



# Red Hat Enterprise Linux 6

## 邏輯卷冊管理程式管理

LVM 管理員指南

版 1



LVM 管理員指南  
版 1

Landmann  
rlandmann@redhat.com

## 法律聲明

Copyright © 2012 Red Hat, Inc. and others.

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

## 摘要

本指南詳述了 LVM 邏輯卷冊管理程式（logical volume manager），包括在叢集環境下執行 LVM 的相關資訊。

# 內容目錄

<b>簡介</b>	<b>5</b>
1. 有關本指南	5
2. 閱讀對象	5
3. 軟體版本	5
4. 相關文件	5
5. 我們需要您的意見！	6
<b>章 1. LVM 邏輯卷測管理程式</b>	<b>7</b>
1.1. 新功能與遭到變更的功能	7
1.1.1. Red Hat Enterprise Linux 6.0 上經過變更, 和新的功能	7
1.1.2. Red Hat Enterprise Linux 6.1 上經過變更, 和新的功能	7
1.1.3. RHEL 6.2 上受到變更以及新的功能	8
1.1.4. RHEL 6.3 上受到變更以及新的功能	8
1.2. 邏輯卷冊	8
1.3. LVM 架構總覽	9
1.4. 叢集邏輯卷測管理員 (CLUSTERED LOGICAL VOLUME MANAGER, CLVM)	10
1.5. 文件總覽	12
<b>章 2. LVM 元件</b>	<b>14</b>
2.1. 實體卷冊	14
2.1.1. LVM 實體卷冊配置	14
2.1.2. 磁碟上的多重分割區	15
2.2. 卷冊群組	15
2.3. LVM 邏輯卷冊	15
2.3.1. 線性邏輯卷冊	16
2.3.2. 等量邏輯卷冊	17
2.3.3. 鏡像邏輯卷冊	18
2.3.4. Snapshot 卷冊	19
<b>章 3. LVM 管理總覽</b>	<b>21</b>
3.1. 在叢集中建立 LVM 卷冊	21
3.2. 邏輯卷冊建立總覽	21
3.3. 在邏輯卷冊上遞增檔案系統	22
3.4. 邏輯卷冊備份	22
3.5. 記錄	22
<b>章 4. 透過 CLI 指令來進行 LVM 管理</b>	<b>24</b>
4.1. 使用 CLI 指令	24
4.2. 實體卷冊管理	25
4.2.1. 建立實體卷冊	25
4.2.1.1. 設定分割區類型	25
4.2.1.2. 初始化實體卷冊	25
4.2.1.3. 掃描區塊裝置	26
4.2.2. 顯示實體卷冊	26
4.2.3. 避免配置於實體卷冊上	27
4.2.4. 重設實體卷冊的大小	27
4.2.5. 移除實體卷冊	27
4.3. 卷冊群組管理	28
4.3.1. 建立卷冊群組	28
4.3.2. 在叢集中建立卷冊群組	29
4.3.3. 新增實體卷冊至卷冊群組中	29
4.3.4. 顯示卷冊群組	29

4.3.5. 掃描磁碟來找尋卷冊群組以便建立快取檔案	30
4.3.6. 由卷冊群組中移除實體卷冊	30
4.3.7. 更改卷冊群組的參數	31
4.3.8. 啟用和停用卷冊群組	31
4.3.9. 移除卷冊群組	32
4.3.10. 分割卷冊群組	32
4.3.11. 結合卷冊群組	32
4.3.12. 備份卷冊群組的 Metadata	32
4.3.13. 為卷冊群組重新命名	32
4.3.14. 將卷冊群組移至另一部系統	33
4.3.15. 重新建立一個卷冊群組目錄	33
4.4. 邏輯卷冊管理	33
4.4.1. 建立線性邏輯卷冊	33
4.4.2. 建立等量卷冊	35
4.4.3. 建立鏡像卷冊	35
4.4.3.1. 鏡像邏輯卷冊失效政策	38
4.4.3.2. 由鏡像邏輯卷冊切割出冗餘映像	38
4.4.3.3. 修復鏡像邏輯裝置	39
4.4.3.4. 更改鏡像卷冊配置	39
4.4.4. 建立快照卷冊 (Snapshot Volumes)	40
4.4.5. 合併 Snapshot 卷冊	41
4.4.6. 一致的裝置號碼	42
4.4.7. 重設邏輯卷冊大小	42
4.4.8. 更改邏輯卷冊群組的參數	42
4.4.9. 重新為邏輯卷冊命名	42
4.4.10. 移除邏輯卷冊	42
4.4.11. 顯示邏輯卷冊	43
4.4.12. 遞增邏輯卷冊	43
4.4.12.1. 延伸等量的卷冊	44
4.4.12.2. 延伸鏡像卷冊	45
4.4.12.3. 以 cling 分配政策延伸邏輯卷冊	46
4.4.13. RAID 邏輯卷冊	47
4.4.13.1. 建立 RAID 邏輯卷冊	48
4.4.13.2. 將線性裝置轉換為 RAID 裝置	50
4.4.13.3. 將 LVM RAID1 邏輯卷冊轉換為一個 LVM 線性邏輯卷冊	51
4.4.13.4. 將鏡像 LVM 裝置轉換為 RAID 1 裝置	51
4.4.13.5. 修改既有 RAID1 裝置中的映像檔數量	52
4.4.13.6. 將 RAID 映像檔分割為各別的邏輯卷冊	54
4.4.13.7. 分割與合併 RAID 映像檔	55
4.4.13.8. 設定 RAID 錯誤政策	57
4.4.13.8.1. 「allocate」RAID 錯誤政策	58
4.4.13.8.2. 「warn」RAID 錯誤政策 (Fault Policy)	59
4.4.13.9. 替換 RAID 裝置	60
4.4.14. 縮減邏輯卷冊	61
4.5. 透過過濾器來控制 LVM 裝置掃描 (LVM DEVICE SCANS)	62
4.6. 線上資料重置 (ONLINE DATA RELOCATION)	63
4.7. 在叢集中啟用各別節點上的邏輯卷冊	63
4.8. LVM 的自訂化回報	63
4.8.1. 格式控制	64
4.8.2. 物件選擇	65
4.8.2.1. pvs 指令	66
4.8.2.2. vgs 指令	68
4.8.2.3. lvs 指令	69

4.8.3. 排序 LVM 報告	73
4.8.4. 指定單位	73
<b>章 5. LVM 配置範例</b>	<b>75</b>
5.1. 在三個磁碟上建立一個 LVM 邏輯卷冊	75
5.1.1. 建立實體卷冊 (Physical Volumes)	75
5.1.2. 建立卷冊群組	75
5.1.3. 建立邏輯卷冊	75
5.1.4. 建立檔案系統	75
5.2. 建立一個磁條邏輯卷冊 (STRIPED LOGICAL VOLUME)	76
5.2.1. 建立實體卷冊 (Physical Volumes)	76
5.2.2. 建立卷冊群組	76
5.2.3. 建立邏輯卷冊	77
5.2.4. 建立檔案系統	77
5.3. 分割卷冊群組	77
5.3.1. 判斷可用空間	78
5.3.2. 移動資料	78
5.3.3. 分割卷冊群組	78
5.3.4. 建立新的邏輯卷冊	79
5.3.5. 製作檔案系統和掛載新的邏輯卷冊	79
5.3.6. 啟用和掛載原本的邏輯卷冊	79
5.4. 由邏輯卷冊中移除磁碟	80
5.4.1. 將扇區移至現有的實體卷冊中	80
5.4.2. 將扇區移至新磁碟上	80
5.4.2.1. 新建實體卷冊	81
5.4.2.2. 新增實體卷冊至卷冊群組中	81
5.4.2.3. 移動資料	81
5.4.2.4. 將舊的實體卷冊由卷冊群組中移除	81
5.5. 在叢集中建立鏡像 LVM 邏輯卷冊	82
<b>章 6. LVM 疑難排解</b>	<b>85</b>
6.1. 疑難排解診斷結果	85
6.2. 顯示錯誤裝置的相關資訊	85
6.3. 由 LVM 鏡像錯誤中復原	86
6.4. 復原實體卷冊的 METADATA	89
6.5. 替換一個遺失的實體卷冊	90
6.6. 將遺失的實體卷冊由一個卷冊群組中移除掉	91
6.7. 邏輯卷冊的可用扇區不足	91
<b>章 7. 利用 LVM GUI 來進行 LVM 管理</b>	<b>92</b>
<b>附錄 A. 裝置映射 (DEVICE MAPPER) 設備</b>	<b>93</b>
A.1. 裝置表格映射	93
A.1.1. Linear 映射目標	94
A.1.2. 等量映射目標	94
A.1.3. 鏡像映射目標	96
A.1.4. 快照和 snapshot-origin 映射目標	98
A.1.5. error 映射目標	99
A.1.6. zero 映射目標	100
A.1.7. multipath 映射目標	100
A.1.8. crypt 映射目標	102
A.2. DMSETUP 指令	103
A.2.1. dmsetup info 指令	103
A.2.2. dmsetup ls 指令	105

A.2.3. dmsetup status 指令	105
A.2.4. dmsetup deps 指令	106
A.3. UDEV DEVICE MANAGER（裝置管理員）的 DEVICE MAPPER 支援	106
A.3.1. udev 與裝置管理員的整合	107
A.3.2. 支援 udev 的指令與介面	108
<b>附錄 B. LVM 配置檔案</b>	<b>110</b>
B.1. LVM 配置檔案	110
B.2. 範例 LVM.CONF 檔案	110
<b>附錄 C. LVM 物件標籤（OBJECT TAGS）</b>	<b>125</b>
C.1. 新增和移除物件標籤	125
C.2. 主機標籤（HOST TAGS）	125
C.3. 利用標籤來控制啓用	126
<b>附錄 D. LVM 卷冊群組 METADATA</b>	<b>127</b>
D.1. 實體卷冊標籤（PHYSICAL VOLUME LABEL）	127
D.2. METADATA 內容	127
D.3. METADATA 範例	128
<b>附錄 E. 修訂歷史</b>	<b>131</b>
<b>索引</b>	<b>133</b>



# 簡介

## 1. 有關本指南

本指南描述了邏輯卷冊管理程式（Logical Volume Manager, LVM），包括有關於在叢集環境下執行 LVM 的相關資訊。

## 2. 閱讀對象

本指南的目標閱讀對象為管理執行 Linux 作業系統的系統管理員。使用者必須熟悉 Red Hat Enterprise Linux 6 以及 GFS2 檔案系統管理。

## 3. 軟體版本

表格 1. 軟體版本

軟體	描述
Red Hat Enterprise Linux 6	請參閱 RHEL 6 或更新版本
GFS2	請參閱 RHEL 6 或更新版本的 GFS2

## 4. 相關文件

預知更多 Red Hat Enterprise Linux 的訊息，請參閱以下資源：

- *安裝指南* — 記載了 Red Hat Enterprise Linux 6 安裝上的相關資訊。
- *建置指南* — 提供了 Red Hat Enterprise Linux 6 的建置、配置與管理上的相關資訊。
- *儲存管理指南* — 提供了如何有效在 Red Hat Enterprise Linux 6 上管理儲存裝置與檔案系統的相關資訊。

欲取得更多有關於 Red Hat Enterprise Linux 6 High Availability 外掛程式和 Resilient Storage 外掛程式上的相關資訊，請參閱下列資源：

- *High Availability 外掛程式總覽* — 提供了 Red Hat High Availability 外掛程式的高層總覽。
- *叢集管理* — 提供了有關於安裝、配置和管理 Red Hat High Availability 外掛程式上的相關資訊。
- *Global File System 2：配置與管理* — 提供了有關於安裝、配置和維護 Red Hat GFS2（Red Hat 全域檔案系統 2）上的相關資訊，並且這項資訊包含在 Resilient Storage 外掛程式中。
- *DM Multipath* — 提供了有關於 Red Hat Enterprise Linux 6 Device-Mapper Multipath 功能使用上的相關資訊。

- **負載平衡器管理** — 提供了有關於透過 Red Hat Load Balancer 外掛程式配置高效能系統和服務上的相關資訊，一組提供了 Linux 虛擬伺服器（LVS），以在多台真實伺服器之間，平衡 IP 負載的整合軟體元件。
- **發行公告** — 提供了最新 Red Hat 產品發行版的相關資訊。

High Availability 外掛程式文件與其它 Red Hat 文件在 Red Hat Enterprise Linux 文件光碟上皆擁有 HTML、PDF 和 RPM 版本，以及線上版本：<http://docs.redhat.com/docs/en-US/index.html>。

## 5. 我們需要您的意見！

如果您發現本指南中有謬誤之處，或任何能改善本文件的方法，我們竭誠歡迎您的意見。請至 <http://bugzilla.redhat.com/> 使用 Bugzilla，並針對於 **Red Hat Enterprise Linux 6** 產品和 **doc-Logical\_Volume\_Manager** 元件提交您的意見。當提交錯誤報告時，請記得提供指南的識別元：

Logical\_Volume\_Manager\_Administration(EN)-6 (2012-6-15-15:20)

如果您有任何改善本文件的建議，請盡可能地詳述內容。如果您發現了錯誤，請告知我們章節與其附近的本文，如此一來我們便能更輕易地找到錯誤。

## 章 1. LVM 邏輯卷管理程式

本章提供了 Red Hat Enterprise Linux 6 初始與後續發行版上，LVM 邏輯卷管理程式新功能上的概要。在那之後，本章亦提供了基本的邏輯卷管理程式（LVM）元件總覽。

### 1.1. 新功能與遭到變更的功能

此部份列出了包含在 Red Hat Enterprise Linux 6 初始，以及後續發行版中的 LVM 邏輯卷管理程式的新功能與變更。

#### 1.1.1. Red Hat Enterprise Linux 6.0 上經過變更，和新的功能

Red Hat Enterprise Linux 6.0 包含了下列文件以及功能更新及變更。

- 您可透過 `lvm.conf` 檔案的 `activation` 部份中的 `mirror_image_fault_policy` 和 `mirror_log_fault_policy` 參數，來定義當裝置失效時，鏡像邏輯卷冊會如何反應。當此參數被設為了 `remove` 時，系統會嘗試移除出錯的裝置，並在沒此裝置的情況下運作。當此參數被設為了 `allocate` 時，系統會嘗試移除出錯的裝置，並嘗試在新裝置上配置空間，以取代失效的裝置；若無適當的裝置與空間可用來替代的話，此政策將會如 `remove` 政策一般運作。欲取得更多有關於 LVM 鏡像失效政策上的相關資訊，請參閱〈[節 4.4.3.1, “鏡像邏輯卷冊失效政策”](#)〉。
- 在 Red Hat Enterprise Linux 6 發行版上，Linux I/O 堆疊已經過改善，以處理廠商所提供的 I/O 限制資訊。這能讓儲存管理工具（包括 LVM）優化資料定位與存取。這項支援可透過更改 `lvm.conf` 檔案中，`data_alignment_detection` 和 `data_alignment_offset_detection` 的預設值來停用，不建議您停用這項支援。

欲取得 LVM 中的資料對稱資訊以及更改 `data_alignment_detection` 和 `data_alignment_offset_detection` 的預設值的關資訊，請參閱 `/etc/lvm/lvm.conf` 檔案的內嵌文件，此文件同時亦記載於〈[附錄 B, LVM 配置檔案](#)〉中。欲取得有關於 RHEL 6 中的 I/O 堆疊與 I/O 限制支援上的一般資訊，請參閱【[儲存裝置管理指南](#)】。

