Red Hat Enterprise Linux 5
管理逻辑卷管理器

LVM 管理员指南
版 3

Landmann

Red Hat Enterprise Linux 5 管理逻辑卷管理器

LVM 管理员指南
版 3

Landmann
rlandmann@redhat.com

摘要

本书介绍了 LVM 逻辑卷管理器（logical volume manager），其中包括在群集环境中运行 LVM 的资料。本文档的内容只适用于 LVM2 发行版本。

目录

简介

1. 关于此手册

本书介绍了 LVM 逻辑卷管理器（LVM），其中包括在群集环境中运行 LVM 的资料。本文档的内容只适用于 LVM2 发行版本。

2. 读者

本书面向的是那些管理运行 Linux 操作系统的系统管理员。要求熟悉红帽企业版 Linux 5 和 GFS 文件系统管理。

3. 软件版本

表 1. 软件版本

软件	描述
RHEL5	请参考 RHEL5 或者更高版本
GFS	请参考 RHEL5 的 GFS 或者更高版本

4. 相关文档

有关使用红帽企业版 Linux 的详情请参考以下资源：

- *《红帽企业版 Linux 安装指南》* — 为您提供有关安装Red Hat Enterprise Linux 5 的信息。
- *《红帽企业版部署指南》* — 提供有关部署、配置和管理红帽企业版 Linux 5 的信息。

有关Red Hat Enterprise Linux 5 的Red Hat Cluster Suite详情请参考以下资源：

- *《Red Hat Cluster Suite总览》* — 为您提供红帽群集套件的高级总览。
- *《配置和管理红帽群集》* — 提供安装、配置和管理红帽群集组件的信息。
- *《全局文件系统：配置和管理》* — 提供安装、配置和维护红帽 GFS（红帽全局文件系统）的信息。
- *《全局文件系统 2：配置和管理》* — 提供安装、配置和维护红帽 GFS2（红帽全局文件系统 2）的信息。
- *《使用设备映射器多路径》* — 提供有关使用Red Hat Enterprise Linux 5 的设备映射器多路径特性的详情。
- *《使用带全局文件系统的 GNBD》* — 提供带红帽 GFS 的全局网络块设备（GNBD）应用总览。
- *《Linux 虚拟服务器管理》* — 提供用 Linux 虚拟服务器（LVS）配置高性能系统和服务的信息。
- *《红帽群集套件发行注记》* — 提供有关红帽群集套件当前发行本的信息。

Red Hat Enterprise Linux 文档光盘和在线文档 http://www.redhat.com/docs/ 提供Red Hat Cluster Suite文档和其它 Red Hat 文档的 HTML、PDF 以及 RPM 版本。

5. 反馈

我们很乐于了解您有关打印错误或者可使本手册更好工作的想法。请在 Bugzilla（http://bugzilla.redhat.com/bugzilla/）中使用 **rh-cs** 提交您的报告。

Be sure to mention the manual's identifier:

```
Bugzilla component: Documentation-cluster
Book identifier: Cluster_Logical_Volume_Manager(EN)-5 (2009-01-05T15:20)
```

By mentioning this manual's identifier, we know exactly which version of the guide you have.

如果您有任何可改进此文档的建议，请尽量说得具体一些。如果您发现任何错误，请将错误所在部分号码以及上下文包括在内，以便我们比较容易找到这些错误之处。

6. 文档约定

本手册使用几个约定来突出某些用词和短语以及信息的某些片段。

在 PDF 版本以及纸版中，本手册使用在 Liberation 字体套件中选出的字体。如果您在您的系统中安装了 Liberation 字体套件，它还可用于 HTML 版本。如果没有安装，则会显示可替换的类似字体。请注意：红帽企业 Linux 5 以及其后的版本默认包含 Liberation 字体套件。

6.1. 排版约定

我们使用四种排版约定突出特定用词和短语。这些约定及其使用环境如下。

单行粗体

用来突出系统输入，其中包括 shell 命令、文件名以及路径。还可用来突出按键以及组合键。例如：

> 要看到文件您当前工作目录中文件 **my_next_bestselling_novel** 的内容，请在 shell 提示符后输入 **cat my_next_bestselling_novel** 命令并按 **Enter** 键执行该命令。

以上内容包括一个文件名，一个 shell 命令以及一个按键，它们都以固定粗体形式出现，且全部与上下文有所区别。

按键组合与单独按键之间的区别是按键组合是使用加号将各个按键连在一起。例如：

> 按 **Enter** 执行该命令。
>
> 按 **Ctrl**+**Alt**+**F2** 切换到虚拟终端。

第一个示例突出的是要按的特定按键。第二个示例突出了按键组合：一组要同时按下的三个按键。

如果讨论的是源码、等级名称、方法、功能、变量名称以及在段落中提到的返回的数值，那么都会以上述形式出现，即**固定粗体**。例如：

> 与文件相关的等级包括用于文件系统的 **filesystem**、用于文件的 **file** 以及用于目录的 **dir**。每个等级都有其自身相关的权限。

比例粗体

这是指在系统中遇到的文字或者短语，其中包括应用程序名称、对话框文本、标记的按钮、复选框以及单选按钮标签、菜单标题以及子菜单标题。例如：

> 在主菜单条中选择**「系统」** → **「首选项」** → **「鼠标」**启动 **鼠标首选项**。在**「按钮」**标签中点击**「惯用左手鼠标」** 复选框并点击 **关闭**切换到主鼠标按钮从左向右（让鼠标适合左手使用）。
>
> 要在 **gedit** 文件中插入特殊字符，请在主菜单栏中选择**「应用程序」** → **「附件」** → **「字符**

> **映射表」**。接下来选择从 **Character Map** 菜单中选择**Search** → 「查找.....」，在「搜索」字段输入字符名称并点击**「下一个」**按钮。此时会在**「字符映射表」**中突出您搜索的字符。双击突出的字符将其放在**「要复制的文本」**字段中，然后点击**「复制」**按钮。现在返回您的文档，并选择 **gedit** 菜单中的**「编辑」** → **「粘贴」**。

以上文本包括应用程序名称、系统范围菜单名称及项目、应用程序特定菜单名称以及按钮和 GUI 界面中的文本，所有都以比例粗体出现并与上下文区别。

固定粗斜体 或者 ***比例粗斜体***

无论固定粗体或者比例粗体，附加的斜体表示是可替换或者变量文本。斜体表示那些不直接输入的文本或者那些根据环境改变的文本。例如：

> 要使用 ssh 连接到远程机器，请在 shell 提示符后输入 **ssh *username@domain.name***。如果远程机器是 **example.com** 且您在该其机器中的用户名为 john，请输入 **ssh john@example.com**。
>
> **mount -o remount *file-system*** 命令会重新挂载命名的文件系统。例如：要重新挂载 **/home** 文件系统，则命令为 **mount -o remount /home**。
>
> 要查看目前安装的软件包版本，请使用 **rpm -q *package*** 命令。它会返回以下结果：***package-version-release***。

请注意上述使用黑斜体的文字 -- username、domain.name、file-system、package、version 和 release。每个字都是一个站位符，可用作您执行命令时输入的文本，也可作为该系统显示的文本。

不考虑工作中显示标题的标准用法，斜体表示第一次使用某个新且重要的用语。例如：

> Publican 是一个 *DocBook* 发布系统。

6.2. 抬升式引用约定

终端输出和源代码列表要与周围文本明显分开。

将发送到终端的输出设定为 **Mono-spaced Roman** 并显示为：

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

源码列表也设为 **Mono-spaced Roman**，但添加下面突出的语法：

```
static int kvm_vm_ioctl_deassign_device(struct kvm *kvm,
                  struct kvm_assigned_pci_dev *assigned_dev)
{
          int r = 0;
          struct kvm_assigned_dev_kernel *match;

          mutex_lock(&kvm->lock);

          match = kvm_find_assigned_dev(&kvm->arch.assigned_dev_head,
                                        assigned_dev->assigned_dev_id);
          if (!match) {
                  printk(KERN_INFO "%s: device hasn't been assigned before, "
                    "so cannot be deassigned\n", __func__);
                  r = -EINVAL;
                  goto out;
          }

          kvm_deassign_device(kvm, match);

          kvm_free_assigned_device(kvm, match);

out:
          mutex_unlock(&kvm->lock);
          return r;
}
```

6.3. 备注及警告

最后，我们使用三种视觉形式来突出那些可能被忽视的信息。

注意

备注是对手头任务的提示、捷径或者备选的解决方法。忽略提示不会造成负面后果，但您可能会错过一个更省事的诀窍。

重要

重要框中的内容是那些容易错过的事情：配置更改只可用于当前会话，或者在应用更新前要重启的服务。忽略‘重要’框中的内容不会造成数据丢失但可能会让您抓狂。

警告

警告是不应被忽略的。忽略警告信息很可能导致数据丢失。

第 1 章 LVM 逻辑卷管理器

本章介绍了逻辑卷管理器（LVM）内容的高级总览。

1.1. 逻辑卷

卷管理生成了物理存储的摘要层，它可允许您创建逻辑卷存储卷。这样就为您在很多方面提供了更大的灵活性，而不仅仅是直接使用物理存储。

逻辑卷提供存储虚拟化。在逻辑卷中您不受物理磁盘大小限制。另外，软件并不了解硬盘存储配置，因此可以在不停止应用程序或者未挂载文件系统的情况下重新设定大小或者移动逻辑卷。这样可以降低操作消耗。

逻辑卷在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，因为您可以将磁盘和分区集合成一个逻辑卷。
- 重新设定存储池大小

  您可以使用检单的软件命令增大或者减小逻辑卷的大小，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。数据可以在磁盘正在使用时进行重新分配。例如，您可以在删除一个热交换磁盘之前将其清空。
- 方便设备命名

  逻辑存储卷可在用户定义的组群中进行管理，这些组群可按您的要求进行命名。
- 磁盘条带

  您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

  逻辑卷为您提供了一个方便配置数据镜像的方法。
- 卷快照

  使用逻辑卷，您可以提取设备快照，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 中实施这些特性进行了论述。

1.2. LVM 构架总览

在 Linux 操作系统的 RHEL 4 中，LVM2 替换了原来的 LVM1 逻辑卷管理器，LVM2 比 LVM1 内核架构更具普遍性。它相对 LVM1 来说有如下改进：

- 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
- 元数据微调
- 元数据冗余副本

LVM2 可向下兼容 LVM1，但不支持 LVM1 的快照和群集。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，比如分区或者整个磁盘。这个设备是作为 LVM *物理卷*（PV）进行初始化的。

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 Logical Volume Components” shows the components of a simple LVM logical volume:

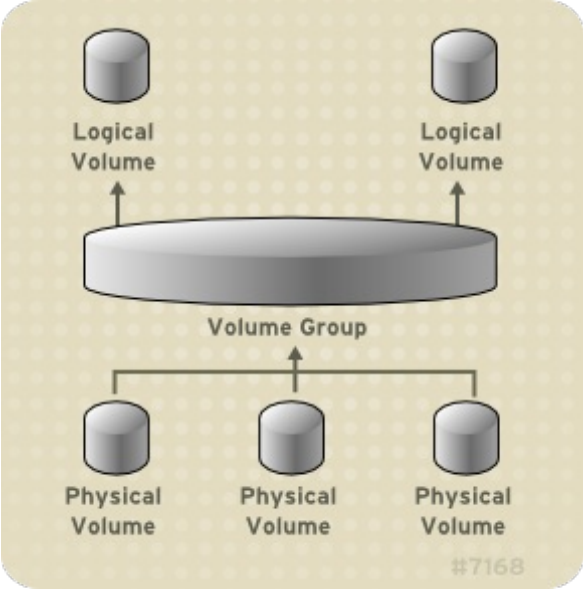

图 **1.1. LVM Logical Volume Components**

For detailed information on the components of an LVM logical volume, see 第 2 章 *LVM 组成*.

1.3. 群集逻辑卷管理器（CLVM）

群集的逻辑卷过滤器（CLVM）是 LVM 的一组群集的扩展。这些扩展允许计算机群集使用 LVM 管理共享存储（例如：在 SAN 中）。

您是否应该使用 CLVM 取决于您的系统要求：

- 如果您的系统中只有一个节点需要访问您正在将其配置为逻辑卷的存储，那么您可以使用 LVM 而不是 LVM 扩展，且在该节点中生成的逻辑卷对该节点来说都是本地的。
- 如果您正在使用群集的系统进行故障切换，其中在任意时间只有那个访问该存储的单一节点是活跃的，您应该使用高性能逻辑卷管理代理（HA-LVM）。有关 HA-LVM 的详情请参考*《配置和管理红帽群集》*。
- 如果您的群集中有一个以上的节点需要访问您的存储，那么该存储会在所有活跃的节点间共享，则您必须使用 CLVM。CLVM 允许用户通过在配置逻辑卷时锁定对物理存储的访问来在共享存储中配置逻辑卷，并使用群集的锁定服务管理共享存储。

要使用 CLVM，则必须运行红帽群集套件软件，其中包括 **clmvd** 守护进程。**clvmd** 守护进程是 LVM 的关键群集扩展。**clvmd** 在每个群集计算机中运行，并在群集中发布 LVM 元数据更新，用相同的逻辑卷视图来显示每个群集计算机。有关安装和管理红帽群集套件的详情请参考*《配置和管理红帽群集》*。

要确定在引导时启动了 **clmvd**，您可对 **clmvd** 服务执行 **chkconfig ... on** 命令，如下：

```
# chkconfig clvmd on
```

如果还没有启动 **clvmd** 守护进程，您可以对 **clvmd** 服务执行 **service ... start** 命令，如下：

```
# service clvmd start
```

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

By default, logical volumes created with CLVM on shared storage are visible to all computers that have access to the shared storage. It is possible, however, to create logical volumes when the storage devices are visible to only one node in the cluster. It is also possible to change the status of a logical volume from a local volume to a clustered volume. For information, see 第 4.3.2 节 “在群集中创建卷组” and 第 4.3.7 节 “修改卷组参数”.

图 1.2 “CLVM 总览” shows a CLVM overview in a Red Hat cluster.

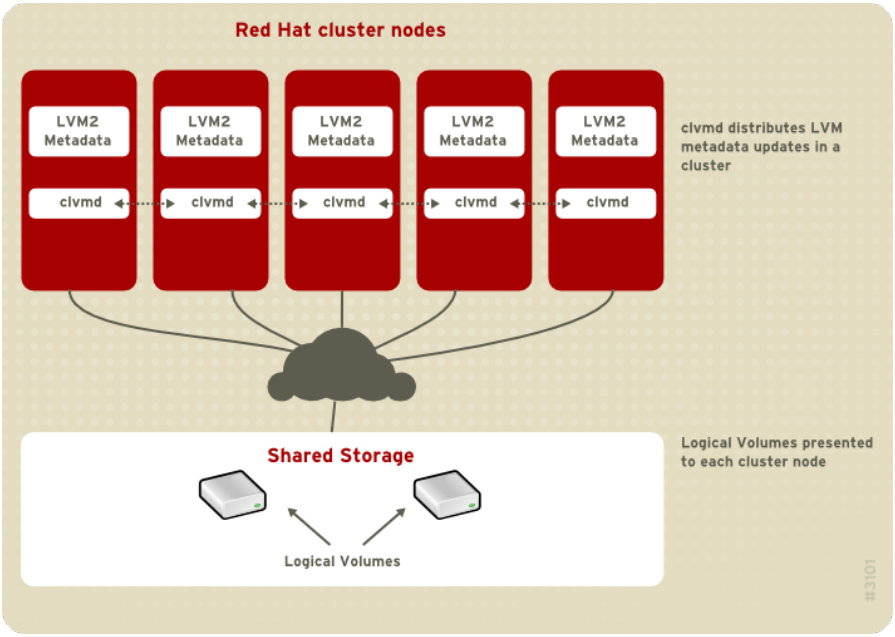

图 1.2. CLVM 总览

注意

在红帽群集套件中使用的共享存储要求您要运行群集逻辑卷管理器守护进程（**clvmd**）或者高可用性逻辑卷管理代理（HA-LVM）。如果您有与操作原因或者您没有正确的权利而无法使用 **clvmd** 守护进程或者 HA-LVM，您就不能在共享磁盘中使用单一事件 LVM，因为这将导致数据崩溃。如果您有任何疑问请与您的红帽服务代表联络。

注意

CLVM requires changes to the **lvm.conf** file for cluster-wide locking. Information on configuring the **lvm.conf** file to support clustered locking is provided within the **lvm.conf** file itself. For information about the **lvm.conf** file, see *附录 B, LVM 配置文件*.

