在系统日志中记录。
- » 如果将 `raid_fault_policy` 字段设定为 `warn`，则该系统会生成警告，并在日志中记录该设备已失败。这样可让用户决定要采取的行动。

只要有足够的设备可供使用，RAID 逻辑卷就会继续运行。

##### 4.4.3.7.1. allocate RAID 错误策略

在下面的示例中，已在 `lvm.conf` 文件中将 `raid_fault_policy` 字段设定为 `allocate`。RAID 逻辑卷的布局如下。

```
# 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)
```

如果 `/dev/sde` 设备失败，则系统日志中会显示出错信息。

```
# 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-H200-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.
```

因为已将 `raid_fault_policy` 字段设定为 `allocate`，因此会使用该卷组中的新设备替换失败的设备。

```
# lvs -a -o name,copy_percent,devices vg
Couldn't find device with uuid 3lugiV-3eSP-AFAR-sdrP-H200-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)
```

注：即使已替换设备的设备，但在显示中仍表示 LVM 无法找到失败的设备。这是因为虽然已从 RAID 逻辑卷中删除那个失败的设备，但尚未从该卷组中删除该设备。要从卷组中删除设备的设备，请运行 `vgreduce -removemissing VG`。

如果已将 `raid_fault_policy` 设定为 `allocate`，但没有剩余的设备，则分配会设备，并让该逻辑卷保持此状态。如果分配失败，则需要修复确定当前，然后停用并重新激活该逻辑卷，如 [第 4.4.3.7.2 节“警告 RAID 出错策略”](#) 所述。另外，还可以替换设备的设备，如 [第 4.4.3.7.3 节“替换 RAID 设备”](#) 所述。

#### 4.4.3.7.2. 警告 RAID 出错策略

在下面的示例中，已将 `lvm.conf` 文件中的 `raid_fault_policy` 字段设定为 `warn`。该 RAID 逻辑卷的布局如下。

```
# 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)
```

如果 `/dev/sdh` 设备失败，则系统日志中会显示出错信息。但在这种情况下，LVM 不会通过替换映象之一尝试自动修复该 RAID 设备。反之，如果该设备失败，则会使用 `lvconvert` 命令的 `--repair` 参数替换失败的设备，如下所示。

```
# lvconvert --repair my_vg/my_lv
/dev/sdh1: read failed after 0 of 2048 at 250994294784: Input/output
error
/dev/sdh1: read failed after 0 of 2048 at 250994376704: Input/output
error
/dev/sdh1: read failed after 0 of 2048 at 0: Input/output error
/dev/sdh1: read failed after 0 of 2048 at 4096: Input/output error
Couldn't find device with uuid fbI0Y0-GX7x-firU-Vy5o-vzwx-vAKZ-feRxFF.
Attempt to replace failed RAID images (requires full device resync)?
[y/n]: y

# lvs -a -o name,copy_percent,devices my_vg
Couldn't find device with uuid fbI0Y0-GX7x-firU-Vy5o-vzwx-vAKZ-feRxFF.
LV          Copy%  Devices
my_lv        64.00
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)
```

注：即使已替换设备的设备，但在显示中仍表示 LVM 无法找到失败的设备。这是因为虽然已从 RAID 逻辑卷中删除那个失败的设备，但尚未从该卷组中删除该设备。要从卷组中删除设备的设备，请运行 **vgreduce --removemissing VG**。

如果该设备失败是一个瞬时失败，或者您可以修复失败的设备，则可以使用 **lvchange** 命令的 **--refresh** 选项开始修复失败的设备。以前是需要停用然后再重新激活该逻辑卷。

使用下面的命令刷新逻辑卷。

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

#### 4.4.3.7.3. 替换 RAID 设备

RAID 与传统的 LVM 镜像不同。LVM 镜像需要删除失败的设备，否则镜像逻辑卷会挂起。RAID 阵列可在有失败设备的情况下继续运行。事实上，在 RAID1 以外的 RAID 类型中，删除某个设备可能意味着转换为低级 RAID（例如：从 RAID6 转换为 RAID5，或者从 RAID4 或者 RAID5 转换为 RAID0）。因此，与其无条件删除失败的设备并可能会分配一个替换的设备，LVM 允许您使用 **lvconvert** 命令的 **--replace** 参数一步到位地替换 RAID 卷中的设备。

**lvconvert --replace** 命令的格式如下。

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

下面的示例创建 RAID1 逻辑卷，然后替换那个卷中的一个设备。

```
# 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)
```

下面的示例创建 RAID1 逻辑卷，然后替换那个卷中的设备，指定用来进行替换的物理卷。

```
# 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)
```

可指定多个 **replace** 参数一次替换多个 RAID 设备，如下所示。

```
# 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)
```

 注意

使用 **lvconvert --replace** 命令指定替换启动器时，永远不要使用阵列中已经使用的驱动器分配替换驱动器。例如：**lv\_rimage\_0** 和 **lv\_rimage\_1** 不应位于同一物理卷中。

**4.4.3.7.4. 擦除 RAID 逻辑卷**

LVM 为 RAID 逻辑卷提供擦除支持。RAID 擦除是读取某个阵列中的所有数据及奇偶校验块，检查其是否一致的过程。

使用 **lvchange** 命令的 **--syncaction** 选项启动 RAID 擦除操作。可指定 **check** 或者 **repair** 操作。**check** 操作针对该阵列，并记录阵列中的差异数字，但不会修复。**repair** 操作会修正发现的差异。

擦除 RAID 逻辑卷的命令格式如下：

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

 注意

**lvchange --syncaction repair vg/raid\_lv** 操作不执行与 **lvconvert --repair vg/raid\_lv** 操作相同的功能。**lvchange --syncaction repair** 操作中该阵列中启动一个后台同步操作，同时 **lvconvert --repair** 操作旨在修复/替换镜像或 RAID 逻辑卷中失败的设备。

为支持新 RAID 擦除操作，**lvs** 命令现在支持两种新的可输出字段：**raid\_sync\_action** 和 **raid\_mismatch\_count**。默认不会输出这些字段。可在 **lvs** 命令中指定 **-o** 参数以显示这些字段，如下。

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

**raid\_sync\_action** 字段显示该 RAID 卷目前正在进行的同步操作。它可以是以下值之一：

- » **idle**：已完成所有同步操作（什么也不做）
- » **resync**：初始化阵列或在机器失败后恢复
- » **recover**：替换阵列中的设备
- » **check**：查找阵列不一致的地方

- » **repair**：查找并修复阵列不一致的地方

**raid\_mismatch\_count** 字段显示 **check** 操作中找到的差异。

**lvs** 命令的 **Cpy%Sync** 字段现在输出 **raid\_sync\_action** 操作的过程，其中包括 **check** 和 **repair**。

**lvs** 命令的 **lv\_attr** 字段显示目前可在 RAID 擦除操作支持中提供额外的识别符。这个字段的字节 9 显示逻辑卷是否正常工作，它现在支持以下识别符。

- » (*m* (不匹配)) 表示在 RAID 逻辑卷中存在差异。如果擦除操作探测到 RAID 的部分有不一致的情况后会显示这个字符。
- » (*r* (刷新)) 表示阵列中的设备已经失败，且内核将其视为失败，即使 LVM 可读取该设备标签，并将其视为可操作。该逻辑卷应该可以刷新以通知内核该设备现在可用；也可以在怀疑该设备失败后将其替换。

有关 **lvs** 命令的详情，请查看 [第 4.8.2 节“对象选择”](#)。

执行 RAID 擦除操作时，**sync**（同步）操作要求的后台 I/O 会被 LVM 设备的其他操作挤掉，比如更新到卷组元数据。这样可能会造成其他 LVM 操作变慢。您可以通过设定恢复限制控制 RAID 逻辑卷的擦除比例。

可通过为 **lvchange** 命令的 **--minrecoveryrate** 或 **--maxrecoveryrate** 选项设定 **sync** 操作的最小和最大 I/O 比例，控制该操作比例。可按照以下方式指定这些选项。

- » **--maxrecoveryrate Rate[bBsSkMmGg]**

为 RAID 逻辑卷设定最大恢复比例，以便其不会排挤正常 I/O 操作。将 *Rate* 指定为该阵列中每个设备的每秒恢复数量。如果没有给出后缀，则假设使用 kiB/sec/device。将恢复比例设定为 0 的含义是不绑定。

- » **--minrecoveryrate Rate[bBsSkMmGg]**

设定 RAID 逻辑卷的最小恢复比例，保证 **sync** 操作取得最小吞吐量，即使有大量常规 I/O。以阵列中的每个设备的每秒数量为单位指定 *Rate*。如果没有给出后缀，则假设使用 kiB/sec/device。

#### 4.4.3.7.5. 控制 RAID 逻辑卷中的 I/O 操作

可使用 **lvchange** 命令的 **--writemostly** 和 **--writebehind** 参数为 RAID1 逻辑卷中的设备控制 I/O 操作。

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

将 RAID1 逻辑卷中的设备标记为 **write-mostly**。除非必要，应避免读取这些驱动器。设定这个参数以保证该驱动器的最小 I/O 操作数。默认行为是为了解决在指定的物理卷设定 **write-mostly** 刷新。可在该物理卷中附加 :*n* 删去 **write-mostly** 标记。在单一命令中可多次指定 **write-mostly** 参数，使其可以一次为逻辑卷中的所有物理卷触发 **write-mostly** 属性。

- » **--[raid]writebehind IOCount**

指定 RAID1 逻辑卷中设备允许的最大未完成写入数，将其标记为 **write-mostly**。超过此数值后，写入变为同步，导致所有对组成设备的写入会在阵列发出完成写入信号前完成。将该值设定为 0 会清除该属性，并允许该系统随机选择该值。

#### 4.4.4. 创建镜像卷

从 Red Hat Enterprise Linux 7.0 发行本开始，LVM 支持 RAID 1/4/5/6/10，如 [第 4.4.3 节“RAID 逻辑卷”](#) 所述。集群无法识别 RAID 逻辑卷。虽然可只在一台机器中创建并激活 RAID 逻辑卷，也可以同时在一台以上机器中激活它们。如果需要非独占镜像卷，则必须使用 **mirror** 片段类型创建卷，如本小节所述。



## 注意

有关使用 **mirror** 片段类型将现有 LVM 设备转换为 RAID1 LVM 设备的详情，请查看 [第 4.4.3.3 节“将镜像 LVM 卷转换为 RAID1 设备”](#)。



## 注意

在集群中创建镜像 LVM 逻辑卷的命令和步骤与在单一节点中使用 **mirror** 片段类型创建镜像 LVM 逻辑卷相同。但要在集群中创建镜像 LVM 卷，则必须运行该集群和集群镜像基础结构，该集群必须处于 **quorate** 状态，且必须在 **lvm.conf** 文件中正确设定锁定类型以便启用集群锁定。有关在集群中创建镜像卷的示例，请查看 [第 5.5 节“在集群中创建镜像 LVM 逻辑卷”](#)。

尝试在集群的多个节点中连续创建并转换多个 LVM 镜像可能会造成这些命令积压。这样会造成需要的操作超时，并进而失败。为避免这个问题，建议只在集群的一个节点中运行创建镜像的命令。

创建镜像卷时，可使用 **lvcreate** 命令的 **-m** 参数指定要生成的副本数。指定 **-m1** 则生成一个镜像，并产生两个文件系统副本：即一个线性逻辑卷外加一个副本。同样，指定 **-m2** 则生成两个镜像，并生成三个文件系统副本。

下面的命令使用单一镜像创建镜像逻辑卷。该卷大小为 50GB，名为 **mirrorlv**，是从卷组 **vg0** 中产生：

```
# lvcreate --type mirror -L 50G -m 1 -n mirrorlv vg0
```

LVM 将要复制的设备分成默认为 512KB 大小的区域。可使用 **lvcreate** 命令的 **-R** 参数以 MB 为单位指定区域大小。还可以通过编辑 **lvm.conf** 文件中的 **mirror\_region\_size** 设定编辑默认区域大小。



## 注意

由于集群架构的限制，无法使用默的 512KB 区域大小创建大于 1.5TB 的镜像。需要较大镜像的用户应将该区域大小从默认值调整为较大的值。无法增大区域大小将造成 LVM 创建操作停滞，同时还可能造成其他 LVM 命令挂起。

为超过 1.5TB 的镜像指定区域大小的一般方法是取镜像的 TB 数，并四舍五入到 2 的下一次幂，使用该数字作为 **lvcreate** 命令的 **-R** 参数值。例如：如果镜像大小为 1.5TB，则可以指定 **-R 2**。如果镜像大小为 3TB，则指定 **-R 4**。如果镜像大小为 5TB，则指定 **-R 8**。

下面的命令创建区域大小为 2MB 的镜像逻辑卷：

```
# lvcreate --type mirror -m 1 -L 2T -R 2 -n mirror vol_group
```

创建镜像后，则会同步镜像区域。对于较大的镜像组件，同步过程可能会较长。创建不需要恢复的新镜像时，可指定 **--nosync** 参数，表示不需要根据第一个设备初始化同步。

LVM 保留一个小日志以便用来记录那些区域与该镜像或多个镜像同步。默认情况下，这个日志是保存在磁盘中，以便其中重启后仍可用，并保证每次机器重启或崩溃后不需要重新同步该镜像。也可使用 **--mirrorlog core** 参数指定将该日志保存在内存中，这样就不需要额外的日志设备，但需要每次重启后重新同步整个镜像。

下面的命令在卷组 **bigvg** 中创建镜像逻辑卷。该逻辑卷名为 **ondiskmirvol**，有一个单一镜像。该卷大小为 12MB，在内存中保存镜像日志。

```
# lvcreate --type mirror -L 12MB -m 1 --mirrorlog core -n ondiskmirvol
bigvg
Logical volume "ondiskmirvol" created
```

该镜像日志是在于创建该镜像分支设备不同的设备中创建。但也可以使用 **vgcreate** 命令的 **--alloc anywhere** 参数在创建该镜像分支设备的同一设备中创建该镜像日志。这样可能会降低性能，但可让您在只有两个底层设备的情况下创建镜像。

下面的命令在创建镜像分支的同一设备中为单一镜像创建镜像逻辑卷。在这个示例中，卷组 **vg0** 由两个设备组成。这个命令在卷组 **vg0** 中创建名为 **mirrorlv**，大小为 500MB 的卷。

```
# lvcreate --type mirror -L 500M -m 1 -n mirrorlv -alloc anywhere vg0
```

### 注意

在集群镜像中，镜像日志管理完全由目前集群 ID 最低的集群节点控制。因此，当存储集群镜像日志的设备在集群的某个子集中不可用时，只要 ID 数最小的集群节点可以访问该集群该日志，则该集群镜像可继续运行，不受任何影响。因为该镜像不会受到影响，因此也不会进行自动修正（修复）。但当 ID 数最低的集群节点无法访问该镜像日志后，则会执行自动操作（无论其他节点是否可以访问该日志）。

要创建自我对称的镜像日志，可指定 **--mirrorlog mirrored** 参数。下面的命令在卷组 **bigvg** 中创建镜像逻辑卷。该逻辑卷名为 **twologvol**，有一个镜像。该卷大小为 12MB，镜像日志为对称的。

```
# lvcreate --type mirror -L 12MB -m 1 --mirrorlog mirrored -n twologvol
bigvg
Logical volume "twologvol" created
```

与标准镜像日志一样，可以使用 **vgcreate** 命令的 **--alloc anywhere** 参数在创建镜像分支的同一设备中创建冗余镜像日志。这样可能会降低性能，但可让您在没有足够底层设备以保证每个设备预期镜像分支处于不同设备的情况下创建冗余镜像日志。

创建镜像后，则会同步镜像区域。对于较大的镜像组件，同步过程可能会较长。创建不需要恢复的新镜像时，可指定 **--nosync** 参数，表示不需要根据第一个设备初始化同步。

可指定使用镜像分支及日志的设备以及该设备中要使用的扩展。要强制将日志记录在特定磁盘中，可指定保存日志的磁盘具体扩展。LVM 不需要遵守命令行中列出的顺序。如果列出任意物理卷，则只能在该设备中保存日志。列表中列出的已分配物理扩展将被忽略。

下面的命令创建使用单一镜像的镜像逻辑卷，且单一日志为不对称。该卷大小为 500MB，名为 **mirrorlv**，由卷组 **vg0** 生成。该镜像的第一个分支位于设备 **/dev/sda1**，第二个镜像位于设备 **/dev/sdb1**，而第三个镜像位于设备 **/dev/sdc1**。

```
# lvcreate --type mirror -L 500M -m 1 -n mirrorlv vg0 /dev/sda1
/dev/sdb1 /dev/sdc1
```

下面的命令创建使用单一镜像的镜像逻辑卷。其大小为 500MB，名为 **mirrorlv**，由卷组 **vg0** 生成。该镜像的第一个分支位于设备 **/dev/sda1** 的第 0-499 扩展，第二个分支位于 **/dev/sdb1** 的第 0-499 扩展，镜像日志起始点为设备 **/dev/sdc1** 的扩展 0. 这些是 1MB 扩展。如果任何指定的扩展已经被分配，则会将其忽略。

```
# lvcreate --type mirror -L 500M -m 1 -n mirrorlv vg0 /dev/sda1:0-499
/dev/sdb1:0-499 /dev/sdc1:0
```

## 注意

可将条带和镜像卷合并为一个逻辑卷。创建逻辑卷的同时指定镜像数（**--mirrors X**）和条带数（**-stripes Y**）即可得到一个由条带连续设备组成的镜像设备。

### 4.4.4.1. 镜像逻辑卷失败策略

您可以使用 **mirror\_image\_fault\_policy** 和 **mirror\_log\_fault\_policy** 参数在 **lvm.conf** 文件的 **activation** 部分定义逻辑卷在设备失败事件中的行为。将这些参数设定为 **activation** 后，系统会尝试移除出错的设备，并在没有这些设备的情况下运行。将这些参数设定为 **allocate** 后，系统会尝试移除出错的设备，并尝试在新设备中分配空间替换失败的设备。如果没有合适的设备，且没有可为替换失败的设备分配的空间，则这个策略与 **remove** 策略的行为一致。

默认情况下，**mirror\_log\_fault\_policy** 参数为 **allocate**。在日志中使用这个策略非常迅速，并可保证在崩溃及重启后可记住同步状态。如果将这个策略设定为 **remove**，则会在日志设备失败，镜像转而使用内存日志后，该镜像不会记住出现故障和重启后的同步状态，因而会重新同步整个镜像。

默认情况下，**mirror\_image\_fault\_policy** 参数为 **remove**。使用这个策略后，如果镜像映象失败，且只还有一个工作的副本，则该镜像会转而使用非镜像设备。为镜像设备将这个策略设定为 **allocate** 后，则要求该镜像设备重新同步该设备。这样会让过程变缓，但会保留该设备的镜像特征。

## 注意

LVM 镜像卷有失败的设备时，会进行两个阶段的恢复。第一个阶段包括删除失败的设备。这样可得到成为线性设备的镜像。在第二个阶段中，如果将 **mirror\_log\_fault\_policy** 参数设定为 **allocate**，则会尝试替换失败的设备。注：但不保证在第二阶段中会选取之前由该镜像使用，但不是失败设备一部分的设备可用。

从 LVM 镜像失败中手动恢复的详情请查看 [第 6.3 节“恢复 LVM 镜像错误”](#)。

### 4.4.4.2. 拆分镜像逻辑卷的冗余映象

可拆分镜像逻辑卷的冗余映象以形成新逻辑卷。要拆分映象，可使用 **lvconvert** 命令的 **--splitmirrors** 参数，指定要拆分的冗余映象数。必须使用该命令的 **--name** 参数指定新拆分逻辑卷的名称。

下面的命令从镜像逻辑卷 **vg/lv** 中拆分名为 **copy** 的新逻辑卷。新逻辑卷包含两个镜像分支。在这个示例中，LVM 选择要拆分的设备。

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

可指定要拆分的设备。下面的命令从镜像逻辑卷 **vg/lv** 中拆分名为 **copy** 的新逻辑卷。这个新逻辑卷包含两个镜像分支，由设备 **/dev/sdc1** 和 **/dev/sde1** 组成。

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

#### 4.4.4.3. 修复镜像逻辑设备

可使用 **lvconvert --repair** 命令在磁盘失败后修复镜像。这样可让该镜像返回一致的状态。**lvconvert --repair** 命令是互动式命令，可提示您确定是否要让该系统尝试替换失败的设备。

- » 可在命令行中指定 **-y** 选项跳过提示并替换所有失败的设备。
- » 可在命令行中指定 **-f** 选项跳过提示且不替换任何失败的设备。
- » 可指定 **--use-policies** 参数使用由 **lvm.conf** 文件中指定的 **mirror\_log\_fault\_policy** 和 **mirror\_device\_fault\_policy** 参数指定的替换策略，以便跳过提示，并仍然为镜像映象及镜像日志指定不同的替换策略。

#### 4.4.4.4. 更改镜像卷配置

可使用 **lvconvert** 命令增加或减少逻辑卷包含的镜像数。这样可让您将逻辑卷从镜像卷转换为线性卷，或将其实从线性卷转换为镜像卷。还可以使用这个命令重新配置现有逻辑卷的其他镜像参数，比如 **corelog**。

将线性卷转换为镜像卷时，基本上均可为现有卷创建镜像分支。这就是说您的卷组必须包含用于该镜像分支及镜像日志的设备和空间。

如果丢失镜像分支，则 LVM 会将该卷转换为线性卷，以便您可以在没有镜像冗余的情况下访问该卷。替换该分支后，可使用 **lvconvert** 命令恢复该镜像，如 [第 6.3 节“恢复 LVM 镜像错误”](#) 所述。

下面的命令将线性逻辑卷 **vg00/lvol1** 转换为镜像逻辑卷。

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

下面的命令将镜像逻辑卷 **vg00/lvol1** 转换为线性逻辑卷，并删除该镜像分支。

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

下面的示例在现有逻辑卷 **vg00/lvol1** 中添加附加镜像分支。这个示例显示使用 **lvconvert** 命令将该卷改为有两个镜像分支的卷钱、后的卷配置。

```
# lvs -a -o name,copy_percent,devices vg00
  LV          Copy%  Devices
  lvol1       100.00  lvol1_mimage_0(0),lvol1_mimage_1(0)
  [lvol1_mimage_0]      /dev/sda1(0)
  [lvol1_mimage_1]      /dev/sdb1(0)
  [lvol1_mlog]         /dev/sdd1(0)
# lvconvert -m 2 vg00/lvol1
vg00/lvol1: Converted: 13.0%
vg00/lvol1: Converted: 100.0%
Logical volume lvol1 converted.
# lvs -a -o name,copy_percent,devices vg00
  LV          Copy%  Devices
  lvol1       100.00
  lvol1_mimage_0(0),lvol1_mimage_1(0),lvol1_mimage_2(0)
  [lvol1_mimage_0]      /dev/sda1(0)
  [lvol1_mimage_1]      /dev/sdb1(0)
  [lvol1_mimage_2]      /dev/sdc1(0)
  [lvol1_mlog]         /dev/sdd1(0)
```

#### 4.4.5. 创建精简配置逻辑卷

可对逻辑卷进行精简配置。这样就可以让您创建比可用扩展大的逻辑卷。可使用精简配置管理剩余空间的存储池，也称精简池，应用程序可使用该池根据需要分配随机数量的设备。然后可以创建绑定到精简池的设备，以便稍后应用程序实际写入该逻辑卷时分配。该精简池可在需要时动态扩展，以便进行符合成本效益的存储空间分配。

## 注意

本小节提供用来创建并增大精简配置逻辑卷的基本命令概述。有关 LVM 精简配置详情及使用 LVM 命令和利用精简配置逻辑卷的详情，请查看 **lvmthin(7)** man page。

## 注意

在集群的节点间不支持精简卷。该精简池及其所有精简卷必须只在一个集群节点中激活。

请执行以下任务创建精简卷：

1. 使用 **vgcreate** 命令创建卷组。
2. 使用 **lvcreate** 命令创建精简池。
3. 使用 **lvcreate** 命令在精简池中创建精简卷。

可使用 **lvcreate** 命令的 **-T**（或者 **--thin**）选项创建精简池或精简卷。还可以使用 **lvcreate** 命令的 **-T** 选项使用单一命令同时在该池中创建精简池和精简卷。

下面的命令使用 **lvcreate** 命令的 **-T** 选项创建位于卷组 **vg001**，名为 **mythinpool** 的精简池中，大小为 100M。注：因为要创建占用物理空间的池，所以必须指定该池的大小。**lvcreate** 命令的 **-T** 选项不会使用参数，它会推理出该命令指定的其他选项所创建的设备类型。

```
# lvcreate -L 100M -T vg001/mythinpool
Rounding up size to full physical extent 4.00 MiB
Logical volume "mythinpool" created
# lvs
LV          VG      Attr       LSize   Pool Origin Data%  Move Log
Copy% Convert
my  mythinpool  vg001  twi-a-tz 100.00m                      0.00
```

下面的命令使用 **lvcreate** 命令的 **-T** 选项在精简池 **vg001/mythinpool** 中创建名为 **thinvolume** 的精简卷。注：在这个示例中要指定虚拟卷大小，且为该卷指定的虚拟卷大小要比其所在池要大。