- 在 RHEL 6 中，Device Mapper 直接提供了 `udev` 整合上的支援。這會同步化 Device Mapper 和所有與 Device Mapper 相關的 `udev`，包括 LVM 裝置。欲取得更多有關於 `udev` 裝置管理員的 Device Mapper 支援上的相關資訊，請參閱〈[節 A.3, “udev Device Manager（裝置管理員）的 Device Mapper 支援”](#)〉。
- 在 RHEL 6 發行版上，當磁碟失效時，您可使用 `lvconvert --repair` 指令來修復鏡像。這會將鏡像復原為正常的狀態。欲取得 `lvconvert --repair` 指令上的相關資訊，請參閱〈[節 4.4.3.3, “修復鏡像邏輯裝置”](#)〉。
- 從 RHEL 6 發行版開始，您將能使用 `lvconvert` 指令的 `--merge` 選項來將 snapshot 合併入它原始的卷冊中。欲取得 snapshot 合併上的相關資訊，請參閱 [節 4.4.5, “合併 Snapshot 卷冊”](#)。
- 由 RHEL 6 發行版起，您將能使用 `lvconvert` 指令的 `--splitmirrors` 引數來切割重複的鏡像邏輯卷冊映像，以形成新的邏輯卷冊。欲取得此選項使用上的相關資訊，請參閱〈[節 4.4.3.2, “由鏡像邏輯卷冊切割出冗餘映像”](#)〉。
- 現在當在建立鏡像邏輯裝置時，您可為透過了使用 `lvcreate` 指令的 `--mirrorlog mirrored` 引數所映射的鏡像邏輯裝置，建立一個鏡像 log。欲取得使用此裝置上的相關資訊，請參閱〈[節 4.4.3, “建立鏡像卷冊”](#)〉。

#### 1.1.2. Red Hat Enterprise Linux 6.1 上經過變更，和新的功能

Red Hat Enterprise Linux 6.1 包含了下列文件和功能更新與變更。

- 從 Red Hat Enterprise Linux 6.1 發行版支援建立鏡像邏輯卷冊的 snapshot 邏輯卷冊。您可如您建立線性或磁條邏輯卷冊一般地建立鏡像卷冊的 snapshot。欲取得更多有關於建立 snapshot 卷冊上的相關資訊，請參閱 [〈節 4.4.4, “建立快照卷冊 \(Snapshot Volumes\)”](#)。
- 當延伸 LVM 卷冊時，您現在已能使用 **lvextend** 指令的 **--alloc cling** 選項，來指定 **cling** 分配政策。此政策將會在與既有邏輯卷冊的最後磁區相同的實體卷冊上選擇空間。若實體卷冊上的空間不足，並且 **lvm.conf** 檔案中已定義了一列標籤，LVM 將會檢查是否有任何標籤已連接至實體卷冊，並嘗試在既有扇區和新扇區之間，比對這些實體卷冊標籤。

欲取得更多有關於透過 **lvextend** 指令的 **--alloc cling** 選項，來延伸 LVM 鏡像卷冊上的相關資訊，請參閱 [〈節 4.4.12.3, “以 cling 分配政策延伸邏輯卷冊”](#)。

- 您現在已可在 **pvchange**、**vgchange** 或 **lvchange** 指令中指定多重 **--addtag** 和 **--deltag** 引數。欲取得有關於新增或移除物件標籤上的相關資訊，請參閱 [〈節 C.1, “新增和移除物件標籤”](#)。
- LVM 物件標籤中允許的字元已擴增，標籤現在已可包含「/」、「=」、「!」、「:」、「#」，以及「&」字元。欲取得 LVM 物件標籤上的相關資訊，請參閱 [〈附錄 C, LVM 物件標籤 \(Object Tags\)〉](#)。
- 您現在已可在單獨的邏輯卷冊中合併 RAID0 (striping) 和 RAID1 (mirroring)。建立邏輯卷冊並同時指定鏡像數量 (**--mirrors X**) 與磁條數量 (**--stripes Y**)，會使鏡像裝置所構成的裝置成為等量。欲取得建立鏡像邏輯卷冊上的相關資訊，請參閱 [〈節 4.4.3, “建立鏡像卷冊”](#)。
- 由 Red Hat Enterprise Linux 6.1 發行版起，若您需要在叢集邏輯卷冊上建立一致的資料備份，您可單獨地啟用卷冊，然後建立 snapshot。欲取得更多有關於在一個節點上，單獨啟用邏輯卷冊的相關資訊，請參閱 [〈節 4.7, “在叢集中啟用各別節點上的邏輯卷冊”](#)。

### 1.1.3. RHEL 6.2 上受到變更以及新的功能

RHEL 6.2 包含了下列文件與功能更新及改變。

- Red Hat Enterprise Linux 6.2 發行版本的 **lvm.conf** 配置檔案支援 **issue\_discards** 參數。當此參數被設定時，如果邏輯卷冊不再使用實體卷冊的空間時，LVM 會發出棄置訊號至邏輯卷冊下方的實體卷冊。  
欲知此參數的更多資訊，請參閱 **/etc/lvm/lvm.conf** 檔案中的文件，或從 [附錄 B, LVM 配置檔案](#) 中取得。

### 1.1.4. RHEL 6.3 上受到變更以及新的功能

RHEL 6.3 包含了下列文件與功能更新和變更。

- 由 RHEL 6.3 發行版起，LVM 支援了 RAID4/5/6 以及新的鏡像實作。欲知 RAID 邏輯卷冊的相關資訊，請參閱 [節 4.4.13, “RAID 邏輯卷冊”](#)。
- 當您建立新的鏡射裝置，且不需要加以修改時，您可以指定 **--nosync** 選項，表示不需要從第一個裝置進行初始同步。欲知建立鏡射卷冊的資訊，請參閱 [〈節 4.4.3, “建立鏡像卷冊”](#)。
- 本手冊現在也記載 snapshot **autoextend** 功能。欲知建立 snapshot 卷冊的資訊，請參閱 [〈節 4.4.4, “建立快照卷冊 \(Snapshot Volumes\)”](#)。

## 1.2. 邏輯卷冊

卷冊管理程式會在實體儲存裝置上建立抽象層，讓您可建立邏輯儲存卷冊。這提供了比直接使用實體儲存裝置還要大的靈活性。當利用邏輯卷冊時，您不會受到實體磁碟大小的限制。此外，硬體儲存配置對軟體來說是隱藏的，如此一來它可在不停止應用程式，或卸載檔案系統的情況下，重設大小或進行移動。這可

降低作業上的成本。

和直接使用實體儲存相較之下，邏輯卷冊提供了下列優點：

- 可變通的容量

當使用邏輯卷冊時，檔案系統可延伸至多重磁碟上，因為您可將磁碟和分割區聚合為一個單獨的邏輯卷冊。

- 可重設大小的儲存池（Resizeable storage pools）

您可在不格式化與重新分割基本磁碟裝置的情況下透過基本的軟體指令來延伸或遞減邏輯卷冊的大小。

- 線上資料重置（Online data relocation）

若要建置更新、更快，或更有彈性的儲存子系統，您可在您的系統啓用時移動資料。資料可在磁碟使用中的時候被重新整理。比方說，您可在移除一個熱插拔磁碟之前先將它清空。

- 方便的裝置命名

邏輯儲存卷冊可管理於用戶定義群組中，並且您可視您的喜好來進行命名。

- 多磁碟記錄塊串操作（Disk striping）

您可建立一個將資料分散在兩個或更多磁碟上的邏輯卷冊。這會顯著地增加總處理能力。

- 卷冊鏡像

邏輯卷冊提供了一個便利的方式來為您的資料配置一個鏡像。

- 卷冊快照（Volume Snapshots）

當使用邏輯卷冊時，您能夠透過產生裝置快照來進行含有一致性的備份，或在不影響真實資料的情況下測試進行變更後的效果。

這些 LVM 中的功能實做描述於此文件剩下的部份中。

## 1.3. LVM 架構總覽

RHEL 4 發行版的 Linux 作業系統上，原本的 LVM1 邏輯卷冊管理程式已被 LVM2 取代，它含有個比 LVM1 更加通用的 kernel 架構。LVM2 針對於 LVM1 提供了下列改善：

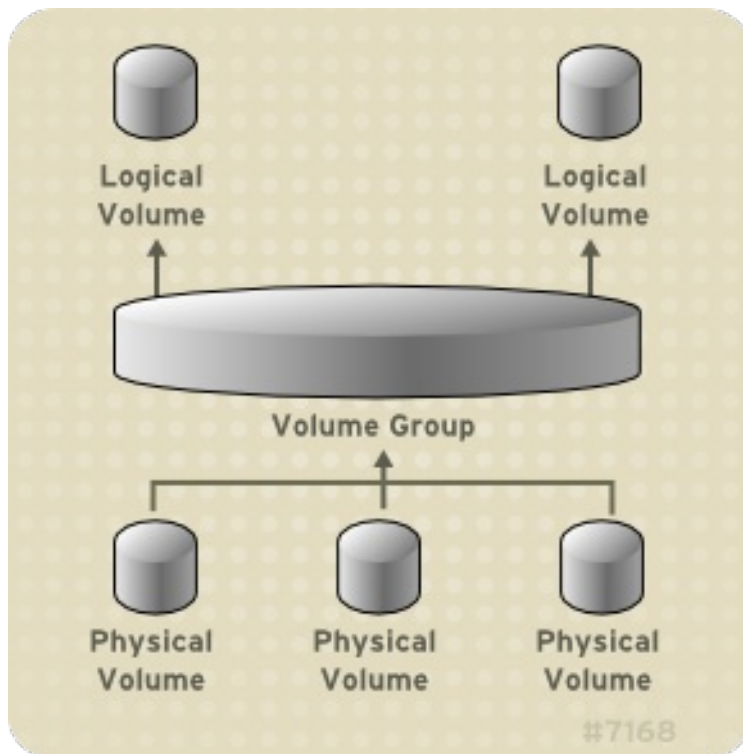
- 可變通的容量
- 更有效率的 metadata 儲存
- 較佳的復原格式
- 新的 ASCII metadata 格式
- metadata 的基元變更
- metadata 的重複副本

LVM2 含有 LVM1 的向後相容性，除了快照與叢集支援。您可透過使用 **vgconvert** 指令來將卷冊群組由 LVM1 格式轉換為 LVM2 格式。如欲取得更多有關於轉換 LVM metadata 格式的相關資訊，請參閱 **vgconvert(8) man page**。

一個 LVM 邏輯卷冊的基本實體儲存裝置是個像是分割區或是完整磁碟的區塊裝置。此裝置被初始化為了一個 LVM 實體卷冊 (PV)。

若要建立一個 LVM 邏輯卷冊，實體卷冊會被併入一個卷冊群組 (volume group, VG) 中。這會建立一個磁碟的空間池，從而，LVM 邏輯卷冊 (LVs) 便能被分配出。這項程序和磁碟被劃分為分割區的方式類似。邏輯卷冊是由檔案系統和應用程式（例如資料庫）所使用的。

〈圖形 1.1, “LVM 邏輯卷冊元件”〉顯示了一個基本 LVM 邏輯卷冊的元件：



圖形 1.1. LVM 邏輯卷冊元件

欲取得一個 LVM 邏輯卷冊元件上的相關資訊，請查看〈[章 2, LVM 元件](#)〉。

## 1.4. 叢集邏輯卷冊管理員 (CLUSTERED LOGICAL VOLUME MANAGER, CLVM)

叢集邏輯卷冊管理程式 (Clustered Logical Volume Manager, CLVM) 為一組 LVM 的叢集延伸。這些延伸允許叢集中的元件透過使用 LVM (比方說在 SAN 上) 來管理共享的儲存裝置。CLVM 屬於 Resilient Storage 外掛程式的一部分。

您是否應使用 CLVM 取決於您的系統需求：

- 若您系統只有一個節點需要存取您配置來作為邏輯卷冊的儲存裝置，那麼您可使用 LVM 並且不使用 CLVM 的延伸，以該節點建立的邏輯卷冊便都會是節點的本地邏輯卷冊。
- 若您使用叢集系統作為容錯服務用，而任何時候皆只有一個存取儲存裝置的節點會啓用的話，您應使用 High Availability Logical Volume Management 代理程式 (HA-LVM)。



- 若您的叢集有超過一個節點需要存取您的儲存裝置並在啓用的節點之間進行共享的話，那麼您就必須使用 CLVM。CLVM 允許用戶透過在配置邏輯卷冊時將實體儲存裝置鎖定以便配置共享儲存裝置上的邏輯卷冊，並使用叢集鎖定服務來管理共享儲存裝置。

若要使用 CLVM，High Availability 外掛程式以及 Resilient Storage 外掛程式軟體，包括 **clvmd** daemon，皆必須要執行。**clvmd** daemon 為 LVM 的關鍵叢集延伸。**clvmd** daemon 會在各個叢集電腦中執行，並在叢集中分配 LVM metadata 的更新，提供各個叢集電腦相同的邏輯卷冊視點。欲取得更多有關於安裝和管理 High Availability 外掛程式的相關資訊，請參閱《叢集管理》。

若要確保 **clvmd** 會在開機時啓動，您可針對於 **clvmd** 服務執行一項 **chkconfig ... on** 指令，如下：

```
# chkconfig clvmd on
```

若 **clvmd** daemon 沒有啓動的話，您可針對於 **clvmd** 服務執行一項 **service ... start** 指令，如下：

```
# service clvmd start
```

在叢集環境中建立 LVM 邏輯卷冊，和在單獨節點上建立 LVM 邏輯卷冊基本上是相同的。LVM 指令本身沒有改變，LVM 圖形化用戶介面亦相同，如〈[章 4, 透過 CLI 指令來進行 LVM 管理](#)〉和〈[章 7, 利用 LVM GUI 來進行 LVM 管理](#)〉所描述。若要啓用您在叢集中所建立的 LVM 卷冊，該叢集架構必須要處於執行狀態中，並且叢集也必須要 **quorate**。