1.4. 文档总览

本文档还包含以下章节：

- 第 2 章 *LVM 组成* describes the components that make up an LVM logical volume.
- 第 3 章 *LVM 管理总览* provides an overview of the basic steps you perform to configure LVM logical volumes, whether you are using the LVM Command Line Interface (CLI) commands or the LVM Graphical User Interface (GUI).
- 第 4 章 *用 CLI 命令管理 LVM* summarizes the individual administrative tasks you can perform with the LVM CLI commands to create and maintain logical volumes.
- 第 5 章 *LVM 配置示例* provides a variety of LVM configuration examples.
- 第 6 章 *LVM 故障排除* provides instructions for troubleshooting a variety of LVM issues.
- 第 7 章 *用 LVM GUI 进行 LVM 管理* summarizes the operating of the LVM GUI.
- 附录 A, *设备映射器（Device Mapper）* describes the Device Mapper that LVM uses to map logical and physical volumes.
- 附录 B, *LVM 配置文件* describes the LVM configuration files.
- 附录 C, *LVM 对象标签* describes LVM object tags and host tags.
- 附录 D, *LVM 卷组元数据* describes LVM volume group metadata, and includes a sample copy of metadata for an LVM volume group.

第 2 章 LVM 组成

本章论述了 LVM 逻辑卷的组成。

2.1. 物理卷

LVM 逻辑卷的基本物理存储单元是块设备，比如分区或者整个磁盘。这个设备是作为 LVM *物理卷*（PV）进行初始化的。将块设备作为物理卷进行初始化会在接近设备起始处放置一个标签。

默认情况下，LVM 标签是放在第二个 512 字节扇区。您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 卷可与其它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 标记在重新引导和整个群集中保持不变。

LVM 标记可识别作为 LVM 物理卷的设备。它为物理卷包含一个随机特定识别符号（UUID）。它还用字节记录块设备的大小，并记录 LVM 元数据在设备中的存储位置。

LVM 元数据包含您的系统中 LVM 卷组的配置详情。在默认情况下，在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副本。LVM 元数据很小，并可以 ASCII 格式保存。

现在，LVM 允许您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。一旦您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了。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。如果您不小心写入了不同于您想要写入的磁盘从而覆盖了磁盘的起始部分，那么您可以使用在设备末尾的元数据第二个副本可让进行恢复。

For detailed information about the LVM metadata and changing the metadata parameters, see 附录 D, *LVM 卷组元数据*.

2.1.1. LVM Physical Volume Layout

图 2.1 “物理卷布局” shows the layout of an LVM physical volume. The LVM label is on the second sector, followed by the metadata area, followed by the usable space on the device.

注意

在 Linux 内核（在整个文档中），每个扇区的大小为 512K。

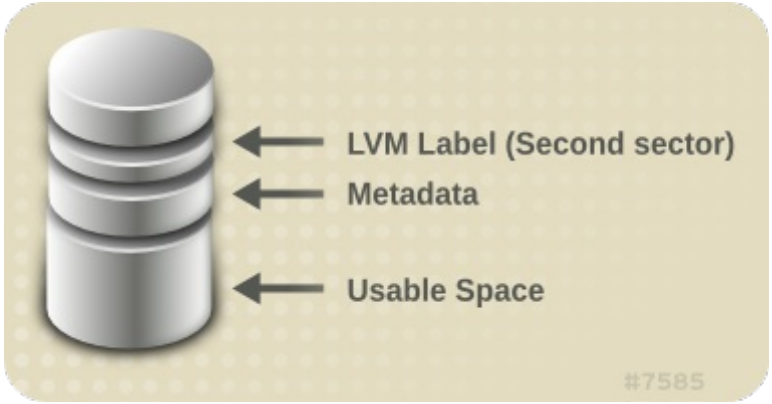

图 **2.1. 物理卷布局**

2.1.2. 一个磁盘中有多个分区

LVM 允许您在磁盘分区外创建物理卷。通常建议您创建可覆盖整个磁盘的单一分区，并将其标记为 LVM 物理卷，理由如下：

- 方便管理

  如果每个正真的磁盘只出现一次会比较容易在系统中追踪硬件，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。
- 条带性能

  LVM 无法告知两个物理卷在同一个物理磁盘中。如果您在两个物理卷在同一个物理磁盘的情况下创建了条状逻辑卷，那么条带就可能在同一磁盘的不同分区中。这可能会降低它的性能而不是提高其性能。

尽管不建议您这样做，但在特别的情况下可能需要您将一个磁盘分成独立的 LVM 物理卷。例如：在有多个磁盘的系统中，当您从现有系统迁移到 LVM 卷的时候，可能需要将数据在分区间转移。另外，如果您有一个很大的磁盘，并且因为管理的原因想要有超过一个卷组，那么对磁盘进行分区是很必要的。如果您的磁盘分区数大于 1，且属于同一卷组，请在创建条状卷的时候小心指定哪些分区是应该包括在逻辑卷中的。

2.2. 卷组

物理卷合并为卷组（VG）。这样就创建了一个磁盘空间池，在它之外可分配逻辑卷。

在卷组中，可用于分配的磁盘空间被分成固定大小的单元，我们称之为扩展。一个扩展就是可被分配的最小单位。在物理卷中，扩展就是物理扩展。

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，卷组会被分成逻辑卷。LVM 逻辑卷有三种类型：*线性*卷、*条状*卷和*镜像*卷。这些在以后的内容中都有论述。

2.3.1. 线性卷

一个线性卷可将多个物理卷集合到一个逻辑卷中。例如：如果您有两个 60GB 的磁盘，您可以创建一个 120GB 的逻辑卷。物理存储是连在一起的。

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

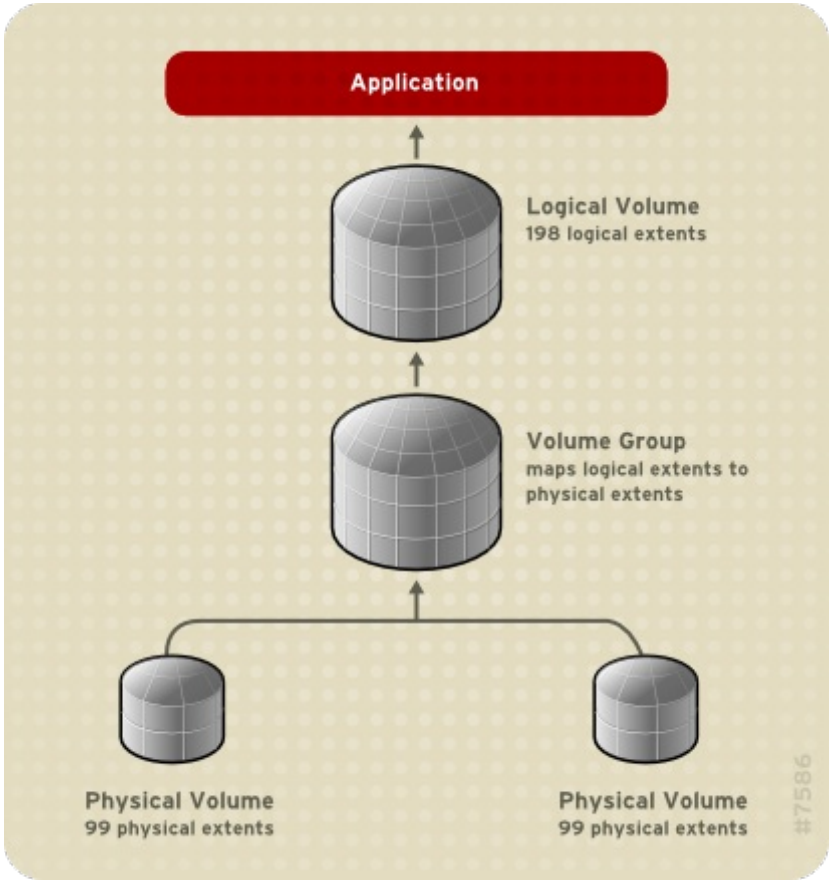

图 **2.2.** 扩展映射

The physical volumes that make up a logical volume do not have to be the same size. 图 2.3 “物理卷大小不同的线性卷” shows volume group **VG1** with a physical extent size of 4MB. This volume group includes 2 physical volumes named **PV1** and **PV2**. The physical volumes are divided into 4MB units, since that is the extent size. In this example, **PV1** is 100 extents in size (400MB) and **PV2** is 200 extents in size (800MB). You can create a linear volume any size between 1 and 300 extents (4MB to 1200MB). In this example, the linear volume named **LV1** is 300 extents in size.

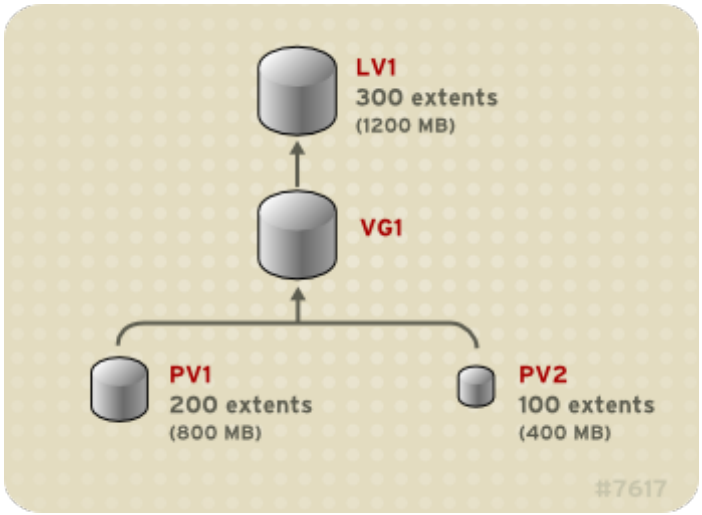

图 **2.3.** 物理卷大小不同的线性卷

You can configure more than one linear logical volume of whatever size you desire from the pool of

physical extents. 图 2.4 “多逻辑卷” shows the same volume group as in 图 2.3 “物理卷大小不同的线性卷”, but in this case two logical volumes have been carved out of the volume group: **LV1**, which is 250 extents in size (1000MB) and **LV2** which is 50 extents in size (200MB).

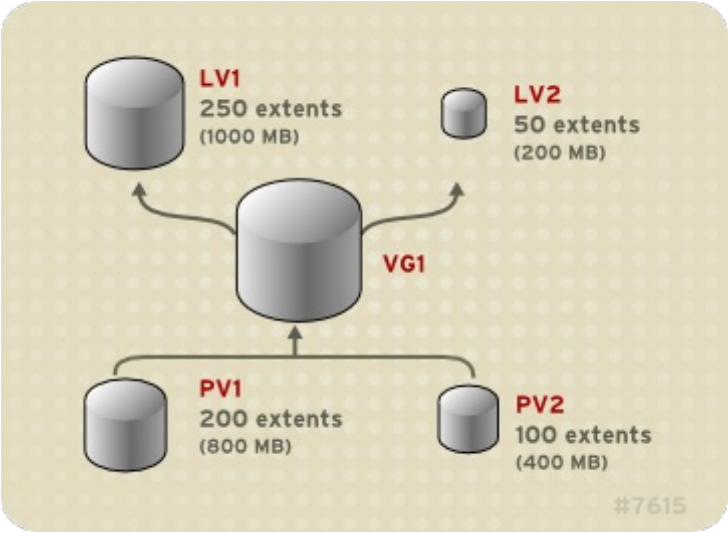

图 **2.4. 多逻辑卷**

2.3.2. 条状逻辑卷

当您向 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。您可以通过创建条状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-round 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状分布。在这个图表中：

- 数据的第一条写入 PV1
- 数据的第二条写入 PV2
- 数据的第三条写入 PV3
- 数据的第四条写入 PV1

在条状逻辑卷中，条的大小不能超过扩展的大小。

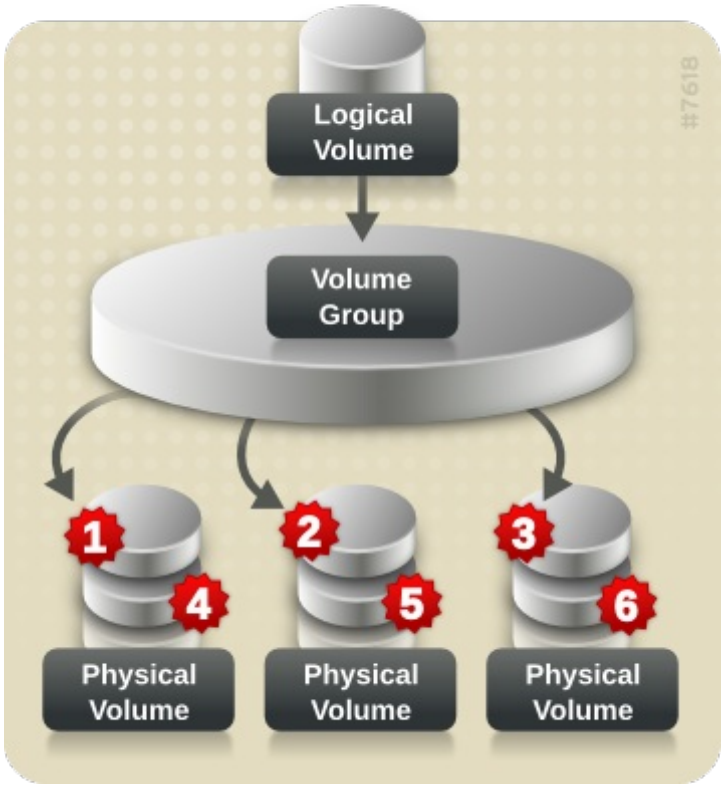

图 **2.5. 在三个 PV 中条状分配数据**

Striped logical volumes can be extended by concatenating another set of devices onto the end of the first set. In order extend a striped logical volume, however, there must be enough free space on the underlying physical volumes that make up the volume group to support the stripe. For example, if you have a two-way stripe that uses up an entire volume group, adding a single physical volume to the volume group will not enable you to extend the stripe. Instead, you must add at least two physical volumes to the volume group. For more information on extending a striped volume, see 第 4.4.9 节 “扩展条状卷”.

2.3.3. 镜像逻辑卷

镜像维护不同设备中的相同的副本。当向一个设备中写入数据时，也会向第二个设备中写入，即镜像保存数据。这提供了在设备失败时的数据保护。当镜像的一个分支失败时，逻辑卷就成为一个线性卷，仍然可访问。

LVM 支持镜像卷。当您创建了一个镜像逻辑卷时，LVM 确定写入基本物理卷的数据被镜像保存到一个独立的物理卷中。使用 LVM，您可以创建有多个镜像的镜像逻辑卷。

LVM 镜像一般以 512KB 为单位分割要复制到区域中的设备。LVM 维护一个小的日志，可用来追踪哪些区域是和镜像同步的。默认情况下，该日志是保存在磁盘中的，这样可以使它在机器重启后得以保存，也可在内存中维护此日志。

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

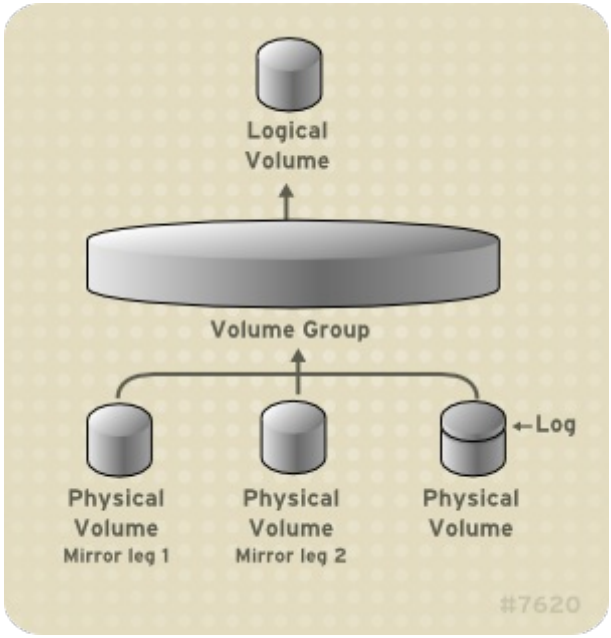

图 **2.6. Mirrored Logical Volume**

注意

在 RHEL 5.3 发行本在群集中还不支持镜像逻辑卷。

For information on creating and modifying mirrors, see 第 4.4.1.3 节 “创建镜像卷”.