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

下面的命令使用 **lvcreate** 命令的 **-T** 选项在该池中创建一个精简池和一个精简卷，方法是同时指定 **lvcreate** 的大小和虚拟卷大小参数。这个命令在卷组 **vg001** 中创建名为 **mythinpool** 的精简池，还会在该池中创建名为 **thinvolume** 的精简卷。

```
# lvcreate -L 100M -T vg001/mythinpool -V 1G -n thinvolume
Rounding up size to full physical extent 4.00 MiB
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
```

还可使用 **lvcreate** 命令的 **--thinpool** 参数创建精简池。与 **-T** 选项不同，**--thinpool** 参数需要给出一个参数，即所要创建精简池逻辑卷的名称。下面的示例指定 **lvcreate** 命令的 **--thinpool** 选项创建位于卷组 **vg001**，大小为 100M，名为 **mythinpool** 的精简池。

```
# lvcreate -L 100M --thinpool mythinpool vg001
Rounding up size to full physical extent 4.00 MiB
Logical volume "mythinpool" created
# lvs
LV VG Attr LSize Pool Origin Data% Move Log Copy%
Convert
mythinpool vg001 twi-a-tz 100.00m 0.00
```

创建池时支持条带。下面的命令在有两个 64kB 条带及一个 256kB 区块的卷组 **vg001** 中创建名为 **pool**，大小为 100M 的精简池。它还会创建一个 1T 精简卷 **vg001/thin\_lv**。

```
# lvcreate -i 2 -I 64 -c 256 -L 100M -T vg001/pool -V 1T --name thin_lv
```

可使用 **lvextend** 命令扩展精简卷的大小。但不能将其减小。

下面的命令可为现有精简池扩充 100M 以重新定义其大小。

```
# 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% Move Log
Copy% Convert
mythinpool vg001 twi-a-tz 200.00m 0.00
thinvolume vg001 Vwi-a-tz 1.00g mythinpool 0.00
```

与其他逻辑卷类型一样，可使用 **lvrename** 命令重命名该卷，使用 **lvremove** 删除该卷，同时可使用 **lvs** 和 **lvdisplay** 命令显示该卷的信息。

默认情况下，**lvcreate** 根据方程式 (**Pool\_LV\_size / Pool\_LV\_chunk\_size \* 64**) 设定精简池元数据逻辑卷的大小。但如果要稍后大幅度增加精简池大小，则应该使用 **lvcreate** 命令的 **--poolmetadatasize** 参数增大这个值。精简池的元数据逻辑卷的支持值在 2MiB 到 16GiB 之间。

可使用 **lvconvert** 命令的 **--thinpool** 参数将现有逻辑卷转换为精简卷。将现有逻辑卷转换为精简池卷时，必须配合使用 **lvconvert** 命令的 **--thinpool** 参数将现有逻辑卷转换为精简池卷的元数据卷。

## 注意

将逻辑卷转换为精简池卷或精简池元数据卷会破坏该逻辑卷的内容，因为在这个实例中，**lvconvert** 不会保留设备的内容，而是会覆盖这些内容。

下面的命令将卷组 **vg001** 中的现有逻辑卷 **lv1** 转换为精简池卷，并将卷组 **vg001** 中的现有逻辑卷 **lv2** 转换为那个精简池卷的元数据卷。

```
# lvconvert --thinpool vg001/lv1 --poolmetadata vg001/lv2
Converted vg001/lv1 to thin pool.
```

#### 4.4.6. 创建快照卷

##### 注意

LVM 支持精简配置快照。有关创建精简配置快照卷的详情，请查看 [第 4.4.7 节“创建精简配置快照卷”](#)。

使用 **lvcreate** 命令的 **-s** 参数创建快照卷。快照卷是可以写入的。

##### 注意

不支持跨集群节点的 LVM 快照。不能在集群的卷组中创建快照卷。但如果需要在集群的逻辑卷中创建一致的数据备份，则可以独占方式激活该卷，然后创建快照。有关以独占方式在某个节点中激活逻辑卷的详情，请查看 [第 4.7 节“在集群的独立节点中激活逻辑卷”](#)。

##### 注意

镜像逻辑卷支持 LVM 快照。

RAID 逻辑卷支持快照。有关创建 RAID 逻辑卷的详情，请查看 [第 4.4.3 节“RAID 逻辑卷”](#)。

LVM 不允许创建超过原始卷与该卷所需元数据大小之和的快照卷。如果指定的快照卷大小超过此数值，则系统会创建原始卷所需大小的快照卷。

默认情况下，正常激活命令会跳过快照卷。有关控制快照卷激活的详情，请查看 [第 4.4.17 节“控制逻辑卷激活”](#)。

下面的命令创建大小为 100MB，名为 **/dev/vg00/snap** 的快照逻辑卷。这样会创建名为 **/dev/vg00/lvol1** 的原始卷的快照。如果原始逻辑卷中包含一个文件系统，则可以在任意目录中挂载该快照逻辑卷，以便访问该文件系统，让原始文件系统继续获得更新的同时运行备份。

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

创建快照逻辑卷后，在 **lvdisplay** 命令中指定原始卷可生成包括所有快照逻辑卷机器状态（活跃或不活跃）的输出结果。

下面的示例显示逻辑卷 **/dev/new\_vg/lvol0** 的状态，已为其创建快照卷 **/dev/new\_vg/newvgsnap**。

```
# lvdisplay /dev/new_vg/lvol0
--- Logical volume ---
LV Name          /dev/new_vg/lvol0
```

```

VG Name          new_vg
LV UUID          LBv1Tz-sr23-0jsI-LT03-nHLC-y8XW-EhC178
LV Write Access  read/write
LV snapshot status source of
                  /dev/new_vg/newvgsnap1 [active]
LV Status        available
# open           0
LV Size          52.00 MB
Current LE       13
Segments         1
Allocation       inherit
Read ahead sectors 0
Block device     253:2

```

默认情况下，**lvs** 命令显示原始卷及每个快照卷目前使用的百分比。以下示例显示了系统 **lvs** 命令的默认输出结果，该系统中包括逻辑卷 **/dev/new\_vg/lvol0**，为其创建的快照为 **/dev/new\_vg/newvgsnap**。

```

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

```



### 警告

因为快照的大小随原始卷的变化而变化，常规使用 **lvs** 命令监控快照卷的百分比，以保证不会满溢就变得很重要。如果快照 100% 填满则会丢失全部数据，因为写入原始卷的未更改部分一定会破坏快照。

除快照本身填满后会失效外，所有那个快照设备中挂载的文件系统都会被强制卸载，以避免访问挂载点时不可避免的文件系统错误。另外，可在 **lvm.conf** 文件中指定 **snapshot\_autoextend\_threshold** 选项。这个选项允许在剩余快照空间低于所设定阈值时可随时自动扩展快照。这个功能要求在卷组中包含尚未分配的空间。

LVM 不允许您创建超过原始元与该卷所需元数据大小总和的快照卷。同样，自动扩展快照也不会将快照卷大小增大为超过该快照所需最大计算大小。快照增长到足以覆盖原始卷后，则不会监控其自动扩展。

设定 **lvm.conf** 文件本身提供的 **snapshot\_autoextend\_threshold** 和 **snapshot\_autoextend\_percent** 的详情。有关 **lvm.conf** 文件的详情，请查看 [附录 B, LVM 配置文件](#)。

#### 4.4.7. 创建精简配置快照卷

Red Hat Enterprise Linux 提供精简配置快照卷支持。有关精简快照卷的优点及局限，请参看 [第 2.3.6 节“精简配置快照卷”](#)。



### 注意

本小节提供用来创建并增大精简配置快照的基本命令概述。有关 LVM 精简配置的详情，以及在精简配置逻辑卷中使用 LVM 命令及程序的详情，请查看 **lvmthin(7) man page**。



## 重要

创建精简快照卷时，不能指定该卷的大小。如果指定 size 参数，所创建快照就不是精简快照卷，也不会使用精简池保存数据。例如：命令 `lvcreate -s vg001/thinvolume -L10M` 不会创建精简快照，即使原始卷是精简卷也不行。

可为精简配置原始卷或非精简配置逻辑卷创建精简快照。

可使用 `lvcreate` 命令的 `--name` 选项指定快照卷名称。下面的命令为精简配置逻辑卷 `vg001/thinvolume` 创建名为 `mysnapshot1` 的精简配置快照卷。

```
# lvcreate -s --name mysnapshot1 vg001/thinvolume
Logical volume "mysnapshot1" created
# lvs
  LV        VG      Attr       LSize   Pool      Origin      Data%
Move Log Copy% Convert
  mysnapshot1  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
```

精简快照卷与其他精简卷有同样的特征。您可以单独激活、扩展、重命名、删除该卷，甚至可以为该卷生成快照。

默认情况下，正常激活命令会跳过快照卷。有关控制快照卷激活的详情，请查看 [第 4.4.17 节“控制逻辑卷激活”](#)。

还可以在非精简配置逻辑卷中创建精简配置快照。因为非精简配置逻辑卷不存在于精简池中，因此将其视为外部原始卷。外部原始卷可由很多精简配置快照卷使用并共享，即使这些快照卷来自不同的精简池。创建该精简配置快照时，外部原始卷必须为不活跃卷，且处于只读状态。

要创建外部原始卷的精简配置快照，则必须指定 `--thinpool` 选项。下面的命令创建只读、不活跃卷 `origin_volume` 的精简快照卷。该精简快照卷名为 `mythinsnap`。然后逻辑卷 `origin_volume` 会成为卷组 `vg001` 中精简快照卷 `mythinsnap` 的精简外部原始卷，使用现有精简池 `vg001/pool`。因为原始卷必须与该快照卷处于同一卷组，因此不需要在指定原始逻辑卷时指定该卷组。

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

可使用下面的命令创建第一个快照卷的第二个精简配置快照卷。

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

## 4.4.8. 合并快照卷

可使用 `lvconvert` 命令的 `--merge` 选项将快照合并到其原始卷中。如果原始卷及快照卷都没有打开，则会立即开始合并。否则会在第一次激活原始卷或快照卷，且两者均为关闭状态时开始合并快照。将快照合并到无法关闭的原始卷时，比如 `root` 文件系统，会延迟到下次激活该原始卷时方才进行。合并开始时，得到的逻辑卷会有原始卷的名称、次要号码及 UUID。在合并过程中，对原始卷的读取或者写入直接指向要合并的快照。合并完成后，则会删除合并的快照。

下面的命令将快照卷 `vg001/lvol1_snap` 合并到期原始卷中。

```
# lvconvert --merge vg001/lvol1_snap
```

可在命令行中指定多个快照，或者使用 LVM 对象标签将多个快照合并到其各自的原始卷中。在下面的示例中，逻辑卷 `vg00/lvol1`、`vg00/lvol2` 和 `vg00/lvol3` 均使用 `@some_tag` 标记。下面的命令将该快照逻辑卷按顺序合并到所有三个卷中：即 `vg00/lvol1`，然后 `vg00/lvol2`，然后 `vg00/lvol3`。如果使用 `--background` 选项，所有快照逻辑卷合并操作都应同时开始。

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

有关标记 LVM 对象的详情，请查看 [附录 C, LVM 对象标签](#)。有关 `lvconvert --merge` 命令的详情，请查看 `lvconvert(8)` man page。

#### 4.4.9. 永久设备号

载入模块时会动态分配主要和次要设备号。有些应用程序在块设备永远使用同一设备（主要和次要）号激活状态时工作状态最佳。可使用下面的参数，通过 `lvcreate` 和 `lvchange` 命令的指定这些内容：

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

使用较大的次要号码以保证尚未将其动态分配给另一个设备。

如果要使用 NFS 导出文件系统，在导出文件中指定 `fsid` 参数可避免在 LVM 中设定持久设备号。

#### 4.4.10. 重新定义逻辑卷大小

可使用 `lvreduce` 命令减小逻辑卷大小。如果该逻辑卷包含一个文件系统，请确定首先减小该文件系统（或使用 LVM GUI），以便逻辑卷总是可至少达到文件系统所需要的大小。

下面的命令将卷组 `vg00` 中逻辑卷 `lvol1` 大小减小 3 个逻辑扩展。

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

#### 4.4.11. 更改逻辑卷组的参数

可使用 `lvchange` 命令更改逻辑卷的参数。有关可更改的参数列表，请查看 `lvchange(8)` man page。

可使用 `lvchange` 命令激活和停用逻辑卷。可使用 `vgchange` 命令同时激活和停用卷组中的所有逻辑卷，如 [第 4.3.8 节“更改卷组参数”](#) 所述。

下面的命令将卷组 `vg00` 中卷 `lvol1` 的权限改为只读。

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

#### 4.4.12. 重命名逻辑卷

请使用 `lvrename` 命令重命名现有逻辑卷。

下面的命令将卷组 `vg02` 中的逻辑卷 `lvol1` 重命名为 `lvnew`。

```
# lvrename /dev/vg02/lvol1 /dev/vg02/lvnew
```

```
# lvrename vg02 lvol1 lvnew
```

有关在集群的独立节点中激活逻辑卷的详情，请查看 [第 4.7 节“在集群的独立节点中激活逻辑卷”](#)。

#### 4.4.13. 删除逻辑卷

请使用 **lvremove** 命令删除不活跃的逻辑卷。如果目前未挂载该逻辑卷，请在将其删除前卸载该卷。另外，在集群的环境中必须在将其删除前停用逻辑卷。

下面的命令从卷组 **testvg** 中删除逻辑卷 **/dev/testvg/testlv**。注：在此情况下尚未停用该逻辑卷。

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

使用 **lvchange -an** 命令删除逻辑卷前，必须明确将其停用，但不会看到让您确定是否要删除某个活跃逻辑卷的提示。

#### 4.4.14. 显示逻辑卷

有三个命令可用来显示 LVM 逻辑卷的属性：**lvs**、**lvdisplay** 和 **lvscan**。

命令 **lvs** 以可配置格式提供逻辑卷信息，每行显示一个逻辑卷。**lvs** 命令提供大量格式控制，并在编写脚本时使用。有关使用 **lvs** 自定义您的输出结果的详情，请查看 [第 4.8 节“LVM 的自定义报告”](#)。

**lvdisplay** 命令以固定格式显示逻辑卷属性（比如大小、布局及映射）。

下面的命令显示 **vg00** 中 **lvol2** 的属性。如果已为这个原始逻辑卷创建快照逻辑卷，这个命令会显示所有快照逻辑卷及其状态（active 或者 inactive）列表。

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

**lvscan** 命令扫描该系统中的所有逻辑卷，并将其列出，如下所示。

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

#### 4.4.15. 扩展逻辑卷

使用 **lvextend** 命令增加逻辑卷的大小。

扩展逻辑卷后，可指明该卷要扩展的大小，或者将其扩展后该卷应该有多大。

下面的命令将逻辑卷 **/dev/myvg/homevol** 扩展为 12GB。

```
# 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
```

下面的命令在 **/dev/myvg/homevol** 逻辑卷中添加另一个 GB。

```
# 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
```

在 **lvcreate** 命令中，可使用 **lvextend** 命令的 **-l** 参数指定逻辑卷增加的扩展数。还可以使用这个参数指定卷组的百分比，或者卷组中剩余可用空间的百分比。下面的命令将名为 **testlv** 的逻辑卷扩展为填满卷组 **myvg** 中的未分配空间。

```
# lvextend -l +100%FREE /dev/myvg/testlv
Extending logical volume testlv to 68.59 GB
Logical volume testlv successfully resized
```

扩展逻辑卷后，需要增大文件系统大小以便与其匹配。

默认情况下，大多数文件系统重新定义大小的工具会将该文件系统的大小增加到底层逻辑卷的大小，这样就不需要考虑为每两个命令指定同样的大小。

#### 4.4.15.1. 扩展条带卷

要增大条带逻辑卷的大小，则必须在底层物理卷中有足够的剩余空间，以便组成卷组支持该条带。例如：如果有一个使用整个卷组的双向条带，在该卷组中添加单一物理卷不会让您有扩展该条带的能力，而必须在该卷组中添加至少两个物理卷方可有此能力。

例如：可使用下面的 **vgs** 命令显示由两个底层物理卷组成的卷组 **vg**。

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

使用该卷组中的整个空间数量创建条带。

```
# 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)
```

注：卷组目前没有剩余空间。

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

下面的命令在卷组中添加另一个物理卷，之后该卷组就有 135G 附加空间。

```
# 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
```

此时无法将该条带逻辑卷扩展到卷组的最大大小，因为需要有两个底层设备方可将数据条带化。

```
# 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
```

要扩展条带逻辑卷，需要添加量一个物理卷，然后再扩展该逻辑卷。在这个示例中，在该卷组中添加两个物理卷，即可将该逻辑卷扩展到该卷组的最大大小。

```
# 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
```

如果没有足够的底层物理设备扩展该条带逻辑卷，且扩展不是条带化也没有关系，则也能够扩展该卷，结果是会得到不平衡的性能。在逻辑卷中添加空间时，默认操作是使用与现有逻辑卷最后片段相同的条带参数，但也可以覆盖那些参数。下面的示例扩展现有条带逻辑卷，使用启动 **lvextend** 命令失败后可用的剩余空间。

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

#### 4.4.15.2. 扩展 RAID 卷

在不执行新 RAID 区域同步的情况下使用 **lvextend** 命令增大 RAID 逻辑卷。

使用 **lvcreate** 命令创建 RAID 逻辑卷时，如果指定 **--nosync** 选项，则在创建该逻辑卷后不会同步 RAID 区域。如果稍后使用 **--nosync** 选项扩展已创建的 RAID 逻辑卷，此时也不会同步该 RAID 扩展。

使用 **lvs** 命令的 **--nosync** 选项决定是否让已创建的现有逻辑卷显示该卷的属性。如果逻辑卷 attribute 字段第一个字符为“R”，则表示创建该 RAID 卷时没有启用初始同步；如果该字符为“r”，则表示创建时启动初始同步。

下面的命令显示名为 **lv** 的 RAID 逻辑卷的属性，创建该卷时没有启动同步，attribute 字段的第一个字符为“R”。该字段的第七个字符为“r”，代表 RAID 目标类型。有关 attribute 字段的含义，请查看 [表 4.4 “lvs 显示字段”](#)。

```
# lvs vg
LV VG Attr LSize Pool Origin Snap% Move Log Cpy%Sync Convert
lv vg Rwi-a-r- 5.00g 100.00
```

如果使用 **lvextend** 命令增大这个逻辑卷，则不会重新同步 RAID 扩展。

如果创建逻辑卷时没有指定 **lvcreate** 命令的 **--nosync** 选项，则可以使用 **lvextend** 命令的 **--nosync** 选项增大该逻辑卷，且无需重新同步该镜像。

下面的示例扩展了 RAID 逻辑卷，创建该卷时没有使用 **--nosync** 选项，表示在创建时同步该 RAID 卷。但这个示例中指定在扩展该卷时不要同步。注：该卷有属性 "r"，但执行附带 **--nosync** 选项的 **lvextend** 命令后，该卷有一个属性 "R"。

```
# lvs vg
  LV   VG   Attr      LSize  Pool Origin Snap%  Move Log Cpy%Sync
Convert
  lv   vg   rwi-a-r-  20.00m                               100.00
# lvextend -L +5G vg/lv --nosync
Extending 2 mirror images.
Extending logical volume lv to 5.02 GiB
Logical volume lv successfully resized
# lvs vg
  LV   VG   Attr      LSize  Pool Origin Snap%  Move Log      Cpy%Sync
Convert
  lv   vg   Rwi-a-r-  5.02g                               100.00
```

如果 RAID 卷为不活跃状态，即使创建该卷时指定了 **--nosync** 选项，也则不会在扩展该卷时自动跳过同步。反之，会提示您是否要全面创新同步该逻辑卷的扩展部分。

### 注意

如果 RAID 卷要执行恢复，则在使用 **--nosync** 选项创建或扩展该卷时不能扩展该逻辑卷。但如果未指定 **--nosync** 选项，就可以在恢复时扩展该 RAID 卷。

#### 4.4.15.3. 使用 **cling** 分配策略扩展逻辑卷

可使用 **lvextend** 命令的 **--alloc cling** 选项指定 **cling** 分配策略扩展 LVM 卷。这个策略会在与现有逻辑卷的最后片段所在的同一物理卷中选择空间。如果在该物理卷中没有足够的空间，并在 **lvm.conf** 文件中定义了一组标签，则 LVM 会检查是否在该物理卷中附加任何标签，并在现有扩展和新扩展之间映射那些物理卷标签。

例如：如果您有逻辑卷是某个单一卷组中两个网站间的镜像，就可以使用 @site1 和 @site2 标签，根据其所在位置标记该物理卷，并在 **lvm.conf** 文件中指定以下行：

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

有关物理卷标签的详情，请查看 [附录 C, LVM 对象标签](#)。

在下面的示例中修改了 **lvm.conf** 文件使其包含以下行：

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

同样在此示例中，创建的卷组 **taft** 由物理卷

**/dev/sdb1**、**/dev/sdc1**、**/dev/sdd1**、**/dev/sde1**、**/dev/sdf1**、**/dev/sdg1** 和 **/dev/sdh1** 组成。该示例不使用 **C** 标签，但这样会显示该 LVM 使用标签选择用于镜像分支的物理卷。

```
# pvs -a -o +pv_tags /dev/sd[bcddefgh]
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
```

下面的命令在卷组 **taft** 中创建一个大小为 100GB 的镜像卷。

```
# 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
```

下面的命令显示用于镜像分支及 RAID 元数据子卷的设备。

```
# 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)
```

下面的命令扩展镜像卷的大小，使用 **cling** 分配策略表示需要使用有相同标签的物理卷扩展的镜像分支。

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

下面的命令显示使用与该分支有相同标签的物理卷扩展的镜像卷。注：忽略使用标签 **C** 的物理卷。

```
# 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)
```

#### 4.4.16. 缩小逻辑卷

要缩小逻辑卷，首先请卸载该文件系统。然后可使用命令 **lvreduce** 缩小该卷。缩小该卷后，重新挂载该文件系统。



## 警告

关键是要在缩小该卷前，减小文件系统或者其在该卷中所在位置的大小，否则可能会有丢失数据的风险。

缩小逻辑卷可释放一些卷组空间，将其分配给该卷组中的其他逻辑卷。

下面的示例将卷组 **vg00** 中的逻辑卷 **lv01** 减少 3 个逻辑扩展。

```
# lvreduce -l -3 vg00/lv01
```

### 4.4.17. 控制逻辑卷激活

可使用 **lvcreate** 或者 **lvchange** 命令的 **-k** 或者 **--setactivationskip {y|n}** 选项在正常激活命令中跳过为逻辑卷添加标签。该标签不适用于停用命令。

可使用 **lvs** 命令确定是否要为逻辑卷设定标志，下面的示例中显示 **k** 属性。

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

默认情况下，将精简快照卷标记为跳过激活。除使用标准的 **-ay** 或者 **--activate y** 选项外，还可使用由 **-K** 或者 **--ignoreactivationskip** 设定的 **k** 属性激活逻辑卷。

下面的命令激活精简逻辑卷。

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

使用 **lvcreate** 命令的 **-kn** 或者 **--setactivationskip n** 选项创建逻辑卷时可关闭持久 "activation skip" 标志。可使用 **lvchange** 命令的 **-kn** 或者 **--setactivationskip n** 选项关闭现有逻辑卷的标签。可使用 **-ky** 或者 **--setactivationskip y** 再打开该标志。

下面的命令创建没有 activation skip 标志的快照逻辑卷

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

下面的命令从快照逻辑卷中删除 activation skip 标签。

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

可使用 **/etc/lvm/lvm.conf** 文件中的 **auto\_set\_activation\_skip** 设置控制默认的跳过激活指定。

## 4.5. 使用过滤器控制 LVM 设备扫描

启动时，会运行 **vgscan** 命令，扫描系统中的块设备，查找 LVM 标签，以决定哪些是物理卷。同时还会读取元数据并建立卷组列表。物理卷的名称保存在系统中每个节点的缓存文件 **/etc/lvm/cache/.cache** 中。之后的命令可读取那个文件以避免重复扫描。

可通过在 **lvm.conf** 配置文件中设定过滤器控制设备 LVM 扫描。**lvm.conf** 文件中的过滤器由一系列简单正则表达式组成，应用于 **/dev** 目录中的设备名以决定接受或者拒绝每个找到的块设备。

下面的示例演示了使用过滤器控制 LVM 扫描的设备。注：这些示例不一定代表最佳实践，因为正则表达式与完整路径名完全匹配。例如：`a/loop/` 等同于 `a/.loop.*/`，并与 `/dev/solooperation/lvol1` 映射。

下面的过滤器添加所有找到的设备，这是默认行为，因为在配置文件中没有配置任何过滤器：

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

下面的过滤器删除 cdrom 设备，以避免在该驱动器中不包含任何介质时会造成延迟：

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

下面的过滤器添加所有回路并删除其他所有块设备：

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

下面的过滤器添加所有回路和 IDE，并删除其他所有块设备：

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

下面的过滤器只在第一个 IDE 驱动器中添加分区 8 并删除其他所有块设备：

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

### 注意

`lvm` 守护进程处于运行状态时，执行 `pvscan --cache device` 命令时不会应用 `/etc/lvm/lvm.conf` 文件中的 `filter` = 设置。要过滤设备，则需要使用 `global_filter` = 设定。LVM 不会打开无法进行全局过滤的设备，且再也不会对其进行扫描。可能会需要使用全局过滤器，例如：在 VM 中使用 LVM 设备，且不想让该物理主机扫描 VM 中设备的内容。