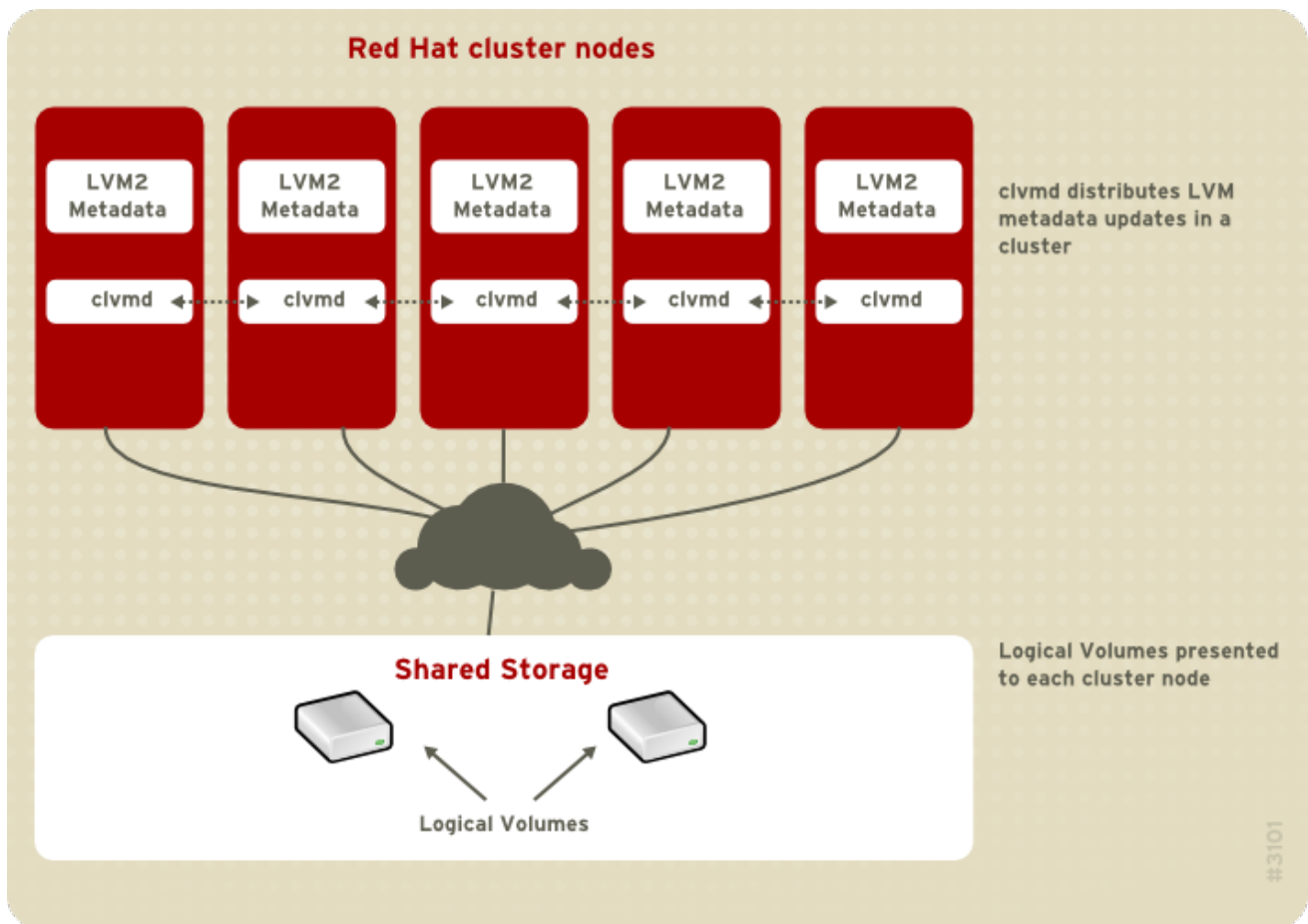
就預設值，擁有共享儲存裝置存取權限的所有電腦，皆可看見透過 CLVM 在該共享儲存裝置上所建立的邏輯卷冊。您亦可在儲存裝置只能讓叢集中的一個節點偵測到的情況下，建立邏輯卷冊。您亦可將邏輯卷冊的狀態，由本機卷冊更改為叢集卷冊。欲取得更多相關資訊，請參閱〈[節 4.3.2, “在叢集中建立卷冊群組”](#)〉和〈[節 4.3.7, “更改卷冊群組的參數”](#)〉。



### 警告

當您在 CLVM 位於共享儲存裝置上時建立卷冊群組時，您必須確認叢集中的所有節點，皆可存取構成卷冊群組的實體卷冊。不支援非對稱式的叢集配置（某些節點能存取儲存裝置，某些則無法存取）。

圖形 1.2, “CLVM 總覽” 顯示叢集中的 CLVM 總覽。



圖形 1.2. CLVM 總覽

**注意**

您需要修改 `lvm.conf` 檔案，CLVM 才能達到叢集全域（cluster-wide）的鎖定。欲取得有關於配置 `lvm.conf` 檔案來支援叢集鎖定上的相關資訊，請查看 `lvm.conf` 這個檔案。欲取得有關於 `lvm.conf` 檔案的相關資訊，請參閱〈[附錄 B, LVM 配置檔案](#)〉。

## 1.5. 文件總覽

本文件剩下的章節包含了下列內容：

- 〈[章 2, LVM 元件](#)〉描述了構成一個 LVM 邏輯卷冊的元件。
- 〈[章 3, LVM 管理總覽](#)〉提供了您配置 LVM 邏輯卷冊所進行的基本步驟之總覽（無論您是使用 LVM 指令列介面〔CLI〕指令或是 LVM 圖形化用戶介面〔GUI〕）。
- 〈[章 4, 透過 CLI 指令來進行 LVM 管理](#)〉概述了您可透過 LVM CLI 指令來執行，並建立和維護邏輯卷冊的各別管理工作。
- 〈[章 5, LVM 配置範例](#)〉提供了各種 LVM 配置的範例。
- 〈[章 6, LVM 疑難排解](#)〉提供了針對於各種 LVM 問題所進行的疑難排解上的指示。
- 〈[章 7, 利用 LVM GUI 來進行 LVM 管理](#)〉概述了 LVM GUI 的作業。
- 〈[附錄 A, 裝置映射 \(Device Mapper\) 設備](#)〉描述了 LVM 使用來映射邏輯和實體卷冊的 Device Mapper。



- 〈[附錄 B, LVM 配置檔案](#)〉描述了 LVM 配置檔案。
- 〈[附錄 C, LVM 物件標籤 \(Object Tags\)](#)〉描述了 LVM 物件標籤和主機標籤。
- 〈[附錄 D, LVM 卷冊群組 Metadata](#)〉描述了 LVM 卷冊群組的 metadata，並包含一個 LVM 卷冊群組的 metadata 範本。

## 章 2. LVM 元件

此章節描述了 LVM 邏輯卷冊的元件。

### 2.1. 實體卷冊

一個 LVM 邏輯卷冊的基本實體儲存裝置就是一些像是分割區或是整個磁碟的區塊裝置。若要使用一個 LVM 邏輯卷冊的裝置，該裝置必須被初始化為實體卷冊（PV）。請在實體卷冊將標籤放置在靠近裝置的起始時將區塊裝置初始化。

就預設值，LVM 標籤會被放置在第二個 512 位元組的磁區中。您可透過將標籤放置在前 4 個磁區中的任何一個磁區上來將此預設值覆寫。在必要的情況下，這能讓 LVM 卷冊和這些磁區的其它用戶並存。

一個 LVM 標籤會提供實體卷冊的正確標示和裝置順序，這是因為系統啟動時，裝置的順序能夠是任意的。LVM 標籤能夠在系統重新啟動的情況下以及叢集的環境中保留。

LVM 標籤會將裝置視為是一個 LVM 實體卷冊。它包含著實體卷冊的亂數唯一識別元（random unique identifier, UUID）。它同時將區塊裝置的大小以位元組來儲存了起來，並且它會記錄 LVM metadata 被儲存在裝置上的哪裡。

LVM metadata 包含了您系統上的 LVM 卷冊群組的配置詳情。就預設值，metadata 會有個副本被保留在卷冊群組中每個實體卷冊中的所有 metadata 區域裡。LVM metadata 非常小並且會被儲存為 ASCII。

目前，LVM 允許您在各個實體卷冊上儲存 0、1 或 2 個相同的 metadata 副本。一旦您配置了實體卷冊上的 metadata 副本數量之後，您之後便無法修改該數量。第一個副本會被儲存在裝置的起始，就在標籤之後不遠的位置上。若有第二個副本的話，它便會被放置在裝置的最後位置上。若您不小心將您磁碟一開始的區域覆寫掉的話，位於裝置最後的第二個 metadata 副本能讓您將 metadata 復原。

如欲取得更多有關於 LVM metadata 以及更改 metadata 參數的相關資訊，請參閱〈[附錄 D, LVM 卷冊群組 Metadata](#)〉。

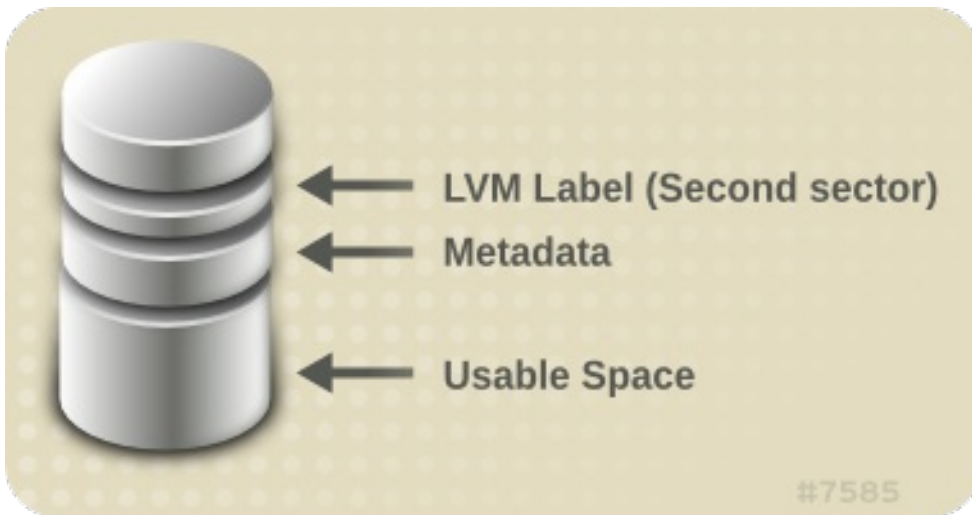
#### 2.1.1. LVM 實體卷冊配置

〈[圖形 2.1, “實體卷冊配置”](#)〉顯示了一個 LVM 實體卷冊的配置。LVM 標籤位於第二個磁區上，接著是 metadata 區域，接著便是裝置上的可用空間。



#### 注意

在 Linux kernel（和本文件）中，磁區的大小將會被視為是 512 個位元組。



圖形 2.1. 實體卷冊配置

### 2.1.2. 磁碟上的多重分割區

LVM 能讓您透過磁碟分割區來建立實體卷冊。一般我們建議您建立一個能夠完全涵蓋一個被標記為 LVM 實體卷冊的磁碟的單獨分割區，理由如下：

- 管理上的方便性

在系統中若各個真實的磁碟只出現一次的話，這將能簡化系統中的硬體追蹤（特別是當磁碟發生錯誤時）。此外，單獨磁碟上的多重實體卷冊可能會造成 kernel 在啟動時發出有關於不明分割區類型的相關警告。

- 等量磁碟效能 (Striping performance)

LVM 無法得知兩個實體卷冊是否位於相同的實體磁碟上。若您在兩個實體卷冊位於相同實體磁碟上時建立了一個等量的邏輯卷冊的話，等量磁碟可能會位於相同磁碟的不同分割區上。這將會造成效能上的降低。

雖然我們不建議，不過您可能會遇到需要將磁碟分割為各別 LVM 實體卷冊的情況。比方說，在一部有幾個磁碟的系統上，當您要將一個現有的系統遷移至 LVM 卷冊時，您可能需要將資料在分割區上進行移動。此外，若您擁有一個非常大的磁碟並且基於管理的原因而希望擁有超過一個卷冊群組的話，那麼您便需要將磁碟進行分割。若您沒有一個含有超過一個分割區的磁碟，並且這些分割區都位於相同卷冊群組中的話，當您在建立等量卷冊時，您應小心注意指定哪個分割區需要包含在邏輯卷冊中。

## 2.2. 卷冊群組

實體卷冊會被合併為卷冊群組 (VG)。這建立了一個磁碟空間的 pool，並且邏輯卷冊可從而被進行分配。

在一個卷冊群組中，可被用來進行分配的磁碟空間會被劃分為固定大小的單位稱為扇區。扇區為能夠被分配的最小空間單元。在實體卷冊中，扇區被稱為實體扇區 (physical extent)。

邏輯卷冊會被分配至與實體扇區相同大小的邏輯扇區中。因此卷冊群組中所有邏輯卷冊的扇區大小都會是相同的。卷冊群組會將邏輯扇區映射至實體扇區。

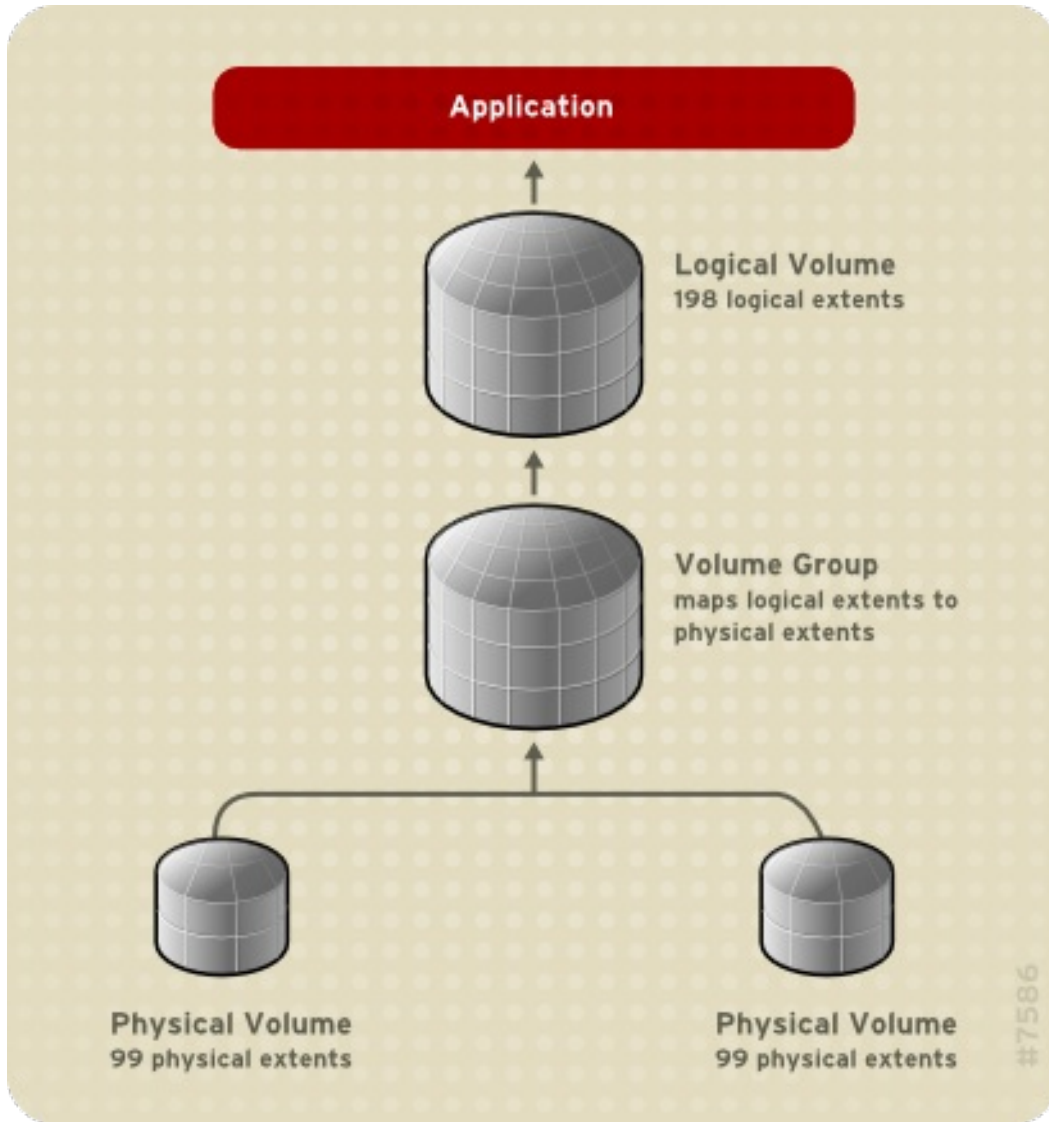
## 2.3. LVM 邏輯卷冊

在 LVM 中，邏輯卷冊會被劃分為邏輯卷冊。LVM 邏輯卷冊的類型有三種：*linear* (線性) 卷冊、*striped* (等量) 卷冊，以及 *mirrored* (鏡像) 卷冊。這些邏輯卷冊描述於下列部份中。