2.3.4. 快照卷

LVM 的快照性能为您提供了在某个特定时刻，在不导致服务中断的情况下创建设备的虚拟映射可能性。在提取快照后，当对原始设备进行修改时，快照特性会和在修改前提取快照一样提供一个修改了的数据区域的副本，以便重建设备的状态。

注意

在群集中不支持跨节点的 LVM 快照。

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

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。快照不是备份过程的替代行为。

如果快照已满，那么就会停止提取快照，因此需要确定源文件系统有足够的空间。您可以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。

当您创建文件系统的快照时，仍可能对源系统有完全的读和写访问。如果快照中的一个块修改了，那么就会标记出那个块，并再不从原始卷中复制这个块。

快照特性有几个用途：

- 最典型的就是，当您需要在逻辑卷中在不影响运行系统连续性的情况下执行备份操作时可提取一个快照，这样可以持续地更新数据。
- 您可以在快照文件系统中执行 **fsck** 命令来检查文件系统的完整性，并确定源文件系统是否需要修复。
- 因为快照是可读/写的，您可以通过获取快照并根据快照进行测试，来进行根据产品数据测试应用程序，而不会影响真实数据。
- 您可以使用 Xen 虚拟机器监视器创建要使用的卷。您可以使用快照特性创建磁盘映射并提取快照，还可以修改特定 domU 时刻的快照。然后您可以为另一个 domU 时刻创建另一个快照并修改它。因为只使用了在源或者快照修改的块存储，所以卷的绝大部分是共享的。

第 3 章 LVM 管理总览

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

For descriptions of the CLI commands you can use to perform LVM administration, see 第 4 章 *用 CLI 命令管理 LVM*. Alternately, you can use the LVM GUI, which is described in 第 7 章 *用 LVM GUI 进行 LVM 管理*.

3.1. 在群集中创建 LVM 卷

To create logical volumes in a cluster environment, you use the Clustered Logical Volume Manager (CLVM), which is a set of clustering extensions to LVM. These extensions allow a cluster of computers to manage shared storage (for example, on a SAN) using LVM. In order to use CLVM, the Red Hat Cluster Suite software, including the **clmvd** daemon, must be started at boot time, as described in 第 1.3 节 “群集逻辑卷管理器（CLVM）”.

在群集环境中创建 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，LVM GUI 界面也一样。要启用您在群集中创建的 LVM 卷，群集构架必须正在运行且群集必须有足够节点。

CLVM requires changes to the **lvm.conf** file for cluster-wide locking. Information on configuring the **lvm.conf** file to support clustered locking is provided within the **lvm.conf** file itself. For information about the **lvm.conf** file, see 附录 B, *LVM 配置文件*.

By default, logical volumes created with CLVM on shared storage are visible to all computers that have access to the shared storage. It is possible, however, to create logical volumes when the storage devices are visible to only one node in the cluster. It is also possible to change the status of a logical volume from a local volume to a clustered volume. For information, see 第 4.3.2 节 “在群集中创建卷组” and 第 4.3.7 节 “修改卷组参数”

注意

红帽群集套件中使用的共享存储要求您正在运行群集逻辑卷管理器守护进程（**clvmd**）或者高度可用逻辑卷管理代理（HA-LVM）。如果由于操作原因您无法使用 **clvmd** 守护进程或者 HA-LVM 之一，或者因为您没有正确的权利，您就不能在共享磁盘中使用单一事件 LVM，因为这将导致数据崩溃。如果您有任何疑问请联络您的红帽服务代表。

有关如何安装红帽群集套件以及设置群集构架的详情请参考 *《配置和管理红帽群集》*。

3.2. 创建逻辑卷总览

以下内容对创建 LVM 逻辑卷的步骤进行了总结。

1. 将您要用作 LVM 卷的分区初始化为物理卷（进行标记）。
2. 创建卷组。
3. 创建逻辑卷。

创建逻辑卷后，您可以生成并挂载文件系统。本文档示例中使用的是 GFS 文件系统。

1. 在逻辑卷中用 **gfs_mkfs** 创建 GFS 文件系统。
2. 用 **mkdir** 命令创建一个新的挂载点。在群集的系统中，在群集的所有节点中创建挂载点。

3. 挂载文件系统。您可能想要在 **fstab** 为系统中的每个节点添加一行。

另外，您可以使用 LVM GUI 创建并挂载 GFS 文件系统。

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，而不是在创建卷的机器中。用于存储的服务器有本地副本，但可以重新生成物理卷中的内容。如果 LVM 版本兼容，您可以将物理卷附加到不同服务器上。

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. 创建一个新的物理卷。
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。

如果您的卷组中有足够的未分配空间，您可以使用那些空间来扩展逻辑卷，而不执行步骤 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，除非您在 **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** 中，元数据归档保存在 **/etc/lvm/archive** 中。元数据归档在 **/etc/lvm/archive** 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。

You can manually back up the metadata to the **/etc/lvm/backup** file with the **vgcfgbackup** command. You can restore metadata with the **vgcfgrestore** command. The **vgcfgbackup** and **vgcfgrestore** commands are described in 第 4.3.12 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

The logging levels are set in the **/etc/lvm/lvm.conf** file, which is described in 附录 B, *LVM 配置文件*.

第 4 章 用 CLI 命令管理 LVM

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

注意

If you are creating or modifying an LVM volume for a clustered environment, you must ensure that you are running the **clvmd** daemon. For information, see see 第 3.1 节 “在群集中创建 LVM 卷”.

4.1. 使用 CLI 命令

LVM CLI 命令有一些通用的特性。

当在命令行中需要容量参数时，可以明确指定单位。如果您不指定单位，那么就使用默认的 KB 或者 MB。LVM CLI 不接受分数。

在命令行参数中为 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，且使用 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。

在命令使用卷组或者逻辑卷名称作为参数时，完整路径名称是可选项。卷组 **vg0** 中的逻辑卷 **lvol0** 可被指定为 **vg0/lvol0**。如果空置需要指定卷组的地方，就会代入所有列出的卷组。在需要指定一组逻辑卷的地方给出了卷组，就会代入那个卷组中的列出的逻辑卷。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，它可多次输入来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 **lvcreate** 命令带 **-v** 参数的输出。

```
# 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
```

您还可以使用 **-vv**、**-vvv** 或者 **-vvvv** 参数来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** 命令带 **-vvvv** 参数时给出的输出的前几行。

```
# 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
...
```

您可以用命令的 **--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。

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

4.2. 物理卷管理

这部分论述了对物理卷不同方面进行管理的命令。

4.2.1. 创建物理卷

下面的子部分论述了创建物理卷的命令。

4.2.1.1. 设定分区类型

如果您将整张磁盘作为您的物理卷使用，那么磁盘就必须没有分区表。对于 DOS 磁盘分区，您应该用 **fdisk** 或者 **cfdisk** 或者等同的命令将分区 id 设为 0x8e。如果将整张磁盘作为一个设备使用就必须擦除分区表，这也就会有效地破坏磁盘中的数据。您可以用以下命令将现有分区表的第一个扇区归零，从而删除分区表：

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

4.2.1.2. 初始化物理卷

使用 **pvcreate** 命令来将一个块设备初始化为一个物理卷。初始化和格式化文件系统类似。

以下命令初始化 **/dev/sdd1**、**/dev/sde1** 和 **/dev/sdf1**，将其作为 LVM 物理卷使用。

```
pvcreate /dev/sdd1 /dev/sde1 /dev/sdf1
```

要初始化分区而不是整张磁盘，请在分区中运行 **pvcreate**。以下给出了将 **/dev/hdb1** 初始化成一个 LVM 物理卷，以便以后成为 LVM 逻辑卷的一部分的示例。

```
pvcreate /dev/hdb1
```

4.2.1.3. 扫描块设备

您可以使用 **lvmdiskscan** 命令来扫描用作物理卷的块设备，示例如下。

```
# 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

用来显示 LVM 物理卷属性的命令有三个：**pvs**、**pvdisplay** 和 **pvscan**。

The **pvs** command provides physical volume information in a configurable form, displaying one line per physical volume. The **pvs** command provides a great deal of format control, and is useful for scripting. For information on using the **pvs** command to customize your output, see 第 4.9 节 “为 LVM 自定义报告”.

pvdisplay 命令为每个物理卷提供详细的多行输出。它用混合格式显示物理属性（大小、扩展、卷组等等）。

以下是 **pvdisplay** 为单一物理卷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-XO8M-mcpsVe
```

pvscan 命令在系统中为物理卷扫描所有支持的 LVM 块设备。

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

You can define a filter in the **lvm.conf** so that this command will avoid scanning specific physical volumes. For information on using filters to control which devices are scanned, see 第 4.6 节 “用过滤器控制 LVM 设备扫描”.

4.2.3. 防止在物理卷中进行分配

您可以使用 **pvchange** 命令防止在一个或者多个物理卷的剩余空间进行物理扩展分配。这在出现磁盘错误或者要删除物理卷时是很必要的。

以下命令不允许在 **/dev/sdk1** 中分配物理扩展。

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

您还可以使用 **pvchange** 命令的 **-xy** 参数来允许在之前禁止进行分配的地方分配扩展。

4.2.4. 重新设置物理卷大小

如果您由于任何原因需要修改基本块设备的大小，请使用 **pvresize** 命令来更新 LVM 的大小。您可以在 LVM 正在使用物理卷的时候使用这个命令。

4.2.5. 删除物理卷

如果 LVM 不再使用某个设备，您可以使用 **pvremove** 命令删除 LVM 标签。执行 **pvremove** 会将空白物理卷的 LVM 元数据归零。

If the physical volume you want to remove is currently part of a volume group, you must remove it from the volume group with the **vgreduce** command, as described in 第 4.3.6 节 “从卷组中删除物理卷”.

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

4.3. 卷组管理

这部分论述用于管理卷组各个方面的命令。

4.3.1. 创建卷组

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

以下命令创建一个名为 **vg1** 的卷组，其中包含物理卷 **/dev/sdd1** 和 **/dev/sde1**。

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

当使用物理卷创建卷组时，默认情况下，它的磁盘空间被分成大小为 4MB 的扩展。这个扩展是增大或者减小逻辑卷容量的最小单位。大量的扩展不会影响逻辑卷的 I/O 性能。

如果默认设置不适用于 **-s** 参数时，您可以使用 **vgcreate** 命令指定扩展的大小。您可以对卷组中物理和逻辑卷的数量进行限制，方法就是用 **vgcreate** 命令的 **-p** 和 **-l** 参数。

默认情况下，卷组根据一般规则分配物理扩展，比如不会将平行条带放在同一个物理卷中。这就是 **normal（标准）** 分配策略。您可以使用 **vgcreate** 命令的 **--alloc** 参数来指定 **contiguous**、**anywhere** 或者 **cling** 分配策略。

contiguous（持续） 策略要求新的扩展要和现有的扩展相邻。如果没有足够的可用扩展来满足分配请求，**normal（标准）** 分配策略就无法使用它们，而 **anywhere（任意）** 分配策略会使用它们，即使不惜通过将两个条带放在同一个物理卷中降低性能。**cling（粘贴）** 分配策略会将新的扩展放到同一个物理卷中作为逻辑卷相同条带中现有的扩展使用。可使用 **vgchange** 命令修改这些策略。

一般说来，分配策略只在很特殊的情况下，也就是说您需要指定非常规或者非标准扩展分配的时候才会使用**标准**策略以外的策略。

LVM 卷组和基本逻辑卷是包含在 **/dev** 目录的设备专用文件目录树中的，布局如下：

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

例如：如果您创建两个卷组 **myvg1** 和 **myvg2**，每个带三个名为 **lvo1**、**lvo2** 和 **lvo3** 的逻辑卷，那么就要创建六个设备专用文件：

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

64 位 CPU 中 LVM 的最大设备大小为 8EB。

4.3.2. 在群集中创建卷组

在群集环境中使用 **vgcreate** 命令创建卷组就如同在单一节点中创建卷组。

默认情况下，在所有可访问共享存储的计算机中都可看到 CLVM 在共享存储中创建的卷组。但有可能在使用带 **-c n** 选项的 **vgcreate** 命令创建的本地卷组，只能在群集的一个节点中看到。

当在群集环境中执行以下命令时，创建的卷组对执行该命令的节点来说是本地的。该命令创建一个名为 **vg1** 的卷组，其中包含物理卷 **/dev/sdd1** 和 **/dev/sde1**。

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

You can change whether an existing volume group is local or clustered with the **-c** option of the **vgchange** command, which is described in 第 4.3.7 节 “修改卷组参数”.

您可以使用 **vgs** 命令查看现有卷组是否为群集卷组，如是则会显示 **c** 属性。以下命令显示卷组 **VolGroup00** 和 **testvg1** 的属性。在这个示例中，**VolGroup00** 不是群集的，而 **testvg1** 是群集的，如 **Attr** 标题下的 **c** 属性所示。

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

For more information on the **vgs** command, see 第 4.3.4 节 “显示卷组”第 4.9 节 “为 LVM 自定义报告”, and the **vgs** man page.

4.3.3. 在卷组中添加物理卷

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

以下命令可将物理卷 **/dev/sdf1** 添加到卷组 **vg1** 中。

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

4.3.4. 显示卷组

您可以使用两个命令来显示 LVM 卷组的属性：**vgs** 和 **vgdisplay**。

The **vgscan** command will also display the volume groups, although its primary purpose is to scan all the disks for volume groups and rebuild the LVM cache file. For information on the **vgscan** command, see 第 4.3.5 节 “为卷组扫描磁盘来建立缓存文件”.

The **vgs** command provides volume group information in a configurable form, displaying one line per volume group. The **vgs** command provides a great deal of format control, and is useful for scripting. For information on using the **vgs** command to customize your output, see 第 4.9 节 “为 LVM 自定义报告”.

vgdisplay 命令以固定格式显示卷组属性（比如大小、扩展、物理卷数量等等）。下面的例子就是对卷组 **new_vg** 执行 **vgdisplay** 命令时的输出结果。如果您没有指定卷组，那么就会列出所有现有的卷组。

```
# 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                0
  Cur LV                1
  Open LV               0
  Max PV                0
  Cur PV                3
  Act PV                3
  VG Size               51.42 GB
  PE Size               4.00 MB
  Total PE              13164
  Alloc PE / Size       13 / 52.00 MB
  Free  PE / Size       13151 / 51.37 GB
  VG UUID               jxQJ0a-ZKk0-OpMO-0118-nlwO-wwqd-fD5D32
```

4.3.5. 为卷组扫描磁盘来建立缓存文件

vgscan 命令在系统中扫描所有支持的磁盘设备来查找 LVM 物理卷和卷组。这样就在 **/etc/lvm/.cache** 中建立了一个 LVM 缓存文件，该文件可维护当前列出的 LVM 设备。

LVM 在系统启动以及进行 LVM 操作时自动运行 **vgscan** 命令，比如您执行 **vgcreate** 命令或者当 LVM 检测到不一致性时。您可能在修改硬件配置时需要手动运行 **vgscan**，这样可以让系统识别在引导时没有出现的新设备。例如：当您在 SAN 中的一个系统中添加了新磁盘或者热插拔一个已经被标记为物理卷的新磁盘时，这应该是很需要的。

You can define a filter in the **lvm.conf** file to restrict the scan to avoid specific devices. For information on using filters to control which devices are scanned, see 第 4.6 节 “用过滤器控制 LVM 设备扫描”.