有关 `lvm.conf` 文件的详情，请查看 [附录 B, LVM 配置文件](#) 及 `lvm.conf(5)` man page。

## 4.6. 在线数据重新定位

可使用 `pvmove` 命令在系统处于使用状态时迁移数据。

`pvmove` 命令将要移动到扇区中的数据分散，并创建临时镜像以便移动每个扇区。有关 `pvmove` 命令操作的详情，请查看 `pvmove(8)` man page。

### 注意

要在集群中执行 `pvmove` 操作，应该确定已安装 `cmirror` 软件包，且 `cmirrord` 服务正在运行。

下面的命令将物理卷 `/dev/sdc1` 中所有已分配的空间移动到该卷组中可用的物理卷中：

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

下面的命令只移动逻辑卷 `MyLV` 的扩展。

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

因为执行 **pmove** 命令需要较长时间，可以在后台运行该命令，以免在前台显示进程更新。下面的命令在后台将物理卷 **/dev/sdc1** 中所有分配的扩展移动到 **/dev/sdf1**。

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

下面的命令以 5 秒为间隔报告该命令移动进程的百分比。

```
# pmove -i5 /dev/sdd1
```

## 4.7. 在集群的独立节点中激活逻辑卷

如果在集群环境中安装了 LVM，则有时可能会需要以独占方式在一个节点中激活逻辑卷。

要以独占方式在一个节点中激活逻辑卷，可使用 **lvchange -aey** 命令。另外，可以使用 **lvchange -aly** 命令在一个本地节点中以非独占方式激活逻辑卷。可稍后在附加节点中同时将其激活。

还可以使用 LVM 标签中独立节点中激活逻辑卷，如 [附录 C, LVM 对象标签](#)。还可以在该配置文件中指定节点激活，如 [附录 B, LVM 配置文件](#) 所述。

## 4.8. LVM 的自定义报告

可使用 **pvs**、**lvs** 和 **vgs** 命令生成简洁且可自定义的 LVM 对象报告。这些命令生成的报告包括每行一个对象的输出结果。每行包含与该对象有关的属性字段顺序列表。有五种方法可用来选择要报告的对象：即根据物理卷、卷组、逻辑卷、物理卷片段及逻辑卷片段报告。

以下小节提供：

- » 用来控制所生成报告格式的命令参数概述。
- » 可为每个 LVM 对象选择的字段列表。
- » 可用来对生成的报告进行排序的命令参数概述。
- » 指定报告输出结果单位简介。

### 4.8.1. 格式控制

是否使用 **pvs**、**lvs** 或者 **vgs** 命令可决定显示字段的默认设置及排序顺序。可使用下面的参数控制这些命令的输出结果：

- » 可使用 **-o** 参数将显示的字段改为默认值以外的内容。例如：下面的输出结果是 **pvs** 命令的默认显示内容（即显示物理卷信息）。

```
# 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
```

下面的命令只显示物理卷名称和大小。

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

- » 可将加号 (+) 于 -o 参数合用，在输出结果中附加一个字段。

下面的示例除显示默认字段外还显示该物理卷的 UUID。

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

- » 在命令中添加 -v 参数可显示额外的字段。例如：pvs -v 命令会在默认字段外显示 DevSize 和 PV UUID 字段。

```
# pvs -v
Scanning for physical volume names
PV          VG      Fmt  Attr  PSize  PFree  DevSize  PV  UUID
/dev/sdb1  new_vg  lvm2  a-    17.14G  17.14G  17.14G  onFF2w-1fLC-ughJ-
D9eB-M7iv-6XqA-dqGeXY
/dev/sdc1  new_vg  lvm2  a-    17.14G  17.09G  17.14G  Joqlch-yWSj-kuEn-
IdwM-01S9-X08M-mcpsVe
/dev/sdd1  new_vg  lvm2  a-    17.14G  17.14G  17.14G  yvfvZK-Cf31-j75k-
dECm-0RZ3-0dGW-tUqkCS
```

- » --noheadings 参数会取消标题行。这在编写脚本时有用。

下面的示例将 --noheadings 参数与 pv\_name 参数联合使用生成所有物理卷的列表：

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

- » --separator separator 参数使用 separator 分离各个字段。

下面的示例使用等号 (=) 分离 pvs 命令的默认输出字段。

```
# 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
```

要在使用 separator 参数时保持字段对齐，请将 separator 参数与 --aligned 参数配合使用。

```
# 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
```

可使用 **lvs** 或者 **vgs** 命令的 **-P** 参数显示有关失败卷的信息，否则该信息不会出现在输出结果中。有关这个参数字段的输出结果详情，请查看 [第 6.2 节“在失败的设备中显示信息”](#)。

有关完整显示参数列表，请查看 **pvs(8)**、**vgs(8)** 和 **lvs(8)** man page。

卷组字段可与物理卷（和物理卷片段）字段或者逻辑卷（和逻辑卷片段）字段混合，但物理卷和逻辑卷字段不能混合。例如：下面的命令会每行显示一个物理卷的输出结果。

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

## 4.8.2. 对象选择

本小节提供可用来使用 **pvs**、**vgs** 和 **lvs** 命令显示 LVM 对象信息的表格。

为方便起见，如果字段名称前缀与该命令的默认匹配，则可将其去掉。例如：在 **pvs** 命令中，**name** 的含义是 **pv\_name**，但在 **vgs** 命令中，会将 **name** 解读为 **vg\_name**。

执行下面的命令与执行 **pvs -o pv\_free** 等同。

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

### 注意

在之后的发行本中，**pvs**、**vgs** 和 **lvs** 输出结果中 **attribute** 字段中的字符数可能会增加。现有字符字段不会更改位置，但可在结尾处添加新字段。为具体属性字符编写脚本时应考虑这个因素，并根据字符与该字段起始位置的相对位置搜索字符，而不是根据其与该字段的截止位置的相对位置进行搜索。例如：要在 **lv\_attr** 字段的第九个字节搜索字符 **p**，可搜索字符串 "**^/.....p/**"，但不应搜索字符串 "**/\*p\$/**"。

## pvs 命令

[表 4.2 “pvs Display 字段”](#) 列出 **pvs** 目录的显示参数以及在标头显示中出现的字段名称及该字段的描述。

表 4.2. **pvs Display 字段**

参数	标头	描述
<b>dev_size</b>	DevSize	该物理卷所在底层设备的大小
<b>pe_start</b>	1st PE	底层设备中的第一个物理扩展偏差
<b>pv_attr</b>	Attr	物理卷状态：(a) 可分配；或者 (x) 导出。

参数	标头	描述
<b>pv_fmt</b>	Fmt	物理卷元数据格式 ( <b>lvm2</b> 或者 <b>lvm1</b> )
<b>pv_free</b>	PFree	物理卷中的剩余空间
<b>pv_name</b>	PV	物理卷名称
<b>pv_pe_alloc_count</b>	Alloc	已使用物理扩展数
<b>pv_pe_count</b>	PE	物理扩展数
<b>pvseg_size</b>	SSize	物理卷的片段大小
<b>pvseg_start</b>	起始	物理卷片段的起始物理扩展
<b>pv_size</b>	PSize	物理卷大小
<b>pv_tags</b>	PV 标签	附加到物理卷的 LVM 标签
<b>pv_used</b>	已使用	该物理卷中目前已使用的空间数量
<b>pv_uuid</b>	PV UUID	该物理卷的 UUID

**pvs** 命令默认显示以下字段：**pv\_name**, **vg\_name**, **pv\_fmt**, **pv\_attr**, **pv\_size**, **pv\_free**。该显示结果按 **pv\_name** 排序。

```
# 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
```

使用 **pvs** 命令的 **-v** 参数值默认显示中添加以下字段：**dev\_size**, **pv\_uuid**。

```
# pvs -v
Scanning for physical volume names
PV      VG      Fmt  Attr PSize   PFree   DevSize PV UUID
/dev/sdb1  new_vg  lvm2 a-    17.14G 17.14G  17.14G onFF2w-1fLC-ughJ-D9eB-
M7iv-6XqA-dqGeXY
/dev/sdc1  new_vg  lvm2 a-    17.14G 17.09G  17.14G Joqlch-yWSj-kuEn-IdwM-
01S9-X08M-mcpsVe
/dev/sdd1  new_vg  lvm2 a-    17.14G 17.13G  17.14G yfvZK-Cf31-j75k-dEcM-
0RZ3-0dGW-tUqkCS
```

可使用 **pvs** 命令的 **--segments** 参数显示每个物理卷片段的信息。一个片段就是一个扩展组。如果要了解逻辑卷是否碎片化，查看片段视图很有帮助。

**pvs --segments** 命令默认显示以下字段：**pv\_name**, **vg\_name**, **pv\_fmt**, **pv\_attr**, **pv\_size**, **pv\_free**, **pvseg\_start**, **pvseg\_size**。结果根据物理卷中的 **pv\_name** 和 **pvseg\_size** 排序。

```
# 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

可使用 **pvs -a** 命令查看已被 LVM 探测到但尚未初始化为 LVM 物理卷的设备。

# 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

## vgs 命令

表 4.3 “vgs Display 字段” 列出 **vgs** 命令的显示参数以及在标头显示中出现的字段名称及该字段的描述。

表 4.3. vgs Display 字段

参数	标头	描述
<b>lv_count</b>	#LV	卷组所包含逻辑卷数
<b>max_lv</b>	MaxLV	卷组允许的最大逻辑卷数 (0 表示无限大)
<b>max_pv</b>	MaxPV	卷组允许的最大物理卷数 (0 表示无限)
<b>pv_count</b>	#PV	定义该卷组的物理卷数
<b>snap_count</b>	#SN	卷组包含的快照数
<b>vg_attr</b>	Attr	卷组状态：(w) 可写入；(r) 只读；(z) 可重新定义大小；(x) 可导出；(p) 部分；以及 (c) 集群。
<b>vg_extent_count</b>	#Ext	卷组中的物理扩展数
<b>vg_extent_size</b>	Ext	卷组中的物理扩展大小
<b>vg_fmt</b>	Fmt	卷组的元数据格式 ( <b>lvm2</b> 或者 <b>lvm1</b> )
<b>vg_free</b>	VFree	卷组中剩余的可用空间大小
<b>vg_free_count</b>	可用	卷组中的可用物理扩展数
<b>vg_name</b>	VG	卷组名称
<b>vg_seqno</b>	Seq	代表卷组修改的次数
<b>vg_size</b>	VSize	卷组大小
<b>vg_sysid</b>	SYS ID	LVM1 System ID

参数	标头	描述
<b>vg_tags</b>	VG 标签	附加到卷组的 LVM 标签
<b>vg_uuid</b>	VG UUID	卷组的 UUID

**vgs** 命令默认显示以下字段：**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
```

使用 **vgs** 命令的 **-v** 参数值默认显示着添加以下字段：**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-ZKk0-0pM0-0118-
  nlw0-wwqd-fD5D32
```

## Ivs 命令

表 4.4 “**lvs 显示字段**” 列出了 **lvs** 命令的显示参数以及在标头显示中的字段名称以及该字段的描述。

表 4.4. **lvs 显示字段**

参数	标头	描述
<b>chunksize</b>	区块	快照卷的单位大小
<b>chunk_size</b>		
<b>copy_percent</b>	Copy%	镜像逻辑卷的同步百分比；也可在使用 <b>pv_move</b> 命令移动物理扩展时使用。
<b>devices</b>	设备	组成逻辑卷的底层设备：即物理卷、逻辑卷及起始物理扩展和逻辑扩展

参数	标头	描述
<b>lv_attr</b>	Attr	<p>逻辑卷状态。逻辑卷的属性字节如下：</p> <p>字节 1：卷类型：(m)镜像卷，(M) 没有初始同步的镜像卷，(o)原始卷，(O)附带合并快照的原始卷，(r)阵列，(R)没有初始同步的阵列，(s)快照，(S)合并快照，(p)pvmmove，(v)虚拟，(i)镜像或阵列映象，(l)未同步的镜像或阵列映象，(l)映象日志设备，(c)底层转换卷，(V)精简卷，(t)精简池，(T)精简池数据，(e)阵列或精简池元数据或池元数据备件，</p> <p>字节 2：授权：(w)写入，(r)只读，(R)非只读卷的只读激活</p> <p>字节 3：分配策略：(a)任意位置，(c)相邻，(i)继承，(l)紧邻，(n)常规。如果该卷目前锁定无法进行分配更改，则该字母会呈大写状态。例如执行 <b>pvmmove</b> 命令时。</p> <p>字节 4：(m)固定镜像</p> <p>字节 5：状态：(a)激活，(s)挂起，(l) 无效快照，(S) 无效挂起快照，(m) 快照合并失败，(M) 挂起快照合并失败，(d) 显示的映射设备不包含表格，(i) 显示的映射设备中包含停用表格。</p> <p>字节 6：设备 (o) 开启</p> <p>字节 7：目标类型：(m)镜像，(r) RAID，(s) 快照，(t) 精简，(u)未知，(v) 虚拟。这样可将有类似内核目标的逻辑卷分在一组。比如镜像映象、镜像日志以及镜像本身分为组 (m)，它们使用原始设备映射器内核驱动程序，使用 md raid 内核驱动程序的类似的 raid 设备则分组为 (r)。使用原始设备映射器驱动程序的快照则分组为 (s)，使用精简配置驱动程序的精简卷快照则分组为 (t)。</p> <p>字节 8：使用前，以设置为 0 的块覆盖新分配了数据的块。</p> <p>字节 9：卷正常情况：(p) 部分正常，(r) 需要刷新，(m) 存在映射错误，(w) 大部分写入。(p) 部分正常表示该系统中缺少这个逻辑卷使用的一个或多个物理卷。(p) 部分正常表示这个 RAID 逻辑卷使用的一个或多个物理卷有写入错误。该写入错误可能是由于该物理卷故障造成，也可能表示该物理卷正在出现问题。应刷新或替换该设备。(m) 存在映射错误表示 RAID 逻辑卷中有阵列不一致的部分。在 RAID 逻辑卷中启动 <b>check</b> 操作就会发现这些不一致之处。（取消该操作，可使用 <b>lvchange</b> 命令在 RAID 逻辑卷中执行 <b>check</b> 和 <b>repair</b>）。(w) 大部分写入表示已将 RAID 1 逻辑卷中的设备标记为大部分写入。</p> <p>字节 10：(k) 跳过激活：将该卷标记为在激活过程中跳过。</p>
<b>lv_kernel_major</b>	KMaj	逻辑卷的实际主要设备数（若未激活则为 -1）
<b>lv_kernel_minor</b>	KMIN	逻辑卷的实际次要设备数（若未激活则为 -1）
<b>lv_major</b>	Maj	逻辑卷的持久主要设备数（若未指定则为 -1）
<b>lv_minor</b>	Min	逻辑卷的持久次要设备数（若未指定则为 -1）
<b>lv_name</b>	LV	逻辑卷名称
<b>lv_size</b>	LSize	逻辑卷大小

<b>lv_kernel_major</b>	KMaj	逻辑卷的实际主要设备数（若未激活则为 -1）
<b>lv_kernel_minor</b>	KMIN	逻辑卷的实际次要设备数（若未激活则为 -1）
<b>lv_major</b>	Maj	逻辑卷的持久主要设备数（若未指定则为 -1）
<b>lv_minor</b>	Min	逻辑卷的持久次要设备数（若未指定则为 -1）
<b>lv_name</b>	LV	逻辑卷名称
<b>lv_size</b>	LSize	逻辑卷大小

参数	标头	描述
<b>lv_tags</b>	LV 标签	附加到逻辑卷的 LVM 标签
<b>lv_uuid</b>	LV UUID	逻辑卷的 UUID
<b>mirror_log</b>	Log	镜像日志所在设备
<b>modules</b>	模块	需要使用这个逻辑卷的对应内核设备映射器目标
<b>move_pv</b>	移动	使用 <b>pvmove</b> 命令创建的临时逻辑卷之源物理卷
<b>origin</b>	Origin	快照卷的原始设备
<b>regionsize</b>	区域	镜像逻辑卷的单元大小
<b>region_size</b>		
<b>seg_count</b>	#Seg	逻辑卷中的片段数
<b>seg_size</b>	SSize	逻辑卷的片段大小
<b>seg_start</b>	起始	逻辑卷中的片段偏移
<b>seg_tags</b>	Seg 标签	附加到逻辑卷片段的 LVM 标签
<b>segtype</b>	类型	逻辑卷的片段类型 (例如: 镜像、条带、线性)
<b>snap_percent</b>	Snap%	目前使用的快照卷百分比
<b>stripes</b>	#Str	逻辑卷中的条状卷或镜像卷数
<b>stripesize</b>	条带	条带逻辑卷的单元大小
<b>stripe_size</b>		

**lvs** 命令默认显示下面的字

段 : **lv\_name**、**vg\_name**、**lv\_attr**、**lv\_size**、**origin**、**snap\_percent**、**move\_pv**、**mirror\_log**、**copy\_percent**、**convert\_lv**。默认根据卷组中的 **vg\_name** 和 **lv\_name** 排序。

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

使用 **lvs** 命令的 **-v** 参数将下面的字段添加到默认显示

中 : **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-sr23-0jsI-LT03-nHLC-y8XW-EhC178
  newvgsnap1 new_vg    1 swi-a-  8.00M  -1  -1 253   5      lvol0     0.20
  1ye1OU-1cIu-o79k-20h2-ZGF0-qCJm-CfbsIx
```

可使用 **lvs** 命令的 **--segments** 参数显示默认栏信息，并强调片段信息。使用 **segments** 参数后，**seg** 前缀为自选项。**lvs --segments** 命令默认显示下面的字

段 : **lv\_name**、**vg\_name**、**lv\_attr**、**stripes**、**segtype**、**seg\_size**。默认显示是根据卷组的 **vg\_name** 和 **lv\_name**，以及该逻辑卷的 **seg\_start** 排序。如果逻辑卷碎片化，这个命令的输出结果会体现此状态。

```
# 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
```

使用 **lvs --segments** 命令的 **-v** 参数在默认显示着添加以下字段：**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
```

下面的示例演示了在已配置逻辑卷的系统中运行 **lvs** 命令的输出结果，以及运行附加 **segments** 参数的 **lvs** 命令的输出结果。

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

### 4.8.3. 将 LVM 报告排序

通常必须生成 **lvs**、**vgs** 或者 **pvs** 命令的整个输出结果，并在可将其排序并正确对齐前保存。可指定 **--unbuffered** 参数在生成后即刻显示未排序的输出结果。

要制定可替换的列排序顺序，请使用任意报告命令的 **-o** 参数。不一定要在输出结果中包含这些字段。

以下示例显示 **pvs** 命令的输出结果，其中显示内容包括物理卷名称、大小及可用空间。

```
# 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
```

以下示例显示同样的输出结果，但根据可用空间字段排序。

```
# 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
```

以下示例表示不需要显示用来排序的字段内容。

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

```
/dev/sdb1 17.14G
```

要显示反向排序，请在 **-O** 参数前指定的字段前添加 **-** 字符。

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

#### 4.8.4. 指定单位

要指定 LVM 报告显示的单位，请使用该报告命令的 **--units** 参数。可指定 (b)、(k)、(m)、(g)、(t)、(e)xabytes、(p) 和 (h)。默认显示为 human-readable (用户可读)。可在 **lvm.conf** 文件的 **global** 部分设置 **units** 参数覆盖默认值。

以下示例采用 MB 指定 **pvs** 命令的输出结果，而不是默认的 GB。

```
# 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
```

默认情况是以 2 的次方数 (1024 的倍数) 显示单位。可使用大写单位 (B、K、M、G、T、H) 以 1000 的倍数显示单位。

下面的命令采用 1024 的倍数 (即默认行为) 显示命令输出结果。

```
# 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
```

下面的命令采用 1000 的倍数显示命令输出结果。

```
# 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
```

还可以指定 (s) 扇区 (默认为 512 字节) 或自定义单位。

下面的示例以扇区数显示 **pvs** 命令的输出结果。

```
# 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
```

下面的示例以 4MB 为单位显示 **pvs** 命令的输出结果。

```
# 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
```

# 第 5 章 LVM 配置示例

本章提供了一些基本 LVM 配置示例。

## 5.1. 在三个磁盘中创建 LVM 逻辑卷

在这个示例中是要创建一个名为 `new_logical_volume` 的逻辑卷，它由磁盘 `/dev/sda1`、`/dev/sdb1` 和 `/dev/sdc1` 组成。

### 5.1.1. 创建物理卷

要在某个卷组中使用磁盘，需要将其标记为 LVM 物理卷。



#### 警告

这个命令会破坏 `/dev/sda1`、`/dev/sdb1` 和 `/dev/sdc1` 中的所有数据。

```
# pvcreate /dev/sda1 /dev/sdb1 /dev/sdc1
Physical volume "/dev/sda1" successfully created
Physical volume "/dev/sdb1" successfully created
Physical volume "/dev/sdc1" successfully created
```

### 5.1.2. 创建卷组

下面的命令可创建卷组 `new_vol_group`。

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

可以使用 `vgs` 命令显示新卷组的属性。

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

### 5.1.3. 创建逻辑卷

下面的命令可在卷组 `new_vol_group` 中创建逻辑卷 `new_logical_volume`。本示例创建的逻辑卷使用了卷组的 2GB 容量。

```
# lvcreate -L 2 G -n new_logical_volume new_vol_group
Logical volume "new_logical_volume" created
```

### 5.1.4. 创建文件系统

下面的命令在逻辑卷中创建了一个 GFS2 文件系统。

```
# mkfs.gfs2 -p lock_nolock -j 1 /dev/new_vol_group/new_logical_volume
```

```
This will destroy any data on /dev/new_vol_group/new_logical_volume.

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

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

Syncing...
All Done
```

下面的命令将挂载逻辑卷并报告文件系统磁盘空间用量。

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

## 5.2. 创建条带逻辑卷

本示例为创建一个名为 `striped_logical_volume` 的条带逻辑卷，并可在磁盘 `/dev/sda1`、`/dev/sdb1` 和 `/dev/sdc1` 间跨磁盘条带分配数据。

### 5.2.1. 创建物理卷

将卷组中要使用的磁盘标记为 LVM 物理卷。



#### 警告

这个命令会破坏 `/dev/sda1`、`/dev/sdb1` 和 `/dev/sdc1` 中的所有数据。

```
# pvcreate /dev/sda1 /dev/sdb1 /dev/sdc1
Physical volume "/dev/sda1" successfully created
Physical volume "/dev/sdb1" successfully created
Physical volume "/dev/sdc1" successfully created
```

### 5.2.2. 创建卷组

以下命令可创建卷组 `volgroup01`。

```
# vgcreate volgroup01 /dev/sda1 /dev/sdb1 /dev/sdc1
Volume group "volgroup01" successfully created
```

可以使用 `vgs` 命令显示新卷组的属性。

```
# vgs
VG          #PV #LV #SN Attr   VSize   VFree
volgroup01      3    0    0 wz--n- 51.45G 51.45G
```

### 5.2.3. 创建逻辑卷

以下命令可使用卷组 **volgroup01** 创建条带逻辑卷 **striped\_logical\_volume**。本示例创建的逻辑卷的大小为 2GB，有三个条带，每个条带的大小为 4Kb。

```
# lvcreate -i 3 -a I 4 -L 2 G -n striped_logical_volume volgroup01
Rounding size (512 extents) up to stripe boundary size (513 extents)
Logical volume "striped_logical_volume" created
```

### 5.2.4. 创建文件系统

下面的命令在逻辑卷中创建了一个 GFS2 文件系统。

```
# mkfs.gfs2 -p lock_nolock -j 1 /dev/volgroup01/stripped_logical_volume
This will destroy any data on /dev/volgroup01/stripped_logical_volume.