### 2.3.1. 線性邏輯卷冊

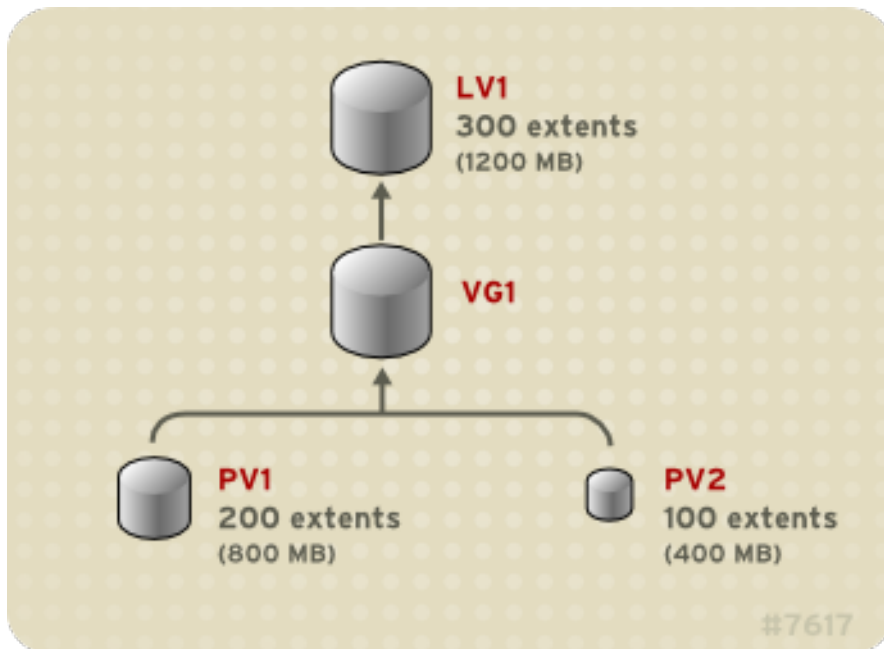
線性卷冊會將多個實體卷冊聚集為單一個邏輯卷冊。比方說，若您有兩個 60GB 的磁碟，您便可建立一個 120GB 的邏輯卷冊。實體儲存裝置會被序連在一起。

建立一個線性卷冊會將一個範圍的實體卷冊，按照順序指派至一個邏輯卷冊的區域中。如〈[圖形 2.2, “扇區映射 \(Extent Mapping\)”](#)〉中所示，邏輯扇區 1 至 99 能夠映射至一個實體卷冊，並且邏輯扇區 100 至 198 可映射至第二個實體卷冊。就應用程式的觀點來看，這是個大小為 198 個扇區的裝置。



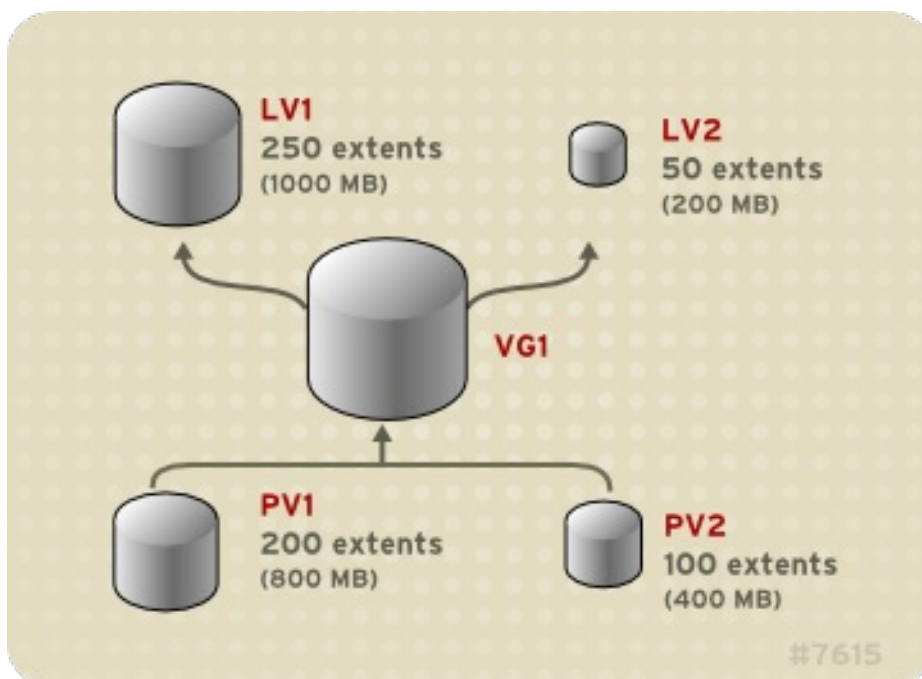
圖形 2.2. 扇區映射 (Extent Mapping)

構成邏輯卷冊的實體卷冊不需有相同大小。〈[圖形 2.3, “含有不公平的實體卷冊的線性卷冊”](#)〉顯示了卷冊群組 **VG1** 和一個大小為 4MB 的實體扇區。此卷冊群組包含了兩個實體卷冊，它們名為 **PV1** 和 **PV2**。實體卷冊會被劃分為 4MB 的單位，因為那就是扇區的大小。在此範例中，**PV1** 的大小為 100 個扇區（400MB）並且 **PV2** 的大小為 200 個扇區（800MB）。您可建立一個大小為 1 至 300 個扇區之間（4MB 至 1200MB）的線性卷冊。在此範例中，名為 **LV1** 的線性卷冊的大小為 300 個扇區。



圖形 2.3. 含有不公平的實體卷冊的線性卷冊

您可藉由實體扇區群來配置數個隨意大小的線性邏輯卷冊。〈圖形 2.4, “多重邏輯卷冊”〉顯示了和〈圖形 2.3, “含有不公平的實體卷冊的線性卷冊”〉之中相同的卷冊群組，不過在此情況下，有兩個邏輯卷冊被由卷冊群組中切割了出來：LV1，大小為 250 個扇區（1000MB）以及 LV2，大小為 50 個扇區（200MB）。



圖形 2.4. 多重邏輯卷冊

### 2.3.2. 等量邏輯卷冊

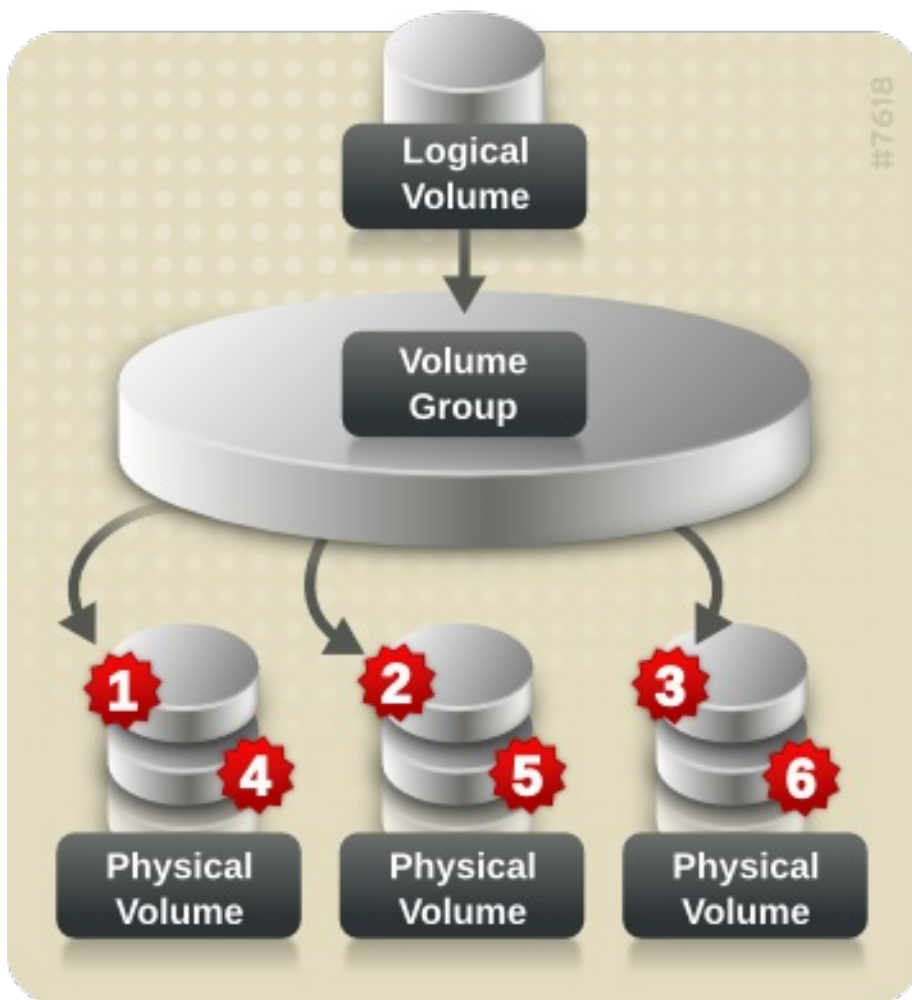
當您將資料寫至一個 LVM 邏輯卷冊時，檔案系統會將資料配置在基本實體卷冊之間。您可藉由建立一個等量的邏輯卷冊來控制資料寫至實體卷冊的方式。對於較大量的讀取和寫入循序來講，這可有效改善資料 I/O 的效率。

建立等量磁碟可藉由循環配置資源（round-robin）的方式，將資料寫入數量已預先決定的實體卷冊，以增強效能。當建立等量磁碟時，I/O 能以平行的方式來進行。在某些情況下，這可能會使等量磁碟中的各個實體卷冊皆達到近線性（near-linear）的效能。

下列描述了資料如何被等量分配在三個實體卷冊之間。在此圖形中：

- 第一個等量磁碟的資料已被寫至 PV1
- 第二個等量磁碟的資料已被寫至 PV2
- 第三個等量磁碟的資料已被寫至 PV3
- 第四個等量磁碟的資料已被寫至 PV1

在等量邏輯卷冊中，等量磁碟的大小不可超過扇區的大小。



圖形 2.5. 將資料等量分配在三個 PV 之間

等量邏輯卷冊可透過將其它的裝置序連至第一組裝置的最後，來進行延伸。然而若要延伸一個等量邏輯卷冊，構成卷冊群組的基本實體卷冊上，必須要有足夠的可用空間，才可支援等量磁碟。比方說，若您擁有一個使用了整個卷冊群組的雙向等量磁碟，那麼將一個單獨的實體卷冊新增至卷冊群組，將無法讓您延伸等量磁碟。反之，您必須新增至少兩個實體卷冊至卷冊群組中。欲取得更多有關於延伸等量卷冊的相關資訊，請參閱〈[節 4.4.12.1, “延伸等量的卷冊”](#)〉。

### 2.3.3. 鏡像邏輯卷冊

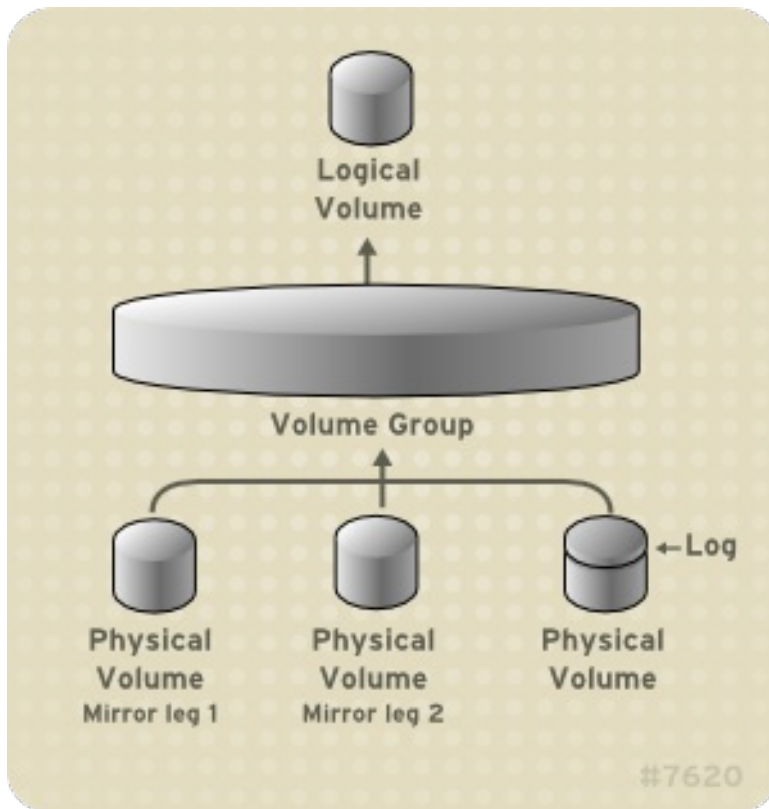


鏡像會將資料的相同副本保留在不同的裝置上。當資料被寫至一個裝置上時，它也會同時被寫至第二個裝置上，並成為該資料的鏡像。這提供了裝置發生錯誤時的保障。當鏡像的其中一個 leg 發生錯誤時，邏輯卷冊會成為一個線性卷冊並且依然能被存取。

LVM 支援鏡像卷冊。當您建立了鏡像邏輯卷冊時，LVM 會確保寫至基本實體卷冊的資料會被 mirror 至另一個實體卷冊上。透過 LVM，您可建立多重鏡像的鏡像邏輯卷冊。

LVM 鏡像會將被複製的裝置劃分為大小各為 512KB 的不同區域。LVM 會保留一個小型的日誌檔，它使用了該日誌檔來記錄哪個區域已和哪些鏡像同步化。這個日誌檔可被保留在磁碟上，如此一來它便可保留一致性並不受系統重新啟動影響，或是它亦可被保留在記憶體中。

〈[圖形 2.6, “鏡像邏輯卷冊”](#)〉顯示了被映射、含有一個鏡像的鏡像邏輯卷冊。在此配置中，日誌檔會被保留在磁碟上。



圖形 2.6. 鏡像邏輯卷冊

欲取得有關於建立或修改鏡像的相關資訊，請參閱〈[節 4.4.3, “建立鏡像卷冊”](#)〉。

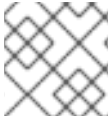
### 2.3.4. Snapshot 卷冊

LVM snapshot 提供了一項在特定一瞬間，建立某個裝置的虛擬映像檔，並且不干擾服務的運作。當針對原始裝置進行變更，並建立了 snapshot 之後，snapshot 功能會建立一份遭到變更後的部份資料之副本，如此一來它便可重建該裝置的狀態。