下面的例子显示了 **vgscan** 命令的输出结果。

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

4.3.6. 从卷组中删除物理卷

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

在您从卷组中删除物理卷之前，您可以使用 **pvdisplay** 命令确定物理卷没有被任何逻辑卷使用。

```
# pvdisplay /dev/hda1

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

如果物理卷仍然在使用，您可能必须使用 **pvmove** 命令将数据迁移到另一个物理卷。然后使用 **vgreduce** 命令删除该物理卷：

以下命令可从卷组 **my_volume_group** 中删除物理卷 **/dev/hda1**。

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

4.3.7. 修改卷组参数

There are several volume group parameters that you can change for an existing volume group with the **vgchange** command. Primarily, however, this command is used to deactivate and activate volume groups, as described in 第 4.3.8 节 “激活和失活卷组”,

下面的命令将卷组 **vg00** 的包含逻辑卷的最大数量改为 128。

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

有关 **vgchange** 命令的卷组参数的论述请参考 **vgchange**(8) man page。

4.3.8. 激活和失活卷组

当您创建一个卷组时，默认情况下它是激活的。这就是说该组中的逻辑卷是可访问，也可修改的。

有一些情况需要您失活卷组，使其无法被内核识别。要失活或者激活卷组，请使用 **vgchange** 命令的 **-a**（**--available**）参数。

下面的例子失活了卷组 **my_volume_group**。

```
vgchange -a n my_volume_group
```

如果启用了群集的锁定，添加‘e’来激活或者失活完全占据一个节点的卷组，或者添加‘l’来激活或者失活只位于本地节点的卷组。带单一主机快照的逻辑卷总是被完全激活，因为他们一次只能在一个节点上使用。

You can deactivate individual logical volumes with the **lvchange** command, as described in 第 4.4.4 节 “修改逻辑卷组的参数”, For information on activating logical volumes on individual nodes in a cluster, see 第 4.8 节 “在群集的独立节点中激活逻辑卷”.

4.3.9. 删除卷组

要删除不包含逻辑卷的卷组，请使用 **vgremove** 命令。

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

4.3.10. 分割卷组

要分割卷组的物理卷并创建新的卷组，请使用 **vgsplit** 命令。

无法在卷组间分割逻辑卷。每个现有的逻辑卷必须完整地存在于物理卷中来构成旧的或者新的卷组。如果需要，您可以使用 **pvmove** 命令来进行强制分割。

下面的例子将新的卷组 **smallvg** 从原来的卷组 **bigvg** 中分割出来。

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

4.3.11. 合并卷组

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

下面的命令将失活卷组 **my_vg** 归并到激活或者失活的卷组 **databases** 中，给出详细的运行时信息。

```
vgmerge -v databases my_vg
```

4.3.12. 备份卷组元数据

在每次修改卷组和逻辑卷配置时都会自动进行元数据备份和归档，除非您在 **lvm.conf** 文件中禁用该功能。在默认情况下，元数据备份保存在 **/etc/lvm/backup** 文件中，元数据归档保存在 **/etc/lvm/archives** 中。您可以手动使用 **vgcfgbackup** 命令将元数据备份到 **/etc/lvm/backup** 文件中。

vgcfrestore 命令可为卷组中的所有物理卷从归档中恢复卷组的元数据。

For an example of using the **vgcfgrestore** command to recover physical volume metadata, see 第 6.4 节 “修复物理卷元数据”.

4.3.13. 重命名卷组

使用 **vgrename** 命令来重新命名一个现有的卷组。

可使用以下命令的之一将现有卷组 **vg02** 重新命名为 **my_volume_group**。

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

```
vgrename vg02 my_volume_group
```

4.3.14. 将卷组移动到其它系统中

您可以将整个 LVM 卷组移动到另一个系统中。建议您使用 **vgexport** 和 **vgimport** 命令进行此操作。

vgexport 使系统无法访问失活卷组，这样可允许您卸去其物理卷。**vgimport** 命令可在 **vgexport** 命令使卷组失活后让机器可以重新访问该卷组。

要将一个卷组从一个系统移动到另一个系统，请执行以下步骤：

1. 确定没有用户正在访问卷组中激活卷中的文件，然后卸载逻辑卷。
2. 使用 **vgchange** 命令的 **-a n** 参数将卷组标记为失活，这样可防止在该卷组中进行任何进一步的操作。
3. 使用 **vgexport** 命令导出卷组。这样可防止您要将其从中删除的系统访问该卷组。

   在您导出卷组后，在执行 **pvscan** 命令时，物理卷会在导出的物理卷中显示，如下。

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

   当关闭系统时，您可以拔出组成该卷组的磁盘并将它们连接到新的系统中。
4. 当将磁盘插入新的系统，使用 **vgimport** 命令导入卷组，并使新的系统可以访问该卷组。
5. 用 **vgchange** 命令的 **-a y** 参数激活卷组。
6. 挂载文件系统使其可用。

4.3.15. 重新创建卷组目录

要重新创建卷组目录和逻辑卷专用文件，请使用 **vgmknodes** 命令。这个命令会检查位于 **/dev** 目录中用来激活逻辑卷的 LVM2 专用文件。它创建所有丢失的专用文件，并删除不使用的文件。

您可以通过在命令中指定 **--mknodes** 参数来将 **vgmknodes** 命令整合到 **vgscan** 命令中。

4.4. 逻辑卷管理

这部分介绍了逻辑卷管理各个方面的命令。

4.4.1. 创建逻辑卷

要创建逻辑卷，请使用 **lvcreate** 命令。您可以按照以下子部分的论述创建线性卷、条状卷和镜像卷。

如果您没有为逻辑卷指定名称，就会使用默认的名称 **lvol#**，其中用逻辑卷的内部号码替换 **#**。

下面的部分为您提供可以在 LVM 中创建的三种逻辑卷的示例。

4.4.1.1. 创建线性卷

当您创建逻辑卷时，该逻辑卷是从使用物理卷可用扩展的卷组中切割下来的，它们构成了卷组。通常逻辑卷会根据下一个可用原则在最大程度上使用基本物理卷中的空间。修改逻辑卷可释放并重新分配物理卷的空间。

下面的命令在卷组 **vg1** 中创建了大小为 10GB 的逻辑卷。

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

下面的命令在创建了块设备 **/dev/testvg/testlv** 的卷组 **testvg** 中创建了大小为 1500MB，名为 **testlv** 的线性逻辑卷。

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

下面的命令在卷组 **vg0** 中使用可用扩展创建了大小为 50GB，名为 **gfslv** 的逻辑卷。

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

您可以使用 **lvcreate** 命令的 **-l** 参数以扩展为单位指定逻辑卷的大小。您还可以使用这个参数来为逻辑卷指定所用卷组的比例。下面的命令创建了名为 **mylv** 逻辑卷，它使用了卷组 **testvol** 总空间的 60%。

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

您还可以使用 **lvcreate** 命令的 **-l** 参数来用卷组中所剩空间的比例指定逻辑卷的大小。下面的命令创建了名为 **yourlv**，使用卷组 **testvol** 中所有未分配空间的逻辑卷。

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

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

下面的命令创建了名为 **mylv**，可充满卷组 **testvg** 的逻辑卷。

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

The underlying physical volumes used to create a logical volume can be important if the physical volume needs to be removed, so you may need to consider this possibility when you create the logical volume. For information on removing a physical volume from a volume group, see 第 4.3.6 节 “从卷组中删除物理卷”.

要创建一个从卷组的特定物理卷中分配出来的逻辑卷，请在 **lvcreate** 命令行的最后指定物理卷或者卷。下面的命令在卷组 **testvg** 中创建了名为 **testlv** 的逻辑卷，并在物理卷 **/dev/sdg1** 中进行分配：

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

您可以指定物理卷中的哪些扩展可用作逻辑卷。下面的例子创建了一个线性逻辑卷，使用卷组 **testvg** 的物理卷 **/dev/sda1** 的扩展 0-25，和物理卷 **/dev/sdb1** 的扩展 50-125。

```
lvcreate -l 100 -n testlv testvg /dev/sda1:0-25 /dev/sdb1:50-125
```

下面的例子创建了线性逻辑卷，使用的是物理卷 **/dev/sda1** 的扩展 0-25，然后继续在扩展 100 处部署逻辑卷。

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

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

4.4.1.2. 创建条状卷

For large sequential reads and writes, creating a striped logical volume can improve the efficiency of the data I/O. For general information about striped volumes, see 第 2.3.2 节 “条状逻辑卷”.

当您创建条状逻辑卷时，请使用 **lvcreate** 命令的 **-i** 参数指定条带的数目。这取决于逻辑卷要进行条带化的物理卷数目。条带的数目不能超过卷组中物理卷的数目（除非使用 **--alloc anywhere** 参数）。

如果构成逻辑卷的基本物理设备的大小不同，条状卷的最大容量由最小的基本设备决定。例如，在有两个分支条状卷中，其容量最大为较小设备的两倍。在有三个分支的条状卷中，其容量是最小设备的三倍。

下面的命令创建了跨两个物理卷，幅度为 64KB 的条状逻辑卷。逻辑卷大小为 50GB，名称为 **gfslv**，它是从卷组 **vg0** 中分割出来的。

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

在线性卷中，您可以指定用于条状卷的物理卷扩展。下面的命令创建了大小为 100 扩展的条状卷，跨两个物理卷，名为 **stripelv**，属于卷组 **testvg**。该条状卷将使用 **/dev/sda1** 的 0-50 区段以及 **/dev/sdb1** 50-100 区段的。

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

4.4.1.3. 创建镜像卷

当您创建一个镜像卷时，您可使用 **lvcreate** 命令的 **-m** 参数来指定数据的备份数目。指定 **-m1** 生成一个镜像，也就是生成两个文件系统副本：一个线性逻辑卷加上一个副本。同样的，指定 **-m2** 会生成两个镜像，也就是生成三个文件系统副本。

下面的命令使用单一镜像创建一个镜像逻辑卷。大小为 50GB，名为 **mirrorlv**，且是从卷组 **vg0** 中分离出来的：

```
lvcreate -L 50G -m1 -n gfslv vg0
```

LVM 镜像将分割复制到区域（默认大小为 512K）的设备。您可以使用 **-R** 参数以 MB 为单位指定区域大小。LVM 维护一个小的日志，可用来追踪哪些区域是和镜像同步的。默认情况下，该日志是保存在磁盘中的，这样可以使它在机器重启后得以保存。您可以使用 **--corelog** 参数令该日志在内存中保存，这样就需要额外的日志设备，但它将需要在每次重启时重新同步化来获得完整镜像。

下面的命令在卷组 **bigvg** 中创建镜像逻辑卷。逻辑卷名称为 **ondiskmirvol**，它有单一镜像。卷大小为 12MB，并在内存中保存镜像日志。

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

镜像日志是在与生成镜像分支的设备不同的设备中生成的。但有可能使用 **vgcreate** 命令的 **--alloc anywhere** 参数在镜像分支之一的同一设备中创建镜像分支。这可能会降低性能，但可让您在只有两个基础设备的情况下创建镜像。

下面的命令使用单一镜像创建一个镜像逻辑卷，其镜像日志位于作为镜像分支之一的设备中。在这个示例中，卷组 **vg0** 只包括两个设备。这个命令创建的镜像卷大小为 50GB，名为 **mirrorlv**，且是从卷组 **vg0** 中分离出来的。

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

创建了镜像时，就会同步镜像区域。对于大量的镜像组件来说，同步进程可能需要一些时间。当您创建一个不需要恢复的新镜像时，您可以指定 **nosync** 参数说明不需要从第一个设备进行初始同步化。

您可以指定镜像日志使用哪些设备并进行记录。您还可以指定要使用的设备扩展。要强制登录到特定磁盘，请在存放扩展的磁盘中明确指定一个扩展。LVM 不一定要考虑命令行中的设备顺序。如果列出任意物理卷，那么那里就是唯一进行磁盘分配的地方。列表中包含的任何已经被分配的物理扩展将会被忽略。

下面的命令创建了带单一镜像的镜像逻辑卷，卷大小为 500GB，名为 **mirrorlv**，是从卷组 **vg0** 中分割出来的。镜像的一支位于设备 **/dev/sda1**，第二支位于设备 **/dev/sdb1**，镜像日志位于 **/dev/sdc1**。

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

下面的命令创建了带单一镜像的镜像卷。卷的大小为 500MB，名为 **mirrorlv**，是从卷组 **vg0** 中分割出来的。镜像的第一个分支在设备 **/dev/sda1** 的第 0 到 499 扩展，镜像的第二个分支位于设备 **/dev/sdb1** 的第 0 到 499 扩展，且镜像日志从 **/dev/sdc1** 的扩展 0 开始。这些是 1MB 的扩展。如果任何指定的扩展已经被分配了，那么它们就会被忽略。

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

注意

在 RHEL 5.3 发行本中，在群集中支持镜像的逻辑卷。

4.4.1.4. 修改镜像卷配置

您可以使用 **lvconvert** 命令将一个逻辑卷从镜像卷转换成线性卷，或者从线性卷转换成镜像卷。您还可以使用这个命令来重新配置现有逻辑卷的其它镜像参数，比如 **corelog**。

当您将一个逻辑卷转换成镜像卷时，通常您需要为现有卷创建镜像分支。就是说您的卷组必须包含用作镜像分支和镜像日志的设备和空间。

If you lose a leg of a mirror, LVM converts the volume to a linear volume so that you still have access to the volume, without the mirror redundancy. After you replace the leg, you can use the **lvconvert** command to restore the mirror. This procedure is provided in 第 6.3 节 “修复 LVM 镜像错误”.

下面的命令可将线性卷 **vg00/lvol1** 转换成镜像卷。

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

下面的命令可将镜像卷 **vg00/lvol1** 转换成线性卷，并删除镜像分支。

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

4.4.2. 持久的设备号码

在载入模块的时候会自动分配主、副设备号码。如果总是用相同的设备（主和副）号码激活块设备，可使有些应用程序获得最佳性能。您可以通过使用以下参数来指定 **lvcreate** 和 **lvchange** 来达到此目的：

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

Use a large minor number to be sure that it hasn't already been allocated to another device dynamically.

如果您使用 NFS 导出一个文件系统，在导出文件中指定 **fsid** 参数可避免在 LVM 中设定持久的设备号码。

4.4.3. 重新设定逻辑卷大小

要修改逻辑卷的大小，请使用 **lvreduce** 命令。如果逻辑卷包含一个文件系统，请确定首先减小文件系统（或者使用 LVM GUI），以便逻辑卷逻辑卷总是至少可达到文件系统所需的大小。

下面的命令将卷组 **vg00** 中的逻辑卷 **lvol1** 的大小减少了三个逻辑扩展。

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

4.4.4. 修改逻辑卷组的参数

要修改逻辑卷参数，请使用 **lvchange** 命令。有关您可以修改的参数列表，请参考 **lvchange**(8) man page。

You can use the **lvchange** command to activate and deactivate logical volumes. To activate and deactivate all the logical volumes in a volume group at the same time, use the **vgchange** command, as described in 第 4.3.7 节 “修改卷组参数”.

下面的命令将卷组 **vg00** 中卷 **lvol1** 的权限改为只读。

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

4.4.5. 重新命名逻辑卷

要重新命名一个现有逻辑卷，请使用 **lvrename** 命令。

下面两个命令之一都可以将卷组 **vg02** 中的逻辑卷 **lvold** 重新命名为 **lvnew**。

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