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

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

Syncing...
All Done
```

下面的命令将挂载逻辑卷并报告文件系统磁盘空间用量。

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

## 5.3. 分割卷组

在本示例中，现有卷组由三个物理卷组成。如果在物理卷中有足够的未使用空间，就可在不添加新磁盘的情况下创建新的卷组。

在初始设定中，逻辑卷 **mylv** 是从卷组 **myvol** 中分割出来的，它依次包含三个物理卷 **/dev/sda1**、**/dev/sdb1** 和 **/dev/sdc1**。

完成这个步骤后，卷组 **myvg** 将包含 **/dev/sda1** 和 **/dev/sdb1**。第二个卷组 **yourvg** 将包含 **/dev/sdc1**。

### 5.3.1. 确定剩余空间

可以使用 **pvscan** 命令确定目前在卷组中有多少可用的剩余空间。

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

### 5.3.2. 移动数据

可以使用 **pvmove** 将 **/dev/sdc1** 中所有使用的物理扩展移动到 **/dev/sdb1** 中。执行 **pvmove** 会消耗较长时候。

```
# 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%
```

移动数据后，可以看到 **/dev/sdc1** 中的所有空间都可用用了。

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

### 5.3.3. 分割卷组

要创建新卷组 **yourvg**，请使用 **vgsplit** 命令分割卷组 **myvg**。

在可以分割卷组前，必须停用该逻辑卷。如果挂载了文件系统，必须在停用该逻辑卷之前卸载文件系统。

可以使用 **lvchange** 命令或者 **vgchange** 命令使逻辑卷失活。以下命令可以使逻辑卷 **mylv** 失活并从卷组 **myvg** 中分割出卷组 **yourvg**，将物理卷 **/dev/sdc1** 移动到新的卷组 **yourvg** 中。

```
# lvchange -a n /dev/myvg/mylv
# vgsplit myvg yourvg /dev/sdc1
Volume group "yourvg" successfully split from "myvg"
```

可以使用 **vgs** 查看两个卷组的属性。

```
# vgs
VG      #PV #LV #SN Attr   VSize   VFree
myvg     2    1    0 wz--n-  34.30G 10.80G
yourvg   1    0    0 wz--n-  17.15G 17.15G
```

### 5.3.4. 创建新逻辑卷

创建新的卷组后，可以创建新的逻辑卷 **yourlv**。

```
# lvcreate -L 5 G -n yourlv yourvg
Logical volume "yourlv" created
```

### 5.3.5. 生成一个文件系统并挂载到新的逻辑卷

可以在新的逻辑卷中生成一个文件系统并挂载它。

```
# mkfs.gfs2 -plock_nolock -j 1 /dev/yourvg/yourlv
This will destroy any data on /dev/yourvg/yourlv.
```

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

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

```
Syncing...
All Done
```

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

### 5.3.6. 激活并挂载原来的逻辑卷

因为必须停用逻辑卷 **mylv**，所以需要在挂载它之前再次激活它。

```
# lvchange -a y /dev/myvg/mylv

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

## 5.4. 从逻辑卷中删除磁盘

本示例演示了如何从现有逻辑卷中删除磁盘。可以替换磁盘，也可以用这个磁盘作为不同卷的一部分。要删除磁盘，必须首先将 LVM 物理卷中的扩展移动到不同的磁盘或者一组磁盘中。

### 5.4.1. 将扩展移动到现有物理卷中

在本示例中，逻辑卷是在卷组 **myvg** 中的四个物理卷中进行分配的。

```
# 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
```

要移动 **/dev/sdb1** 的扩展，以便可以将其从卷组中删除。

如果在该卷组的其他物理卷中没有足够的剩余扩展，则可以在想要删除的设备中执行不带选项的 **pvmove** 命令，那么就会在其他设备中分配扩展。

```
# pvmove /dev/sdb1
/dev/sdb1: Moved: 2.0%
...
/dev/sdb1: Moved: 79.2%
...
/dev/sdb1: Moved: 100.0%
```

完成 **pvmove** 命令后，扩展的分配如下：

```
# 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
```

使用 **vgreduce** 命令从卷组中删除物理卷 **/dev/sdb1**。

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

现在可以物理删除这个磁盘或者将其分配给其他用户。

### 5.4.2. 将扩展移动到新磁盘中

在本示例中，在卷组 **myvg** 的三个物理卷中按以下方法分配逻辑卷：

```
# 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
```

要将 `/dev/sdb1` 的扩展移动到新设备 `/dev/sdd1` 中。

#### 5.4.2.1. 创建新物理卷

在 `/dev/sdd1` 中创建新物理卷。

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

#### 5.4.2.2. 在卷组中添加新物理卷

将 `/dev/sdd1` 添加到现有卷组 `myvg` 中。

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

#### 5.4.2.3. 移动数据

使用 `pvmove` 将数据从 `/dev/sdb1` 转移到 `/dev/sdd1` 中。

```
# pvmove /dev/sdb1 /dev/sdd1
/dev/sdb1: Moved: 10.0%
...
/dev/sdb1: Moved: 79.7%
...
/dev/sdb1: Moved: 100.0%
[root@tng3-1]# pvs -o+pv_used
PV          VG      Fmt  Attr PSize  PFree  Used
/dev/sda1    myvg   lvm2  a-    17.15G  7.15G  10.00G
/dev/sdb1    myvg   lvm2  a-    17.15G  17.15G   0
/dev/sdc1    myvg   lvm2  a-    17.15G  15.15G  2.00G
/dev/sdd1    myvg   lvm2  a-    17.15G  15.15G  2.00G
```

#### 5.4.2.4. 删除卷组中的旧物理卷

将数据从 `/dev/sdb1` 中移走后，就可以将其从卷组中删除。

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

现在可以将这个磁盘重新分配给其他卷组，或者将其从系统中删除。

### 5.5. 在集群中创建镜像 LVM 逻辑卷

在集群中创建镜像 LVM 逻辑卷与在单一节点中使用 **mirror** 片段类型创建镜像 LVM 逻辑卷所使用的命令和过程相同。但要在集群中创建镜像 LVM 逻辑卷，该集群和集群镜像必须处于运行状态，该集群必须可以仲裁，且必须在 **lvm.conf** 文件中正确设定锁定类型以便启用集群锁定，可以是直接锁定，也可以如[第 3.1 节“在集群中创建 LVM 卷”](#)所述使用 **lvmconf** 命令锁定。

在 Red Hat Enterprise Linux 7 中使用 Pacemaker 管理集群。只有与 Pacemaker 集群联合使用方可支持集群的 LVM 逻辑卷，且必须将其配置为集群资源。

下面是在集群中创建镜像 LVM 卷的步骤。

1. 安装集群软件及 LVM 软件包，启动该集群软件，并创建该集群。必须为该集群配置 fencing。文档《高可用附加组件管理》中提供了创建集群及为集群中的节点配置 fencing 的步骤示例。文档《高可用附加组件管理》还提供更多有关集群组件配置的详情。
2. 要创建可为集群中所有节点共享的镜像逻辑卷，则必须在该集群每个节点的 **lvm.conf** 文件中正确设定锁定类型。默认情况下是将锁定类型设定为 **local**。要更改这个设置，请在集群的每个节点中执行以下命令启用集群的锁定：

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

3. 为集群设定 **d1m** 资源。将该资源作为克隆的资源创建，以便其可在该集群的每个节点中运行。

```
# pcs resource create d1m ocf:pacemaker:controld op monitor
interval=30s on-fail=fence clone interleave=true ordered=true
```

4. 将 **clvmd** 配置为集群资源。创建资源时只将 **d1m** 资源作为克隆资源创建，以便其可在该集群的所有节点中运行。注：必须设定 **with\_cmirrord=true** 参数，以便在所有运行 **clvmd** 的节点中启用 **cmirrord** 守护进程。

```
# pcs resource create clvmd ocf:heartbeat:clvm with_cmirrord=true
op monitor interval=30s on-fail=fence clone interleave=true
ordered=true
```

如果已配置 **clvmd** 资源，但没有指定 **with\_cmirrord=true** 参数，则可以使用下面的命令更新该资源以便其包含该参数：

```
# pcs resource update clvmd with_cmirrord=true
```

5. 设定 **clvmd** 和 **d1m** 相依性及启动顺序。必须在 **d1m** 之后启动 **clvmd**，且必须在 **d1m** 所在的同一节点中运行。

```
# pcs constraint order start d1m-clone then clvmd-clone
# pcs constraint colocation add clvmd-clone with d1m-clone
```

6. 创建镜像。第一步是创建物理卷。下面的命令创建三个物理卷，其中两个物理卷将作为镜像的分支，第三个物理卷将包含镜像日志。

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

7. 创建卷组。这个示例创建了由前一步创建的三个物理卷组成的卷组 **vg001**。

```
# vgcreate vg001 /dev/xvdb1 /dev/xvdb2 /dev/xvdc1
Clustered volume group "vg001" successfully created
```

请注意：**vgcreate** 命令的输出结果表示卷组是集群的。可以使用 **vgs** 命令确认卷组是集群的，该命令可显示卷组属性。如果卷组是集群的，它会显示 c 属性。

```
vgs vg001
VG          #PV #LV #SN Attr   VSize   VFree
vg001        3    0    0 wz--nc 68.97G 68.97G
```

- 创建镜像逻辑卷。这个示例在卷组 **vg001** 中创建逻辑卷 **mirrorlv**。这个卷有一个镜像分支。这个示例指定使用物理卷的哪些扩展作为逻辑卷。

```
# lvcreate --type mirror -l 1000 -m 1 vg001 -n mirrorlv
/dev/xvdb1:1-1000 /dev/xvdb2:1-1000 /dev/xvdc1:0
Logical volume "mirrorlv" created
```

可以使用 **lvs** 命令显示创建镜像的过程。以下示例显示 47% 的镜像被同步，然后是 91%，当镜像完成时为 100%。

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

在系统日志中会记录镜像完成：

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

- 可以使用 **lvs** 命令的 **-o +devices** 选项显示镜像的配置，其中包括组成镜像分支的设备。可以看到在这个示例中逻辑卷是由两个线性映像和一个日志组成。

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

```
[mirrorlv_mimage_1] vg001      iwi-ao  3.91G
/dev/xvdb2(1)
[mirrorlv_mlog]     vg001      lwi-ao  4.00M
/dev/xvdc1(0)
```

可以使用 **lvs** 命令的 **seg\_pe\_ranges** 选项显示数据布局。可以使用这个选项验证您的布局是正确的冗余。这个命令的输出会显示 PE 范围，格式与 **lvcreate** 和 **lvresize** 命令的输入格式相同。

```
# lvs -a -o +seg_pe_ranges --segments
PE Ranges
mirrorlv_mimage_0:0-999 mirrorlv_mimage_1:0-999
/dev/xvdb1:1-1000
/dev/xvdb2:1-1000
/dev/xvdc1:0-0
```

## 注意

有关从 LVM 镜像卷的一个分支失败中恢复的详情请参考 [第 6.3 节“恢复 LVM 镜像错误”](#)。

## 第 6 章 LVM 故障排除

本章提供了对不同 LVM 问题进行故障排除的操作方法。

### 6.1. 故障排除诊断

如果某个命令没有按照预期执行，则可以使用以下方法收集诊断信息：

- » 使用任意命令的 **-v**、**-vv**、**-vvv** 或者 **-vvvv** 参数提高输出信息的详细程度。
- » 如果问题与逻辑卷激活有关，请在配置文件的‘log’部分设定‘activation = 1’，并在运行命令时使用 **-vvvv** 选项。检查输出结果后，确定将此参数重新设为 0，以防止在机器可用内存较少时出现机器锁定现象。
- » 运行 **lvm dump** 命令可为诊断提供信息转储。有关详情请参考 **lvm dump(8)** man page。
- » 执行 **lvs -v**、**pvs -a** 或者 **dmsetup info -c** 命令以获得额外的系统信息。
- » 检查 **/etc/lvm/backup** 文件中最后的元数据备份和 **/etc/lvm/archive** 中的归档版本。
- » 运行 **lvm dumpconfig** 命令检查现有配置信息。
- » 检查 **/etc/lvm** 目录中的 **.cache** 文件，了解包含物理卷设备的记录。

### 6.2. 在失败的设备中显示信息。

可以使用 **lvs** 或者 **vgs** 命令的 **-P** 参数显示那些没有出现在输出结果中的失败卷的信息。该参数甚至允许一些内部元数据不完全统一时的操作。例如：如果组成卷组 **vg** 的某个设备失败，**vgs** 命令可能会显示以下输出信息。

```
# vgs -o +devices
Volume group "vg" not found
```

如果已为 **vgs** 指定了 **-P** 选项，那么该卷组虽仍然不可用，但可以看到更多有关失败设备的信息。

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

在这个示例中，失败的设备导致卷组中的线性和条带逻辑卷均失败。不带 **-P** 选项的 **lvs** 命令会显示以下输出结果。

```
# lvs -a -o +devices
Volume group "vg" not found
```

使用 **-P** 选项显示失败的逻辑卷。

```
# lvs -P -a -o +devices
Partial mode. Incomplete volume groups will be activated read-only.
LV VG Attr LSize Origin Snap% Move Log Copy% Devices
linear vg -wi-a- 20.00G unknown
```

```
device(0)
  stripe vg    -wi-a- 20.00G                               unknown
device(5120),/dev/sda1(0)
```

下面的例子显示在镜像逻辑卷的一个分支出错时，带 **-P** 选项的 **pvs** 和 **lvs** 命令的输出结果。

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

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

## 6.3. 恢复 LVM 镜像错误

本小节提供一个恢复示例，即从由于物理卷底层设备宕机，同时将 **mirror\_log\_fault\_policy** 参数设定为 **remove**，从而造成 LVM 镜像卷的一支失败，需要手动恢复的情况。有关设定 **mirror\_log\_fault\_policy** 参数的详情，请查看 [第6.3节“恢复LVM镜像错误”](#)。

当一个镜像分支失败时，LVM 将镜像卷转换成线性卷，此时可和以前一样继续操作但没有镜像冗余。此时可以在系统中添加新磁盘来替换物理设备，并重建该镜像。

以下命令创建将作为镜像使用的物理卷。

```
# pvcreate /dev/sd[abcdefg][12]
Physical volume "/dev/sda1" successfully created
Physical volume "/dev/sda2" successfully created
Physical volume "/dev/sdb1" successfully created
Physical volume "/dev/sdb2" successfully created
Physical volume "/dev/sdc1" successfully created
Physical volume "/dev/sdc2" successfully created
Physical volume "/dev/sdd1" successfully created
Physical volume "/dev/sdd2" successfully created
Physical volume "/dev/sde1" successfully created
Physical volume "/dev/sde2" successfully created
Physical volume "/dev/sdf1" successfully created
Physical volume "/dev/sdf2" successfully created
```

```
Physical volume "/dev/sdg1" successfully created
Physical volume "/dev/sdg2" successfully created
Physical volume "/dev/sdh1" successfully created
Physical volume "/dev/sdh2" successfully created
```

以下命令创建卷组 **vg** 和镜像卷 **groupfs**。

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

可以使用 **lvs** 命令验证作为镜像分支的镜像卷及底层设备布局，以及镜像日志。请注意：在第一个示例中，镜像还没有被完全同步。应该在 **Copy%** 字段显示 100.00 之后才继续操作。

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

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

在这个示例中，镜像 **/dev/sda1** 的主要分支失败。任何镜像卷的写入操作都会让 LVM 探测失败的镜像。出现这种情况后，LVM 会将镜像转换成单一线性卷。在这个示例中是因为执行了 **dd** 命令。

```
# dd if=/dev/zero of=/dev/vg/groupfs count=10
10+0 records in
10+0 records out
```

可以使用 **lvs** 命令确定该设备现在已经是线性设备了。因为是失败的磁盘，所以会发生 I/O 错误。

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

此时应该仍然可以使用逻辑卷，但没有镜像冗余。

要重建镜像卷，则需要替换坏的驱动器，并重新创建该物理卷。如果使用同一个磁盘而不是换一个新磁盘，则在运行 **pvcreate** 命令时将看到“inconsistent”的警告提示。执行 **vgreduce --removemissing** 命令即可避免出现该警告。

```
# pvcreate /dev/sdi[12]
Physical volume "/dev/sdi1" successfully created
Physical volume "/dev/sdi2" successfully created

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

下面可以使用新的物理卷扩展原来的卷组。

```
# vgextend vg /dev/sdi[12]
Volume group "vg" successfully extended

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

将线性卷转换回其原始镜像状态。

```
# lvconvert -m 1 /dev/vg/groupfs /dev/sdi1 /dev/sdb1 /dev/sdc1
Logical volume mirror converted.
```

可以使用 **lvs** 命令验证已恢复到镜像状态。

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

## 6.4. 恢复物理卷元数据

如果不小心覆盖或者破坏了卷组物理卷元数据区域，则会看到出错信息显示元数据区域不正确，或者系统无法使用特定的 UUID 找到物理卷。此时可能需要通过在物理卷的元数据区域写入新的元数据来恢复物理卷数据，指定相同的 UUID 作为丢失的元数据。



### 警告

在正常的 LVM 逻辑卷中应该不会进行这个操作过程。如果指定了不正确的 UUID，则会丢失您的数据。

下面的例子显示排序的输出结果，可以看到您的元数据是丢了还是被破坏了。

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

通过查看 **/etc/lvm/archive** 目录，应该可以找到被覆盖的物理卷 UUID。在文件 **VolumeGroupName\_xxxx.vg** 中查找该卷组最后的有效归档 LVM 元数据。

另外，还可以找到失活的卷并设定 **partial** (-P) 选项，这样就可以找到丢失的被破坏的物理卷的 UUID。

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

使用 **pvccreate** 的 **--uuid** 和 **--restorefile** 选项恢复物理卷。下面的例子使用上述 UUID **FmGRh3-zhok-iVI8-7qTD-S5BI-MAEN-NYM5Sk** 将 **/dev/sdh1** 设备标记为物理卷。这个命令使用 **VG\_00050.vg** 中的元数据信息，即使用该卷组最新的好归档元数据恢复物理卷标签。**restorefile** 参数让 **pvccreate** 生成与卷组中旧的物理卷兼容的新物理卷，确保新的元数据不会被放在包含旧的物理卷元数据的区域（这有可能发生。例如：如果原始 **pvccreate** 命令使用了控制元数据放置位置的命令行参数，或者使用了应用不同默认选项的软件版本创建物理卷时，就会发生这种情况）。**pvccreate** 命令仅覆盖 LVM 元数据区域，不会影响现有的数据区域。

```
# pvccreate --uuid "FmGRh3-zhok-iVI8-7qTD-S5BI-MAEN-NYM5Sk" --
restorefile /etc/lvm/archive/VG_00050.vg /dev/sdh1
Physical volume "/dev/sdh1" successfully created
```

然后就可以使用 **vgcfgrestore** 命令恢复卷组的元数据。

```
# vgcfgrestore VG
Restored volume group VG
```

现在可以显示逻辑卷。

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

下面的命令激活卷并显示激活的卷。

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

如果磁盘中的 LVM 元数据至少使用了覆盖其数据的空间大小，这个命令可以恢复物理卷。如果覆盖元数据的数据超过了元数据区域，那么就有可能损害到卷中的数据。此时可以使用 **fsck** 命令修复那些数据。

## 6.5. 替换丢失的物理卷

如果物理卷失败或者需要替换，则可以标记一个新的物理卷来替换那个已经在现有卷组中丢失的物理卷，过程与修复物理卷元数据相同，详见 [第6.4节“恢复物理卷元数据”](#)。您可以使用 **vgdisplay** 命令的 **--partial** 和 **--verbose** 选项显示不再出现的物理卷的 UUID 和大小。如果想要使用同样大小的物理卷进行替换，则可以使用 **pvccreate** 命令，选项为 **--restorefile** 和 **--uuid** 初始化与丢失的物理卷有相同 UUID 的新设备。然后就可以使用 **vgcfgrestore** 命令恢复卷组的元数据。

## 6.6. 从卷组中删除丢失的物理卷。

如果丢失了物理卷，则可以用 **vgchange** 命令的 **--partial** 选项激活卷组中剩下的物理卷。可以使用 **vgreduce** 命令的 **--removemissing** 选项删除所有使用卷组中那些物理卷的逻辑卷。

建议运行 **vgreduce** 命令，使用 **--test** 选项确定要破坏的数据。

如果立即使用 **vgcfgrestore** 命令将卷组的元数据恢复到之前的状态，则与大多数 LVM 操作一样，**vgreduce** 命令在某种意义上是可逆的。例如：如果使用 **vgreduce** 命令的 **--removemissing** 参数，而不带 **--test** 参数，就可以找到要保留的已删除逻辑卷。此时仍可用替换物理卷，并使用另一个 **vgcfgrestore** 命令将卷组返回到之前的状态。

## 6.7. 逻辑卷没有足够的可用扩展

虽然根据 **vgdisplay** 或者 **vgs** 命令的输出结果，您认为有足够的扩展，但此时如果创建逻辑卷，则可能得到这样的出错信息“Insufficient free extents（没有足够的可用扩展）”。这是因为这些命令采用十进制处理数字以便提供可读的输出结果。要指定确切的大小，请使用可用物理扩展计数，而不是用字节来计算逻辑卷的大小。

在默认情况下，**vgdisplay** 命令的输出结果提示可用物理扩展的行。

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

另外还可以使用 **vgs** 的 **vg\_free\_count** 和 **vg\_extent\_count** 参数显示可用扩展和扩展的总数。

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

有 8780 个可用物理扩展，现在可以运行以下命令，使用小写 l 选项使用扩展而不是字节作为单位：

```
# lvcreate -l 8780 -n testlv testvg
```

这样就会使用卷组中的所有可用扩展。

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

另外还可用通过使用 **lvcreate** 命令的 **-l** 参数，使用卷组中一定比例的剩余可用空间扩大逻辑卷。详情请参考 [第 4.4.1 节“创建线性逻辑卷”](#)。

# 附录 A. 设备映射器 (Device Mapper)

Device Mapper 是一个为卷管理提供构架的内核驱动程序。它提供一个创建映射设备的通用方法，该设备可作为逻辑卷使用。它并不具体了解卷组或者元数据格式。

Device Mapper 是很多高级技术的基础。除 LVM 外，Device Mapper 多路径和 **dmraid** 命令也使用 Device Mapper。Device Mapper 的应用程序界面是 **ioctl** 系统调用。用户界面是 **dmsetup** 命令。

LVM 逻辑卷是使用 Device Mapper 激活的。每个逻辑卷都会被转换成映射的设备，每个片段都会被转换成映射列表中描述设备的一行。Device Mapper 支持各种映射目标，包括线性映射、条带映射和错误映射。例如：两张磁盘可连成一个带一对线性映射的逻辑卷，每个映射对应一个磁盘。当 LVM 创建卷时，它会生成一个底层 device-mapper 设备，使用 **dmsetup** 命令可对其进行查询。有关映射列表中设备格式的详情，请参考 [第 A.1 节“设备表映射”](#)。有关使用 **dmsetup** 命令查询某个设备的详情，请参考 [第 A.2 节“dmsetup 命令”](#)。

## A.1. 设备表映射

映射设备是根据指定如何使用支持的设备表映射映射设备的每个逻辑扇区的表格定义。映射设备表由以下格式行组成：

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

在设备映射表的第一行中，**start** 参数必须等于 0。每行中的 **start + length** 参数必须与下一行的 **start** 相等。在映射表中指定哪个映射参数取决于在该行中指定的 **mapping** 类型。

Device Mapper 的大小总是以扇区 (512 字节) 为单位指定。

将某个设备指定为 Device Mapper 的映射参数后，就可以在该文件系统（比如 `/dev/hda`）中根据其设备名称进行参考，或者在使用 **major:minor** 格式时根据主要号码和次要号码进行参考。首选 **major:minor** 格式，因为这样可避免查找路径名。

以下显示了某设备的映像表示例。在这个表中有四个线性目标：

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

每行的前两个参数是片段起始块以及该片段的长度。下一个关键字是映射目标，在此示例中全部是 **linear**。该行的其余部分由用于 **linear** 目标的参数组成。

下面的小节描述了以下映射的格式：

- » 线性映射
- » 条带映射
- » 镜像映射
- » 快照和原始快照映射
- » 错误映射
- » 零映射
- » 多路径映射

» 加密映射

### A.1.1. 线性映射目标

线性映射目标将块的连续行映射到另一个块设备中。线性目标的格式如下：

```
start length linear device offset
```

**start**

虚拟设备中的起始块

**length**

这个片段的长度

**device**

块设备，被该文件系统中的设备名称或者主号码和副号码以 **major:minor** 的格式参考

**offset**

该设备中映射的起始偏移

以下示例显示了起始块位于虚拟设备 0，片段长度为 1638400，major:minor 号码对为 8:2，起始偏移为 41146992 的线性目标。

```
0 16384000 linear 8:2 41146992
```

以下示例演示了将设备参数指定为 **/dev/hda** 的线性目标。

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

### A.1.2. 条带映射目标

条带映射目标支持所有跨物理设备的条带。它使用条带数目和成条的组集大小以及设备名称和扇区对作为参数。条状目标的格式如下：

```
start length striped #stripes chunk_size device1 offset1 ... deviceN
offsetN
```

每个条块都有一组 **device** 和 **offset** 参数。

**start**

虚拟设备中的起始块

**length**

这个片段的长度

**#stripes**

虚拟设备的条数

**chunk\_size**

切换到下一个条之前写入每个条的扇区数，必须至少是内核页面大小的两倍

#### **device**

块设备，可被该文件系统中的设备名称或者主号码和副号码以格式 **major:minor** 参考。

#### **offset**

该设备中映射的起始偏移

以下示例显示了一个有三个条，且组集大小为 128 的条状目标：

```
0 73728 striped 3 128 8:9 384 8:8 384 8:7 9789824
```

**0**

虚拟设备中的起始块

**73728**

这个片段的长度

**striped 3 128**

三个设备中组集大小为 128 块的条带

**8:9**

第一个设备的 major:minor 号码

**384**

第一个设备中映射的起始偏移

**8:8**

第二个设备的 major:minor 号码

**384**

第二个设备中映射的起始偏移

**8:7**

第三个设备的 major:minor 号码

**9789824**

第三个设备中映射的起始偏移

以下示例显示了含有两个 256KiB 条，使用文件系统中的设备名称而不是主号码和副号码指定设备参数的条状目标。

```
0 65536 striped 2 512 /dev/hda 0 /dev/hdb 0
```

### A.1.3. 镜像映射目标

镜像映射目标支持镜像逻辑设备映像。镜像目标格式如下：

```
start length mirror log_type #logargs logarg1 ... logargN #devs device1
offset1 ... deviceN offsetN
```