#### 注意

LVM snapshot 在叢集環境中的節點之間，並不受到支援。您無法在叢集化的卷冊群組中建立 snapshot 卷冊。



### 注意

LVM 鏡像邏輯卷冊不支援 LVM snapshot。

因為 snapshot 只會複製建立了 snapshot 之後受到更改的資料部份，因此 snapshot 功能只需要少量的儲存空間。比方說，對於一個極少更新的原始裝置來說，該裝置容量的 3 到 5 % 就已足夠容納 snapshot。



### 注意

檔案系統的 snapshot 副本屬於虛擬副本，而不是實際的檔案系統媒介備份。Snapshot 無法取代備份程序。

Snapshot 的大小可掌管用來儲存原始卷冊上的變更的所需空間。比方說，若您製作了一個 snapshot，並且完全地覆寫了原始的卷冊，此 snapshot 的大小必須至少要和原始卷冊的大小相同，以保存變更。您必須根據預期的變更程度來配置 snapshot。比方說，一個大部分為唯讀的卷冊的暫時性 snapshot（例如 `/usr`）之所需空間，會比擁有較多寫入次數的卷冊的長期性 snapshot 之所需空間還要少。

若 snapshot 滿出的話，該 snapshot 將會無法使用，因為它將無法再追蹤原始卷冊上的變更。您應定時監控 snapshot 的大小。Snapshot 的大小能夠完全地重設，因此若您有足夠的儲存容量，您便能增加 snapshot 卷冊的大小來避免它被 drop 掉。相反的，若您發現 snapshot 卷冊的大小比您所需的還要大，您可減少該卷冊的大小，以釋出其它邏輯卷冊所需要的空間。

當您建立了 snapshot 檔案系統時，原始裝置的完整讀取和寫入存取權限還是可被保留。若 snapshot 上有一小區塊受到變更的話，該區塊會被標記並且永遠不會被由原始卷冊中複製出去。

Snapshot 功能有幾種用途：

- 一般來講，snapshot 會使用於當您需要在邏輯卷冊上進行備份，而不影響持續進行資料更新的即時系統的情況下。
- 您可在一個 snapshot 檔案系統上執行 **fsck** 這項指令，以檢查檔案系統的整合性，並判斷原始的檔案系統是否需要進行檔案系統修復。
- 因為 snapshot 為可讀寫（read/write），因此您可透過製作一個 snapshot，並針對該 snapshot 執行測試的方式，來使用應用程式針對於生產資料進行測試，並且完全不動到真實的資料。
- 您可建立 LVM 卷冊，以搭配 Red Hat 虛擬化使用。LVM snapshot 可被使用來建立虛擬客座映像的 snapshot。這些 snapshot 可提供便利的方式來修改既有客座端，或以最少額外儲存空間來建立新客座端。欲取得在 Red Hat Virtualization 中建立以 LVM 為基礎的儲存集區之資訊，請參閱《[虛擬管理指南](#)》。

欲取得有關於建立 snapshot 卷冊上的相關資訊，請參閱〈[節 4.4.4, “建立快照卷冊（Snapshot Volumes）”](#)〉。

從 RHEL 6 發行版開始，您將可使用 **lvconvert** 指令的 **--merge** 選項來將 snapshot 合併入它原始的卷冊中。此功能的用途之一，就是當您遺失資料或檔案時，進行系統復原，或是當您需要將系統回復為先前的狀態時。當您合併了 snapshot 卷冊之後，形成的邏輯卷冊將會擁有原始卷冊的名稱、minor 數字，以及 UUID，並且合併的 snapshot 將會被移除。欲取得使用此選項上的相關資訊，請參閱〈[節 4.4.5, “合併 Snapshot 卷冊”](#)〉。



## 章 3. LVM 管理總覽

本章提供了您用來配置 LVM 邏輯卷冊的管理程序總覽。本章主要是要讓使用者理解各個包含的步驟。欲取得一般 LVM 配置程序上的特定逐步範例，請查看〈[章 5, LVM 配置範例](#)〉。

如欲取得您可用來執行 LVM 管理的 CLI 指令之描述，請查看〈[章 4, 透過 CLI 指令來進行 LVM 管理](#)〉。另外，您亦可使用 LVM GUI（描述於〈[章 7, 利用 LVM GUI 來進行 LVM 管理](#)〉中）。

### 3.1. 在叢集中建立 LVM 卷冊

若要在叢集環境下建立邏輯卷冊的話，您需使用 Clustered Logical Volume Manager (CLVM)，也就是一組 LVM 的叢集延伸。這些延伸功能可允許叢集中的電腦透過 LVM 來管理共享的儲存裝置（比方說在 SAN 上）。若要使用 CLVM，您必須在開機時啟用 High Availability 外掛程式和 Resilient Storage 外掛程式軟體，包括 `clvmd` daemon，如〈[節 1.4, “叢集邏輯卷冊管理員 \(Clustered Logical Volume Manager, CLVM\)”](#)〉中所描述。

在叢集環境中建立 LVM 邏輯卷冊和在單節點上建立 LVM 邏輯卷冊相似。LVM 指令本身以及 LVM GUI 介面並沒有不同。若要啟用您在叢集中所建立的 LVM 卷冊，該叢集設備必須要是執行中並且為 `quorate`。

您需要修改 `lvm.conf` 檔案，CLVM 才能達到叢集全域 (cluster-wide) 的鎖定。如欲取得有關於配置 `lvm.conf` 檔案來支援叢集鎖定，請查看 `lvm.conf` 這個檔案。欲取得有關於 `lvm.conf` 檔案的相關資訊，請參閱〈[附錄 B, LVM 配置檔案](#)〉。

就預設值，透過 CLVM 在共享儲存裝置上建立的邏輯卷冊，對於可存取該共享儲存裝置的所有電腦來說，都是可見的。您亦可建立邏輯卷冊，並且儲存裝置只可被叢集中的一個節點看見。您亦可將邏輯卷冊的狀態，由本機卷冊更改為叢集卷冊。欲取得相關資訊，請參閱〈[節 4.3.2, “在叢集中建立卷冊群組”](#)〉和〈[節 4.3.7, “更改卷冊群組的參數”](#)〉。



#### 警告

當您在 CLVM 位於共享儲存裝置上時建立卷冊群組時，您必須確認叢集中的所有節點，皆可存取構成卷冊群組的實體卷冊。不支援非對稱式的叢集配置（某些節點能存取儲存裝置，某些則無法存取）。

欲取得有關於如何安裝 High Availability 外掛程式，和設定叢集架構的相關資訊，請參閱【[叢集管理](#)】。

欲取得在叢集中建立鏡像邏輯卷冊的相關範例，請參閱〈[節 5.5, “在叢集中建立鏡像 LVM 邏輯卷冊”](#)〉。

### 3.2. 邏輯卷冊建立總覽

下列為建立 LVM 邏輯卷冊所需步驟的概要。

1. 初始化您將會使用來作為實體卷冊的 LVM 卷冊分割區（這會將它們標記下來）。
2. 建立卷冊群組。
3. 建立邏輯卷冊

建立了邏輯卷冊之後，您便可建立並掛載檔案系統。本文件中的範例使用了 GFS2 檔案系統。



## 注意

儘管 GFS2 檔案系統可實做於獨立的系統中，或是實做為 Red Hat Enterprise Linux 6 發行版叢集配置的一部分，Red Hat 並不支援使用 GFS2 作為單節點的檔案系統。Red Hat 將會持續支援單節點的 GFS2 檔案系統，以進行叢集檔案系統的 snapshot 掛載（比方說作為備份用途）。

1. 藉由 **mkfs.gfs2** 指令來在邏輯卷冊上建立一個 GFS2 檔案系統。
2. 透過 **mkdir** 指令來建立新的掛載點。在叢集的系統中，請在叢集中所有的節點上建立掛載點。
3. 掛載檔案系統。您可能會希望為系統中的各個節點附加一行行列至 **fstab** 檔案中。

此外，您可使用 LVM GUI 來建立與掛載 GFS2 檔案系統。

LVM 的建立基於各別的機器，因為 LVM 設定資訊的儲存位置位於實體卷冊上而不是卷冊所被建立於的機器上。使用該儲存裝置的伺服器含有本地副本，不過能藉由實體卷冊上所含有的內容重新建立。若 LVM 版本相容的話，您可將實體卷冊連至一個不同的伺服器上。

### 3.3. 在邏輯卷冊上遞增檔案系統

若要在邏輯卷冊上遞增檔案系統的話，請執行下列步驟：

1. 建立一個新的實體卷冊
2. 將包含著邏輯卷冊以及您所希望遞增的檔案系統的卷冊群組延伸來包含新的實體卷冊。
3. 將邏輯卷冊延伸來包含新的實體卷冊。
4. 遞增檔案系統。

若您的卷冊群組中含有足夠的未使用空間，您可使用這個空間來延伸邏輯卷冊而不用進行步驟 1 和 2。

### 3.4. 邏輯卷冊備份

Metadata 的備份和 archive 會在每個卷冊群組和邏輯卷冊配置遭到變更時自動被建立，除非在 **lvm.conf** 檔案中停用。就預設值，metadata 的備份儲存於 **/etc/lvm/backup** 檔案中，並且 metadata archive 則儲存在 **/etc/lvm/archive** 檔案中。Metadata archive 儲存在 **/etc/lvm/archive** 檔案中的保留時間長短，和 archive 檔案的數量取決於您在 **lvm.conf** 檔案中所設的參數。每日的系統備份，皆應包含著 **/etc/lvm** 目錄的內容於備份中。

請注意，metadata 備份不會備份包含在邏輯卷冊中的用戶與系統資料。

您可透過 **vgcfgbackup** 指令來手動式地將 metadata 備份至 **/etc/lvm/backup** 檔案。您可利用 **vgcfgrestore** 指令來恢復 metadata。**vgcfgbackup** 和 **vgcfgrestore** 指令的描述位於〈[節 4.3.12, “備份卷冊群組的 Metadata”](#)〉中。

### 3.5. 記錄

所有輸出的訊息都會通過一個記錄模組並含有針對於下列的各別記錄層級選項：

- 標準輸出/錯誤
- syslog

- 日誌檔案
- 外部記錄功能

記錄層可於 `/etc/lvm/lvm.conf` 檔案中設置（描述於〈[附錄 B, LVM 配置檔案](#)〉中）。

## 章 4. 透過 CLI 指令來進行 LVM 管理

本章節包含了透過 LVM 指令列介面（CLI）來建立與維護邏輯卷冊的各別管理作業的概述。



### 注意

若您要建立或是修改一個叢集環境下的 LVM 卷冊的話，您必須確認您有執行 **clvmd** daemon。如欲取得更多相關資訊，請參閱〈節 3.1, “在叢集中建立 LVM 卷冊”〉。

### 4.1. 使用 CLI 指令

所有 LVM CLI 指令都有幾個通用的功能。

當在指令列引數中需要使用到大小時，單位可被明確地指定。若您不指定單位的話，那麼預設值便會被使用，一般會是 KB 或是 MB。LVM CLI 指令不接受分數。

當在指令列引數中指定單位時，LVM 並不區分大小寫；比方說，指定 M 或 m 都是相等的，在此情況下 2 進位（1024 的倍數）會被使用。不過，當在指令中指定了 **--units** 引數時，小寫表示單位為 1024 的倍數，而大寫則表示單位為 1000 的倍數。

當指令使用卷冊群組或是邏輯卷冊名稱為引數時，完整路徑名稱為非必要的。一個稱為 **vg0** 的卷冊群組中的 **lv010** 邏輯卷冊可被指定為 **vg0/lv010**。當需要一系列必要的卷冊群組，不過卻被保留為空白時，該卷冊群組中的所有邏輯卷冊就會被帶入。比方說，**lvdisplay vg0** 指令將會顯示 **vg0** 卷冊群組中所有的邏輯卷冊。

所有的 LVM 指令都接受 **-v** 引數，您可多重輸入該引數來增加輸出的詳細度。比方說，下列範例顯示了 **lvcreate** 指令的預設輸出。

```
# lvcreate -L 50MB new_vg
Rounding up size to full physical extent 52.00 MB
Logical volume "lv010" created
```

下列指令顯示了 **lvcreate** 指令以及 **-v** 引數的輸出。

```
# 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 lv010
Creating volume group backup "/etc/lvm/backup/new_vg" (seqno 5).
Found volume group "new_vg"
Creating new_vg-lv010
Loading new_vg-lv010 table
Resuming new_vg-lv010 (253:2)
Clearing start of logical volume "lv010"
Creating volume group backup "/etc/lvm/backup/new_vg" (seqno 5).
Logical volume "lv010" created
```

您也能使用 **-vv**、**-vvv** 或是 **-vvvv** 引數來顯示更加詳細的指令執行資訊。目前，**-vvvv** 引數提供了大量的資訊。下列範例僅顯示了 **lvcreate** 指令以及 **-vvvv** 引數的輸出的前幾個行列。

```
# 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
...
```

您可透過使用指令的 **--help** 引數來顯示任何 LVM CLI 指令的協助畫面。

```
# commandname --help
```

若要顯示某項指令的 man page，請執行 **man** 指令：

```
# man commandname
```

**man lvm** 指令提供了有關於 LVM 的一般線上資訊。

所有的 LVM 物件皆會被藉由 UUID 內部參照，當您建立物件時，UUID 將會被分配。當您移除了一個稱為 **/dev/sdf** 的實體卷冊（卷冊群組的一部分），然後再將它放回去時，卻發現它已成為了 **/dev/sdk** 時，這將會非常有幫助。LVM 還是有辦法找到實體卷冊，因為它會藉由實體卷冊的 UUID 來進行辨識，而非藉由實體卷冊的裝置名稱。欲取得在建立實體卷冊時，指定實體卷冊 UUID 上的相關資訊，請參閱〈[節 6.4, “復原實體卷冊的 Metadata”](#)〉。

## 4.2. 實體卷冊管理

此部份描述了進行各種實體卷冊管理的指令。

### 4.2.1. 建立實體卷冊

下列小部分描述了幾項使用來建立實體卷冊的指令。

#### 4.2.1.1. 設定分割區類型

若您要使用整個磁碟裝置來作為您的實體卷冊，該磁碟上必須沒有分割表。對於 DOS 磁碟分割區來講，分割區 id 應被透過使用 **fdisk** 或是 **cfdisk** 指令來設為 0x8e。當使用整個磁碟裝置時，只有分割表必須被清除掉，如此一來便能有效地清除該磁碟上的所有資料。您可藉由使用下列指令來將第一個磁區化零以便移除現有的分割表：

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

#### 4.2.1.2. 初始化實體卷冊

您可使用 **pvccreate** 指令來初始化一個將被用來作為實體卷冊的區塊裝置。初始化和格式化檔案系統是相似的。

下列指令初始化了 **/dev/sdd**、**/dev/sde** 和 **/dev/sdf** 為 LVM 實體卷冊，以事後作為 LVM 邏輯卷冊一部份使用。

■

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

若要初始化分割區而不是整個磁碟：請在分割區上執行 **pvcreate** 指令。下列範例將 **/dev/hdb1** 作為 LVM 實體卷冊初始化並作為事後 LVM 邏輯卷冊的一部分來使用。