```
lvrename vg02 lvold lvnew
```

For more information on activating logical volumes on individual nodes in a cluster, see 第 4.8 节 “在群集的独立节点中激活逻辑卷”.

4.4.6. 删除逻辑卷

要删除一个失活逻辑卷，请使用 **lvremove** 命令。您必须在它可被删除之前使用 **umount** 命令关闭逻辑卷。另外，在群集的环境中，您必须在删除逻辑卷之前将其失活。

如果逻辑卷目前是挂载状态，请在删除它之前将其卸载。

下面的命令可从卷组 **testvg** 中删除逻辑卷 **/dev/testvg/testlv**。注意：在这种情况下，逻辑卷还没有被失活。

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

您可以使用 **lvchange -an** 命令在删除逻辑卷之前明确将其失活，这样就不会出现询问您是否删除某个激活逻辑卷的提示。

4.4.7. 显示逻辑卷

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

The **lvs** command provides logical volume information in a configurable form, displaying one line per

logical volume. The **lvs** command provides a great deal of format control, and is useful for scripting. For information on using the **lvs** command to customize your output, see 第 4.9 节 “为 LVM 自定义报告”.

lvdisplay 命令用混合格式显示物理属性（大小、布局和映射）。

下面的命令显示 **vg00** 中 **lvol2** 的属性。如果已经为原始逻辑卷创建了快照逻辑卷，这个命令还会显示所有快照逻辑卷及其状态（激活或者失活）的列表。

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

lvscan 命令扫描系统中所有逻辑卷并将其列出，如下:

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

4.4.8. 增大逻辑卷

要增大逻辑卷的大小，请使用 **lvextend** 命令。

在扩展逻辑卷后，您将需要增大相关的文件系统以使之与其匹配。

当您扩展逻辑卷时，您可以指定您想要增大的量，或者在您扩展它之后，它应该是多大。

下面的命令将逻辑卷 **/dev/myvg/homevol** 增大到 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
```

下面的命令为逻辑卷 **/dev/myvg/homevol** 添加了另一个 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
```

和使用 **lvcreate** 命令一样，您可以使用 **lvextend** 命令的 **-l** 参数指定扩展的数目，并据此增加逻辑卷的大小。您还可以使用此参数指定卷组的比例或者卷组中剩余空间的比例。下面的命令扩展了名为 **testlv** 的逻辑卷，并使其使用卷组 **myvg** 所有未被分配的空间。

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

在您扩展完逻辑卷后，有必要增大文件系统的大小与之进行匹配。

在默认情况下，大多数重新定义文件系统大小的工具都会将文件系统的大小增加到基本逻辑卷的大小，这样您就不必担心为两个命令指定相同的容量了。

4.4.9. 扩展条状卷

要增加条状逻辑卷的大小，基本物理卷中必须有足够的可用空间，以便让卷组支持它。例如，如果您有一个使用了这个卷组的双向条带，那么在卷组中添加一个物理卷将会使您无法扩展条带，反之，您必须在卷组中添加至少两个物理卷。

例如：考虑卷组 **vg** 包括两个基本物理卷，参见 **vgs** 命令的结果。

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

您可以使用整个卷组空间创建一个条带。

```
# 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)
```

注意：那个卷组中已经没有剩余空间了。

```
# 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了另一个物理卷，那么就有了 135G 的额外空间。

```
# 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
```

此时您不能扩展条状逻辑卷来充满卷组，因为需要两个基本设备来将数据按条状保存。

```
# 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
```

要扩展条状逻辑卷，请添加另一个物理卷，然后扩展该逻辑卷。在这个示例中，在卷组中添加两个物理卷，我们就可以将逻辑卷 5A 扩展成卷组的大小。

```
# 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
```

如果您没有足够的基本物理设备来扩展条状逻辑卷，那么在扩展不是条状的情况下也可能扩大卷，但可能导致性能不平衡。当为逻辑卷添加空间时，默认操作是使用与现有逻辑卷最新片段相同的条状参数，但您可覆盖那些参数。下面的例子是在启动 **lvextend** 命令失败后，使用剩余的可用空间扩大了条状逻辑卷。

```
# 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
```

4.4.10. 缩小逻辑卷

要减小逻辑卷的大小，首先卸载文件系统。您可以接着使用 **lvreduce** 命令缩小卷，缩小卷后，重新挂载文件系统。

警告

在缩小卷自身大小之前，减小文件系统或者卷中任何事物的大小是很重要的，否则您可能会丢失数据。

缩小逻辑卷，释放一些卷组，可将其分配给卷组中的其它逻辑卷。

下面的例子是将卷组 **vg00** 中的逻辑卷 **lvol1** 的减小三个逻辑扩展。

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

4.5. 创建快照卷

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

注意

在群集中不支持跨节点的 LVM 快照。

Since LVM snapshots are not cluster-aware, they require exclusive access to a volume. For information on activating logical volumes on individual nodes in a cluster, see 第 4.8 节 “在群集的独立节点中激活逻辑卷”.

以下命令创建了一个快照卷，大小为 100MB，名为 **/dev/vg00/snap**。它是名为 **/dev/vg00/lvol1** 的源逻辑卷的快照。如果源逻辑卷有一个文件系统，您可以将快照逻辑卷挂载到任意目录下，以便访问该文件系统的内容来在源文件系统不断更新的情况下进行备份。

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

在您创建了快照逻辑卷之后，对源逻辑卷执行 **lvdisplay** 命令产生的输出结果包含了所有扩展逻辑卷及其状态（激活或者失活）的列表。

下面的命令显示逻辑卷 **/dev/new_vg/lvol0** 的状态，并为其生成了快照卷 **/dev/new_vg/newvgsnap**。

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

默认情况下，**lvs** 命令显示快照卷的源逻辑卷以及目前所使用的快照卷的比例。下面的例子显示了在系统中执行 **lvs** 命令的默认输出结果，其中包括逻辑卷 **/dev/new_vg/lvol0** 以及为其生成的快照卷 **/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
```

注意

因为快照在源卷有变化时会增大，所以常规使用 **lvs** 命令监控快照卷的比例以确定它还没有被填满是很重要的。使用了 100% 的快照卷会完全丢失，因为写入源卷中不修改的部分的操作如果不破坏快照是无法成功的。

4.6. 用过滤器控制 LVM 设备扫描

在启动时会运行 **vgscan** 命令来扫描系统中的块设备以查找 LVM 标签来确定哪些是物理卷，并读取元数据建立卷组列表。物理卷的名称被保存在系统每个节点的缓冲文件中，即 **/etc/lvm/.cache**。之后运行的命令可以读取该文件以避免重复扫描。

您可以在 **lvm.conf** 配置文件中通过设置过滤器来控制 LVM 要扫描的设备。过滤器由一组简单正则表达式组成，使用 **/dev** 目录中的设备名称来确定接受还是拒绝找到的块设备。

下面的例子显示用过滤器控制 LVM 要扫描设备的方法。请注意：有些例子不一定是最好的实践方法，因为正则表达式根据完整路径名称自由匹配。例如：**a/.*loop.*/** 和 **a/loop/** 及 **/dev/solooperation/lvol1** 都对映。

下面的过滤器添加所有找到的设备，这是配置文件中没有配置过滤器的默认行为：

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

下面的过滤器会删除光驱以避免在驱动器中没有介质时造成延迟：

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

下面的过滤器添加所有回路设备并删除其它块设备：

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

下面的过滤器添加所有回路设备和 IDE 设备，同时删除所有其它块设备：

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

下面的过滤器只添加第一个 IDE 驱动器中的分区 8，同时删除所有其它块设备：

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

For more information on the **lvm.conf** file, see 附录 B, *LVM 配置文件* and the **lvm.conf**(5) man page.

4.7. 在线数据重定位

您可以使用 **pvmove** 命令在系统还在使用时移动数据。

pvmove 命令将数据分成片段，并生成临时镜像来移动每个片段。有关 **pvmove** 命令操作的详细内容请参考 **pvmove**(8) man page。

Because the **pvmove** command uses mirroring, it is not cluster-aware and needs exclusive access to a volume. For information on activating logical volumes on individual nodes in a cluster, see 第 4.8 节 “在群集的独立节点中激活逻辑卷”.

下面的命令将物理卷 **/dev/sdc1** 中所有分配了的空间都移动到卷组中其它可用物理卷中：

```
pvmove /dev/sdc1
```

下面的命令只移动逻辑卷 **MyLV** 中的扩展：

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

因为执行 **pvmove** 命令要花很长时间，您可能想要在后台运行此命令以避免在前台显示更新的过程。下面的命令在后台将所有分配给物理卷 **/dev/sdc1** 的扩展移动到 **/dev/sdf1**。

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

下面的命令以 5 秒为间隔，以百分比形式报告移动的过程。

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

4.8. 在群集的独立节点中激活逻辑卷

如果您在群集环境中安装了 **lvm**，您可能需要多次在一个节点完全激活逻辑卷。例如： **pvmove** 命令，且需要对卷的完全访问。LVM 快照也需要对卷的完全访问。

要在一个节点上完全激活逻辑卷，请使用 **lvchange -aey** 命令。另外，您可以使用 **lvchange -aly** 来只激活本地节点中的逻辑卷，而不是所有逻辑卷。您可以晚些时候在附加节点上同时激活它们。

You can also activate logical volumes on individual nodes by using LVM tags, which are described in 附录 C, *LVM 对象标签*. You can also specify activation of nodes in the configuration file, which is described

in 附录 B, *LVM 配置文件*.

4.9. 为 LVM 自定义报告

您可以使用 **pvs**、**lvs** 和 **vgs** 命令得到一份 LVM 对象的简洁自定义报告。这些命令生成的报告包括每行一个对象的输出结果。每行包含有关对象属性字段排序列表。选择要报告的对象有五种方法：根据物理卷、卷组、逻辑卷、物理卷片段和逻辑卷片段。

以下部分提供了：

- 您可以用来扩展生成报告格式的参数概述。
- 您可以为每个 LVM 对象选择的字段列表。
- 您可以用来对生成的报告进行排序的命令参数总结。
- 指定报告输出结果单位的步骤。

4.9.1. 格式控制

无论您使用 **pvs**、**lvs** 或者 **vgs** 命令，都要确定默认字段显示和排列顺序。您可以使用以下参数来控制这些命令的输出结果：

- 您可以使用 **-o** 参数将字段显示的内容改成任意内容，而不只是默认的内容。例如：**pvs** 命令默认显示如下（显示有关物理卷的信息）：

```
# 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
```

  您可以用下面的命令只显示物理卷的名称和大小。

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

- 您可以用 (+) 符号在输出结果中附加一个字段，它通常与 -o 参数合用。
  下面的例子除默认字段外还显示物理卷 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** 参数使其包括一些额外的字段。例如：**pvs -v** 命令将在默认字段之外显示 **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-XO8M-mcpsVe
  /dev/sdd1  new_vg lvm2 a-   17.14G 17.14G  17.14G yvfvZK-Cf31-j75k-dECm-
0RZ3-0dGW-tUqkCS
```

- **--noheadings** 参数制止标题行。这在写脚本时很有用。

  下面的命令合并使用 **--noheadings** 和 **pv_name** 参数，将生成所有物理卷的列表。

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

- **--separator** ***分隔符*** 参数使用 ***分隔符*** 来分隔每个字段。这在对输出结果运行 **grep** 命令的脚本中很有用。

  下面的例子使用等号（=）分隔 **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** 参数时让字段对齐，请联合使用 **separator** 和 **--aligned** 参数。

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

You can use the **-P** argument of the **lvs** or **vgs** command to display information about a failed volume that would otherwise not appear in the output. For information on the output this argument yields, see 第 6.2 节 “在失败的设备中显示信息。”.

有关显示参数的完整列表请参考 **pvs**(8)、**vgs**(8) 和 **lvs**(8) man page。

卷组字段可以与物理卷（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：下面的命令将在输出结果中每行显示一个物理卷。

```
# 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
```

4.9.2. 对象选择

这部分提供了一组列表，表中列出的有关 LVM 对象的信息您可以用 **pvs**、**vgs** 和 **lvs** 命令获得。

为方便起见，字段名称前缀如果与命令默认字段匹配就可以去掉。例如：在 **pvs** 命令中，**name** 的指的是 **pv_name**，但使用 **vgs** 命令时，**name** 被理解为 **vg_name**。

执行以下命令和执行 **pvs -o pv_free** 的结果相同。

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

pvs 命令

表 4.1 “pvs 显示字段” lists the display arguments of the **pvs** command, along with the field name as it appears in the header display and a description of the field.

表 4.1. pvs 显示字段

参数	标题	描述
dev_size	DevSize	创建物理卷的基本设备的大小
pe_start	1st PE	在基本设备中调整到第一个物理扩展的起始位置
pv_attr	Attr	物理卷状态：可分配（a）或者导出的（x）。
pv_fmt	Fmt	物理卷的元数据格式（**lvm2** 或者 **lvm1**）
pv_free	PFree	物理卷中剩余的可用空间
pv_name	PV	物理卷名称
pv_pe_alloc_count	Alloc	已经使用的物理扩展数目
pv_pe_count	PE	物理扩展数量
pvseg_size	SSize	物理卷的片段大小
pvseg_start	Start	物理卷片段的起始物理扩展
pv_size	PSize	物理卷的大小
pv_tags	PV Tags	附加到物理卷的 LVM 标签
pv_used	Used	目前物理卷中已经使用的空间量
pv_uuid	PV UUID	物理卷的 UUID

默认情况下 **pvs** 命令显示以下字段：**pv_name**、**vg_name**、**pv_fmt**、**pv_attr**、**pv_size**、**pv_free**。结果根据 **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
```

使用带 **-v** 参数的 **pvs** 命令向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-
XO8M-mcpsVe
  /dev/sdd1  new_vg lvm2 a-   17.14G 17.13G  17.14G yvfvZK-Cf31-j75k-dECm-0RZ3-
0dGW-tUqkCS
```

您可以使用 **pvs** 命令的 **--segments** 参数显示每个物理卷片段的信息。一个片段就是一组扩展。查看片段在想要查看逻辑卷是否有很多碎片时很有用。

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**。结果按照物理卷中 **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
```

您可以使用 **pvs -a** 查看被 LVM 侦测出来但还没有初始化为 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 4.2 “vgs 显示字段” lists the display arguments of the **vgs** command, along with the field name as it appears in the header display and a description of the field.

表 4.2. vgs 显示字段

参数	标题	描述
lv_count	#LV	卷组中含有的逻辑卷数目
max_lv	MaxLV	卷组中最多可用逻辑卷数量（如果没有限制就是 0）
max_pv	MaxPV	卷组中最多允许的物理卷数目（如果没有限制就是 0）
pv_count	#PV	定义卷组的物理卷数目
snap_count	#SN	卷组包含的快照数目
vg_attr	Attr	卷组状态：可写入（w）、只读（r）、可重新定义大小（z）、导出的（x）、不完整的（p）和群集的（c）。
vg_extent_count	#Ext	卷组中的物理扩展数目
vg_extent_size	Ext	卷组中物理扩展的大小
vg_fmt	Fmt	卷组（**lvm2** 或者 **lvm1**）的元数据格式
vg_free	VFree	卷组中剩余可用空间
vg_free_count	Free	卷组中可用物理扩展数目
vg_name	VG	卷组名称
vg_seqno	Seq	代表修正卷组的数字
vg_size	VSize	卷组大小
vg_sysid	SYS ID	LVM1 系统 ID
vg_tags	VG Tags	附加到卷组中的 LVM 标签
vg_uuid	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**，并根据 **vg_name** 排序。

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

使用带 **-v** 参数的 **vgs** 命令向默认显示中添加以下字段：**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-OpMO-0118-nlwO-wwqd-
fD5D32
```

lvs 命令

表 4.3 “lvs 显示字段” lists the display arguments of the **lvs** command, along with the field name as it

appears in the header display and a description of the field.