**start**

虚拟设备中的起始块

**length**

这个片段的长度

**log\_type**

可能的日志类型及其参数如下：

**core**

镜像是本地的，镜像日志保存在核内存中。这个日志类型有 1-3 个参数：

*regionsize [[no]sync] [block\_on\_error]*

**disk**

镜像是本地的，镜像日志保存在磁盘中。这个日志类型可附加 2-4 个参数：

*logdevice regionsize [[no]sync] [block\_on\_error]*

**clustered\_core**

镜像是集群的，镜像日志保存在 core 内存中。这个日志类型可附加 2-4 个参数：

*regionsize UUID [[no]sync] [block\_on\_error]*

**clustered\_disk**

镜像是集群的，镜像日志保存在磁盘中。这个日志类型可附加 3-5 个参数：

*logdevice regionsize UUID [[no]sync] [block\_on\_error]*

LVM 保存一个小日志用来跟踪与该镜像或者多个镜像同步的区域。*regionsize* 参数指定这些区域的大小。

在集群环境中，*UUID* 参数是与镜像日志设备关联的特定识别符，以便可通过该集群维护日志状态。

可使用自选的 **[no]sync** 参数将镜像指定为 "in-sync" 或者 "out-of-sync"。**block\_on\_error** 参数是用来让镜像对错误做出响应，而不是忽略它们。

**#log\_args**

将在映射中指定的日志参数数目

**logargs**

镜像的日志参数；提供的日志参数数目是由 **#log-args** 参数指定的，且有效日志参数由 **log\_type** 参数决定。

**#devs**

镜像中的分支数目；为每个分支指定一个设备和一个偏移

**device**

每个镜像分支的块设备，使用该文件系统中的设备名称或者主号码和副号码以 *major:minor* 的格式参考。每个镜像分支都有一个块设备和误差，如 *#devs* 参数中所示。

#### **offset**

设备中映射的起始偏移。每个镜像分支都有一个块设备和偏移，如 *#devs* 参数中所示。

下面的示例演示了集群映像的镜像映像目标，并将该映像的映像日志保存在磁盘中。

```
0 52428800 mirror clustered_disk 4 253:2 1024 UUID block_on_error 3 253:3
0 253:4 0 253:5 0
```

**0**

虚拟设备中的起始块

**52428800**

这个片段的长度

**mirror clustered\_disk**

镜像目标，其日志类型指定该镜像为集群镜像，并在磁盘中保存期镜像日志。

**4**

附带 4 个镜像日志参数

**253:2**

日志设备的 major:minor 号码

**1024**

镜像日志用来跟踪进行同步的区域大小

**UUID**

镜像日志设备的 UUID，该设备上用来维护集群吞吐量信息。

**block\_on\_error**

镜像应该对错误进行响应

**3**

镜像中的分支数

**253:3 0 253:4 0 253:5 0**

构成镜像每个分支的设备的 major:minor 号码和偏移

### A.1.4. 快照和原始快照映射目标

创建某个卷的第一个 LVM 快照时，使用了四个 Device Mapper 设备：

1. 包含源卷原始映射表的**线性**映射设备。
2. 作为源卷即写即拷 (copy-on-write, COW) 设备使用的**有线性**映射的设备；每次写入时，会将原始数据保存在每个快照的 COW 设备中以便保持不更改可见内容（直到 COW 设备写满为止）。

3. 附带快照映射及 #1 和 #2 的设备，它是可见快照卷。
4. “原始”卷（使用源卷使用的设备号码），其列表由来自设备 #1 的“snapshot-origin”映射替换。

用来创建这些设备的固定命名方案，例如：您可以使用以下命令生成名为 **base** 的 LVM 卷以及基于该卷的名为 **snap** 快照卷。

```
# lvcreate -L 1G -n base volumeGroup
# lvcreate -L 100M --snapshot -n snap volumeGroup/base
```

这样会生成四个设备，可以使用下面的命令查看：

```
# dmsetup table|grep volumeGroup
volumeGroup-base-real: 0 2097152 linear 8:19 384
volumeGroup-snap-cow: 0 204800 linear 8:19 2097536
volumeGroup-snap: 0 2097152 snapshot 254:11 254:12 P 16
volumeGroup-base: 0 2097152 snapshot-origin 254:11

# ls -ll /dev/mapper/volumeGroup-
brw----- 1 root root 254, 11 29 ago 18:15 /dev/mapper/volumeGroup-
base-real
brw----- 1 root root 254, 12 29 ago 18:15 /dev/mapper/volumeGroup-
snap-cow
brw----- 1 root root 254, 13 29 ago 18:15 /dev/mapper/volumeGroup-snap
brw----- 1 root root 254, 10 29 ago 18:14 /dev/mapper/volumeGroup-base
```

**snapshot-origin** 目标的格式如下：

```
start length snapshot-origin origin
```

**start**

虚拟设备中的起始块

**length**

这个片段的长度

**origin**

快照基础卷

**snapshot-origin** 通常有一个或者多个基于它的快照。会将读取操作直接与后备设备映射。每次写入时，会将原始数据保存在每个快照的 COW 设备中以便使其不更改可见内容（直到 COW 设备写满为止）。

**快照** 目标的格式如下：

```
start length snapshot origin COW-device P|N chunkszie
```

**start**

虚拟设备中的起始块

**length**

这个片段的长度

***origin***

快照基础卷

***cow-device***

保存更改组集数据的设备

**P|N**

P (持久) 或者N (不持久) ; 表示是否可在重启后保留快照。对于瞬时快照 (N) 必须将 less metadata 保存在磁盘中；内核可将其保存在内存中。

***chunksize***

将保存到 COW 设备中的有数据块更改的扇区大小

以下示例显示了起始设备为 254:11 的 **snapshot-origin** 目标。

```
0 2097152 snapshot-origin 254:11
```

以下示例显示了起始设备为 254:11、COW 设备为 254:12 的 **snapshot-origin** 目标。这个快照设备在重启后仍然保留，且保存在 COW 设备中的数据块大小为 16 个扇区。

```
0 2097152 snapshot 254:11 254:12 P 16
```

**A.1.5. 错误映射目标**

如果有错误映射目标，任何对映射的扇区的 I/O 操作会失败。

错误映射可用来进行测试。要测试某个设备在失败后如何动作，可以创建一个有坏扇区的设备映射，或者您可以换出一个镜像分支并用错误目标替换。

错误目标可用于出错的设备，是一种避免超时并在实际设备中重试的方法。失败后重新部署 LVM 元数据时可将其作为中间目标使用。

错误映射目标除 *start* 和 *length* 参数外不使用其它参数。

以下示例显示的是错误目标。

```
0 65536 error
```

**A.1.6. 零映射目标**

零映射目标是与 `/dev/zero` 等同的块设备。对这个映射的读取操作会返回零块。写入这个映射的数据会被丢弃，但写入操作会成功。零映射目标除 *start* 和 *length* 参数外没有其它参数。

以下示例显示了大小为 16Tb 设备的零目标。

```
0 65536 zero
```

**A.1.7. 多路径映射目标**

多路径映射目标支持多路径的设备的映射。多路径目标的格式如下：

```
start length multipath #features [feature1 ... featureN] #handlerargs
[handlerarg1 ... handlerargN] #pathgroups pathgroup pathgroupargs1 ...
pathgroupargsN
```

每个路径组群都有一组 **pathgroupargs** 参数。

### **start**

虚拟设备中的起始块

### **length**

这个片段的长度

### **#features**

在那些功能之后是多路径功能的数目。如果这个参数是 0，则没有 **feature** 参数，且下一个设备映射参数为 **#handlerargs**。目前只有一个功能可使用 **multipath.conf** 文件中的 **features** 属性设置，即 **queue\_if\_no\_path**。这说明如果没有路径可用，则将这个多路径的设备设定为队列 I/O。

在下面的示例中，只在所有尝试使用这些路径测试失败后，才将 **multipath.conf** 文件的 **no\_path\_retry** 属性设定为 I/O 操作队列。在这个示例中，只有使用路径检查程序在进行指定次数的检查后才会显示该映射。

```
0 71014400 multipath 1 queue_if_no_path 0 2 1 round-robin 0 2 1
66:128 \
1000 65:64 1000 round-robin 0 2 1 8:0 1000 67:192 1000
```

在所有路径检查程序完成指定数目的检查并失败后，会出现如下映射。

```
0 71014400 multipath 0 0 2 1 round-robin 0 2 1 66:128 1000 65:64
1000 \
round-robin 0 2 1 8:0 1000 67:192 1000
```

### **#handlerargs**

那些参数后是硬件处理程序参数的数目。硬件处理程序指定在切换路径组或者处理 I/O 错误时用来执行具体硬件的动作。如果将其设定为 0，那么下一个参数则为 **#pathgroups**。

### **#pathgroups**

路径组的数目。路径组是一组 multipathed 设备进行负载平衡的路径。每个路径组都有一组 **pathgroupargs** 参数。

### **pathgroup**

下一个要尝试的路径组。

### **pathgroupargs**

每个路径组均由以下参数组成：

```
pathselector #selectorargs #paths #pathargs device1 ioreqs1 ...
deviceN ioreqsN
```

路径组中的每个路径都有一组路径参数。

***pathselector***

指定用来决定使用这个路径组中的哪个路径进行下一个 I/O 操作的算法。

**#*selectorargs***

在多路径映射中这个参数后的路径选择程序参数的数目。目前，这个参数的值总是 0。

**#*paths***

这个路径组中的路径数目。

**#*pathargs***

在这个组群中为每个路径指定的路径参数数目。目前，这个数值总是 1，即 **ioreqs** 参数。

***device***

该路径的块设备，使用主号码和副号码以 **major:minor** 格式参考

***ioreqs***

切换到当前组群的下一个路径前路由到这个路径的 I/O 请求数目。

图 A.1 “多路径映射目标” 显示带两个路径组群的多路径目标格式。

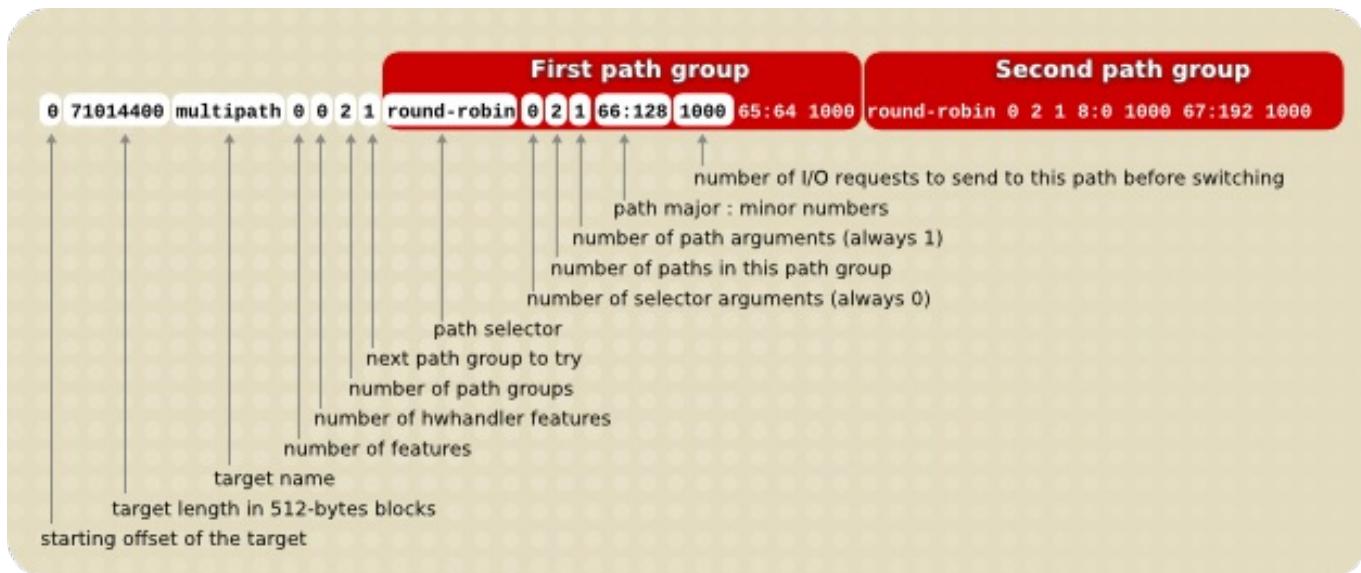


图 A.1. 多路径映射目标

以下示例显示对同一个多路径设备的一个纯故障排除目标定义。在这个目标中有四个路径组，其中每个路径组只有一个路径，以便多路径的设备每次只能使用一个路径。

```
0 71014400 multipath 0 0 4 1 round-robin 0 1 1 66:112 1000 \
round-robin 0 1 1 67:176 1000 round-robin 0 1 1 68:240 1000 \
round-robin 0 1 1 65:48 1000
```

下面的示例显示为同一个多路径设备完全展开（多总线）目标定义。在这个目标中只有一个路径组，其中包含所有路径。在这个设定中，多路径将所有负载平均分配到所有路径中。

```
0 71014400 multipath 0 0 1 1 round-robin 0 4 1 66:112 1000 \
67:176 1000 68:240 1000 65:48 1000
```

有关多路径的详情请参考《使用设备映射器多路径》文档。

### A.1.8. 加密映射目标

加密目标会加密通过指定设备的所有数据。它使用内核 Crypto API。

加密目标的格式如下：

```
start length crypt cipher key IV-offset device offset
```

**start**

虚拟设备中的起始块

**length**

这个片段的长度

**cipher**

Cipher 包含 **cipher[-chainmode]-ivmode[:iv options]**。

**cipher**

可用密码位于 /proc/crypto (例如 :aes)。

**chainmode**

永远使用 **cbc**。不要使用 **ebc**，它不使用初始向量 (IV)。

**ivmode[:iv options]**

IV 是一个用来区分加密法的初始向量。IV 模式是 **plain** 或者 **essiv:hash**。-plain 的 **ivmode** 使用扇区号码 (加 IV 误差) 作为 IV。-essiv 的 **ivmode** 是一个改进，可避免水印弱点。

**key**

加密密钥，以十六进制提供

**IV-offset**

初始向量 (IV) 偏移

**device**

块设备，被该文件系统中的设备名称或者主号码和副号码以 **major:minor** 的格式参考

**offset**

该设备中映射的起始偏移

以下是加密目标示例。

```
0 2097152 crypt aes-plain 0123456789abcdef0123456789abcdef 0 /dev/hda 0
```

## A.2. dmsetup 命令

**dmsetup** 命令是一个用来与 Device Mapper 沟通的命令行封装器 (wrapper)。可使用 **dmsetup** 命令的 **info**、**ls**、**status** 和 **deps** 选项查看 LVM 设备的常规信息，如以下小节所述。

有关 **dmsetup** 命令的额外选项和功能，请参考 **dmsetup(8)** man page。

### A.2.1. dmsetup info 命令

**dmsetup info device** 命令提供有关 Device Mapper 设备概述。如果没有指定设备名称，则输出所有目前配置的 Device Mapper 设备信息。如果指定了设备，那么这个命令只会生成该设备信息。

**dmsetup info** 命令提供以下分类信息：

#### Name

设备名称。LVM 设备以用小横线分隔的卷组名称和逻辑卷名称表示。在源名称中小横线会转换为两个小横线。在标准 LVM 操作过程中，不应使用这种格式的 LVM 设备名称直接指定 LVM 设备，而是应该使用 **vg/lv** 指定。

#### State

可能的设备状态是 **SUSPENDED**、**ACTIVE** 和 **READ-ONLY**。**dmsetup suspend** 命令将设备状态设定为 **SUSPENDED**。当挂起某个设备时，会停止对该设备的所有 I/O 操作。使用 **dmsetup resume** 命令可将设备状态恢复到 **ACTIVE**。

#### Read Ahead

系统对正在进行读取操作的任意打开文件的预读数据块数目。默认情况下，内核会自动选择一个合适的值。可使用 **dmsetup** 命令的 **--readahead** 选项更改这个值。

#### Tables present

这个类型的可能状态为 **LIVE** 和 **INACTIVE**。**INACTIVE** 状态表示已经载入了表格，且会在 **dmsetup resume** 命令将某个设备状态恢复为 **ACTIVE** 时进行切换，届时表格状态将为 **LIVE**。有关详情请参考 **dmsetup** man page。

#### Open count

打开参考计数表示打开该设备的次数。**mount** 命令会打开一个设备。

#### Event number

目前收到的事件数目。使用 **dmsetup wait n** 命令允许用户等待第 n 个事件，收到该事件前阻断该调用。

#### Major, minor

主设备号码和副设备号码

#### Number of targets

组成某个设备的片段数目。例如：一个跨三个磁盘的线性设备会有三个目标。线性设备由某个磁盘起始和结尾，而不是中间组成的线性设备有两个目标。

#### UUID

该设备的 UUID。

以下是 **dmsetup info** 命令的部分输出示例。

```
# dmsetup info
Name: testgfsvg-testgfslv1
State: ACTIVE
Read Ahead: 256
Tables present: LIVE
Open count: 0
Event number: 0
Major, minor: 253, 2
Number of targets: 2
UUID: LVM-
K528WUGQgPadNXYcFrrf9LnPlUMswgkCkpgPIgYzSvigM7SfewCypddNSWtNzc2N
...
Name: VolGroup00-LogVol00
State: ACTIVE
Read Ahead: 256
Tables present: LIVE
Open count: 1
Event number: 0
Major, minor: 253, 0
Number of targets: 1
UUID: LVM-
t0cS1kqFV9drb0X1Vr8sxeYP0tqcrpdegyqj5lZxe45JMG1mvtqLmbLpBcenh2L3
```

## A.2.2. dmsetup ls 命令

可以使用 **dmsetup ls** 命令列出映射的设备的设备名称列表。可以使用 **dmsetup ls --target target\_type** 命令列出至少有一个指定类型目标的设备。有关 **dmsetup ls** 的其他选项，请参考 **dmsetup ls** man page。

以下示例演示了用来列出目前配置的映射设备名称的命令。

```
# dmsetup ls
testgfsvg-testgfslv3      (253:4)
testgfsvg-testgfslv2      (253:3)
testgfsvg-testgfslv1      (253:2)
VolGroup00-LogVol01        (253:1)
VolGroup00-LogVol00        (253:0)
```

以下示例演示了用来列出目前配置的镜像映射设备名称的命令。

```
# dmsetup ls --target mirror
lock_stress-grant--02.1722      (253, 34)
lock_stress-grant--01.1720      (253, 18)
lock_stress-grant--03.1718      (253, 52)
lock_stress-grant--02.1716      (253, 40)
lock_stress-grant--03.1713      (253, 47)
lock_stress-grant--02.1709      (253, 23)
lock_stress-grant--01.1707      (253, 8)
lock_stress-grant--01.1724      (253, 14)
lock_stress-grant--03.1711      (253, 27)
```

在多路径或者其它 device mapper 装置中堆叠的 LVM 配置文件可能过于复杂。**dmsetup ls** 命令提供了一个 **--tree** 选项，可以树形式显示设备间的相依性，如下所示。

```
# dmsetup ls --tree
vgtest-lvmir (253:13)
├─vgtest-lvmir_mimage_1 (253:12)
│ └─mpathep1 (253:8)
│    └─mpathe (253:5)
│        ├─ (8:112)
│        └─ (8:64)
├─vgtest-lvmir_mimage_0 (253:11)
│ └─mpathcp1 (253:3)
│    └─mpathc (253:2)
│        ├─ (8:32)
│        └─ (8:16)
└─vgtest-lvmir_mlog (253:4)
  └─mpathfp1 (253:10)
    └─mpathf (253:6)
        ├─ (8:128)
        └─ (8:80)
```

### A.2.3. dmsetup status 命令

**dmsetup status device** 命令提供指定设备中每个目标的状态信息。如果没有指定设备名称，输出结果是所有目前配置的设备映射器设备信息。可以使用 **dmsetup status --target target\_type** 命令列出那些至少有一个指定类型目标的设备。

以下示例演示了用来列出在所有目前配置映射设备中目标状态的命令。

```
# dmsetup status
testgfsvg-testgfslv3: 0 312352768 linear
testgfsvg-testgfslv2: 0 312352768 linear
testgfsvg-testgfslv1: 0 312352768 linear
testgfsvg-testgfslv1: 312352768 50331648 linear
VolGroup00-LogVol01: 0 4063232 linear
VolGroup00-LogVol00: 0 151912448 linear
```

### A.2.4. dmsetup deps 命令

**dmsetup deps device** 命令为指定设备的映射列表参考的设备提供 (major , minor) 对列表。如果没有指定设备名称，则输出所有目前配置的设备映射器设备信息。

以下示例演示了用来列出所有目前配置的映射设备相依性的命令。

```
# dmsetup deps
testgfsvg-testgfslv3: 1 dependencies : (8, 16)
testgfsvg-testgfslv2: 1 dependencies : (8, 16)
testgfsvg-testgfslv1: 1 dependencies : (8, 16)
VolGroup00-LogVol01: 1 dependencies : (8, 2)
VolGroup00-LogVol00: 1 dependencies : (8, 2)
```

以下示例演示了用来只列出设备 **lock\_stress-grant--02.1722** 相依性的命令：

```
# dmsetup deps lock_stress-grant--02.1722
3 dependencies : (253, 33) (253, 32) (253, 31)
```

## A.3. Device Mapper 支持 udev 设备管理器

**udev** 设备管理器的主要任务是在 `/dev` 目录中提供设置节点的动态方法。在用户空间中使用**udev** 程序规则创建这些节点。在由内核直接发送的 **udev** 事件中执行这些规则即可添加、删除或者更改具体设备。这为热插拔支持提供了方便且集中的机制。

除创建实际节点外，**udev** 设备管理器还可以创建用户可命名的任意符号链接，必要时为用户提供在 `/dev` 目录中自由选择自定义命名和目录结构。

每个 **udev** 事件都包含有关要处理设备的基本信息，比如名称、所属于系统、设备类型、使用的主号码和副号码以及事件类型。因此，如果能够访问 `/sys` 命令中的所有信息，就是说也可以使用 **udev** 规则访问，则用户就可以利用基于此信息的简单过滤器，并根据此信息有条件地运行这些规则。

**udev** 设备管理器还提供集中设置节点权限的方法。用户可轻松添加自定义规则组来为任意设备定义权限，所有设备都是由处理该事件时可用的字节信息指定。

还可在 **udev** 规则中直接添加程序钩。**udev** 设备管理器可调用这些程序以便进一步提供处理该事件的过程。另外，该程序还可导出环境变量作为这个过程的结果。任意给出的结果都可进一步作为信息补充资源的规则使用。

任意使用 **udev** 的程序库的软件都可接受并处理带全部可用信息的 **udev** 事件，因此该进程不只是与 **udev** 守护进程绑定。

### A.3.1. 使用 Device Mapper 的 udev 整合

Device Mapper 提供对 **udev** 整合的直接支持。这样就将 Device Mapper 与所有与 Device Mapper 设备关联的 **udev** 进程同步，包括 LVM 设备。需要同步是因为 **udev** 守护进程中的规则程序与设备更改源的程序形成并行处理（比如 `dmsetup` 和 LVM）。没有这个支持，那么当用户尝试删除仍被前一个更改事件形成的 **udev** 规则打开并处理的设备时通常就会出问题，尤其在对那个设备的两次更改时间间隔非常短的时候。

Red Hat Enterprise Linux 为 Device Mapper 设备以及 LVM 提供官方支持的 **udev** 规则。[表 A.1 “Device-Mapper 设备的 udev 规则”](#) 中总结了这些规则，它们安装在 `/lib/udev/rules.d` 目录中。

表 A.1. Device-Mapper 设备的 udev 规则

文件名	描述
<b>10-dm.rules</b>	<p>包含基本/常规 Device Mapper 规则，并在 <code>/dev/mapper</code> 中使用 <code>/dev/dm-N</code> 目标创建符号链接，其中 N 是内核动态分配给设备的数字 (<code>/dev/dm-N</code> 是一个节点)。</p> <p>备注：<code>/dev/dm-N</code> 节点绝对不可用于访问该设备的脚本，因为 N 数字是动态分配的，并根据设备激活的顺序改变。因此应使用 <code>/dev/mapper</code> 目录中的真实名称。这个布局是为支持创建节点/符号链接的 <b>udev</b> 要求。</p>
<b>11-dm-lvm.rules</b>	<p>包含可用于 LVM 设备并为卷组逻辑卷创建符号链接的规则。该符号链接是在 <code>/dev/vgname</code> 目录中使用 <code>/dev/dm-N</code> 目标创建。</p> <p>备注：要保持为 Device Mapper 子系统命名所有未来规则使用统一的标准，udev 规则应采用以下格式：<b>11-dm-subsystem_name.rules</b>。所有提供 <b>udev</b> 规则的 <code>libdevmapper</code> 用户都应采用这个标准。</p>
<b>13-dm-disk.rules</b>	<p>包含一般适用于所有 Device Mapper 设备以及在 <code>/dev/disk/by-id</code>、<code>/dev/disk/by-uuid</code> 和 <code>/dev/disk/by-label</code> 目录中创建符号链接的规则。</p>

文件名	描述
<b>95-dm-notify.rules</b>	包含使用 <b>libdevmapper</b> 通知等待进程的规则（与 LVM 和 <b>dmsetup</b> 类似）。通知在所有之前的规则实施后执行以确定完成所有 <b>udev</b> 进程。然后恢复通知的进程。
<b>69-dm-lvm-metad.rules</b>	包含可触发 LVM 在系统中新出现块设备中进行扫描的挂钩，并在可能时执行所有 LVM 自动激活。这个挂钩支持 <b>lvm</b> 守护进程，后者是使用 <b>lvm.conf</b> 文件的 <b>use_lvmetad=1</b> 设定。在集群环境中不支持 <b>lvmetad</b> 守护进程及自动激活。