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

#### 4.2.1.3. 掃描區塊裝置

您可透過 **lvmdiskscan** 指令來掃描可用來作為實體卷冊的區塊裝置，如下列範例所示。

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

#### 4.2.2. 顯示實體卷冊

您可使用三項指令來顯示 LVM 實體卷冊的內容：**pvs**、**pvdisplay** 以及 **pvscan**。

**pvs** 指令會將實體卷冊資訊以可配置的格式顯示（一個實體卷冊一行）。**pvs** 指令提供了許多格式控制，並且對於進行 script 相當有幫助。如欲取得使用 **pvs** 指令來自訂化您的輸出的相關資訊，請參閱〈[節 4.8, “LVM 的自訂化回報”](#)〉。

**pvdisplay** 指令會為各個實體卷冊提供詳細的多行輸出。它會以一個固定的格式來顯示實體的內容（大小、扇區、卷冊群組等等）。

下列範例顯示了針對於單獨實體卷冊所執行的 **pvdisplay** 指令的輸出。

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

**pvscan** 指令會掃描系統中所有受支援的 LVM 區塊裝置來作為實體卷冊。

下列指令顯示了所有發現的實體裝置：

```
# 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]
```

您可在 **lvm.conf** 中定義一個過濾器，如此一來這項指令將會避免掃描特定的實體卷冊。欲取得如何使用過濾器來控制哪些裝置需被掃描的相關資訊，請查看 [〈節 4.5, “透過過濾器來控制 LVM 裝置掃描 \(LVM Device Scans\)”](#)。

### 4.2.3. 避免配置於實體卷冊上

您可透過使用 **pvchange** 指令來避免將實體扇區分配在一個或更多實體卷冊的可用空間上。若有磁碟錯誤或是若您要將實體卷冊移除的話，這將會是必要的。

下列指令會使實體扇區無法分配在 **/dev/sdk1** 上。

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

您也可使用 **pvchange** 指令的 **-xy** 引數來允許分配。

### 4.2.4. 重設實體卷冊的大小

若您基於任何理由需要更改一個區塊裝置的大小的話，請使用 **pvresize** 指令來將 LVM 更改為新的大小。您可在 LVM 使用實體卷冊時執行這項指令。

### 4.2.5. 移除實體卷冊

若 LVM 已不再需要使用某個裝置，您可透過 **pvremove** 指令來將 LVM 標籤移除掉。執行 **pvremove** 指令可將一個空的實體卷冊上的 LVM metadata 零化掉。



若您希望移除的實體卷冊目前屬於卷冊群組的一部分，您便必須透過使用 **vgreduce** 指令來將它由卷冊群組中移除（如〈[節 4.3.6, “由卷冊群組中移除實體卷冊”](#)〉中所述）。

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

## 4.3. 卷冊群組管理

此部份描述了執行各種卷冊群組管理的指令。

### 4.3.1. 建立卷冊群組

若要由一個或更多個實體卷冊建立卷冊群組，請使用 **vgcreate** 指令。**vgcreate** 這項指令會藉由名稱來建立一個新的卷冊群組，並新增至少一個實體卷冊至此卷冊群組當中。

下列指令會建立一個名為 **vg1** 的卷冊群組，它將會包含 **/dev/sdd1** 和 **/dev/sde1** 這兩個實體卷冊。

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

當實體卷冊被使用來建立卷冊群組時，就預設值它的磁碟空間會被平分為幾個 4MB 扇區。這個扇區為邏輯卷冊大小可被遞增或遞減的最小值。大量扇區將不會對於 I/O 邏輯卷冊的效能造成任何影響。

若是預設的扇區大小不適合，您可透過使用 **vgcreate** 指令的 **-s** 選項來指定扇區大小。您可透過使用 **vgcreate** 指令的 **-p** 和 **-l** 引數來限制卷冊群組所能擁有的實體或邏輯卷冊數量。

就預設值，卷冊群組會針對於一般常規（例如不將平行磁帶放置在相同實體卷冊上）來分配實體扇區。這是 **normal** 分配政策。您可使用 **vgcreate** 指令的 **--alloc** 引數來指定 **contiguous**、**anywhere** 或 **cling** 的分配政策。

**contiguous** 政策會需要新的扇區和既有的扇區相鄰。若有足夠的可用扇區來滿足分配請求，不過 **normal** 分配政策不使用它們的話，**anywhere** 分配政策則會使用它們，儘管將兩個磁帶放置在相同的實體卷冊上會降低效能。**cling** 政策會將新的扇區放置在與邏輯卷冊相同磁帶中的既有扇區相同的實體卷冊上。這些政策可透過使用 **vgchange** 指令來進行變更。

欲取得合併使用 **cling** 政策與 LVM 標籤來指定在延伸 LVM 卷冊時，要使用哪個額外實體卷冊上的相關資訊，請參閱〈[節 4.4.12.3, “以 cling 分配政策延伸邏輯卷冊”](#)〉。

一般來講，**normal** 以外的分配政策只有在特殊的情況下（當您需要指定不尋常或是非標準扇區分配時），才需要使用到。

LVM 卷冊群組和基本的邏輯卷冊包含在位於 **/dev** 目錄中的裝置特殊檔案目錄樹中，格式如下：

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

比方說，若您建立了兩個卷冊群組 **myvg1** 和 **myvg2**，各個都含有三個稱為 **lv01**、**lv02** 和 **lv03** 的邏輯卷冊，這將會建立六個裝置特殊檔案：

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



LVM 的裝置大小在 64 位元 CPU 上最大值為 8 百萬兆位元組（Exabytes）。

### 4.3.2. 在叢集中建立卷冊群組

在叢集環境中，您可透過 **vgcreate** 指令來建立卷冊群組，就如同您在單獨的節點上建立它們一樣。

就預設值，透過 CLVM 在共享儲存裝置上所建立的卷冊群組能讓所有可存取該共享儲存裝置的電腦看見。不過您也可透過使用 **vgcreate** 指令的 **-c n** 選項來建立一個本機、只有叢集中的一個節點可看見的卷冊群組。

當在叢集環境中執行下列指令時，它會在指令被執行的節點上，建立一個本機卷冊群組。這項指令會建立一個名為 **vg1** 並且包含著實體卷冊 **/dev/sdd1** 和 **/dev/sde1** 的本機卷冊。

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

您可透過使用 **vgchange** 指令的 **-c** 選項來將現有的卷冊群組，更改為本機或是叢集卷冊群組，如〈[節 4.3.7, “更改卷冊群組的參數”](#)〉中所描述。

您可透過 **vgs** 指令來檢查一個現有的卷冊群組是否是個叢集卷冊群組。若此卷冊已位於叢集中，這項指令便會顯示 **c** 這個屬性。下列指令顯示了卷冊群組 **VolGroup00** 和 **testvg1** 的屬性。在此範例中，**VolGroup00** 尚未加入叢集中，而 **testvg1** 則已加入了叢集，如 **Attr** 標題下的 **c** 屬性所示。

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

如欲取得更多有關於 **vgs** 指令的相關資訊，請參閱 [節 4.3.4, “顯示卷冊群組”](#) [節 4.8, “LVM 的自訂化回報”](#) 和 **vgs** 的 man page。

### 4.3.3. 新增實體卷冊至卷冊群組中

若要新增額外的實體卷冊至現有的卷冊群組，請使用 **vgextend** 指令。**vgextend** 這項指令會藉由新增一個或更多個可用的實體卷冊來遞增卷冊群組的容量。

下列指令會將 **/dev/sdf1** 實體卷冊新增至卷冊群組 **vg1** 中。

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

### 4.3.4. 顯示卷冊群組

您可使用兩項指令來顯示 LVM 卷冊群組的內容：**vgs** 和 **vgdisplay**。

**vgscan** 指令能掃描卷冊群組的所有磁碟、重建 LVM 快取檔案，以及顯示卷冊群組。欲取得 **vgscan** 指令上的相關資訊，請參閱〈[節 4.3.5, “掃描磁碟來找尋卷冊群組以便建立快取檔案”](#)〉。

**vgs** 指令會以可配置的格式來提供卷冊群組資訊（一個卷冊群組一行）。**vgs** 指令提供了許多格式控制，並且對於進行 script 相當有幫助。如欲取得有關於使用 **vgs** 指令來自訂化您的輸出的相關資訊，請參閱〈[節 4.8, “LVM 的自訂化回報”](#)〉。

**vgdisplay** 指令會以固定的格式來顯示卷冊群組內容（例如大小、扇區、實體卷冊數量等等）。下列範例顯示了 **vgdisplay** 指令針對於 **new\_vg** 卷冊群組的輸出。若您不指定卷冊群組的話，所有現有的卷冊群組皆會顯示。

```
# 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-OpM0-0118-nlw0-wwqd-fD5D32
```

#### 4.3.5. 掃描磁碟來找尋卷冊群組以便建立快取檔案

**vgscan** 指令會掃描系統中所有受支援的磁碟裝置並尋找 LVM 實體卷冊和卷冊群組。這會將 LVM 快取建立在 `/etc/lvm/cache/.cache` 檔案中，該檔案含有一列現有的 LVM 裝置清單。

LVM 會在系統啟動時，自動地執行 **vgscan** 指令，並且也會在進行 LVM 作業時的其它時候（例如當您執行一項 **vgcreate** 指令，或是當 LVM 偵測到不一致的狀況時）執行。



#### 注意

當您更改您的硬體設定時，您可能需要手動式地執行 **vgscan** 指令，這會使得系統能看見系統開機時所不存在的新裝置。這可能會是必要的，比方說當您新增磁碟至一部 SAN 上的系統，或是當您熱插拔一個被標記為實體卷冊的新磁碟時。

您可在 `lvm.conf` 檔案中定義一個過濾器，以限制掃描避免特定裝置。欲取得有關於使用過濾器來控制欲掃描之裝置的相關資訊，請參閱〈[節 4.5, “透過過濾器來控制 LVM 裝置掃描 \(LVM Device Scans\)”](#)〉。

下列範例顯示了一項 **vgscan** 指令的輸出。

```
# 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. 由卷冊群組中移除實體卷冊

若要由卷冊群組中移除未使用的實體卷冊，請使用 **vgreduce** 指令。**vgreduce** 指令會藉由移除一個或更多個空的實體卷冊來縮減卷冊群組的容量。這會釋放出需要被使用於不同卷冊群組中或是由系統中移除的實體卷冊。

在由一個卷冊群組中移除實體卷冊之前，您可藉由使用 **pvdisk** 指令來確認實體卷冊並未被任何邏輯卷冊使用中。

```
# pvdisk /dev/hda1

-- Physical volume ---
PV Name           /dev/hda1
VG Name           myvg
PV Size           1.95 GB / NOT usable 4 MB [LVM: 122 KB]
PV#               1
PV Status          available
Allocatable        yes (but full)
Cur LV           1
PE Size (KByte)    4096
Total PE           499
Free PE            0
Allocated PE       499
PV UUID            Sd44tK-9IRw-SrMC-M0kn-76iP-iftz-0VSen7
```

若實體卷冊依然使用中，您便必須透過使用 **pvmove** 指令來將資料遷移至另一個實體卷冊上。然後使用 **vgreduce** 指令來移除掉實體卷冊：

下列指令會將實體卷冊 **/dev/hda1** 由 **my\_volume\_group** 卷冊群組中移除。

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

#### 4.3.7. 更改卷冊群組的參數

**vgchange** 指令可被使用來停用和啓用卷冊群組，如〈節 4.3.8, “啓用和停用卷冊群組”〉中所示。您亦可使用這項指令，來為既有的卷冊群組更改數個卷冊群組參數。

下列指令將會把卷冊群組 **vg00** 的最大邏輯卷冊數量更改為 128。

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

如欲取得 **vgchange** 指令所能更改的卷冊群組參數的相關資訊，請參閱 **vgchange(8)** man page。

#### 4.3.8. 啓用和停用卷冊群組

當您建立卷冊群組時，就預設值它會是被啓用的。這代表在該群組中的邏輯卷冊可被存取並且可能會有變更。

在幾種情況下您將需要停用卷冊群組，並使 kernel 無法偵測到該卷冊群組。若要停用或啓用卷冊群組，請使用 **vgchange** 指令的 **-a (--available)** 引數。

下列範例將會停用卷冊群組 **my\_volume\_group**。

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

若叢集鎖定被啓用的話，請附加「e」來將卷冊群組啓用或停用於一個專屬的節點上，或是附加「l」來啓用或停用本機節點上的卷冊群組。只有單獨主機 snapshot 的邏輯卷冊總是會特別地被啓用，因為它們一次只能使用於一個節點上。

您可透過 **lvchange** 指令來停用各別的邏輯卷冊，如〈[節 4.4.8, “更改邏輯卷冊群組的參數”](#)〉中所示。欲取得有關於在叢集中的各別節點上，啟用邏輯卷冊的相關資訊，請參閱〈[節 4.7, “在叢集中啟用各別節點上的邏輯卷冊”](#)〉。

### 4.3.9. 移除卷冊群組

若要移除不包含邏輯卷冊的卷冊群組，請使用 **vgremove** 指令。

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

### 4.3.10. 分割卷冊群組

若要分割卷冊群組的實體卷冊並建立新的卷冊群組，請使用 **vgsplit** 指令。

邏輯卷冊無法在邏輯群組之間分割。所有現有的邏輯卷冊都必須整個位於實體卷冊上並形成舊的或新的卷冊群組。不過若有必要的話，您可使用 **pvmove** 指令來強制分割。

下列範例由原本的 **bigvg** 卷冊群組分割出了新的卷冊群組 **smallvg**。

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

### 4.3.11. 結合卷冊群組

若要將兩個卷冊群組結合為一個單獨的卷冊群組，請使用 **vgmerge** 指令。您可將一個未啟用的「來源」卷冊和一個啟用或未啟用的「目標」卷冊結合在一起，不過卷冊的實體扇區大小必須相等並且概括了這兩個卷冊群組的實體和邏輯卷冊必須符合目標卷冊群組的限制。

下列指令會將未啟用的卷冊群組 **my\_vg** 合併入啟用，或未啟用的卷冊群組 **databases** 中，並提供詳細的 runtime 資訊。

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

### 4.3.12. 備份卷冊群組的 Metadata

Metadata 備份和 archive 會在每個卷冊群組和邏輯卷冊配置遭到變更時被自動地建立，除非您在 **lvm.conf** 檔案中將它停用。就預設值，metadata 備份儲存在 **/etc/lvm/backup** 檔案中，並且 metadata archive 則儲存在 **/etc/lvm/archives** 檔案中。您可透過 **vgcfgbackup** 指令來手動式地將 metadata 備份至 **/etc/lvm/backup** 檔案中。

**vgcfgrestore** 指令會將卷冊群組的 metadata 由 archive 復原至卷冊群組中所有的實體卷冊中。

欲取得使用 **vgcfgrestore** 指令來復原實體卷冊 metadata 上的相關範例，請參閱〈[節 6.4, “復原實體卷冊的 Metadata”](#)〉。

### 4.3.13. 為卷冊群組重新命名

使用 **vgrename** 指令來為現有的卷冊群組重新命名。

下列兩項指令皆可將現有的卷冊群組重新由 **vg02** 命名為 **my\_volume\_group**

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