表 4.3. lvs 显示字段

参数	标题	描述
chunksize **chunk_size**	Chunk	快照卷的单位大小
copy_percent	Copy%	镜像卷的同步化比例，还可在使用 **pv_move** 命令移动物理扩展时使用。
devices	Devices	组成逻辑卷的基本设备：物理卷、逻辑卷以及起始物理和逻辑扩展
lv_attr	Attr	逻辑卷状态。逻辑卷属性字节如下： 字节 1：卷类型：镜像（m）、不带初始同步的镜像（M ）、源（o）、pvmove（p）、快照（s）、不可用快照（S）、虚拟（v） 字节 2：权限：可写入（w），只读（r） 字节 3：分配策略：持续（c）、正常（n）、任意（a）、继承（i）。如果在修改分配时锁定了卷，就会显示成大写，比如在执行 **pvmove** 命令的时候。 字节 4：固定的副号码（m） 字节 5：激活（a）、暂停的（s）、不可用快照（I）、不可用暂停快照（S）、不带表格的映射设备（d）、带未激活表格的映射设备（i） 字节 6：设备开放（o）
lv_kernel_major	KMaj	逻辑卷的真实主设备号码（如果是未激活就减 1）
lv_kernel_minor	KMIN	逻辑卷的真实副设备号码（如果是未激活就减 1）
lv_major	Maj	逻辑卷持久的主设备号码（如果未指定就减 1）
lv_minor	Min	逻辑卷持久的副设备号码（如果未指定就减 1）
lv_name	LV	逻辑卷名称
lv_size	LSize	逻辑卷的大小
lv_tags	LV Tags	附加到逻辑卷的 LV 标签
lv_uuid	LV UUID	逻辑卷的 UUID
mirror_log	Log	镜像分支所在设备
modules	Modules	使用此逻辑卷符合内核设备映射器目标需要
move_pv	Move	用 **pvmove** 命令创建的临时逻辑卷的源物理卷
origin	Origin	快照卷的源设备
regionsize **region_size**	Region	镜像逻辑卷的单元大小
seg_count	#Seg	逻辑卷中片段的数目
seg_size	SSize	逻辑卷中片段的大小

seg_start	Start	修正逻辑卷中的片段
seg_tags	Seg Tags	附加到逻辑卷片段的 LVM 标签
segtype	Type	逻辑卷的片段类型（例如：镜像、条状、线性）
snap_percent	Snap%	已经使用的快照卷的比例
stripes	#Str	逻辑卷中条带或者镜像的数目
stripesize **stripe_size**	Stripe	条状逻辑卷中条带的单位大小

lvs 命令默认显示以下字段：**lv_name**、**vg_name**、**lv_attr**、**lv_size**、**origin**、**snap_percent**、**move_pv**、**mirror_log**、**copy_percent**。默认显示根据卷组中 **vg_name** 和 **lv_name** 排序。

```
# 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
```

使用带 **-v** 参数的 **lvs** 命令在默认显示结果中添加以下字段：**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 LV UUID
  lvol0      new_vg    1 owi-a- 52.00M  -1  -1 253  3
LBy1Tz-sr23-OjsI-LT03-nHLC-y8XW-EhCl78
  newvgsnap1 new_vg    1 swi-a-  8.00M  -1  -1 253  5    lvol0    0.20
1ye1OU-1cIu-o79k-20h2-ZGF0-qCJm-CfbsIx
```

您可以使用 **lvs** 命令的 **--segments** 参数显示强调片段信息的默认列。当您使用 **segments** 参数时，**seg** 前缀是可选的。**lvs --segments** 命令默认显示以下字段：**lv_name**、**vg_name**、**lv_attr**、**stripes**、**segtype**、**seg_size**。默认显示根据卷组的 **vg_name** 和 **lv_name** 排序。如果逻辑卷中有碎片，那么会在此命令的输出结果中显示出来。

```
# 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
```

使用带 **-v** 参数的 **lvs --segments** 命令向默认显示中添加以下字段：**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
```

下面的例子显示在配置了一个逻辑卷的系统的 **lvs** 命令的默认输出结果以及指定了 **segments** 参数的 **lvs** 命令输出结果。

```
# 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
```

4.9.3. LVM 报告排序

通常，**lvs**、**vgs** 或者 **pvs** 命令的完整输出结果在正确排序和对齐之前必须在内部生成并保存。您可以指定 **--unbuffered** 参数来尽快显示未排序的输出。

要指定另外的列表顺序进行排序，请使用任意报告命令的 **-O** 参数。在输出中不一定要包含这些输出字段。

下面的例子显示 **pvs** 命令的输出结果，包括物理卷名称、大小和可用空间。

```
# 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
```

下面的例子显示相同的输出结果，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
```

下面的例子表明您不需要根据您要排序的字段显示。

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

要显示逆向排序，请在 **-O** 参数后您指定的字段前添加 **-** 符号。

```
# 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
```

4.9.4. 指定单位

要指定 LVM 报告显示的单位，请使用报告命令的 **--units** 参数。您可以指定字节（b）、千字节（k）、兆字节（m）、千兆字节（g）、兆兆字节（t）、艾字节（e）、拍字节（p）以及可读。默认显示是可读。您可以通过在 **lvm.conf** 文件的 **global** 部分设定 **units** 参数来覆盖默认设置。

下面的例子指定 **pvs** 命令的输出结果以兆为单位，而不是默认的千兆为单位。

```
# 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
```

默认情况下，单位显示为 2 的乘方（乘 1024）。您通过大写单位说明（B、K、M、G、T、H）指定将单位显示为乘 1000。

下面的命令以默认行为，即乘 1024 显示输出结果。

```
# 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
```

下面的命令以乘 1000 显示输出结果。

```
#  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
```

您还可以指定扇区（sector，定义为 512K），或者自定义单位。

下面的命令以扇区的数目显示 **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
```

下面的例子显示了 **pvs** 命令的输出结果，单位为 4MB。

```
# 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

本示例为创建一个名为 **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，您需要将它们标记为 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。

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

5.1.2. 创建卷组

下面的命令可创建卷组 **new_vol_group**。

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

您可以使用 **vgs** 命令来显示新卷组的属性。

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

5.1.3. 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。

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

5.1.4. 创建文件系统

以下命令在逻辑卷中创建了一个 GFS 文件系统。

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

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

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

Syncing...
All Done
```

下面的命令将逻辑卷挂载到文件系统并报告磁盘空间用量。

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

5.2. 创建条状逻辑卷

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

5.2.1. 创建物理卷

将卷组中您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。

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

5.2.2. 创建卷组

下面的命令可创建卷组 **striped_vol_group**。

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

您可以使用 **vgs** 命令来显示新卷组的属性。

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

5.2.3. 创建逻辑卷

下面的命令可在卷组 **striped_vol_group** 中创建条状逻辑卷 **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。

```
[root@tng3-1 ~]# lvcreate -i3 -I4 -L2G -nstriped_logical_volume
striped_vol_group
  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

以下命令在逻辑卷中创建了一个 GFS 文件系统。

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

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

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

Syncing...
All Done
```

下面的命令将逻辑卷挂载到文件系统并报告磁盘空间用量。

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

您可以使用 **pvscan** 命令来确定在卷组中目前有多少可用的剩余空间。

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

5.3.2. 转移数据

您可以使用 **pvmove** 将 **/dev/sdc1** 中所有使用的物理扩展移动到 **/dev/sdb1** 中。执行 **pvmove** 会花一些时候。

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

转移完数据后，您可以看到 **/dev/sdc1** 中的所有空间都可用了。

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

5.3.3. 分割卷组

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

在您可以分割卷组前，必须使逻辑卷失活。如果挂载了文件系统，您必须在失活逻辑卷之前卸载文件系统。

您可以使用 **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。

```
[root@tng3-1 ~]# lvchange -a n /dev/myvg/mylv
[root@tng3-1 ~]# vgsplit myvg yourvg /dev/sdc1
  Volume group "yourvg" successfully split from "myvg"
```

您可以使用 **vgs** 查看两个卷组的属性。

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

5.3.4. 创建新逻辑卷

创建新的卷组后，您可以创建新的逻辑卷 **yourlv**。

```
[root@tng3-1 ~]# lvcreate -L5G -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

您可以在新的逻辑卷中生成一个文件系统并挂载它。

```
[root@tng3-1 ~]#  gfs_mkfs -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

因为您必须使逻辑卷 **mylv** 失活，所以您需要在挂载它之前再次激活它。

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

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

5.4. 从逻辑卷中删除磁盘

本示例告诉您如何从现有逻辑卷中删除磁盘，您可以替换磁盘，也可以用这个磁盘作为不同卷的一部分。要删除磁盘，您必须首先将 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。

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

我们想要移动 **/dev/sdb1** 的扩展，以便可以将其从卷组中删除。

如果在卷组的其它物理卷中没有足够的剩余扩展，您可以在您想要删除的设备中执行不带选项的 **pvmove** 命令，那么扩展就会被分配到其它设备中。

```
[root@tng3-1 ~]# 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：

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

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

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

现在可以物理删除这个磁盘或者将其分配给其它用户。

5.4.2. 将扩展移动到新磁盘中

在本示例中，逻辑卷在卷组 **myvg** 中按以下方法分配：

```
[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
```

我们想要将 **/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。

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

5.4.2.2. 将新物理卷添加到卷组中

将 **/dev/sdd1** 添加到现有卷组 **myvg** 中。

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

5.4.2.3. 转移数据

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

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

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

5.4.2.4. 删除卷组中的旧物理卷

您将数据从 **/dev/sdb1** 中移走后，您就可以将它从卷组中删除了。

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

现在您可以将这个磁盘重新分配给其它卷组，或者将其从系统中删除。

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，您可以用以下方法收集诊断信息：

- 使用命令的 **-v**、**-vv**、**-vvv** 或者 **-vvvv** 选现提高输出信息的详细程度。
- If the problem is related to the logical volume activation, set 'activation = 1' in the 'log' section of the configuration file and run the command with the **-vvvv** argument. After you have finished examining this output be sure to reset this parameter to 0, to avoid possible problems with the machine locking during low memory situations.
- 运行 **lvmdump** 命令可为诊断提供信息转储。有关详情请参考 **lvmdump**(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。
- 通过运行 **lvm dumpconfig** 命令检查现有配置信息。
- 检查 **/etc/lvm** 中的 **.cache** 文件来了解哪些设备中有物理卷。

6.2. 在失败的设备中显示信息。

您可以使用 **lvs** 或者 **vgs** 命令的 **-P** 选项来显示那些没有出现在输出结果中的失败卷的信息。该选择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。

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

如果您为 **vgs** 指定了 **-P** 选项，那么该卷组虽仍然不可用，但您可能看到更多有关失败设备的信息。

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

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

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

使用 **-P** 选项显示失败的逻辑卷。

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

下面的例子显示在镜像逻辑卷的一支出错时，带 **-P** 选项的 **pvs** 和 **lvs** 命令的输出结果。

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

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

6.3. 修复 LVM 镜像错误

这部分提供了一个修复示例，即 LVM 镜像的一个分支失败，是因为物理卷的基本设备死机。当一个镜像分支失败时，LVM 将镜像卷转换成线性卷，并在没有镜像冗余的之前继续进行操作。在那个时候，您可以在系统中添加一个新的磁盘来替换物理设备，并重建镜像。

以下命令创建将用于镜像的物理卷。

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

以下命令创建卷组 **vg** 和镜像卷 **groupfs**。

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

您可以使用 **lvs** 命令确定镜像卷、用于镜像分支的基本设备以及镜像分支的布局。请注意：在第一个示例中，镜像还没有被完全同步，您应该在 **Copy%** 字段显示 100.00 之后才继续操作。

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

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

在这个示例中，镜像 **/dev/sda1** 的主要分支失败。任何对镜像卷的写入操作都会导致 LVM 去检测失败的镜像。这个时候，LVM 会将镜像转换成单一线性卷。在这里，引起转发的因素是我们执行了 **dd** 命令。

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

您可以使用 **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。

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

在这里，您应该仍然可以使用逻辑卷，但没有镜像冗余。

To rebuild the mirrored volume, you replace the broken drive and recreate the physical volume. If you use the same disk rather than replacing it with a new one, you will see "inconsistent" warnings when you run the **pvcreate** command.

```
[root@link-08 ~]# pvcreate /dev/sda[12]
  Physical volume "/dev/sda1" successfully created
  Physical volume "/dev/sda2" 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/sda1           lvm2 [603.94 GB]
  PV /dev/sda2           lvm2 [603.94 GB]
  Total: 16 [2.11 TB] / in use: 14 [949.65 GB] / in no VG: 2 [1.18 TB]
```

下面您可以使用新的物理卷来扩展原来的卷组。

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

[root@link-08 ~]# pvscan
  PV /dev/sdb1   VG vg   lvm2 [67.83 GB / 67.10 GB free]
  PV /dev/sdb2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdc1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdc2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdd1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdd2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sde1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sde2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdf1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdf2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdg1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdg2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdh1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdh2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sda1   VG vg   lvm2 [603.93 GB / 603.93 GB free]
  PV /dev/sda2   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   ]
```

将线性卷转换回它原来的镜像状态。

```
[root@link-08 ~]# lvconvert -m 1 /dev/vg/groupfs /dev/sda1 /dev/sdb1 /dev/sdc1
  Logical volume mirror converted.
```

您可以使用 **lvs** 命令确定恢复到镜像状态。

```
[root@link-08 ~]# lvs -a -o +devices
  LV                VG   Attr   LSize   Origin Snap%  Move Log          Copy%
Devices
  groupfs           vg   mwi-a- 752.00M                    groupfs_mlog 68.62
groupfs_mimage_0(0),groupfs_mimage_1(0)
  [groupfs_mimage_0] vg   iwi-ao 752.00M
/dev/sdb1(0)
  [groupfs_mimage_1] vg   iwi-ao 752.00M
/dev/sda1(0)
  [groupfs_mlog]     vg   lwi-ao   4.00M
/dev/sdc1(0)
```

6.4. 修复物理卷元数据

如果不小心覆盖或者破坏了卷组物理卷元数据区域，您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。您可能需要通过在物理卷的元数据区域写入新的元数据来修复物理卷数据，指定相同的 UUID 作为丢失的元数据。

警告

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

下面的例子显示排序的输出解个，您可以看到您的元数据是丢了还是被破坏了。

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

通过查看 **/etc/lvm/archive** 目录，您可能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，您可以找到失活的卷并设定 **partial** （**-P**）选项，这样您就可以找到丢失的被破坏的物理卷的 UUID。

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

使用 **pvcreate** 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，即卷组 最新的归档元数据恢复了物理卷标签。**restorefile** 参数让 **pvcreate** 生成一个与卷组中久的物理卷兼容的新物理卷，确保新的元数据不会被放在久的物理卷所做的数据区域（这有可能发生。例如：如果原来的 **pvcreate** 命令使用了控制元数据放置位置的参数，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）。**pvcreate** 命令仅覆盖 LVM 元数据区域，不会影响现有的数据区域。

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

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

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

现在您可以显示逻辑卷。

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

下面的命令激活卷并显示激活的卷。

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

如果磁盘中的 LVM 元数据使用至少覆盖了它的数据的空间大小，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。您可能可以使用 **fsck** 命令修复那些数据。

6.5. 替换丢失的物理卷

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

6.6. 从卷组中删除丢失的物理卷。

如果您丢失了物理卷，您可以用 **vgchange** 命令的 **--partial** 选项激活卷组中剩下的物理卷。您可以使用 **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。

建议您运行 **vgreduce** 命令，使用 **--test** 选项来确定您要破坏的数据。

和大多数 LVM 操作一样，**vgreduce** 命令在某种意义上是可逆的，即您立即使用 **vgcfgrestore** 命令将卷组的元数据恢复到之前的状态。例如：如果您使用 **vgreduce** 命令的 **--removemissing** 参数，而不带 **--test** 参数，您会找到您要保留的已删除的逻辑卷，您仍可用替换物理卷，并使用另一个 **vgcfgrestore** 命令来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

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

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，您可以使用 **vgs** 的 **vg_free_count** 和 **vg_extent_count** 选项显示可用扩展和扩展的总数。

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

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

```
# lvcreate -l8780 -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
```

Alternately, you can extend the logical volume to use a percentage of the remaining free space in the volume group by using the **-l** argument of the **lvcreate** command. For information, see 第 4.4.1.1 节 “创建线性卷”.