可以使用 **12-dm-permissions.rules** 文件添加额外的自定义权限规则。该文件不安装在 **/lib/udev/rules** 目录中，它位于 **/usr/share/doc/device-mapper-version** 目录。**12-dm-permissions.rules** 文件是包含如何设置权限提示的模板，它是根据示例中给出的匹配规则生成的。该文件包含一些常见情况的示例。您可以编辑这个文件并手动将其放在 **/etc/udev/rules.d** 目录中，在此文件不会受更新的影响，因此可保留其设置。

这些规则设定所有可在处理事件的过程中被其它规则使用的基本变量。

以下是在 **10-dm.rules** 中设定的变量：

- » **DM\_NAME** : Device Mapper 设备名称
- » **DM\_UUID** : Device Mapper 设备 UUID
- » **DM\_SUSPENDED** : Device Mapper 的挂起状态
- » **DM\_UDEV\_RULES\_VSN** : **udev** 规则版本（这主要用于检查之前提到变量的所有其它规则，这些变量直接由官方 Device Mapper 规则设定）

以下是在 **11-dm-lvm.rules** 中设置的变量：

- » **DM\_LV\_NAME** : 逻辑卷名称
- » **DM\_VG\_NAME** : 卷组名称
- » **DM\_LV\_LAYER** : LVM 层名称

所有这些变量都可用于 **12-dm-permissions.rules** 文件，来为具体 Device Mapper 设备定义权限，如 **12-dm-permissions.rules** 文件所述。

### A.3.2. 支持 udev 的命令和界面

表 A.2 “支持 udev 的 dmsetup 命令”总结了支持 **udev** 整合的 **dmsetup** 命令。

表 A.2. 支持 udev 的 dmsetup 命令

命令	描述
<b>dmsetup udevcomplete</b>	用于通知 udev 已经完成规则处理并解锁等待的进程（从 <b>95-dm-notify.rules</b> 的 <b>udev</b> 规则调用）。
<b>dmsetup udevcomplete_all</b>	用于在调整过程中手动解锁所有等待进程。
<b>dmsetup udevcookies</b>	用于在 debug 过程中显示所有现有 cookies（系统范围的信号）。
<b>dmsetup udevcreatecookie</b>	用于手动创建 cookie（信号）。这在同一同步资源中运行多个进程时有用。
<b>dmsetup udevreleasemode</b>	用于等待所有与同步 cookie 中的所有进程关联的 <b>udev</b> 进程。

支持 **udev** 整合的 **dmsetup** 选项如下。

**--udevcookie**

需要为所有我们要加入 udev 事务的所有 dmsetup 进程定义。它可与 **udevcreatecookie** 和 **udevreleasemode** 合用：

```
COOKIE=$(dmsetup udevcreatecookie)
dmsetup command --udevcookie $COOKIE ....
dmsetup command --udevcookie $COOKIE ....
...
dmsetup command --udevcookie $COOKIE ....
dmsetup udevreleasemode --udevcookie $COOKIE
```

除使用 **--udevcookie** 选项外，还可以只将该变量导出到该进程的环境中：

```
export DM_UDEV_COOKIE=$(dmsetup udevcreatecookie)
dmsetup command ...
dmsetup command ...
...
dmsetup command ...
```

**--noudevrules**

禁用 udev 规则。**libdevmapper** 自己会生成节点/符号链接（老方法）。如果 **udev** 无法正常工作，则使用这个选项调试。

**--noudevsync**

禁用 **udev** 同步。这也可用于调试。

有关 **dmsetup** 命令及其选项的详情请参考 **dmsetup(8)** man page。

LVM 命令支持以下支持 **udev** 整合的选项：

- » **--noudevrules**：在 **dmsetup** 命令中禁用 **udev** 规则。
- » **--noudevsync**：在 **dmsetup** 命令中禁用 **udev** 同步。

**lvm.conf** 文件包含以下支持 **udev** 整合的选项：

- » **udev\_rules**：为所有 LVM2 命令在全局启用/禁用 **udev\_rules**。
- » **udev\_sync**：为所有 LVM 命令在全局启用/禁用 **udev** 同步。

有关 **lvm.conf** 文件选项的详情请查看 **lvm.conf** 文件的行间注释。

## 附录 B. LVM 配置文件

LVM 支持多配置文件。系统启动时，会从根据环境变量 **LVM\_SYSTEM\_DIR** 指定的目录载入 **lvm.conf** 配置文件，默认该变量为 **/etc/lvm**。

**lvm.conf** 文件可指定要载入的额外配置文件。之后设定的文件会覆盖之前设定的文件。请运行 **lvm dumpconfig** 命令在载入所有配置文件后显示使用的设定。

有关载入额外配置文件的详情，请参考 [第 C.2 节“主机标签”](#)。

### B.1. LVM 配置文件

以下是用于 LVM 配置的文件：

#### **/etc/lvm/lvm.conf**

由工具读取的中央配置文件。

#### **etc/lvm/lvm\_hosttag.conf**

对于每个主机标签，如果存在额外的额外配置文件 **lvm\_hosttag.conf**，就会读取它。如果该文件定义了新标签，那么会另将配置文件附加在要读取的文件列表中。有关主机标签的详情，请参考 [第 C.2 节“主机标签”](#)。

除 LVM 配置文件外，运行 LVM 的系统还包含以下可影响 LVM 系统设置的文件：

#### **/etc/lvm/cache/.cache**

设备名称过滤器缓存文件（可配置）。

#### **/etc/lvm/backup/**

自动卷组元数据备份目录（可配置）。

#### **/etc/lvm/archive/**

自动卷组元数据归档目录（可根据目录路径和归档历史记录途径进行配置）。

#### **/var/lock/lvm**

在单一主机配置中，这是可防止平行工具运行时破坏元数据的锁定文件；而在集群中，这是在集群范围内使用的 DLM。

### B.2. **lvm dumpconfig** 命令

可使用 **lvm** 命令的 **dumpconfig** 选项显示当前 LVM 配置或将该配置保存为一个文件。**lvm dumpconfig** 命令有不同的功能，如下所述：

- » 可转储与任意标签配置文件合并的当前 lvm 配置。
- » 可转储所有与默认值不同的当前配置设定值。
- » 可在具体 LVM 版本转储所有在当前 LVM 版本中引进的新配置设定。
- » 可转储所有可修改配置设定，可以是整个配置，也可以是独立的命令或元数据配置。有关 LVM 配置文件的详情，请查看 [第 B.3 节“LVM 配置文件”](#)。
- » 可以只为 LVM 的具体版本转储配置设定。

- » 可验证当前配置。

有关支持功能的完整列表及指定 **lvm dumpconfig** 选项的信息，请查看 **lvm-dumpconfig** man page。

## B.3. LVM 配置文件

LVM 配置文件是一组用来在不同环境或使用情况下形成某些特征的可自定义配置设定。通常配置文件的名称就可以体现其环境或使用情况。LVM 配置文件覆盖现有配置。

LVM 可识别两组 LVM 配置文件：命令配置文件和元数据配置文件。

- » 命令配置文件是用来在全局 LVM 命令层覆盖所选配置设定。该配置文件在开始执行 LVM 命令时应用，并贯穿 LVM 命令执行的整个过程。可在执行 LVM 命令时指定 **--commandprofile ProfileName** 选项，以便应用命令配置文件。
- » 元数据配置文件是用来覆盖在卷组/逻辑卷层中选择的配置设定。它是独立应用于每个要执行的卷组/逻辑卷。因此，每个卷组/逻辑卷均可使用其元数据保存该配置文件名称，以便下次处理该卷组/逻辑卷时使用。该配置文件是自动应用。如果该卷组及其任意逻辑卷有不同的配置文件定义，则首选为逻辑卷定义的配置文件。
  - 可在使用 **vgcreate** 或 **lvcreate** 命令创建逻辑卷时，指定 **--metadataprofile ProfileName** 选项，将元数据配置文件附加到卷组或逻辑卷中。
  - 可在使用 **vgcreate** 或 **lvcreate** 命令创建逻辑卷时，指定 **--metadataprofile ProfileName** 或者 **--detachprofile** 选项，将元数据配置文件附加到现有卷组或逻辑卷中，或从中将其移除。
  - 可使用 **vgs** 和 **lvs** 命令的 **-o vg\_profile** 和 **-o lv\_profile** 输出结果选项显示目前附加到卷组或逻辑卷中的元数据配置文件。

命令配置文件允许的选项集与元数据配置文件允许的选项集相互排斥。这些设定只能用于这两个集合之一，无法混合使用，同时 LVM 工具也会拒绝混合使用这些设定的配置文件。

LVM 提供几个预先定义的配置文件。LVM 配置文件默认保存在 **/etc/lvm/profile** 目录中。可在 **/etc/lvm/lvm.conf** 文件中更改 **profile\_dir** 设定改变其位置。每个配置文件设定均保存在 **profile** 目录的 **ProfileName.profile** 文件中。在 LVM 命令中参考该配置文件时会省略 **.profile** 后缀。

可使用不同值创建附加配置文件。因此 LVM 提供 **command\_profile\_template.profile** 文件（用于命令配置文件）和 **metadata\_profile\_template.profile** 文件（用于元数据配置文件），这些文件包含每个配置文件类型可自定义的所有设定。可复制这些模板配置文件，并根据需要进行编辑。

另外，可使用 **lvm dumpconfig** 命令为配置文件的给定部分为每种配置文件类型生成新的配置。下面的命令创建一个名为 **ProfileName.profile**，组成 **section** 中设置的新命令配置文件。

```
lvm dumpconfig --file ProfileName.profile --type profilable-command
section
```

下面的命令创建一个名为 **ProfileName.profile**，组成 **section** 中设置的新命令配置文件。

```
lvm dumpconfig --file ProfileName.profile --type profilable-metadata
section
```

如果没有指定这个部分，则会报告所有可配置的设定。

## B.4. lvm.conf 文件示例

以下是 `lvm.conf` 配置文件示例。您的配置文件可能和这个文件有所不同。

```

# This is an example configuration file for the LVM2 system.
# It contains the default settings that would be used if there was no
# /etc/lvm/lvm.conf file.
#
# Refer to 'man lvm.conf' for further information including the file
layout.
#
# To put this file in a different directory and override /etc/lvm set
# the environment variable LVM_SYSTEM_DIR before running the tools.
#
# N.B. Take care that each setting only appears once if uncommenting
# example settings in this file.

# This section allows you to set the way the configuration settings are
handled.
config {

    # If enabled, any LVM2 configuration mismatch is reported.
    # This implies checking that the configuration key is understood
    # by LVM2 and that the value of the key is of a proper type.
    # If disabled, any configuration mismatch is ignored and default
    # value is used instead without any warning (a message about the
    # configuration key not being found is issued in verbose mode only).
    checks = 1

    # If enabled, any configuration mismatch aborts the LVM2 process.
    abort_on_errors = 0

    # Directory where LVM looks for configuration profiles.
    profile_dir = "/etc/lvm/profile"
}

# This section allows you to configure which block devices should
# be used by the LVM system.
devices {

    # Where do you want your volume groups to appear ?
    dir = "/dev"

    # An array of directories that contain the device nodes you wish
    # to use with LVM2.
    scan = [ "/dev" ]

    # If set, the cache of block device nodes with all associated
    symlinks
        # will be constructed out of the existing udev database content.
        # This avoids using and opening any inapplicable non-block devices or
        # subdirectories found in the device directory. This setting is
    applied
        # to udev-managed device directory only, other directories will be
    scanned
        # fully. LVM2 needs to be compiled with udev support for this setting
    to
}

```

```

# take effect. N.B. Any device node or symlink not managed by udev in
# udev directory will be ignored with this setting on.
obtain_device_list_from_udev = 1

# If several entries in the scanned directories correspond to the
# same block device and the tools need to display a name for device,
# all the pathnames are matched against each item in the following
# list of regular expressions in turn and the first match is used.
# preferred_names = [ ]

# Try to avoid using undescriptive /dev/dm-N names, if present.
preferred_names = [ "^/dev/mpath/", "^/dev/mapper/mpath",
"/^/dev/[hs]d" ]

# A filter that tells LVM2 to only use a restricted set of devices.
# The filter consists of an array of regular expressions. These
# expressions can be delimited by a character of your choice, and
# prefixed with either an 'a' (for accept) or 'r' (for reject).
# The first expression found to match a device name determines if
# the device will be accepted or rejected (ignored). Devices that
# don't match any patterns are accepted.

# Be careful if there are symbolic links or multiple filesystem
# entries for the same device as each name is checked separately
against
    # the list of patterns. The effect is that if the first pattern in
the
    # list to match a name is an 'a' pattern for any of the names, the
device
        # is accepted; otherwise if the first pattern in the list to match a
name
            # is an 'r' pattern for any of the names it is rejected; otherwise it
is
            # accepted.

# Don't have more than one filter line active at once: only one gets
used.

# Run vgscan after you change this parameter to ensure that
# the cache file gets regenerated (see below).
# If it doesn't do what you expect, check the output of 'vgscan -
vvvv'.

# If lvmetad is used, then see "A note about device filtering while
# lvmetad is used" comment that is attached to global/use_lvmetad
setting.

# By default we accept every block device:
filter = [ "a/.*/" ]

# Exclude the cdrom drive
# filter = [ "r|/dev/cdrom|" ]

# When testing I like to work with just loopback devices:
# filter = [ "a/loop/", "r/.*/" ]

```

```

# Or maybe all loops and ide drives except hdc:
# filter =[ "a|loop|", "r|/dev/hdc|", "a|/dev/ide|", "r|.*|" ]

# Use anchors if you want to be really specific
# filter = [ "a|^/dev/hda8$|", "r|.*/" ]

# Since "filter" is often overridden from command line, it is not
suitable
# for system-wide device filtering (udev rules, lvmtd). To hide
devices
# from LVM-specific udev processing and/or from lvmtd, you need to
set
# global_filter. The syntax is the same as for normal "filter"
# above. Devices that fail the global_filter are not even opened by
LVM.

# global_filter = []

# The results of the filtering are cached on disk to avoid
# rescanning dud devices (which can take a very long time).
# By default this cache is stored in the /etc/lvm/cache directory
# in a file called '.cache'.
# It is safe to delete the contents: the tools regenerate it.
# (The old setting 'cache' is still respected if neither of
# these new ones is present.)
# N.B. If obtain_device_list_from_udev is set to 1 the list of
# devices is instead obtained from udev and any existing .cache
# file is removed.
cache_dir = "/etc/lvm/cache"
cache_file_prefix = ""

# You can turn off writing this cache file by setting this to 0.
write_cache_state = 1

# Advanced settings.

# List of pairs of additional acceptable block device types found
# in /proc/devices with maximum (non-zero) number of partitions.
# types = [ "fd", 16 ]

# If sysfs is mounted (2.6 kernels) restrict device scanning to
# the block devices it believes are valid.
# 1 enables; 0 disables.
sysfs_scan = 1

# By default, LVM2 will ignore devices used as component paths
# of device-mapper multipath devices.
# 1 enables; 0 disables.
multipath_component_detection = 1

# By default, LVM2 will ignore devices used as components of
# software RAID (md) devices by looking for md superblocks.
# 1 enables; 0 disables.
md_component_detection = 1

# By default, if a PV is placed directly upon an md device, LVM2

```

```

# will align its data blocks with the md device's stripe-width.
# 1 enables; 0 disables.
md_chunk_alignment = 1

# Default alignment of the start of a data area in MB.  If set to 0,
# a value of 64KB will be used.  Set to 1 for 1MiB, 2 for 2MiB, etc.
# default_data_alignment = 1

# By default, the start of a PV's data area will be a multiple of
# the 'minimum_io_size' or 'optimal_io_size' exposed in sysfs.
# - minimum_io_size - the smallest request the device can perform
#   w/o incurring a read-modify-write penalty (e.g. MD's chunk size)
# - optimal_io_size - the device's preferred unit of receiving I/O
#   (e.g. MD's stripe width)
# minimum_io_size is used if optimal_io_size is undefined (0).
# If md_chunk_alignment is enabled, that detects the optimal_io_size.
# This setting takes precedence over md_chunk_alignment.
# 1 enables; 0 disables.
data_alignment_detection = 1

# Alignment (in KB) of start of data area when creating a new PV.
# md_chunk_alignment and data_alignment_detection are disabled if
set.
# Set to 0 for the default alignment (see: data_alignment_default)
# or page size, if larger.
data_alignment = 0

# By default, the start of the PV's aligned data area will be shifted
by
# the 'alignment_offset' exposed in sysfs.  This offset is often 0 but
# may be non-zero; e.g.: certain 4KB sector drives that compensate
for
# windows partitioning will have an alignment_offset of 3584 bytes
# (sector 7 is the lowest aligned logical block, the 4KB sectors
start
# at LBA -1, and consequently sector 63 is aligned on a 4KB
boundary).
# But note that pvcreate --dataalignmentoffset will skip this
detection.
# 1 enables; 0 disables.
data_alignment_offset_detection = 1

# If, while scanning the system for PVs, LVM2 encounters a device-
mapper
# device that has its I/O suspended, it waits for it to become
accessible.
# Set this to 1 to skip such devices.  This should only be needed
# in recovery situations.
ignore_suspended_devices = 0

# ignore_lvm_mirrors:  Introduced in version 2.02.104
# This setting determines whether logical volumes of "mirror" segment
# type are scanned for LVM labels.  This affects the ability of
# mirrors to be used as physical volumes.  If 'ignore_lvm_mirrors'
# is set to '1', it becomes impossible to create volume groups on top
# of mirror logical volumes - i.e. to stack volume groups on mirrors.

```

```

#
# Allowing mirror logical volumes to be scanned (setting the value
to '0')
# can potentially cause LVM processes and I/O to the mirror to become
# blocked. This is due to the way that the "mirror" segment type
handles
# failures. In order for the hang to manifest itself, an LVM command
must
# be run just after a failure and before the automatic LVM repair
process
# takes place OR there must be failures in multiple mirrors in the
same
# volume group at the same time with write failures occurring moments
# before a scan of the mirror's labels.
#
# Note that these scanning limitations do not apply to the LVM RAID
# types, like "raid1". The RAID segment types handle failures in a
# different way and are not subject to possible process or I/O
blocking.
#
# It is encouraged that users set 'ignore_lvm_mirrors' to 1 if they
# are using the "mirror" segment type. Users that require volume
group
# stacking on mirrored logical volumes should consider using the
"raid1"
# segment type. The "raid1" segment type is not available for
# active/active clustered volume groups.
#
# Set to 1 to disallow stacking and thereby avoid a possible
deadlock.
ignore_lvm_mirrors = 1

# During each LVM operation errors received from each device are
counted.
# If the counter of a particular device exceeds the limit set here,
no
# further I/O is sent to that device for the remainder of the
respective
# operation. Setting the parameter to 0 disables the counters
altogether.
disable_after_error_count = 0

# Allow use of pvcreate --uuid without requiring --restorefile.
require_restorefile_with_uuid = 1

# Minimum size (in KB) of block devices which can be used as PVs.
# In a clustered environment all nodes must use the same value.
# Any value smaller than 512KB is ignored.

# Ignore devices smaller than 2MB such as floppy drives.
pv_min_size = 2048

# The original built-in setting was 512 up to and including version
2.02.84.
# pv_min_size = 512

```

```

    # Issue discards to a logical volume's underlying physical volume(s)
when
        # the logical volume is no longer using the physical volumes' space
(e.g.
            # lvremove, lvreduce, etc). Discards inform the storage that a
region is
                # no longer in use. Storage that supports discards advertise the
protocol
                    # specific way discards should be issued by the kernel (TRIM, UNMAP,
or
                        # WRITE SAME with UNMAP bit set). Not all storage will support or
benefit
                            # from discards but SSDs and thinly provisioned LUNs generally do.
If set
    # to 1, discards will only be issued if both the storage and kernel
provide
        # support.
        # 1 enables; 0 disables.
        issue_discards = 0
}

# This section allows you to configure the way in which LVM selects
# free space for its Logical Volumes.
allocation {

    # When searching for free space to extend an LV, the "cling"
    # allocation policy will choose space on the same PVs as the last
    # segment of the existing LV. If there is insufficient space and a
    # list of tags is defined here, it will check whether any of them are
    # attached to the PVs concerned and then seek to match those PV tags
    # between existing extents and new extents.
    # Use the special tag "@*" as a wildcard to match any PV tag.

    # Example: LVs are mirrored between two sites within a single VG.
    # PVs are tagged with either @site1 or @site2 to indicate where
    # they are situated.

    # cling_tag_list = [ "@site1", "@site2" ]
    # cling_tag_list = [ "@*" ]

    # Changes made in version 2.02.85 extended the reach of the 'cling'
    # policies to detect more situations where data can be grouped
    # onto the same disks. Set this to 0 to revert to the previous
    # algorithm.
    maximise_cling = 1

    # Whether to use blkid library instead of native LVM2 code to detect
    # any existing signatures while creating new Physical Volumes and
    # Logical Volumes. LVM2 needs to be compiled with blkid wiping
support
    # for this setting to take effect.
    #
    # LVM2 native detection code is currently able to recognize these
signatures:
        #      - MD device signature
        #      - swap signature
}

```

```

# - LUKS signature
# To see the list of signatures recognized by blkid, check the output
# of 'blkid -k' command. The blkid can recognize more signatures than
# LVM2 native detection code, but due to this higher number of
signatures
# to be recognized, it can take more time to complete the signature
scan.
use_blkid_wiping = 1

# Set to 1 to wipe any signatures found on newly-created Logical
Volumes
# automatically in addition to zeroing of the first KB on the LV
# (controlled by the -Z/--zero y option).
# The command line option -W/--wipesignatures takes precedence over
this
# setting.
# The default is to wipe signatures when zeroing.
#
wipe_signatures_when_zeroing_new_lvs = 1

# Set to 1 to guarantee that mirror logs will always be placed on
# different PVs from the mirror images. This was the default
# until version 2.02.85.
mirror_logs_require_separate_pvs = 0

# Set to 1 to guarantee that cache_pool metadata will always be
# placed on different PVs from the cache_pool data.
cache_pool_metadata_require_separate_pvs = 0

# Specify the minimal chunk size (in kiB) for cache pool volumes.
# Using a chunk_size that is too large can result in wasteful use of
# the cache, where small reads and writes can cause large sections of
# an LV to be mapped into the cache. However, choosing a chunk_size
# that is too small can result in more overhead trying to manage the
# numerous chunks that become mapped into the cache. The former is
# more of a problem than the latter in most cases, so we default to
# a value that is on the smaller end of the spectrum. Supported
values
# range from 32(kiB) to 1048576 in multiples of 32.
# cache_pool_chunk_size = 64

# Set to 1 to guarantee that thin pool metadata will always
# be placed on different PVs from the pool data.
thin_pool_metadata_require_separate_pvs = 0

# Specify chunk size calculation policy for thin pool volumes.
# Possible options are:
# "generic"      - if thin_pool_chunk_size is defined, use it.
#                   Otherwise, calculate the chunk size based on
#                   estimation and device hints exposed in sysfs:
#                   the minimum_io_size. The chunk size is always
#                   at least 64KiB.
#
# "performance"   - if thin_pool_chunk_size is defined, use it.
#                   Otherwise, calculate the chunk size for
#                   performance based on device hints exposed in

```

```

#      sysfs: the optimal_io_size. The chunk size is
#      always at least 512KiB.
# thin_pool_chunk_size_policy = "generic"

# Specify the minimal chunk size (in KB) for thin pool volumes.
# Use of the larger chunk size may improve performance for plain
# thin volumes, however using them for snapshot volumes is less
efficient,
# as it consumes more space and takes extra time for copying.
# When unset, lvm tries to estimate chunk size starting from 64KB
# Supported values are in range from 64 to 1048576.
# thin_pool_chunk_size = 64

# Specify discards behaviour of the thin pool volume.
# Select one of "ignore", "nopassdown", "passdown"
# thin_pool_discards = "passdown"

# Set to 0, to disable zeroing of thin pool data chunks before their
# first use.
# N.B. zeroing larger thin pool chunk size degrades performance.
# thin_pool_zero = 1
}

# This section that allows you to configure the nature of the
# information that LVM2 reports.
log {

    # Controls the messages sent to stdout or stderr.
    # There are three levels of verbosity, 3 being the most verbose.
    verbose = 0

    # Set to 1 to suppress all non-essential messages from stdout.
    # This has the same effect as -qq.
    # When this is set, the following commands still produce output:
    # dumpconfig, lvdisplay, lvdiskscan, lvs, pvck, pvdiskdisplay,
    # pvs, version, vgcfgrestore -l, vgdisplay, vgs.
    # Non-essential messages are shifted from log level 4 to log level 5
    # for syslog and lvm2_log_fn purposes.
    # Any 'yes' or 'no' questions not overridden by other arguments
    # are suppressed and default to 'no'.
    silent = 0

    # Should we send log messages through syslog?
    # 1 is yes; 0 is no.
    syslog = 1

    # Should we log error and debug messages to a file?
    # By default there is no log file.
    #file = "/var/log/lvm2.log"

    # Should we overwrite the log file each time the program is run?
    # By default we append.
    overwrite = 0

    # What level of log messages should we send to the log file and/or
    syslog?
}