```
# vgrename vg02 my_volume_group
```

#### 4.3.14. 將卷冊群組移至另一部系統

您可將整個 LVM 卷冊群組移至另一部系統上。我們建議您使用 **vgexport** 和 **vgimport** 指令來這麼作。

**vgexport** 指令會使得系統無法存取一個未啓用的卷冊群組，這能讓您將它的實體卷冊拆卸。**vgimport** 指令會使得卷冊群組在由 **vgexport** 指令使其停用後能再次地被系統存取。

若要將卷冊群組由一部系統移至另一部系統，請執行下列步驟：

1. 請確認沒有用戶正在存取卷冊群組中啓用中卷冊上的檔案，然後再將邏輯卷冊卸載。
2. 請使用 **vgchange** 指令的 **-a n** 引數來將卷冊群組標記為停用，這可避免在卷冊群組上再有任何活動進行。
3. 請使用 **vgexport** 指令來匯出卷冊群組。這可避免它被您要從之移除的系統存取它。

當您匯出了卷冊群組後，當您執行 **pvscan** 指令時，實體卷冊便會顯示在一個已匯出的卷冊群組中。

```
# 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]
...
```

當系統關閉後，您便可將構成卷冊群組的磁碟拔除然後將它們連接至新的系統上。

4. 當磁碟被插入新的系統上時，請使用 **vgimport** 指令來匯入卷冊群組，並使新的系統能夠存取它。
5. 請使用 **vgchange** 指令的 **-a y** 引數來啓用卷冊群組。
6. 掛載檔案系統來使它能被使用。

#### 4.3.15. 重新建立一個卷冊群組目錄

若要重新建立卷冊群組目錄以及邏輯卷冊特殊檔案，請使用 **vgmknodes** 指令。這項指令會檢查啓用邏輯卷冊所需的 **/dev** 目錄中的 LVM2 特殊檔案。它會建立任何遺失的特殊檔案，並移除未使用的特殊檔案。

您可藉由指定 **vgscan** 指令的 **mknodes** 引數，來將 **vgmknodes** 指令合併入 **vgscan** 指令。

### 4.4. 邏輯卷冊管理

此部份描述了執行可進行各種邏輯卷冊管理的各項指令。

#### 4.4.1. 建立線性邏輯卷冊

若要建立邏輯卷冊，請使用 **lvcreate** 指令。若您不為邏輯卷冊指定一組名稱的話，預設的 **lvol#** 就會被使用，而 **#** 代表邏輯卷冊的內部號碼。

當您建立邏輯卷冊時，邏輯卷冊會由一個使用實體卷冊上可用扇區所組成的卷冊群組來形成。一般來講，邏輯卷冊會使用掉基本實體卷冊上的所有可用空間。修改邏輯卷冊可空出和重新分配實體卷冊中的空間。

由 RHEL 6.3 發行版起，您可使用 LVM 來建立、顯示、重新命名、使用，和移除 RAID 邏輯卷冊。欲取得 RAID 邏輯卷冊上的相關資訊，請參閱 [節 4.4.13, “RAID 邏輯卷冊”](#)。

下列指令將會在 **vg1** 卷冊群組中，建立大小為 10 GB 的邏輯卷冊。

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

下列指令將會在 **testvg** 卷冊群組中，建立一個 1,500 MB、名為 **testlv** 的線性邏輯卷冊，並建立 **/dev/testvg/testlv** 這個區塊裝置。

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

下列指令會由 **vg0** 卷冊群組中的可用扇區，建立一個 50 GB、名為 **gfslv** 的邏輯卷冊。

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

您可使用 **lvcreate** 指令的 **-l** 引數來指定扇區中邏輯卷冊的大小。您亦可使用此引數來指定使用於邏輯卷冊的卷冊群組比例。下列指令將會建立一個稱為 **mylv** 的邏輯卷冊，它使用了卷冊群組 **testvol** 中 60% 的總空間。

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

您亦可使用 **lvcreate** 指令的 **-l** 引數來指定卷冊群組中作為邏輯卷冊的剩下空間的比例。下列指令將會建立一個稱為 **yourlv** 的邏輯卷冊，它將會使用卷冊群組 **testvol** 中所有未分配的空間。

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

您可使用 **lvcreate** 指令的 **-l** 引數，來建立一個使用了整個卷冊群組的邏輯卷冊。還有另一個方式可使用來建立使用整個卷冊群組的邏輯卷冊，那即是使用 **vgdisplay** 指令來找尋「Total PE」大小，然後使用這些結果來作為 **lvcreate** 指令的輸入。

下列指令將會建立一個稱為 **mylv** 的邏輯卷冊，該邏輯卷冊會將稱為 **testvg** 的卷冊群組填滿。

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

若實體卷冊需要被移除的話，使用來建立邏輯卷冊的基本實體卷冊就相當重要，因此當您建立邏輯卷冊時，您可能需要考慮到這個可能性。欲取得有關於由某個卷冊群組，移除實體卷冊上的相關資訊，請參閱 [〈節 4.3.6, “由卷冊群組中移除實體卷冊”〉](#)。

若要建立一個由卷冊群組中的實體卷冊分配的邏輯卷冊，請在 **lvcreate** 指令列後方指定實體卷冊或卷冊。下列指令將會在 **testvg** 卷冊群組中建立一個由實體卷冊 **/dev/sdg1** 分配，並稱為 **testlv** 的邏輯卷冊。

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



您可指定實體卷冊的哪些扇區可用來作為邏輯卷冊。下列範例透過了卷冊群組 **testvg** 中的實體卷冊 **/dev/sda1** 中的扇區 0 至 24，以及實體卷冊 **/dev/sdb1** 中的扇區 50 至 124 建立了一個線性邏輯卷冊。

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

下列範例由 **/dev/sda1** 實體卷冊的扇區 0 至 25 建立了一個線性邏輯卷冊，然後在扇區 100 繼續編排邏輯卷冊。

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

設置邏輯卷冊的扇區如何被分配的預設政策為 **inherit**，並且所套用的政策和卷冊群組所使用的相同。這些政策可透過使用 **lvchange** 指令來更改。如欲取得分配政策上的相關資訊，請參閱〈[節 4.3.1, “建立卷冊群組”](#)〉。

#### 4.4.2. 建立等量卷冊

針對於大量的連續讀取和寫入，建立等量的邏輯卷冊可改善資料 I/O 的效率。欲取得更多有關等量卷冊的相關資訊，請參閱〈[節 2.3.2, “等量邏輯卷冊”](#)〉。

當您建立等量邏輯卷冊時，您可透過 **lvcreate** 指令的 **-i** 引數來指定磁條的數量。這可決定邏輯卷冊需要被分散至多少實體卷冊上。磁條的數量不可大於卷冊群組中的實體卷冊數量（除非使用了 **--alloc anywhere** 引數）。

若構成等量邏輯卷冊的基本實體裝置的大小不同的話，等量卷冊的最大大小便會取決於最小的基本裝置。比方說，在一個 two-legged 的磁條中，最大大小將會等於較小裝置大小的兩倍。在一個 three-legged 的磁條中，最大大小則等於最小裝置大小的三倍。

下列指令將會在兩個實體卷冊上建立等量邏輯卷冊，並且磁條為 64KB。邏輯卷冊大小為 50 GB、名為 **gfslv**，並由卷冊群組 **vg0** 中所分割出。

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

和線性卷冊相同，您可指定您將會使用於磁條的實體卷冊扇區。下列指令將會在兩個實體卷冊上，建立一個大小為 100 扇區的等量卷冊，名為 **stripelv**，並且位於卷冊群組 **testvg** 中。該磁條會使用 **/dev/sda1** 的磁區 0-49 以及 **/dev/sdb1** 的磁區 50-99。

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

#### 4.4.3. 建立鏡像卷冊



##### 注意

由 RHEL 6.3 發行版起，LVM 支援了 RAID4/5/6 以及新的鏡像實作。欲取得新實作上的相關資訊，請參閱 [節 4.4.13, “RAID 邏輯卷冊”](#)。

## 注意

在叢集中建立鏡像 LVM 邏輯卷冊，需使用與在單獨節點上建立鏡像 LVM 邏輯卷冊時的相同指令與程序。然而，若要在叢集中建立鏡像 LVM 卷冊，叢集與叢集鏡像基礎結構必須要運作，叢集必須要 quorate，並且 **lvm.conf** 檔案中的鎖定類型必須要正確設置，以啓用叢集鎖定。欲取得在叢集中建立鏡像卷冊的範例，請參閱〈[節 5.5, “在叢集中建立鏡像 LVM 邏輯卷冊”](#)〉。

若嘗試快速地由叢集中的多重節點，執行多重 LVM 鏡像建立與轉換指令，可能會造成這些指令成為待處理狀態。這可能會造成某些請求的作業因逾時而失效。若要避免此問題，建議您由叢集中的一個節點上，執行叢集鏡像建立指令。

當您要建立鏡像卷冊時，您需要以 **lvcreate** 指令的 **-m** 引數來指定複製的資料數量。指定 **-m1** 的話會建立一個鏡像，並產生出兩份檔案系統：一個 linear 邏輯卷冊和一個副本。相同地，指定了 **-m2** 的話會建立兩個鏡像，並產生出三份檔案系統。

下列指令會建立一個單鏡像的鏡像邏輯卷冊。卷冊大小為 50 GB，名為 **mirrorlv** 並由卷冊群組 **vg0** 中所分割出：

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

LVM 鏡像會將被複製入區域中的裝置切割，預設大小為 512KB。您可使用 **lvcreate** 指令的 **-R** 引數來將區域大小指定為 MB。您亦可藉由編輯 **lvm.conf** 檔案中的 **mirror\_region\_size** 設定來更改預設的區域大小。

## 注意

基於叢集基礎結構中的限制，大於 1.5TB 的叢集鏡像無法以預設大小 512KB 建立。需要大型鏡像的使用者，應增加預設的區域大小。若沒增加區域大小將會造成 LVM 建立程序停滯，並且同時可能也會使其它 LVM 指令停滯。

當為鏡像指定大小超過 1.5TB 的區域時，您可以 TB 作為您鏡像大小的單位，將其四捨五入為下個 2 的次方，並使用該數字作為 **lvcreate** 指令的 **-R** 引數。比方說，若您的鏡像大小為 1.5TB，您可指定 **-R 2**。若您的鏡像大小為 3TB，您可指定 **-R 4**。當鏡像大小為 5TB 時，您可指定 **-R 8**。

下列指令將建立一個區域大小為 2MB 的鏡像邏輯卷冊：

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

當鏡像被建立後，鏡像區域會被同步化。對於大型的鏡像元件來說，同步程序可能會花上許多時間。由 RHEL 6.3 發行版起，當您要建立一個無須再生的新鏡像時，您可指定 **--nosync** 引數來顯示來自於第一個裝置的初始同步化是非必要的。

LVM 含有一個小型的日誌，它會使用該日誌來記錄哪個區域和哪個鏡像已同步化。就預設值，該日誌會被存放在磁碟上，並且系統重新啓動後依然可保有一致性。您亦可透過 **--mirrorlog core** 引數來指定將此日誌存放在記憶體中；這可除去額外日誌裝置的需要，不過若要如此，每當系統重新啓動時，整個鏡像就必須被重新同步化。

下列指令將會由 **bigvg** 卷冊群組建立鏡像邏輯卷冊。該邏輯卷冊名為 **ondiskmirvol** 並且含有單鏡像。卷冊大小為 12 MB 並且將 mirror log 存放在記憶體中。



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

mirror log 會被建立在一個與任何 mirror leg 都會被建立於的裝置區隔開的裝置上。不過您亦可透過使用 **vgcreate** 指令的 **--alloc anywhere** 引數來在和其中一個 mirror leg 的裝置相同的裝置上建立一個 mirror log。這可能會使效能降低，不過它卻能讓您建立鏡像，儘管您只有兩個基本裝置。

下列指令會建立一個單鏡像的鏡像邏輯卷冊，它的 mirror log 位於與其中一個 mirror leg 的裝置相同的裝置上。在此範例中，**vg0** 這個卷冊群組只包含著兩個裝置。這項指令所建立的鏡像卷冊的大小為 500 MB，名為 **mirrorlv**，並且是由卷冊群組 **vg0** 中所分割出的：

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

### 注意

對於叢集鏡像來說，mirror log 管理必須為目前最低叢集 ID 的叢集節點完全負責。因此，當持有叢集 mirror log 的裝置在叢集的子集上無法使用時，叢集鏡像可在不造成任何影響的情況下，繼續進行作業，只要最低 ID 的叢集節點保留了 mirror log 的存取權限即可。因為鏡像不會受到干擾，因此也不會進行自動修正（修復）的動作。然而，當最低 ID 的叢集節點失去了 mirror log 的存取權限時，自動動作將會啟動（無論由其它節點對於 log 的存取權限為何）。

若要建立本身被映射的 mirror log，您可指定 **--mirrorlog mirrored** 引數。下列指令將會由 **bigvg** 卷冊群組建立鏡像邏輯卷冊。該邏輯卷冊名為 **twologvol** 並且含有單鏡像。卷冊大小為 12 MB 並且將 mirror log 已被映射，而存放在各別的裝置上。

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

和標準的 mirror log 相同，您可藉由使用 **vgcreate** 指令的 **--alloc anywhere** 引數，來在與鏡像 leg 相同的裝置上建立冗餘鏡像 log。這可能會使效能降低，不過它卻能讓您建立冗餘鏡像 log，儘管您沒有足夠的基本裝置，以存放 log 於各別裝置上。

當鏡像被建立後，鏡像區域會被同步化。對於大型的鏡像元件來說，同步程序可能會花上許多時間。當您要建立一個無須再生的新鏡像時，您可指定 **--nosync** 引數來顯示來自於第一個裝置的初始同步化是非必要的。

您可指定使用來作為 mirror leg 和 log 的裝置為何，以及該使用裝置的哪個扇區。若要強制將日誌存放在某個特定磁碟上，請在磁碟上確切指定一個欲存放日誌的扇區。LVM 並不一定會遵照指令列中所列出的裝置順序。若有任何實體卷冊被列出的話，那將會是唯一被進行分配的空間。任何包含在清單中，並且已被分配的實體扇區皆會被忽略掉。

下列指令會建立一個含有單獨鏡像的鏡像邏輯卷冊。卷冊大小為 500 MB，名為 **mirrorlv**，並且是由卷冊群組 **vg0** 中所切割出來的。mirror 的第一個 leg 位於 **/dev/sda1** 裝置上，mirror 的第二個 leg 位於 **/dev/sdb1** 裝置上，mirror logs 則位於 **/dev/sdc1** 上。

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

下列指令會建立一個含有單獨鏡像的鏡像邏輯卷冊。卷冊大小為 500 MB，名為 **mirrorlv**，並且是由卷冊群組 **vg0** 中所切割出來的。mirror 的第一個 leg 位於 **/dev/sda1** 裝置的扇區 0 至 499 上，mirror 的第二個 leg 位於 **/dev/sdb1** 裝置的扇區 0 至 499 上，並且 mirror log 由 **/dev/sdc1** 裝置的扇區 0 上起始。這些為 1MB 的扇區。若有任何指定的扇區早已被分配的話，它們將會被忽略掉。

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



### 注意

由 Red Hat Enterprise Linux 6.1 發行版起，您將可在單獨邏輯卷冊中結合 striping 與 mirroring。當建立邏輯卷冊並同時指定鏡像數量 (**--mirrors X**) 和磁條數量 (**--stripes Y**) 時，會使鏡像裝置所構成的裝置成為等量。

#### 4.4.3.1. 鏡像邏輯卷冊失效政策

您可透過在 `lvm.conf` 檔案的 `activation` 部份中使用 `mirror_image_fault_policy` 和 `mirror_log_fault_policy` 參數，以定義當裝置失效時，鏡像邏輯卷冊會如何運作。當這些參數被設為了 **remove** 時，系統將會嘗試移除出錯的裝置，並在無此裝置的情況下運作。當此參數被設為了 **allocate** 時，系統將會嘗試移除出錯的裝置，並嘗試在新裝置上配置空間，以取代失效的裝置；若無適當的裝置和空間可配置取代，此政策的運作方式是與 **remove** 政策相似的。

就預設值，`mirror_log_fault_policy` 參數會被設為 **allocate**。為 log 使用此政策既快速，亦可在當機或重新開機時，記住同步狀態。若您將此政策設為了 **remove**，當 log 裝置失效時，鏡像便會轉換為使用一種記憶體中的日誌，並且在當機或重新開機時，該鏡像將不會記住它的同步狀態，並且整個鏡像將會被重新同步。

就預設值，`mirror_image_fault_policy` 參數會被設為 **remove**。當使用此政策時，若鏡像映像失效，而只有一份完整鏡像的話，鏡像便會轉換為非映射的裝置。若要將鏡像裝置的政策設為 **allocate**，鏡像需要與裝置同步；這是一項緩慢的程序，不過它可保留裝置的鏡像特性。



### 注意

當 LVM 鏡像發生了裝置失效的問題時，將會進行一種兩階段的復原程序。第一階段包含移除失效的裝置。這可能會造成鏡像被縮減為線性裝置。第二階段，若 `mirror_log_fault_policy` 參數被設為了 **allocate**，它將會嘗試置換所有失效的裝置。然而請注意，若有其它可用裝置，您無法保證在第二階段中，先前由未失效的鏡像所使用的裝置會被選擇。

欲取得由失效的 LVM 鏡像進行手動式復原的相關資訊，請參閱〈[節 6.3, “由 LVM 鏡像錯誤中復原”](#)〉。

#### 4.4.3.2. 由鏡像邏輯卷冊切割出冗餘映像

您可將鏡像邏輯卷冊的冗餘映像切割出來形成一個新的邏輯卷冊。若要切割映像，請使用 `lvconvert` 指令的 **--splitmirrors** 引數，並指定欲切割的冗餘映像數量。您必須使用指令的 **--name** 引數來為新切割的邏輯卷冊指定一組名稱。

下列指令將由鏡像邏輯卷冊 **vg/lv** 切割一個名為 **copy** 的新邏輯卷冊。新邏輯卷冊包含了兩個 mirror leg。在此範例中，LVM 選擇要切割哪些裝置。

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

您可指定要切割哪些裝置。下列指令將會由鏡像邏輯卷冊 **vg/lv** 中，切割出一個名為 **copy** 的新邏輯卷冊。新邏輯卷冊包含了兩個含有 **/dev/sdc1** 和 **/dev/sde1** 裝置的新邏輯卷冊。

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

#### 4.4.3.3. 修復鏡像邏輯裝置

當磁碟失效時，您可使用 **lvconvert --repair** 指令來修復鏡像。這會使鏡像恢復為原本正常的狀態。**lvconvert --repair** 指令是一項互動式的指令，它會提示詢問您是否希望系統嘗試替換掉任何失效的裝置。

- 若要跳過提示並替換所有失效的裝置，請在指令列上指定 **-y** 選項。
- 若要跳過提示，並不替換任何失效的裝置，請在指令列上指定 **-f** 選項。
- 若要跳過提示並還是要為鏡像映像和鏡像 log 替換不同的政策，您可指定 **--use-policies** 引數，以使用 **lvm.conf** 檔案中的 **mirror\_log\_fault\_policy** 和 **mirror\_device\_fault\_policy** 參數所指定的裝置替換政策。

#### 4.4.3.4. 更改鏡像卷冊配置

您可透過使用 **lvconvert** 指令來新增或減少邏輯卷冊所包含的鏡像數量。這能讓您將邏輯卷冊由鏡像卷冊轉換為線性卷冊，或由線性卷冊轉換為鏡像卷冊。您亦可使用這項指令來重新配置既有邏輯卷冊的其它鏡像參數，例如 **corelog**。

當您將線性卷冊轉換為鏡像卷冊時，您基本上就是在為一個現有的卷冊建立 mirror leg。這代表您的卷冊群組必須包含著 mirror leg 以及 mirror log 的裝置與空間。

若您失去了鏡像的其中一個 leg，LVM 便會將卷冊轉換為一個線性卷冊，如此一來您便能在無冗餘鏡像的情況下繼續存取卷冊。當您替換了 leg 之後，您可使用 **lvconvert** 指令來將鏡像復原。此步驟提供於〈節 6.3, “由 LVM 鏡像錯誤中復原”〉之中。

下列指令會將線性邏輯卷冊 **vg00/lvol1** 轉換為鏡像邏輯卷冊。

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

下列指令會將鏡像邏輯卷冊 **vg00/lvol1** 轉換為線性邏輯卷冊，並移除 mirror leg。

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

下列範例新增了額外的 mirror leg 至既有的邏輯卷冊 **vg00/lvol1** 上。此範例顯示了 **lvconvert** 指令在將卷冊改變為擁有兩個 mirror leg 之前與之後的卷冊配置。

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