第 7 章 用 LVM GUI 进行 LVM 管理

除了命令行界面（CLI），LVM 还提供用来配置 LVM 逻辑卷的图形用户界面（GUI）。您可以通过输入 **system-config-lvm** 启用此工具。*红帽企业版 Linux 部署指南*中关于配置的 LVM 的章节提供了使用此工具进行配置的具体步骤。

另外，LVM GUI 还可作为 Conga 管理界面的一部分。有关使用带 Conga 的 LVM GUI 的信息请参考 Conga 的在线帮助。

设备映射器（Device Mapper）

设备映射器是一个为卷管理提供通用构架的内核驱动程序。它提供可用来创建用作逻辑卷设备的映射设备的通用方法。它不一定要特别了解卷组或者元数据格式。

设备映射器为一组高级技术提供了基础。除 LVM 之外，设备映射器多路径和 **dmraid** 命令也使用设备映射器。设备映射器的应用程序界面是**ioctl** 系统调用。用户界面是 **dmsetup** 命令。

LVM logical volumes are activated using the Device Mapper. Each logical volume is translated into a mapped device. Each segment translates into a line in the mapping table that describes the device. The Device Mapper supports a variety of mapping targets, including linear mapping, striped mapping, and error mapping. So, for example, two disks may be concatenated into one logical volume with a pair of linear mappings, one for each disk. When LVM2 creates a volume, it creates an underlying device-mapper device that can be queried with the **dmsetup** command. For information about the format of devices in a mapping table, see 第 A.1 节 “设备列表映射”. For information about using the **dmsetup** command to query a device, see 第 A.2 节 “dmsetup 命令”.

A.1. 设备列表映射

映射的设备是由一个列表定义的，该列表指定如何使用支持的设备列表映射将设备的每个逻辑分段行进行匹配。映射设备的列表由以下格式行组成：

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

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

设备映射器中的大小总是以扇区（512 字节）为单位指定。

当将某个设备指定为设备映射器中的映射参数，它就被该文件系统（比如 **/dev/hda**）中的设备名称或者主号码和副号码以 ***major*:*minor*** 的格式进行参考。首选 major:minor 格式因为这样可避免查找路径名称。

以下显示了某设备的映像列表示例。在这个列表中有四个线性对象：

```
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对象，在此示例中全部是 **linear**。该行的其余部分包括用于线**性**对象的参数。

以下部分描述了以下映射的格式：

- 线性
- 条状
- 镜像
- 快照以及 snapshot-origin
- 错误
- 零
- 多路径
- 加密

A.1.1. 线性映射对象

线性映射对象将块的连续行映射到另一个块设备中。线性对象的格式如下：

```
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误差

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，起始误差为 41146992 的线性对象。

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

以下示例是含有在设备 **/dev/hda** 中指定的设备参数的线性对象。

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

A.1.2. 条状映射对象

条状映射对象支持所有跨物理设备的条块。它使用条块数目和成条的组集大小以及设备名称和扇区对作为参数。条状对象的格式如下：

```
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误差

以下示例显示了一个有三个条，且组集大小为 128 的条状对象：

```
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

8:9
第一个设备的 major:minor 号码

384
第一个设备中映射的起始误差

8:8
第二个设备的 major:minor 号码

384
第二个设备中映射的起始误差

8:7
第三个设备的 major:minor 号码

9789824
第三个设备中映射的起始误差

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对象。

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

A.1.3. 镜像映射对象

镜像映射对象支持镜像的逻辑设备。镜像对象格式如下：

```
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。这个日志类型有 2-4 个参数：

logdevice regionsize [[**no**]**sync**] [**block_on_error**]

clustered_core

镜像是群集的，镜像日志保存在核内存中。这个日志类型有 2-4 个参数：

regionsize UUID [[**no**]**sync**] [**block_on_error**]

clustered_disk

镜像是群集的，镜像日志保存在磁盘中。这个日志类型有 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。

在群集环境中，***UUID*** 参数是与镜像日志设备关联的特定识别符，以便可通过该群集维护日志状态。

The optional **[no]sync** argument can be used to specify the mirror as "in-sync" or "out-of-sync". The **block_on_error** argument is used to tell the mirror to respond to errors rather than ignoring them.

#log_args

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误差。

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

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

以下示例显示了某个镜像日志保存在磁盘中的群集镜像的镜像映射对象。

```
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

日志类型指定其为群集镜像且镜像日志保存在磁盘中的镜像对象

4

附带 4 个镜像日志参数

253:2

日志设备的 major:minor 号码

1024

镜像日志用来跟踪哪些进行同步的区域大小

UUID

镜像日志的 UUID，用来通过群集维护日志信息

block_on_error

镜像应该响应错误

3

镜像中的分支

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

构成镜像的每个分支的设备的 major:minor 号码和误差

A.1.4. 快照以及 snapshot-origin 映射对象

当您生成某个卷的第一个 LVM 快照时，要使用四个设备映射器设备：

1. 包含源卷原始映射列表线**性**映射的设备。
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. 带**快照**映射合并 #1 和 #2 的设备，它是可见快照卷
4. The "original" volume (which uses the device number used by the original source volume), whose table is replaced by a "snapshot-origin" mapping from device #1.

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
```

这产生四个设备，您可以使用以下命令浏览：

```
# 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 对象的格式如下：

```
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 一般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 设备中以便保持其不更改的可见内容（直到 COW 设备写满为止）。

快照对象的格式如下：

```
start length snapshot origin COW-device P|N chunksize
```

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
保存更改组集的设备

P|N
P（持久）或者N（不持久）；指示快照是否可在重启后保留。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 设备中的有数据更改的组集的扇区的大小

以下示例显示了起始设备为 254:11 的 **snapshot-origin** 对象。

```
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** 对象。这个快照设备在重启后仍然保留，且保存在 COW 设备中的数据组集大小为 16 个扇区。

```
0 2097152 snapshot 254:11 254:12 P 16
```

A.1.5. 错误映射对象

如果有错误映射对象，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，您可以创建一个设备映射，且在该设备中部有一个坏扇区，或者您可以换出一个镜像分支并用错误对象替换之。

错误对象可用于出错的设备，是一种避免超时或者在实际设备中重试的方法。您在失败时重新部署 LVM 元数据时可将其作为中间对象使用。

错误映射对象除 ***start*** 和 ***length*** 参数外不使用其它参数。

以下示例显示的是错误对象。

```
0 65536 error
```

A.1.6. 零映射对象

零映射对象是与 **/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**映射对象除 ***start*** 和 ***length*** 参数外没有其它参数。

以下示例显示了一个 16Tb 设备的**零**对象。

```
0 65536 zero
```

A.1.7. 多路径映射对象

多路径映射对象支持多路径的设备的映射。**多路径**对象的格式如下：

```
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

在那些特性之后是多路径特性的数目。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***。目前有一个支持的多路径特性 **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。

例如：如果只在尝试使用所有路径后将其标记为失败时从才将 **multipath.conf** 文件的 **no_path_retry** 选项设定为只有 I/O 操作的队列，该映射应显示如下除非所有路径检查程序进行的指定检查次数都是失败的。

```
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器参数的数目。硬件处理器指定在切换路径组或者处理 I/O 错误时用来执行

硬件特定的动作。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。一个路径组是一组多路径的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。

pathgroupsargs

每个路径组包括以下参数：

```
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 “多路径映射对象” shows the format of a multipath target with two path groups.

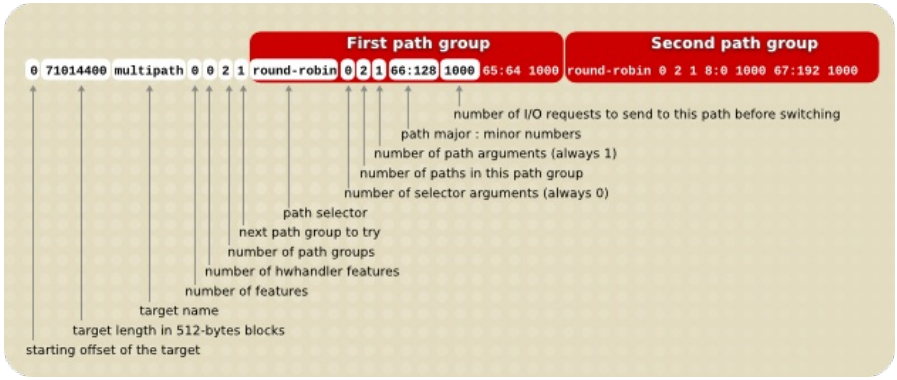

图 **A.1. 多路径映射对象**

以下示例显示对同一个多路径设备的一个纯故障排除对象定义。在这个对象中有四个路径组，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
```

以下示例显示为同一个多路径设备完全展开（多总线）对象定义。在这个对象中只有一个路径组，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. 加密映射对象

加密对象会加密通过指定设备的所有数据。它使用内核 Crypto API。

加密对象的格式如下：

```
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

总是使用 **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，在 hex 中提供

IV-offset

初始向量（IV）误差

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误差

以下是**加密**对象示例。

```
0 2097152 crypt aes-plain 0123456789abcdef0123456789abcdef 0 /dev/hda 0
```

A.2. dmsetup 命令

dmsetup 命令是一个用来与设备映射器沟通的命令行封装器（wrapper）。对于 LVM 设备的一般系统信息，您可发现 **dmsetup** 命令的 **info**、**ls**、**status** 和 **deps** 选项是有用的，如以下部分所述。

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* 命令提供有关设备映射器设备概述。如果您没有指定设备名称，则输出所余目前配置的设备映射器设备信息。如果您指定了一个设备，那么这个命令只会生成那个设备的信息。

dmsetup info 命令提供以下分类中的信息：

Name

该设备的名称。使用以连字符分开的卷组名称和逻辑卷名称表示 LVM 设备。会将原始名称中的连字符转换成两个连字符。

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，所有到那个设备的 I/O 操作都会停止。**dmsetup resume** 命令将设备状态恢复到 **ACTIVE**。

Read Ahead

系统对任意正在进行读取操作的打开的文件提前读取的数据块的数目。模热情况下，内核会自动选择一个合适的值。您可使用 **dmsetup** 命令的 **--readahead** 选项更改这个值。

Tables present

Possible states for this category are **LIVE** and **INACTIVE**. An **INACTIVE** state indicates that a table has been loaded which will be swapped in when a **dmsetup resume** command restores a device state to **ACTIVE**, at which point the table's state becomes **LIVE**. For information, see the **dmsetup** man page.

Open count

打开参考计数指示该打开该设备的次数。**mount** 命令会打开一个设备。

Event number

The current number of events received. Issuing a **dmsetup wait *n*** command allows the user to wait for the n'th event, blocking the call until it is received.

Major, minor

主设备号码和副设备号码

Number of targets

组成一个设备的片段数目。例如：一个跨越三个磁盘的线性设备会有三个对象。由一个磁盘起始和结尾，而不是中间组成的线性设备有两个设备。

UUID

该设备的 UUID。

以下是 **dmsetup info** 命令的部分输出示例。

```
[root@ask-07 ~]# dmsetup info
Name:              testgfsvg-testgfslv1
State:             ACTIVE
Read Ahead:        256
Tables present:    LIVE
Open count:        0
Event number:      0
Major, minor:      253, 2
Number of targets: 2
UUID: LVM-K528WUGQgPadNXYcFrrf9LnPlUMswgkCkpgPIgYzSvigM7SfeWCypddNSWtNzc2N
...
Name:              VolGroup00-LogVol00
State:             ACTIVE
Read Ahead:        256
Tables present:    LIVE
Open count:        1
Event number:      0
Major, minor:      253, 0
Number of targets: 1
UUID: LVM-tOcS1kqFV9drb0X1Vr8sxeYP0tqcrpdegyqj5lZxe45JMGlmvtqLmbLpBcenh2L3
```

A.2.2. dmsetup ls 命令

您可以使用 **dmsetup ls** 命令列出映射的设备的设备名称列表。您可以使用 **dmsetup ls --target *target_type*** 命令列出至少有一个指定类型的对象的设备。**dmsetup ls** 的其它选项请参考 **dmsetup ls** man page。

以下示例显示列出目前配置的映射设备的设备名称的命令。

```
[root@ask-07 ~]# dmsetup ls
testgfsvg-testgfslv3    (253, 4)
testgfsvg-testgfslv2    (253, 3)
testgfsvg-testgfslv1    (253, 2)
VolGroup00-LogVol01     (253, 1)
VolGroup00-LogVol00     (253, 0)
```

以下示例显示列出目前配置的镜像映射的设备名称的命令。

```
[root@grant-01 ~]# dmsetup ls --target mirror
lock_stress-grant--02.1722      (253, 34)
lock_stress-grant--01.1720      (253, 18)
lock_stress-grant--03.1718      (253, 52)
lock_stress-grant--02.1716      (253, 40)
lock_stress-grant--03.1713      (253, 47)
lock_stress-grant--02.1709      (253, 23)
lock_stress-grant--01.1707      (253, 8)
lock_stress-grant--01.1724      (253, 14)
lock_stress-grant--03.1711      (253, 27)
```

A.2.3. dmsetup status 命令

dmsetup status *device* 命令提供指定设备中每个对象的状态信息。如果您没有指定设备名称，输出会是所余目前配置的设备映射器设备信息。您可以使用 **dmsetup status --target *target_type*** 命令列出那些至少有一个指定类型的对象的设备。

以下示例显示列出在所有目前配置的映射设备中的对象状态的命令。

```
[root@ask-07 ~]# dmsetup status
testgfsvg-testgfslv3: 0 312352768 linear
testgfsvg-testgfslv2: 0 312352768 linear
testgfsvg-testgfslv1: 0 312352768 linear
testgfsvg-testgfslv1: 312352768 50331648 linear
VolGroup00-LogVol01: 0 4063232 linear
VolGroup00-LogVol00: 0 151912448 linear
```

A.2.4. 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。如果您没有指定设备名称，则输出所有目前配置的设备映射器设备信息。

以下示例显示列出在所有目前配置的映射设备相依性的命令。

```
[root@ask-07 ~]# dmsetup deps
testgfsvg-testgfslv3: 1 dependencies    : (8, 16)
testgfsvg-testgfslv2: 1 dependencies    : (8, 16)
testgfsvg-testgfslv1: 1 dependencies    : (8, 16)
VolGroup00-LogVol01: 1 dependencies     : (8, 2)
VolGroup00-LogVol00: 1 dependencies     : (8, 2)
```

以下示例显示只列出设备 **lock_stress-grant--02.1722** 相依性的命令：

```
[root@grant-01 ~]# dmsetup deps lock_stress-grant--02.1722
3 dependencies  : (253, 33) (253, 32) (253, 31)
```

LVM 配置文件

LVM 支持多配置文件。在系统启动时，会从用环境变量 **LVM_SYSTEM_DIR** 指定的目录中载入 **lvm.conf** 配置文件，该变量在 **/etc/lvm** 中是默认设置。

lvm.conf 文件可指定要载入的额外配置文件。后来设定的文件会覆盖之前设定的文件。要在载入所有配置文件后显示使用的设定，请运行 **lvm dumpconfig** 命令。

For information on loading additional configuration files, see 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
For each host tag, an extra configuration file is read if it exists: **lvm_*hosttag*.conf**. If that file defines new tags, then further configuration files will be appended to the list of tiles to read in. For information on host tags, see 第 C.2 节 “主机标签”.

除了 LVM 配置文件之外，运行 LVM 的系统会包含以下可影响 LVM 系统设置的文件：

/etc/lvm/.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
在单主机配置这，锁定文件可防止平行工具运行时破坏元数据；而在群集中，使用的群集范围的 DLM。

B.2. lvm.conf 文件示例

以下是 **lvm.conf** 配置文件示例。这个配置文件是 RHEL 5.3 发行本的默认文件。如果您的系统运行的是不同的 RHEL 5 发行本，有些默认设置可能不同。

```
[root@tng3-1 lvm]# cat /etc/lvm/lvm.conf
# 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.