```

```

# There are 6 syslog-like log levels currently in use - 2 to 7
inclusive.
# 7 is the most verbose (LOG_DEBUG).
level = 0

# Format of output messages
# Whether or not (1 or 0) to indent messages according to their
severity
indent = 1

# Whether or not (1 or 0) to display the command name on each line
output
command_names = 0

# A prefix to use before the message text (but after the command
name,
# if selected). Default is two spaces, so you can see/grep the
severity
# of each message.
prefix = "  "

# To make the messages look similar to the original LVM tools use:
# indent = 0
# command_names = 1
# prefix = " -- "

# Set this if you want log messages during activation.
# Don't use this in low memory situations (can deadlock).
# activation = 0

# Some debugging messages are assigned to a class and only appear
# in debug output if the class is listed here.
# Classes currently available:
#   memory, devices, activation, allocation, lvmtd, metadata,
cache,
#   locking
# Use "all" to see everything.
debug_classes = [ "memory", "devices", "activation", "allocation",
                  "lvmtd", "metadata", "cache", "locking" ]
}

# Configuration of metadata backups and archiving. In LVM2 when we
# talk about a 'backup' we mean making a copy of the metadata for the
# *current* system. The 'archive' contains old metadata configurations.
# Backups are stored in a human readable text format.
backup {

    # Should we maintain a backup of the current metadata configuration ?
    # Use 1 for Yes; 0 for No.
    # Think very hard before turning this off!
    backup = 1

    # Where shall we keep it ?
    # Remember to back up this directory regularly!
    backup_dir = "/etc/lvm/backup"
}

```

```

# Should we maintain an archive of old metadata configurations.
# Use 1 for Yes; 0 for No.
# On by default. Think very hard before turning this off.
archive = 1

# Where should archived files go ?
# Remember to back up this directory regularly!
archive_dir = "/etc/lvm/archive"

# What is the minimum number of archive files you wish to keep ?
retain_min = 10

# What is the minimum time you wish to keep an archive file for ?
retain_days = 30
}

# Settings for the running LVM2 in shell (readline) mode.
shell {

    # Number of lines of history to store in ~/.lvm_history
    history_size = 100
}

# Miscellaneous global LVM2 settings
global {
    # The file creation mask for any files and directories created.
    # Interpreted as octal if the first digit is zero.
    umask = 077

    # Allow other users to read the files
    #umask = 022

    # Enabling test mode means that no changes to the on disk metadata
    # will be made. Equivalent to having the -t option on every
    # command. Defaults to off.
    test = 0

    # Default value for --units argument
    units = "h"

    # Since version 2.02.54, the tools distinguish between powers of
    # 1024 bytes (e.g. KiB, MiB, GiB) and powers of 1000 bytes (e.g.
    # KB, MB, GB).
    # If you have scripts that depend on the old behaviour, set this to 0
    # temporarily until you update them.
    si_unit_consistency = 1

    # Whether or not to communicate with the kernel device-mapper.
    # Set to 0 if you want to use the tools to manipulate LVM metadata
    # without activating any logical volumes.
    # If the device-mapper kernel driver is not present in your kernel
    # setting this to 0 should suppress the error messages.
    activation = 1

    # If we can't communicate with device-mapper, should we try running
}

```

```

# the LVM1 tools?
# This option only applies to 2.4 kernels and is provided to help you
# switch between device-mapper kernels and LVM1 kernels.
# The LVM1 tools need to be installed with .lvm1 suffices
# e.g. vgscan.lvm1 and they will stop working after you start using
# the new lvm2 on-disk metadata format.
# The default value is set when the tools are built.
# fallback_to_lvm1 = 0

# The default metadata format that commands should use - "lvm1" or
"lvm2".
# The command line override is -M1 or -M2.
# Defaults to "lvm2".
# format = "lvm2"

# Location of proc filesystem
proc = "/proc"

# Type of locking to use. Defaults to local file-based locking (1).
# Turn locking off by setting to 0 (dangerous: risks metadata
corruption
# if LVM2 commands get run concurrently).
# Type 2 uses the external shared library locking_library.
# Type 3 uses built-in clustered locking.
# Type 4 uses read-only locking which forbids any operations that
might
# change metadata.
# N.B. Don't use lvmetad with locking type 3 as lvmetad is not yet
# supported in clustered environment. If use_lvmetad=1 and
locking_type=3
# is set at the same time, LVM always issues a warning message about
this
# and then it automatically disables lvmetad use.
locking_type = 1

# Set to 0 to fail when a lock request cannot be satisfied
immediately.
wait_for_locks = 1

# If using external locking (type 2) and initialisation fails,
# with this set to 1 an attempt will be made to use the built-in
# clustered locking.
# If you are using a customised locking_library you should set this
to 0.
fallback_to_clustered_locking = 1

# If an attempt to initialise type 2 or type 3 locking failed,
perhaps
# because cluster components such as clvmd are not running, with this
set
# to 1 an attempt will be made to use local file-based locking (type
1).
# If this succeeds, only commands against local volume groups will
proceed.
# Volume Groups marked as clustered will be ignored.
fallback_to_local_locking = 1

```

```

# Local non-LV directory that holds file-based locks while commands
are
# in progress. A directory like /tmp that may get wiped on reboot is
OK.
locking_dir = "/run/lock/lvm"

# Whenever there are competing read-only and read-write access
requests for
# a volume group's metadata, instead of always granting the read-only
# requests immediately, delay them to allow the read-write requests
to be
# serviced. Without this setting, write access may be stalled by a
high
# volume of read-only requests.
# NB. This option only affects locking_type = 1 viz. local file-based
# locking.
prioritise_write_locks = 1

# Other entries can go here to allow you to load shared libraries
# e.g. if support for LVM1 metadata was compiled as a shared library
use
#   format_libraries = "liblvm2format1.so"
# Full pathnames can be given.

# Search this directory first for shared libraries.
#   library_dir = "/lib"

# The external locking library to load if locking_type is set to 2.
#   locking_library = "liblvm2clusterlock.so"

# Treat any internal errors as fatal errors, aborting the process
that
# encountered the internal error. Please only enable for debugging.
abort_on_internal_errors = 0

# Check whether CRC is matching when parsed VG is used multiple
times.
# This is useful to catch unexpected internal cached volume group
# structure modification. Please only enable for debugging.
detect_internal_vg_cache_corruption = 0

# If set to 1, no operations that change on-disk metadata will be
permitted.
# Additionally, read-only commands that encounter metadata in need
of repair
# will still be allowed to proceed exactly as if the repair had been
# performed (except for the unchanged vg_seqno).
# Inappropriate use could mess up your system, so seek advice first!
metadata_read_only = 0

# 'mirror_segtypes_default' defines which segtype will be used when
the
# shorthand '-m' option is used for mirroring. The possible options
are:
#

```

```

# "mirror" - The original RAID1 implementation provided by LVM2/DM.
It is
    #           characterized by a flexible log solution (core, disk,
mirrored)
        #   and by the necessity to block I/O while reconfiguring in the
        #   event of a failure.
    #
    #   There is an inherent race in the dmeventd failure handling
    #   logic with snapshots of devices using this type of RAID1 that
    #   in the worst case could cause a deadlock.
    #       Ref: https://bugzilla.redhat.com/show_bug.cgi?id=817130#c10
    #
    # "raid1" - This implementation leverages MD's RAID1 personality
through
        #           device-mapper. It is characterized by a lack of log
options.
        #   (A log is always allocated for every device and they are placed
        #   on the same device as the image - no separate devices are
        #   required.) This mirror implementation does not require I/O
        #   to be blocked in the kernel in the event of a failure.
        #   This mirror implementation is not cluster-aware and cannot be
        #   used in a shared (active/active) fashion in a cluster.
    #
    # Specify the '--type <mirror|raid1>' option to override this default
    # setting.
    mirror_sectype_default = "raid1"

    # 'raid10_sectype_default' determines the segment types used by
default
        # when the '--stripes/-i' and '--mirrors/-m' arguments are both
specified
            # during the creation of a logical volume.
            # Possible settings include:
            #
            # "raid10" - This implementation leverages MD's RAID10 personality
through
                #           device-mapper.

                #
                # "mirror" - LVM will layer the 'mirror' and 'stripe' segment types.
It
    #           will do this by creating a mirror on top of striped sub-
LVs;
    #           effectively creating a RAID 0+1 array. This is
suboptimal
    #           in terms of providing redundancy and performance.
Changing to
    #           this setting is not advised.
    # Specify the '--type <raid10|mirror>' option to override this
default
    # setting.
    raid10_sectype_default = "raid10"

    # The default format for displaying LV names in lvdisplay was changed
    # in version 2.02.89 to show the LV name and path separately.
    # Previously this was always shown as /dev/vgname/lvname even when
that

```

```

# was never a valid path in the /dev filesystem.
# Set to 1 to reinstate the previous format.
#
# lvdisplay_shows_full_device_path = 0

# Whether to use (trust) a running instance of lvmtd. If this is
set to
# 0, all commands fall back to the usual scanning mechanisms. When
set to 1
# *and* when lvmtd is running (automatically instantiated by
making use of
# systemd's socket-based service activation or run as an initscripts
service
# or run manually), the volume group metadata and PV state flags are
obtained
# from the lvmtd instance and no scanning is done by the individual
# commands. In a setup with lvmtd, lvmtd udev rules *must* be set
up for
# LVM to work correctly. Without proper udev rules, all changes in
block
# device configuration will be *ignored* until a manual 'pvscan --'
cache'
# is performed. These rules are installed by default.
#
# If lvmtd has been running while use_lvmtd was 0, it MUST be
stopped
# before changing use_lvmtd to 1 and started again afterwards.
#
# If using lvmtd, the volume activation is also switched to
automatic
# event-based mode. In this mode, the volumes are activated based on
# incoming udev events that automatically inform lvmtd about new
PVs
# that appear in the system. Once the VG is complete (all the PVs are
# present), it is auto-activated. The
activation/auto_activation_volume_list
# setting controls which volumes are auto-activated (all by default).
#
# A note about device filtering while lvmtd is used:
# When lvmtd is updated (either automatically based on udev events
# or directly by pvscan --cache <device> call), the devices/filter
# is ignored and all devices are scanned by default. The lvmtd
always
# keeps unfiltered information which is then provided to LVM commands
# and then each LVM command does the filtering based on
devices/filter
# setting itself.
# To prevent scanning devices completely, even when using lvmtd,
# the devices/global_filter must be used.
# N.B. Don't use lvmtd with locking type 3 as lvmtd is not yet
# supported in clustered environment. If use_lvmtd=1 and
locking_type=3
# is set at the same time, LVM always issues a warning message about
this
# and then it automatically disables lvmtd use.
use_lvmtd = 1

```

```

# Full path of the utility called to check that a thin metadata
device
# is in a state that allows it to be used.
# Each time a thin pool needs to be activated or after it is
deactivated
# this utility is executed. The activation will only proceed if the
utility
# has an exit status of 0.
# Set to "" to skip this check. (Not recommended.)
# The thin tools are available as part of the device-mapper-
persistent-data
# package from https://github.com/jthornber/thin-provisioning-tools.
#
# thin_check_executable = "/usr/sbin/thin_check"

# Array of string options passed with thin_check command. By default,
# option "-q" is for quiet output.
# With thin_check version 2.1 or newer you can add "--ignore-non-
fatal-errors"
# to let it pass through ignorable errors and fix them later.
#
# thin_check_options = [ "-q" ]

# Full path of the utility called to repair a thin metadata device
# is in a state that allows it to be used.
# Each time a thin pool needs repair this utility is executed.
# See thin_check_executable how to obtain binaries.
#
# thin_repair_executable = "/usr/sbin/thin_repair"

# Array of extra string options passed with thin_repair command.
# thin_repair_options = [ "" ]

# Full path of the utility called to dump thin metadata content.
# See thin_check_executable how to obtain binaries.
#
# thin_dump_executable = "/usr/sbin/thin_dump"

# If set, given features are not used by thin driver.
# This can be helpful not just for testing, but i.e. allows to avoid
# using problematic implementation of some thin feature.
# Features:
#   block_size
#   discards
#   discards_non_power_2
#   external_origin
#   metadata_resize
#   external_origin_extend
#
# thin_disabled_features = [ "discards", "block_size" ]
}

activation {
    # Set to 1 to perform internal checks on the operations issued to
    # libdevmapper. Useful for debugging problems with activation.
}

```

```

# Some of the checks may be expensive, so it's best to use this
# only when there seems to be a problem.
checks = 0

# Set to 0 to disable udev synchronisation (if compiled into the
binaries).
# Processes will not wait for notification from udev.
# They will continue irrespective of any possible udev processing
# in the background. You should only use this if udev is not running
# or has rules that ignore the devices LVM2 creates.
# The command line argument --nodevsync takes precedence over this
setting.
# If set to 1 when udev is not running, and there are LVM2 processes
# waiting for udev, run 'dmsetup udevcomplete_all' manually to wake
them up.
udev_sync = 1

# Set to 0 to disable the udev rules installed by LVM2 (if built with
# --enable-udev_rules). LVM2 will then manage the /dev nodes and
symlinks
# for active logical volumes directly itself.
# N.B. Manual intervention may be required if this setting is changed
# while any logical volumes are active.
udev_rules = 1

# Set to 1 for LVM2 to verify operations performed by udev. This
turns on
# additional checks (and if necessary, repairs) on entries in the
device
# directory after udev has completed processing its events.
# Useful for diagnosing problems with LVM2/udev interactions.
verify_udev_operations = 0

# If set to 1 and if deactivation of an LV fails, perhaps because
# a process run from a quick udev rule temporarily opened the device,
# retry the operation for a few seconds before failing.
retry_deactivation = 1

# How to fill in missing stripes if activating an incomplete volume.
# Using "error" will make inaccessible parts of the device return
# I/O errors on access. You can instead use a device path, in which
# case, that device will be used to in place of missing stripes.
# But note that using anything other than "error" with mirrored
# or snapshotted volumes is likely to result in data corruption.
missing_stripe_filler = "error"

# The linear target is an optimised version of the striped target
# that only handles a single stripe. Set this to 0 to disable this
# optimisation and always use the striped target.
use_linear_target = 1

# How much stack (in KB) to reserve for use while devices suspended
# Prior to version 2.02.89 this used to be set to 256KB
reserved_stack = 64

# How much memory (in KB) to reserve for use while devices suspended

```

```

reserved_memory = 8192

# Nice value used while devices suspended
process_priority = -18

# If volume_list is defined, each LV is only activated if there is a
# match against the list.
#
# "vgname" and "vgname/lvname" are matched exactly.
# "@tag" matches any tag set in the LV or VG.
# "@*" matches if any tag defined on the host is also set in the LV
or VG
#
# If any host tags exist but volume_list is not defined, a default
# single-entry list containing "@*" is assumed.
#
# volume_list = [ "vg1", "vg2/lvol1", "@tag1", "@*" ]

# If auto_activation_volume_list is defined, each LV that is to be
# activated with the autoactivation option (--activate ay/-a ay) is
# first checked against the list. There are two scenarios in which
# the autoactivation option is used:
#
# - automatic activation of volumes based on incoming PVs. If all
the
# PVs making up a VG are present in the system, the
autoactivation
# is triggered. This requires lvmtd (global/use_lvmtd=1) and
udev
# to be running. In this case, "pvscan --cache -aay" is called
# automatically without any user intervention while processing
# udev events. Please, make sure you define
auto_activation_volume_list
# properly so only the volumes you want and expect are
autoactivated.
#
# - direct activation on command line with the autoactivation
option.
# In this case, the user calls "vgchange --activate ay/-a ay" or
# "lvchange --activate ay/-a ay" directly.
#
# By default, the auto_activation_volume_list is not defined and all
# volumes will be activated either automatically or by using --
activate ay/-a ay.
#
# N.B. The "activation/volume_list" is still honoured in all cases so
even
# if the VG/LV passes the auto_activation_volume_list, it still needs
to
# pass the volume_list for it to be activated in the end.

# If auto_activation_volume_list is defined but empty, no volumes
will be
# activated automatically and --activate ay/-a ay will do nothing.
#
# auto_activation_volume_list = []

```

```

# If auto_activation_volume_list is defined and it's not empty, only
matching
# volumes will be activated either automatically or by using --
activate ay/-a ay.
#
# "vgname" and "vgname/lvname" are matched exactly.
# "@tag" matches any tag set in the LV or VG.
# "@*" matches if any tag defined on the host is also set in the LV
or VG
#
# auto_activation_volume_list = [ "vg1", "vg2/lvol1", "@tag1", "@*" ]

# If read_only_volume_list is defined, each LV that is to be
activated
# is checked against the list, and if it matches, it is activated
# in read-only mode. (This overrides '--permission rw' stored in the
# metadata.)
#
# "vgname" and "vgname/lvname" are matched exactly.
# "@tag" matches any tag set in the LV or VG.
# "@*" matches if any tag defined on the host is also set in the LV
or VG
#
# read_only_volume_list = [ "vg1", "vg2/lvol1", "@tag1", "@*" ]

# Each LV can have an 'activation skip' flag stored persistently
against it.
# During activation, this flag is used to decide whether such an LV
is skipped.
# The 'activation skip' flag can be set during LV creation and by
default it
# is automatically set for thin snapshot LVs. The
'auto_set_activation_skip'
# enables or disables this automatic setting of the flag while LVs
are created.
# auto_set_activation_skip = 1

# For RAID or 'mirror' segment types, 'raid_region_size' is the
# size (in KiB) of each:
# - synchronization operation when initializing
# - each copy operation when performing a 'pvmove' (using 'mirror'
segtype)
# This setting has replaced 'mirror_region_size' since version
2.02.99
raid_region_size = 512

# Setting to use when there is no readahead value stored in the
metadata.
#
# "none" - Disable readahead.
# "auto" - Use default value chosen by kernel.
readahead = "auto"

# 'raid_fault_policy' defines how a device failure in a RAID logical
# volume is handled. This includes logical volumes that have the

```

```

following
  # segment types: raid1, raid4, raid5*, and raid6*.
  #
  # In the event of a failure, the following policies will determine
what
  # actions are performed during the automated response to failures
(when
    # dmeventd is monitoring the RAID logical volume) and when
'lvconvert' is
    # called manually with the options '--repair' and '--use-policies'.
    #
    # "warn" - Use the system log to warn the user that a device in the
RAID
    #     logical volume has failed. It is left to the user to run
    #     'lvconvert --repair' manually to remove or replace the failed
    #     device. As long as the number of failed devices does not
    #     exceed the redundancy of the logical volume (1 device for
    #     raid4/5, 2 for raid6, etc) the logical volume will remain
    #     usable.
    #
    # "allocate" - Attempt to use any extra physical volumes in the
volume
        #     group as spares and replace faulty devices.
        #
        raid_fault_policy = "warn"

    # 'mirror_image_fault_policy' and 'mirror_log_fault_policy' define
    # how a device failure affecting a mirror (of "mirror" segment type)
is
        # handled. A mirror is composed of mirror images (copies) and a log.
        # A disk log ensures that a mirror does not need to be re-synced
        # (all copies made the same) every time a machine reboots or crashes.
        #
        # In the event of a failure, the specified policy will be used to
determine
            # what happens. This applies to automatic repairs (when the mirror is
being
            # monitored by dmeventd) and to manual lvconvert --repair when
            # --use-policies is given.
            #
            # "remove" - Simply remove the faulty device and run without it. If
            #             the log device fails, the mirror would convert to using
            #             an in-memory log. This means the mirror will not
            #             remember its sync status across crashes/reboots and
            #             the entire mirror will be re-synced. If a
            #             mirror image fails, the mirror will convert to a
            #             non-mirrored device if there is only one remaining good
            #             copy.
            #
            # "allocate" - Remove the faulty device and try to allocate space on
            #             a new device to be a replacement for the failed device.
            #             Using this policy for the log is fast and maintains the
            #             ability to remember sync state through crashes/reboots.
            #             Using this policy for a mirror device is slow, as it
            #             requires the mirror to resynchronize the devices, but
it

```

```

# will preserve the mirror characteristic of the device.
# This policy acts like "remove" if no suitable device and
# space can be allocated for the replacement.
#
# "allocate_anywhere" - Not yet implemented. Useful to place the log
device
# temporarily on same physical volume as one of the mirror
#
# images. This policy is not recommended for mirror
devices
# since it would break the redundant nature of the
mirror. This
# policy acts like "remove" if no suitable device and
space can
# be allocated for the replacement.

mirror_log_fault_policy = "allocate"
mirror_image_fault_policy = "remove"

# 'snapshot_autoextend_threshold' and 'snapshot_autoextend_percent'
define
# how to handle automatic snapshot extension. The former defines when
the
# snapshot should be extended: when its space usage exceeds this many
# percent. The latter defines how much extra space should be
allocated for
# the snapshot, in percent of its current size.
#
# For example, if you set snapshot_autoextend_threshold to 70 and
# snapshot_autoextend_percent to 20, whenever a snapshot exceeds 70%
usage,
# it will be extended by another 20%. For a 1G snapshot, using up
700M will
# trigger a resize to 1.2G. When the usage exceeds 840M, the snapshot
will
# be extended to 1.44G, and so on.
#
# Setting snapshot_autoextend_threshold to 100 disables automatic
# extensions. The minimum value is 50 (A setting below 50 will be
treated
# as 50).

snapshot_autoextend_threshold = 100
snapshot_autoextend_percent = 20

# 'thin_pool_autoextend_threshold' and
'thin_pool_autoextend_percent' define
# how to handle automatic pool extension. The former defines when the
# pool should be extended: when its space usage exceeds this many
# percent. The latter defines how much extra space should be
allocated for
# the pool, in percent of its current size.
#
# For example, if you set thin_pool_autoextend_threshold to 70 and
# thin_pool_autoextend_percent to 20, whenever a pool exceeds 70%
usage,
# it will be extended by another 20%. For a 1G pool, using up 700M

```

```

will
  # trigger a resize to 1.2G. When the usage exceeds 840M, the pool
will
  # be extended to 1.44G, and so on.
#
# Setting thin_pool_autoextend_threshold to 100 disables automatic
# extensions. The minimum value is 50 (A setting below 50 will be
treated
# as 50).

thin_pool_autoextend_threshold = 100
thin_pool_autoextend_percent = 20

# While activating devices, I/O to devices being (re)configured is
# suspended, and as a precaution against deadlocks, LVM2 needs to pin
# any memory it is using so it is not paged out. Groups of pages
that
# are known not to be accessed during activation need not be pinned
# into memory. Each string listed in this setting is compared
against
# each line in /proc/self/maps, and the pages corresponding to any
# lines that match are not pinned. On some systems locale-archive
was
# found to make up over 80% of the memory used by the process.
# mlock_filter = [ "locale/locale-archive", "gconv/gconv-
modules.cache" ]

# Set to 1 to revert to the default behaviour prior to version
2.02.62
# which used mlockall() to pin the whole process's memory while
activating
# devices.
use_mlockall = 0

# Monitoring is enabled by default when activating logical volumes.
# Set to 0 to disable monitoring or use the --ignoremonitoring
option.
monitoring = 1

# When pvmove or lvconvert must wait for the kernel to finish
# synchronising or merging data, they check and report progress
# at intervals of this number of seconds. The default is 15 seconds.
# If this is set to 0 and there is only one thing to wait for, there
# are no progress reports, but the process is awoken immediately the
# operation is complete.
polling_interval = 15
}

#####
# Advanced section #
#####

# Metadata settings
#
# metadata {

```

```

# Default number of copies of metadata to hold on each PV. 0, 1 or
2.
# You might want to override it from the command line with 0
# when running pvcreate on new PVs which are to be added to large
VGs.

# pvmetadatacopies = 1

# Default number of copies of metadata to maintain for each VG.
# If set to a non-zero value, LVM automatically chooses which of
# the available metadata areas to use to achieve the requested
# number of copies of the VG metadata. If you set a value larger
# than the total number of metadata areas available then
# metadata is stored in them all.
# The default value of 0 ("unmanaged") disables this automatic
# management and allows you to control which metadata areas
# are used at the individual PV level using 'pvchange
# --metadataignore y/n'.

# vgmetadatacopies = 0

# Approximate default size of on-disk metadata areas in sectors.
# You should increase this if you have large volume groups or
# you want to retain a large on-disk history of your metadata
changes.

# pvmetasize = 255

# List of directories holding live copies of text format metadata.
# These directories must not be on logical volumes!
# It's possible to use LVM2 with a couple of directories here,
# preferably on different (non-LV) filesystems, and with no other
# on-disk metadata (pvmetadatacopies = 0). Or this can be in
# addition to on-disk metadata areas.
# The feature was originally added to simplify testing and is not
# supported under low memory situations - the machine could lock up.
#
# Never edit any files in these directories by hand unless you
# are absolutely sure you know what you are doing! Use
# the supplied toolset to make changes (e.g. vgcfgrestore).

# dirs = [ "/etc/lvm/metadata", "/mnt/disk2/lvm/metadata2" ]
#}

# Event daemon
#
dmeventd {
    # mirror_library is the library used when monitoring a mirror
device.
    #
    # "libdevmapper-event-lvm2mirror.so" attempts to recover from
    # failures. It removes failed devices from a volume group and
    # reconfigures a mirror as necessary. If no mirror library is
    # provided, mirrors are not monitored through dmeventd.

    mirror_library = "libdevmapper-event-lvm2mirror.so"
}

```

```
# snapshot_library is the library used when monitoring a snapshot
device.
#
# "libdevmapper-event-lvm2snapshot.so" monitors the filling of
# snapshots and emits a warning through syslog when the use of
# the snapshot exceeds 80%. The warning is repeated when 85%, 90% and
# 95% of the snapshot is filled.

snapshot_library = "libdevmapper-event-lvm2snapshot.so"

# thin_library is the library used when monitoring a thin device.
#
# "libdevmapper-event-lvm2thin.so" monitors the filling of
# pool and emits a warning through syslog when the use of
# the pool exceeds 80%. The warning is repeated when 85%, 90% and
# 95% of the pool is filled.

thin_library = "libdevmapper-event-lvm2thin.so"

# Full path of the dmeventd binary.
#
# executable = "/usr/sbin/dmeventd"
}
```

## 附录 C. LVM 对象标签

LVM 标签是可将有相同类型的 LVM2 对象分为同一组的用词。标签可以附加到对象中，比如物理卷、卷组、逻辑卷、片段。也可将标签附加到集群配置的主机中。无法为快照添加标签。

可在命令行的 PV、VG 或者 LV 参数中使用标签。标签应该有 @ 作为前缀以避免混淆。每个标签都可用所有对象都拥有的标签取代来扩大范围，标签的类型根据它在命令行的位置确定。

LVM 标签是最长可为 1024 个字符的字符串。LVM 标签可使用小横线开始。

有效标签仅由有限字符范围组成。允许的字符包括 [A-Za-z0-9\_+.-]。从 Red Hat Enterprise Linux 6.1 发行本开始，允许的字符列表扩大为可包含 "/"、"="、"!"、":"、"#" 和"&" 字符。

只能为卷组中的对象添加标签。如果从卷组中删除物理卷，它们就会丢失其标签。这是因为标签是作为卷组元数据的一部分保存的，并在删除物理卷时被删除。无法为快照添加标签。

以下命令列出所有带 **database** 标签的逻辑卷。

```
lvs @database
```

以下命令列出目前活跃的主机标签。

```
lvm tags
```

### C.1. 添加和删除对象标签

请使用 **pvchange** 命令的 **--addtag** 或者 **--deletag** 选项在物理卷中添加或者删除标签。

请使用 **vgchange** 或 **vgcreate** 命令的 **--addtag** 或者 **--deletag** 选项在卷组中添加或者删除标签，。

请请使用 **lvchange** 或 **lvcreate** 命令的 **--addtag** 或者 **--deletag** 选项在逻辑卷中添加或者删除标签，。

可以在 **pvchange**、**vgchange** 或者 **lvchange** 命令中指定多个 **--addtag** 和 **--deletag** 参数。例如：下面的命令可从卷组 **grant** 中删除标签 **T9** 和 **T10**，同时添加标签 **T13** 和 **T14**。

```
vgchange --deletag T9 --deletag T10 --addtag T13 --addtag T14 grant
```

### C.2. 主机标签

在集群配置中，可以在配置文件中定义主机标签。如果在 **tags** 部分设定 **hosttags = 1**，就会自动使用机器的主机名定义主机标签。这样可允许在所有机器中使用通用配置文件，以便其有该文件的相同的副本，但会根据主机名有不同的动作。

有关配置文件的详情请参考 [附录 B, LVM 配置文件](#)。

对于每个主机标签，会读取存在额外的配置文件 **lvm\_hosttag.conf**。如果那个文件定义了新的标签，那么会在要读取的文件列表中添加进一步的配置文件。

例如：下面配置文件中的条目总是定义 **tag1**，如果主机名为 **host1** 则定义 **tag2**。

```
tags { tag1 { } tag2 { host_list = ["host1"] } }
```

### C.3. 使用标签控制激活

可以在配置文件中指定在那个主机中只应该激活某个逻辑卷。例如：下面的条目作为激活请求的过滤器使用（比如 **vgchange -ay**），且只激活 **vg1/lvol0** 以及那些在该主机的元数据中带 **database** 标签的逻辑卷和卷组。

```
activation { volume_list = ["vg1/lvol0", "@database"] }
```

有一个特殊的映射“@\*”，只有该机器中任意元数据标签与任意主机标签匹配时方可匹配。

另一个例子就是，想象在该集群的每台机器，其配置文件中均有以下条目：

```
tags { hosttags = 1 }
```

如果只在主机 **db2** 中激活 **vg1/lvol2**，请执行以下操作：

1. 可在集群中的任意主机中运行 **lvchange --addtag @db2 vg1/lvol2**。
2. 运行 **lvchange -ay vg1/lvol2**。

这个解决方案包括将主机名保存在卷组元数据中。

## 附录 D. LVM 卷组元数据

卷组的配置详情被称为元数据。默认情况下，卷组的每个物理卷元数据区域中均保存完全相同的元数据副本。

如果卷组包含很多物理卷，那么有很多元数据的冗余副本不是很有效。可以使用 **pvccreate** 命令的 **--metadatacopies 0** 选项创建没有任何元数据副本的物理卷。选择物理卷中包含的元数据副本数目后就无法再进行修改。选择零副本将在修改配置时提高更新速度。注意：虽然任何时候每个卷组必须至少包含一个带元数据区域的物理卷（除非您使用高级配置设置允许您在文件系统中保存卷组元数据）。如果将来要分割卷组，那么每个卷组至少需要一个元数据副本。

核心元数据以 ASCII 格式保存。元数据区域是一个环形缓冲。新的元数据会附加在旧的元数据之后，然后会更新其起始指针。

可使用 **pvccreate** 命令的 **--metadatasize** 选项指定元数据区域的大小。对于包含数百个物理卷和逻辑卷的卷组来说，默认大小可能太小。

### D.1. 物理卷标签

默认情况下，**pvccreate** 命令会在第二个 512 字节扇区放置物理卷标签。这个标签可选择性地放在前四个扇区中的任意一个，因为扫描物理卷标签的 LVM 工具会检查前四个扇区。物理卷标签以字符串 **LABELONE** 开始。

物理卷标签包含：

- » 物理卷 UUID
- » 以字节为单位的块设备大小
- » 以 NULL 结尾的数据区域位置列表
- » 以 NULL 结尾的元数据区域位置列表

元数据位置以偏移和大小（单位：字节）形式保存。标签中有大约 15 个位置的空间，但 LVM 工具目前仅使用 3 个位置：即单数据区域以及最多两个元数据区域。

### D.2. 元数据内容

卷组元数据包含：

- » 何时以及如何创建该卷组的信息
- » 卷组信息：

卷组信息包括：

- » 名称和唯一 id
- » 更新元数据时增大的版本数
- » 任意属性：读/写？重新定义大小？
- » 它可能包含的所有对物理卷和逻辑卷数量的管理限制
- » 扩展大小（以扇区为单位，大小为 512 字节）
- » 一个未排序物理卷列表组成的卷组，每个都附带：
  - » 它的 UUID，用来决定包含它的块设备

- 所有属性，比如物理卷是否可分配
  - 物理卷中第一个扩展的偏移（在扇区中）
  - 扩展的数目
- » 未排序的逻辑卷列表，每个逻辑卷都包含
- 排序的逻辑卷片段列表。每个片段的元数据都包括用于排序的物理卷片段或者逻辑卷片段的映射

### D.3. 元数据示例

以下是名为 **myvg** 卷组的 LVM 卷组元数据示例。

```
# Generated by LVM2: Tue Jan 30 16:28:15 2007

contents = "Text Format Volume Group"
version = 1

description = "Created *before* executing 'lvextend -L+5G /dev/myvg/mylv
               /dev/sdc'"

creation_host = "tng3-1"           # Linux tng3-1 2.6.18-8.el5 #1 SMP Fri
Jan 26 14:15:21 EST 2007 i686
creation_time = 1170196095        # Tue Jan 30 16:28:15 2007

myvg {
    id = "0zd3UT-wbYT-1DHq-1MPs-EjoE-0o18-wL28X4"
    seqno = 3
    status = ["RESIZEABLE", "READ", "WRITE"]
    extent_size = 8192             # 4 Megabytes
    max_lv = 0
    max_pv = 0

    physical_volumes {

        pv0 {
            id = "ZBW5qW-dXF2-0bGw-ZCad-2R1V-phwu-1c1RFt"
            device = "/dev/sda"      # Hint only

            status = ["ALLOCATABLE"]
            dev_size = 35964301     # 17.1491 Gigabytes
            pe_start = 384
            pe_count = 4390 # 17.1484 Gigabytes
        }

        pv1 {
            id = "ZHEZJW-MR64-D3QM-Rv7V-Hxsa-zU24-wztY19"
            device = "/dev/sdb"      # Hint only

            status = ["ALLOCATABLE"]
            dev_size = 35964301     # 17.1491 Gigabytes
            pe_start = 384
            pe_count = 4390 # 17.1484 Gigabytes
        }
    }
}
```

```

pv2 {
    id = "wCoG4p-55Ui-9tbp-VTEA-j06s-RAVx-UREW0G"
    device = "/dev/sdc"      # Hint only

    status = ["ALLOCATABLE"]
    dev_size = 35964301      # 17.1491 Gigabytes
    pe_start = 384
    pe_count = 4390 # 17.1484 Gigabytes
}

pv3 {
    id = "hGlUwi-zsBg-39FF-do88-pHxY-8XA2-9WKIiA"
    device = "/dev/sdd"      # Hint only

    status = ["ALLOCATABLE"]
    dev_size = 35964301      # 17.1491 Gigabytes
    pe_start = 384
    pe_count = 4390 # 17.1484 Gigabytes
}
}

logical_volumes {

    mylv {
        id = "GhUYSF-qVM3-rzQo-a6D2-o0aV-LQet-Ur90F9"
        status = ["READ", "WRITE", "VISIBLE"]
        segment_count = 2

        segment1 {
            start_extent = 0
            extent_count = 1280      # 5 Gigabytes

            type = "striped"
            stripe_count = 1        # linear

            stripes = [
                "pv0", 0
            ]
        }
        segment2 {
            start_extent = 1280
            extent_count = 1280      # 5 Gigabytes

            type = "striped"
            stripe_count = 1        # linear

            stripes = [
                "pv1", 0
            ]
        }
    }
}
}

```

## 附录 E. 修订历史

<b>修订 0.2-7.2</b>	<b>Wed Aug 31 2016</b>	<b>Leah Liu</b>
完成翻译、校对		
<b>修订 0.2-7.1</b>	<b>Wed Aug 31 2016</b>	<b>Leah Liu</b>
与 XML 源 0.2-7 版本同步的翻译文件		
<b>修订 0.2-7</b>	<b>Mon Feb 16 2015</b>	<b>Steven Levine</b>
用于 7.1 GA 发行本的版本		
<b>修订 0.2-6</b>	<b>Thu Dec 11 2014</b>	<b>Steven Levine</b>
用于 7.1 Beta 发行本的版本		
<b>修订 0.2-3</b>	<b>Tue Dec 2 2014</b>	<b>Steven Levine</b>
解决: #1136078 使用移植到 Red Hat Enterprise Linux 6.6 中的新功能更新文档。		
解决: #1022850, #1111395 更新文档以体现 dm-cache 和 lvm 缓存支持。		
解决: #1102841 论述用于精简池快照的 -k 和 -K 选项。		
解决: #1093227, #1103916, #1055662, #1121277, #1121767, #1113756 阐明并改正文档中的小问题。		
解决: #1093059 添加 lvm 配置文件文档。		
解决: #1030639, #1009575 论述使用 lvmetad 时的全局过滤器要求。		
解决: #1103924 记录 dumpconfig 功能。		
解决: #1158595 论述 vgreduce 的 --mirrosonly 选项。		
解决: #987074 添加 lvm 命令一节。		
<b>修订 0.2-2</b>	<b>Wed Nov 5 2014</b>	<b>Steven Levine</b>
添加缓存卷参考，并记录 vgreduce 的 --removemirrors 选项。		
<b>修订 0.2-1</b>	<b>Fri Oct 24 2014</b>	<b>Steven Levine</b>
使用为版本 6.6 添加的新内容更新。		
<b>修订 0.1-23</b>	<b>Thu Sep 11 2014</b>	<b>Steven Levine</b>
重建以便修复 html-single 格式问题。		
<b>修订 0.1-22</b>	<b>Mon Jun 2 2014</b>	<b>Steven Levine</b>

## 7.0 GA 发行版本

## 修订 0.1-20

Mon May 19 2014

Steven Levine

7.0 版本的草稿更新

解决: #986445, #1082115, #986449

记录 RAID 支持

解决: #1056587

删除过期的工具参考

解决: #1070098

改正小的排版错误。

解决: #794809

论述更新的快照支持

## 修订 0.1-13

Fri May 9 2014

Steven Levine

重建以便体现风格更改

## 修订 0.1-5

Fri Dec 6 2013

Steven Levine

Bata 发行本。

## 修订 0.1-1

Wed Jan 16 2013

Steven Levine

建立文档的 Red Hat Enterprise Linux 6 版本分支。

# 索引

## 符号

[/lib/udev/rules.d directory, 使用 Device Mapper 的 udev 整合](#)[停用卷组, 激活和停用卷组](#)

- 仅用于一个节点, [激活和停用卷组](#)
- 只用于本地卷, [激活和停用卷组](#)

## 元数据

- 备份, [逻辑卷备份](#), [备份卷组元数据](#)
- 恢复, [恢复物理卷元数据](#)

[元数据守护进程, 元数据守护进程 \(lvmtd\)](#)

## 分区

- 多个, [一个磁盘中有多个分区](#)

[分区类型, 设定, 设定分区类型](#)[分配, LVM 分配](#)

- 策略, [创建卷组](#)
- 防止, [防止在物理卷中分配](#)

## 创建

- 卷组, [创建卷组](#)

- 卷组，集群的，在集群中创建卷组
- 条带逻辑卷，示例，[创建条带逻辑卷](#)
- 物理卷，[创建物理卷](#)
- 逻辑卷，[创建线性逻辑卷](#)
- 逻辑卷，示例，在三个磁盘中创建 LVM 逻辑卷
- 集群中的 LVM 卷，在集群中创建 LVM 卷

## 创建 LVM 卷

- 概述，[创建逻辑卷概述](#)

## 初始化

- 分区，[初始化物理卷](#)
- 物理卷，[初始化物理卷](#)

## 删除

- 物理卷，[删除物理卷](#)
- 逻辑卷，[删除逻辑卷](#)
- 逻辑卷中的磁盘，[从逻辑卷中删除磁盘](#)

## 单元，命令行，使用 CLI 命令

### 卷组

- vgs 显示参数，[vgs 命令](#)
- 停用，[激活和停用卷组](#)
- 减少，[从卷组中删除物理卷](#)
- 分割
  - 示例过程，[分割卷组](#)
- 创建，[创建卷组](#)
- 创建集群，在集群中创建卷组
- 删除，[删除卷组](#)
- 合并，[组合卷组](#)
- 在系统间移动，[将卷组移动到其他系统](#)
- 增长，[在卷组中添加物理卷](#)
- 定义，[卷组](#)
- 扩展，[在卷组中添加物理卷](#)
- 拆分，[分割卷组](#)
- 显示，[显示卷组](#)，[LVM 的自定义报告](#)，[vgs 命令](#)
- 更改参数，[更改卷组参数](#)
- 激活，[激活和停用卷组](#)
- 管理，[常规](#)，[卷组管理](#)
- 组合，[组合卷组](#)
- 缩小，[从卷组中删除物理卷](#)
- 重命名，[重新命名卷组](#)

## 命令行单元，使用 CLI 命令

### 在线数据重新定位，[在线数据重新定位](#)

### 块设备

- 扫描，[扫描块设备](#)

### 增大逻辑卷

- 逻辑卷，在逻辑卷中增大文件系统

### 备份

- 元数据，[逻辑卷备份](#)，[备份卷组元数据](#)

- 文件, [逻辑卷备份](#)

帮助显示, [使用 CLI 命令](#)

归档文件, [逻辑卷备份](#)

快照卷

- 定义, [快照卷](#)

快照逻辑卷

- 创建, [创建快照卷](#)

扩展

- 分配, [创建卷组](#), [LVM 分配](#)
- 定义, [卷组](#), [创建卷组](#)

扫描

- 块设备, [扫描块设备](#)

扫描设备, 过滤器, [使用过滤器控制 LVM 设备扫描](#)

报告格式, LVM 设备, [LVM 的自定义报告](#)

故障排除, [LVM 故障排除](#)

数据重新定位, 在线, [在线数据重新定位](#)

文件系统

- 增大逻辑卷, [在逻辑卷中增大文件系统](#)

日志, [日志](#)

显示

- 卷组, [显示卷组](#), `vgs` 命令
- 物理卷, [显示物理卷](#), `pvs` 命令
- 输出结果排序, [将 LVM 报告排序](#)
- 逻辑卷, [显示逻辑卷](#), `lvs` 命令

条带逻辑卷

- 创建, [创建条带卷](#)
- 创建示例, [创建条带逻辑卷](#)
- 增长, [扩展条带卷](#)
- 定义, [条带逻辑卷](#)
- 扩展, [扩展条带卷](#)

永久设备号, [永久设备号](#)

没有足够可用扩展信息, [逻辑卷没有足够的可用扩展](#)

激活卷组, [激活和停用卷组](#)

- 只用于本地卷, [激活和停用卷组](#)
- 独立节点, [激活和停用卷组](#)

激活逻辑卷

- 独立节点, [在集群的独立节点中激活逻辑卷](#)

物理卷

- `pvs` 显示参数, [pvs 命令](#)
- 从卷组中删除, [从卷组中删除物理卷](#)
- 创建, [创建物理卷](#)
- 初始化, [初始化物理卷](#)

- 删除，[删除物理卷](#)
- 删除丢失的卷，[从卷组中删除丢失的物理卷。](#)
- 定义，[物理卷](#)
- 布局，[LVM 物理卷布局](#)
- 恢复，[替换丢失的物理卷](#)
- 插图，[LVM 物理卷布局](#)
- 显示，[显示物理卷，LVM 的自定义报告](#)，[pvs 命令](#)
- 添加到卷组中，在[卷组中添加物理卷](#)
- 管理，常规，[物理卷管理](#)
- 重新定义大小，[重新定义物理卷大小](#)

## 物理扩展

- 防止分配，[防止在物理卷中分配](#)

## 管理流程，[LVM 管理概述](#)

### 精简卷

- 创建，[创建精简配置逻辑卷](#)

### 精简快照卷，[精简配置快照卷](#)

### 精简配置快照卷，[精简配置快照卷](#)

### 精简配置快照逻辑卷

- 创建，[创建精简配置快照卷](#)

### 精简配置逻辑卷，[精简配置逻辑卷（精简卷）](#)

- 创建，[创建精简配置逻辑卷](#)

### 线性逻辑卷

- 创建，[创建线性逻辑卷](#)
- 定义，[线性卷](#)
- 转换为镜像，[更改镜像卷配置](#)

### 缓存卷，[缓存卷](#)

#### 缓存文件

- 构建，[为卷组扫描磁盘以构建缓存文件](#)

### 设备号

- 主要，[永久设备号](#)
- 次要，[永久设备号](#)
- 永久，[永久设备号](#)

### 设备大小，最大，[创建卷组](#)

### 设备扫描过滤器，[使用过滤器控制 LVM 设备扫描](#)

### 设备特殊文件目录，[创建卷组](#)

### 设备的设备

- 显示，在[失败的设备中显示信息。](#)

### 设备路径名，[使用 CLI 命令](#)

### 详细输出，[使用 CLI 命令](#)

### 路径名，[使用 CLI 命令](#)

### 过滤器，[使用过滤器控制 LVM 设备扫描](#)

### 逻辑卷

- lvs 显示参数，[lvs 命令](#)
- 创建，[创建线性逻辑卷](#)
- 创建示例，在[三个磁盘中创建 LVM 逻辑卷](#)
- 删除，[删除逻辑卷](#)
- 定义，[逻辑卷，LVM 逻辑卷](#)
- 快照，[创建快照卷](#)
- 扩展，[扩展逻辑卷](#)
- 显示，[显示逻辑卷，LVM 的自定义报告](#)，[lvs 命令](#)
- 更改参数，[更改逻辑卷组的参数](#)
- 本地访问，在[集群的独立节点中激活逻辑卷](#)
- 条带，[创建条带卷](#)
- 激活，[控制逻辑卷激活](#)
- 独占访问，在[集群的独立节点中激活逻辑卷](#)
- 管理，常规，[逻辑卷管理](#)
- 精简配置，[创建精简配置逻辑卷](#)
- 精简配置快照，[创建精简配置快照卷](#)
- 线性，[创建线性逻辑卷](#)
- 缩小，[缩小逻辑卷](#)
- 重命名，[重命名逻辑卷](#)
- 重新定义大小，[重新定义逻辑卷大小](#)
- 镜像，[创建镜像卷](#)

## 配置示例，[LVM 配置示例](#)

### 重命名

- 卷组，[重新命名卷组](#)
- 逻辑卷，[重命名逻辑卷](#)

### 重新定义大小

- 物理卷，[重新定义物理卷大小](#)
- 逻辑卷，[重新定义逻辑卷大小](#)

### 镜像的逻辑卷

- 失败修复，[恢复 LVM 镜像错误](#)

### 镜像逻辑卷

- 创建n，[创建镜像卷](#)
- 设备策略，[镜像逻辑卷失败策略](#)
- 转换为线性，[更改镜像卷配置](#)
- 重新配置，[更改镜像卷配置](#)
- 集群的，在[集群中创建镜像 LVM 逻辑卷](#)

## 集群环境，[集群逻辑卷管理器（CLVM）](#)，[在集群中创建 LVM 卷](#)

A

**archive** 文件，[备份卷组元数据](#)

B

**backup** 文件，[备份卷组元数据](#)

C

**CLVM**

- 定义，[集群逻辑卷管理器（CLVM）](#)

**clvmd** 守护进程，[集群逻辑卷管理器（CLVM）](#)

## L

**lvchange 命令 , 更改逻辑卷组的参数**

**lvconvert 命令 , 更改镜像卷配置**

**lvcreate 命令 , 创建线性逻辑卷**

**lvdisplay 命令 , 显示逻辑卷**

**lvextend 命令 , 扩展逻辑卷**

## LVM

- 卷组 , 定义 , [卷组](#)
- 历史记录 , [LVM 构架概述](#)
- 帮助 , [使用 CLI 命令](#)
- 日志 , [日志](#)
- 构架概述 , [LVM 构架概述](#)
- 标签 , [物理卷](#)
- 物理卷管理 , [物理卷管理](#)
- 物理卷 , 定义 , [物理卷](#)
- 目录结构 , [创建卷组](#)
- 组件 , [LVM 构架概述](#) , [LVM 组件](#)
- 自定义报告格式 , [LVM 的自定义报告](#)
- 逻辑卷管理 , [逻辑卷管理](#)
- 集群的 , [集群逻辑卷管理器 \(CLVM\)](#)

**LVM1 , [LVM 构架概述](#)**

**LVM2 , [LVM 构架概述](#)**

**lvmdiskscan 命令 , 扫描块设备**

**lvmetad 守护进程 , 元数据守护进程 ([lvmetad](#))**

**lvreduce 命令 , 重新定义逻辑卷大小 , 缩小逻辑卷**

**lvremove 命令 , 删除逻辑卷**

**lvrename 命令 , 重命名逻辑卷**

**lvs 命令 , [LVM 的自定义报告](#) , [lvs 命令](#)**

- 显示参数 , [lvs 命令](#)

**lvscan 命令 , 显示逻辑卷**

## M

**man page 显示 , [使用 CLI 命令](#)**

**mirror\_image\_fault\_policy 配置参数 , [镜像逻辑卷失败策略](#)**

**mirror\_log\_fault\_policy 配置参数 , [镜像逻辑卷失败策略](#)**

## P

**pvdisplay 命令 , [显示物理卷](#)**

**pvmove 命令 , [在线数据重新定位](#)**

**pvremove 命令 , [删除物理卷](#)**

**pvresize 命令 , [重新定义物理卷大小](#)**

**pvs 命令 , [LVM 的自定义报告](#)**

- 显示参数 , [pvs 命令](#)

**pvscan 命令 , [显示物理卷](#)**

R

RAID 逻辑卷，[RAID 逻辑卷](#)

- 增长，[扩展 RAID 卷](#)
- 扩展，[扩展 RAID 卷](#)

`rules.d` directory，[使用 Device Mapper 的 udev 整合](#)

U

udev 规则，[使用 Device Mapper 的 udev 整合](#)

udev 设备管理器，[Device Mapper 支持 udev 设备管理器](#)

V

`vgcfbackup` 命令，[备份卷组元数据](#)

`vgcfrestore` 命令，[备份卷组元数据](#)

`vgchange` 命令，[更改卷组参数](#)

`vgcreate` 命令，[创建卷组，在集群中创建卷组](#)

`vgdisplay` 命令，[显示卷组](#)

`vgexport` 命令，[将卷组移动到其他系统](#)

`vgextend` 命令，[在卷组中添加物理卷](#)

`vgimport` 命令，[将卷组移动到其他系统](#)

`vgmerge` 命令，[组合卷组](#)

`vgmknodes` 命令，[重新创建卷组目录](#)

`vgreduce` 命令，[从卷组中删除物理卷](#)

`vgrename` 命令，[重新命名卷组](#)

`vgs` 命令，[LVM 的自定义报告](#)

- 显示参数，[vgs 命令](#)

`vgscan` 命令，[为卷组扫描磁盘以构建缓存文件](#)

`vgsplit` 命令，[分割卷组](#)