# 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 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 there are symbolic links or multiple filesystem
    # entries for the same device as each name is checked separately against
    # the list of patterns.  The effect is that if any name matches any 'a'
    # pattern, the device is accepted; otherwise if any name matches any 'r'
    # pattern it is rejected; otherwise it is accepted.

    # Don't have more than one filter line active at once: only one gets used.

    # Run vgscan after you change this parameter to ensure that
    # the cache file gets regenerated (see below).
    # If it doesn't do what you expect, check the output of 'vgscan -vvvv'.


    # By default we accept every block device:
    filter = [ "a/.*/" ]

    # Exclude the cdrom drive
    # filter = [ "r|/dev/cdrom|" ]

    # When testing I like to work with just loopback devices:
    # filter = [ "a/loop/", "r/.*/" ]
```

```
    # Or maybe all loops and ide drives except hdc:
    # filter =[ "a|loop|", "r|/dev/hdc|", "a|/dev/ide|", "r|.*|" ]

    # Use anchors if you want to be really specific
    # filter = [ "a|^/dev/hda8$|", "r/.*/" ]

    # The results of the filtering are cached on disk to avoid
    # rescanning dud devices (which can take a very long time).
    # By default this cache is stored in the /etc/lvm/cache directory
    # in a file called '.cache'.
    # It is safe to delete the contents: the tools regenerate it.
    # (The old setting 'cache' is still respected if neither of
    # these new ones is present.)
    cache_dir = "/etc/lvm/cache"
    cache_file_prefix = ""

    # You can turn off writing this cache file by setting this to 0.
    write_cache_state = 1

    # Advanced settings.

    # List of pairs of additional acceptable block device types found
    # in /proc/devices with maximum (non-zero) number of partitions.
    # types = [ "fd", 16 ]

    # If sysfs is mounted (2.6 kernels) restrict device scanning to
    # the block devices it believes are valid.
    # 1 enables; 0 disables.
    sysfs_scan = 1

    # By default, LVM2 will ignore devices used as components of
    # software RAID (md) devices by looking for md superblocks.
    # 1 enables; 0 disables.
    md_component_detection = 1

    # By default, if a PV is placed directly upon an md device, LVM2
    # will align its data blocks with the the chunk_size exposed in sysfs.
    # 1 enables; 0 disables.
    md_chunk_alignment = 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
}

# This section that allows you to configure the nature of the
# information that LVM2 reports.
log {

    # Controls the messages sent to stdout or stderr.
    # There are three levels of verbosity, 3 being the most verbose.
    verbose = 0

    # Should we send log messages through syslog?
    # 1 is yes; 0 is no.
    syslog = 1
```

```
    # Should we log error and debug messages to a file?
    # By default there is no log file.
    #file = "/var/log/lvm2.log"

    # Should we overwrite the log file each time the program is run?
    # By default we append.
    overwrite = 0

    # What level of log messages should we send to the log file and/or syslog?
    # There are 6 syslog-like log levels currently in use - 2 to 7 inclusive.
    # 7 is the most verbose (LOG_DEBUG).
    level = 0

    # Format of output messages
    # Whether or not (1 or 0) to indent messages according to their severity
    indent = 1

    # Whether or not (1 or 0) to display the command name on each line output
    command_names = 0

    # A prefix to use before the message text (but after the command name,
    # if selected).  Default is two spaces, so you can see/grep the severity
    # of each message.
    prefix = "  "

    # To make the messages look similar to the original LVM tools use:
    #   indent = 0
    #   command_names = 1
    #   prefix = " -- "

    # Set this if you want log messages during activation.
    # Don't use this in low memory situations (can deadlock).
    # activation = 0
}

# Configuration of metadata backups and archiving.  In LVM2 when we
# talk about a 'backup' we mean making a copy of the metadata for the
# *current* system.  The 'archive' contains old metadata configurations.
# Backups are stored in a human readeable text format.
backup {

    # Should we maintain a backup of the current metadata configuration ?
    # Use 1 for Yes; 0 for No.
    # Think very hard before turning this off!
    backup = 1

    # Where shall we keep it ?
    # Remember to back up this directory regularly!
    backup_dir = "/etc/lvm/backup"

    # Should we maintain an archive of old metadata configurations.
    # Use 1 for Yes; 0 for No.
    # On by default.  Think very hard before turning this off.
    archive = 1

    # Where should archived files go ?
    # Remember to back up this directory regularly!
    archive_dir = "/etc/lvm/archive"

    # What is the minimum number of archive files you wish to keep ?
```

```
    retain_min = 10

    # What is the minimum time you wish to keep an archive file for ?
    retain_days = 30
}

# Settings for the running LVM2 in shell (readline) mode.
shell {

    # Number of lines of history to store in ~/.lvm_history
    history_size = 100
}


# Miscellaneous global LVM2 settings
global {
    library_dir = "/usr/lib64"

    # 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"

    # 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 "lvm1" if compiled in, else "lvm2".
    # format = "lvm1"

    # 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.
    locking_type = 3

    # If using external locking (type 2) and initialisation fails,
    # with this set to 1 an attempt will be made to use the built-in
    # clustered locking.
    # If you are using a customised locking_library you should set this to 0.
    fallback_to_clustered_locking = 1

    # If an attempt to initialise type 2 or type 3 locking failed, perhaps
    # because cluster components such as clvmd are not running, with this set
    # to 1 an attempt will be made to use local file-based locking (type 1).
    # If this succeeds, only commands against local volume groups will proceed.
    # Volume Groups marked as clustered will be ignored.
    fallback_to_local_locking = 1

    # Local non-LV directory that holds file-based locks while commands are
    # in progress.  A directory like /tmp that may get wiped on reboot is OK.
    locking_dir = "/var/lock/lvm"

    # 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"
}

activation {
    # How to fill in missing stripes if activating an incomplete volume.
    # Using "error" will make inaccessible parts of the device return
    # I/O errors on access.  You can instead use a device path, in which
    # case, that device will be used to in place of missing stripes.
    # But note that using anything other than "error" with mirrored
    # or snapshotted volumes is likely to result in data corruption.
    missing_stripe_filler = "error"

    # How much stack (in KB) to reserve for use while devices suspended
    reserved_stack = 256

    # How much memory (in KB) to reserve for use while devices suspended
    reserved_memory = 8192

    # Nice value used while devices suspended
    process_priority = -18

    # If volume_list is defined, each LV is only activated if there is a
    # match against the list.
    #   "vgname" and "vgname/lvname" are matched exactly.
    #   "@tag" matches any tag set in the LV or VG.
    #   "@*" matches if any tag defined on the host is also set in the LV or VG
    #
    # volume_list = [ "vg1", "vg2/lvol1", "@tag1", "@*" ]
```

```
    # Size (in KB) of each copy operation when mirroring
    mirror_region_size = 512

    # Setting to use when there is no readahead value stored in the metadata.
    #
    # "none" - Disable readahead.
    # "auto" - Use default value chosen by kernel.
    readahead = "auto"

    # 'mirror_image_fault_policy' and 'mirror_log_fault_policy' define
    # how a device failure affecting a mirror is handled.
    # A mirror is composed of mirror images (copies) and a log.
    # A disk log ensures that a mirror does not need to be re-synced
    # (all copies made the same) every time a machine reboots or crashes.
    #
    # In the event of a failure, the specified policy will be used to
    # determine what happens:
    #
    # "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.
    #            Currently this is not implemented properly and behaves
    #            similarly to:
    #
    # "allocate_anywhere" - Operates like "allocate", but it does not
    #            require that the new space being allocated be on a
    #            device is not part of the mirror.  For a log device
    #            failure, this could mean that the log is allocated on
    #            the same device as a mirror device.  For a mirror
    #            device, this could mean that the mirror device is
    #            allocated on the same device as another mirror device.
    #            This policy would not be wise for mirror devices
    #            because 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_device_fault_policy = "remove"
}


####################
# 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

    # Approximate default size of on-disk metadata areas in sectors.
    # You should increase this if you have large volume groups or
    # you want to retain a large on-disk history of your metadata changes.

    # pvmetadatasize = 255

    # List of directories holding live copies of text format metadata.
    # These directories must not be on logical volumes!
    # It's possible to use LVM2 with a couple of directories here,
    # preferably on different (non-LV) filesystems, and with no other
    # on-disk metadata (pvmetadatacopies = 0). Or this can be in
    # addition to on-disk metadata areas.
    # The feature was originally added to simplify testing and is not
    # supported under low memory situations - the machine could lock up.
    #
    # Never edit any files in these directories by hand unless you
    # you are absolutely sure you know what you are doing! Use
    # the supplied toolset to make changes (e.g. vgcfgrestore).

    # dirs = [ "/etc/lvm/metadata", "/mnt/disk2/lvm/metadata2" ]
#}

# Event daemon
#
dmeventd {
    # mirror_library is the library used when monitoring a mirror device.
    #
    # "libdevmapper-event-lvm2mirror.so" attempts to recover from
    # failures.  It removes failed devices from a volume group and
    # reconfigures a mirror as necessary. If no mirror library is
    # provided, mirrors are not monitored through dmeventd.

    mirror_library = "libdevmapper-event-lvm2mirror.so"

    # snapshot_library is the library used when monitoring a snapshot device.
    #
    # "libdevmapper-event-lvm2snapshot.so" monitors the filling of
    # snapshots and emits a warning through syslog, when the use of
    # snapshot exceedes 80%. The warning is repeated when 85%, 90% and
    # 95% of the snapshot are filled.

    snapshot_library = "libdevmapper-event-lvm2snapshot.so"
}
```

LVM 对象标签

LVM 标签是一个可将有相同类型的 LVM2 对象分成同一组的单词。可将标签附加到对象中，对象可以是物理卷、卷组、逻辑卷、片段。也可将标签附加到群集配置的主机中。无法为快照添加标签。

可在命令行的 PV、VG 或者 LV 参数中赋予标签。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 标签是使用 [A-Za-z0-9_+.-] 的字符串，最长为 128 个字符，它们不可以连字符开始。

只能为卷组中的对象添加标签。如果从卷组中删除物理卷，它们就会丢失它们的标签。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掉。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
```

C.1. 添加和删除对象标签

要从物理卷中添加或者删除标签，请使用 **pvchange** 命令的 **--addtag** 或者 **--deltag** 选项。

要从卷组中添加或者删除标签，请使用 **vgchange** 或 **vgcreate** 命令的 **--addtag** 或者 **--deltag** 选项。

要在逻辑卷中添加或者删除标签，请请使用 **lvchange** 或 **lvcreate** 命令的 **--addtag** 或者 **--deltag** 选项。

C.2. 主机标签

In a cluster configuration, you can define host tags in the configuration files. If you set **hosttags = 1** in the **tags** section, a host tag is automatically defined using the machine's hostname. This allow you to use a common configuration file which can be replicated on all your machines so they hold identical copies of the file, but the behavior can differ between machines according to the hostname.

For information on the configuration files, see 附录 B, *LVM 配置文件*.

对于每个主机标签，如果存在额外的配置文件 lvm_***hosttag***.conf，就会读取它。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**，且在主机名为 **host1** 定义 **tag2**。

```
tags { tag1 { }  tag2 { host_list = ["host1"] } }
```

C.3. 使用标签控制激活

您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" ] }
```

There is a special match "@*" that causes a match only if any metadata tag matches any host tag on that machine.

另一个例子就是，考虑一下，在哪里群集中的每一台机器都在配置文件中有以下条目：

```
tags { hosttags = 1 }
```

如果您想要只在主机 **db2** 中激活 **vg1/lvol2**，请执行以下操作：

1. 可从群集中的任意主机运行 **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。

LVM 卷组元数据

卷组的配置详情就是元数据。默认情况是在卷组的每个物理卷的元数据区域都会保存一个相同的元数据备份。LVM 元数据很小，它以 ASCII 格式保存。

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。您可以使用 **pvcreate** 命令的 **--metadatacopies 0** 选项创建没有任何元数据副本的物理卷。一旦您选择了物理卷将包含的元数据副本的数目，您将无法修改它。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）。如果您试图在将来分割卷组，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，然后会更新开始的指示点。

您可以使用 **pvcreate** 命令 **--metadatasize** 选项指定元数据区域的大小。默认的大小对于有很多逻辑卷或者物理卷的卷组来说太小了。

D.1. 物理卷标签

默认情况下，**pvcreate** 命令会在第二个 512 字节部分放置物理卷标签。这个标签可选择性地放在前四个部分的任意一个中，因为扫描物理卷标签的 LVM 工具会检查前四个部分。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
- 数据区域位置的 NULL 终止列表
- 元数据区域位置的 NULL 终止列表

元数据位置以偏差和大小（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: 这个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
- 无论何时更新元数据时增大的版本数目
- 任意属性：读/写？可重新定义大小？
- 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）
- 一个没有排序的物理卷列表组成卷组，每个都带有：
  - 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
  - 在物理卷中第一个扩展的开始调节（在扇区中）
  - 扩展的数目

- 没有配需的逻辑卷列表，每个都包含
  - 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

以下显示了名为 **myvg** 的卷组的 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-lDHq-lMPs-EjoE-0o18-wL28X4"
        seqno = 3
        status = ["RESIZEABLE", "READ", "WRITE"]
        extent_size = 8192              # 4 Megabytes
        max_lv = 0
        max_pv = 0

        physical_volumes {

                pv0 {
                        id = "ZBW5qW-dXF2-0bGw-ZCad-2RlV-phwu-1c1RFt"
                        device = "/dev/sda"     # Hint only

                        status = ["ALLOCATABLE"]
                        dev_size = 35964301     # 17.1491 Gigabytes
                        pe_start = 384
                        pe_count = 4390 # 17.1484 Gigabytes
                }

                pv1 {
                        id = "ZHEZJW-MR64-D3QM-Rv7V-Hxsa-zU24-wztY19"
                        device = "/dev/sdb"     # Hint only

                        status = ["ALLOCATABLE"]
                        dev_size = 35964301     # 17.1491 Gigabytes
                        pe_start = 384
                        pe_count = 4390 # 17.1484 Gigabytes
                }

                pv2 {
                        id = "wCoG4p-55Ui-9tbp-VTEA-jO6s-RAVx-UREW0G"
                        device = "/dev/sdc"     # Hint only

                        status = ["ALLOCATABLE"]
                        dev_size = 35964301     # 17.1491 Gigabytes
                        pe_start = 384
                        pe_count = 4390 # 17.1484 Gigabytes
                }

                pv3 {
                        id = "hGlUwi-zsBg-39FF-do88-pHxY-8XA2-9WKIiA"
                        device = "/dev/sdd"     # Hint only

                        status = ["ALLOCATABLE"]
                        dev_size = 35964301     # 17.1491 Gigabytes
                        pe_start = 384
                        pe_count = 4390 # 17.1484 Gigabytes
```

```
                }
        }
        logical_volumes {

                mylv {
                        id = "GhUYSF-qVM3-rzQo-a6D2-o0aV-LQet-Ur9OF9"
                        status = ["READ", "WRITE", "VISIBLE"]
                        segment_count = 2

                        segment1 {
                                start_extent = 0
                                extent_count = 1280     # 5 Gigabytes

                                type = "striped"
                                stripe_count = 1        # linear

                                stripes = [
                                        "pv0", 0
                                ]
                        }
                        segment2 {
                                start_extent = 1280
                                extent_count = 1280     # 5 Gigabytes

                                type = "striped"
                                stripe_count = 1        # linear

                                stripes = [
                                        "pv1", 0
                                ]
                        }
                }
        }
}
```

修订记录

修订 3-6.400	2013-10-31	Rüdiger Landmann
Rebuild with publican 4.0.0		

修订 3-6	2012-07-18	Anthony Towns
Rebuild for Publican 3.0		

修订 1.0-0	Thu Jan 29 2009	

索引

A

B

C

D

E

M

U

V