```
[lvol1_mlog] /dev/sdd1(0)
```

#### 4.4.4. 建立快照卷冊（Snapshot Volumes）

使用 **lvcreate** 指令的 **-s** 引數來建立 snapshot 卷冊。Snapshot 卷冊是可寫入的。



##### 注意

叢集中，在節點之間並不支援 LVM snapshot。您無法在叢集卷冊群組中建立一個 snapshot 卷冊。然而從 Red Hat Enterprise Linux 6.1 發行版開始，若您需要在叢集邏輯卷冊上建立一致的資料備份，您可單獨啓用卷冊，並建立 snapshot。欲取得在單一節點上，單獨啓用邏輯卷冊的相關資訊，請參閱 [〈節 4.7, “在叢集中啓用各別節點上的邏輯卷冊”〉](#)。



##### 注意

由 Red Hat Enterprise Linux 6.1 發行版起，鏡像邏輯卷冊支援了 LVM snapshot。

由 RHEL 6.3 發行版起，RAID 邏輯卷冊支援了 snapshot。欲取得 RAID 邏輯卷冊上的相關資訊，請參閱 [節 4.4.13, “RAID 邏輯卷冊”](#)。

下列指令建立了一個大小為 100 MB 名為 **/dev/vg00/snap** 的 snapshot 邏輯卷冊。它建立了名為 **/dev/vg00/lvol1** 的原始邏輯卷冊的 snapshot。若原始的邏輯卷冊包含了一個檔案系統，您可將 snapshot 邏輯卷冊掛載在一個任意的目錄上以便在原始檔案系統進行更新的同時存取檔案系統的內容來進行備份。

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

在您建立了 snapshot 邏輯卷冊後，利用 **lvdisplay** 指令來指定原始的卷冊，可產生出包含著一個含有所有 snapshot 邏輯卷冊與其狀態（啓用或停用）之清單的輸出。

下列範例顯示了邏輯卷冊 **/dev/new\_vg/lvol0** 的狀態，並且有個 snapshot 卷冊 **/dev/new\_vg/newvgsnap** 已被建立。

```
# lvdisplay /dev/new_vg/lvol0
--- Logical volume ---
LV Name                /dev/new_vg/lvol0
VG Name                new_vg
LV UUID                LBy1Tz-sr23-0jsI-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
```

就預設值，**lvs** 指令會顯示初始卷冊以及各個 snapshot 卷冊目前被使用到的比例。下列範例顯示了在一部包含著邏輯卷冊 **/dev/new\_vg/lvol0** 的系統上輸入 **lvs** 指令的預設輸出，並且有個 snapshot 卷冊 **/dev/new\_vg/newvgsnap** 已被建立。

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



#### 警告

因為 snapshot 的大小會隨著原始卷冊的改變而增加，所以請記得時常透過 **lvs** 指令來監控 snapshot 卷冊的比例以確保它不會滿出。一個用滿 100% 的 snapshot 基本上一定會發生問題，因為若要寫至未修改的初始卷冊部份中一定得將該 snapshot 損毀才能成功。

由 RHEL 6.2 發行版起，新增了兩項 snapshot 的相關功能。首先，除了 snapshot 本身在滿出時會被無效化，任何掛載於該 snapshot 上的檔案系統也會被強制卸載，避免存取掛載點時發生必然的檔案系統錯誤。第二，您可在 **lvm.conf** 檔案中指定 **snapshot\_autoextend\_threshold** 選項。此選項能在每當剩餘的 snapshot 空間低於您所設置的門檻值時，自動延伸 snapshot。若要使用這項功能，卷冊群組中需要含有未分配的空間。

有關於設定 **snapshot\_autoextend\_threshold** 和 **snapshot\_autoextend\_percent** 的相關資訊位於 **lvm.conf** 檔案中。欲取得有關於 **lvm.conf** 檔案上的相關資訊，請參閱 [附錄 B, LVM 配置檔案](#)。

### 4.4.5. 合併 Snapshot 卷冊

從 RHEL 6 發行版開始，您將可使用 **lvconvert** 指令的 **--merge** 選項，來將 snapshot 併入其原始卷冊中。若原始卷冊和 snapshot 卷冊兩者皆未開啓的話，合併程序便會即刻啓動。否則，合併將會在原始或 snapshot 卷冊第一次啓用和兩者皆關閉時啓動。若要將 snapshot 合併入一個無法關閉的原始卷冊（比方說 **root** 檔案系統）中，這項合併程序將會被延後，直到原始卷冊下次啓用時。當進行合併程序時，所產生的邏輯卷冊將會擁有原始卷冊的名稱、minor 號碼，以及 UUID。當合併程序在進行時，對於原始卷冊所進行的讀取和寫入，將會看似它們是被導向至即將被合併的 snapshot。當合併程序完成時，合併的 snapshot 將會被移除。

下列指令會將 snapshot 卷冊 **vg00/lvol1\_snap** 併入它的原始卷冊中。

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

您可在指令列上指定多重 snapshot，或是您亦可使用 LVM 物件標籤，來指定將多個 snapshot 與和它們相應的原始卷冊合併。在以下範例中，邏輯卷冊 **vg00/lvol1**、**vg00/lvol2** 和 **vg00/lvol3** 皆被標記了 **@some\_tag**。下列指令會序列式地合併所有三個卷冊的 snapshot 邏輯卷冊：

**vg00/lvol1**、**vg00/lvol2**，然後 **vg00/lvol3**。若使用了 **--background** 選項的話，所有 snapshot 邏輯卷冊的合併程序將會平行式地啓動。

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



欲取得標記 LVM 物件上的相關資訊，請參閱〈[附錄 C, LVM 物件標籤 \(Object Tags\)](#)〉。欲取得更多有關於 **lvconvert --merge** 指令上的相關資訊，請參閱 **lvconvert(8)** man page。

#### 4.4.6. 一致的裝置號碼

主要與次要裝置號碼會於模組載入時被動態式地分配。有些應用程式可因為區塊裝置總是透過相同的裝置（主要與次要）號碼來啟用，而得到最佳的效果。您可透過 **lvcreate** 和 **lvchange** 指令，並使用下列引數來進行指定：

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

使用一個較大的次要號碼，來確認它尚未被動態式地分配至其它裝置上。

若您希望使用 NFS 來匯出檔案系統的話，在 **exports** 檔案中指定 **fsid** 參數可能能夠避免在 LVM 中設置一致性的裝置號碼的必要性。

#### 4.4.7. 重設邏輯卷冊大小

若要更改邏輯卷冊的大小，請使用 **lvreduce** 指令。若邏輯卷冊包含著一個檔案系統，請確認先減少檔案系統（或使用 LVM GUI），如此一來邏輯卷冊大小就總是能至少和檔案系統所預期的大小相等。

下列指令會使 **vg00** 卷冊群組中的邏輯卷冊 **lv011** 的大小減少 3 個邏輯扇區。

```
# lvreduce -l -3 vg00/lv011
```

#### 4.4.8. 更改邏輯卷冊群組的參數

若要更改邏輯卷冊的參數，請使用 **lvchange** 指令。如欲取得一列包含著您可更改的參數之清單，請查看 **lvchange(8)** man page。

您可使用 **lvchange** 指令來啟用和停用邏輯卷冊。若要同時啟用和停用某個卷冊群組中的所有邏輯卷冊，請使用 **vgchange** 指令，如〈[節 4.3.7, “更改卷冊群組的參數”](#)〉中所描述。

下列指令會將卷冊群組 **vg00** 中的 **lv011** 卷冊之權限更改為唯讀。

```
# lvchange -pr vg00/lv011
```

#### 4.4.9. 重新為邏輯卷冊命名

若要為現有的邏輯卷冊重新命名，請使用 **lvrename** 指令。

下列兩項指令皆可將卷冊群組 **vg02** 中的邏輯卷冊 **lvold** 重新命名為 **lvnew**。

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

```
# lvrename vg02 lvold lvnew
```

欲取得更多有關於如何啟用叢集中，各別節點上的邏輯卷冊的相關資訊，請參閱〈[節 4.7, “在叢集中啟用各別節點上的邏輯卷冊”](#)〉。

#### 4.4.10. 移除邏輯卷冊

若要移除一個非啟用中的邏輯卷冊，請使用 **lvremove** 這項指令。若該邏輯卷冊目前已掛載，請在將此卷冊移除前將它卸載。此外，在一個叢集環境中，您必須先將邏輯卷冊停用才可將之移除。

下列指令會將邏輯卷冊 **/dev/testvg/testlv** 由卷冊群組 **testvg** 中移除。請注意，在此情況下，邏輯卷冊並未被停用。

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

您能夠透過 **lvchange -an** 指令來在移除某個邏輯卷冊前先將它停用，在此情況下，您將不會看見一個提示要求您驗證是否要將一個啟用中的邏輯卷冊移除掉。

#### 4.4.11. 顯示邏輯卷冊

您可使用三項指令來顯示 LVM 邏輯卷冊的內容：**lvs**、**lvdisplay** 以及 **lvscan**。

**lvs** 指令會以可配置的格式來提供邏輯卷冊資訊，並以一行一個邏輯卷冊的格式顯示。**lvs** 指令提供了大量的格式控制，並有助於編寫 script。欲取得有關於如何使用 **lvs** 指令來自訂您的輸出的相關資訊，請參閱〈[節 4.8, “LVM 的自訂化回報”](#)〉。

**lvdisplay** 指令可利用固定的格式來顯示邏輯卷冊的內容（像是大小、格式和映射）。

下列指令顯示了 **vg00** 中的 **lv012** 的屬性。若是已為此原始邏輯卷冊建立了 snapshot 邏輯卷冊的話，這項指令便會顯示一系列含有所有 snapshot 邏輯卷冊以及其狀態（啟用中或停用中）的清單。

```
# lvdisplay -v /dev/vg00/lv012
```

**lvscan** 指令會掃秒系統中所有的邏輯卷冊並將它們列出，如下列範例。

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

#### 4.4.12. 遞增邏輯卷冊

若要增加邏輯卷冊的大小，請使用 **lvextend** 指令。

當您延伸邏輯卷冊時，您可指定要將卷冊延伸多少或是當您將它延伸後它應該要多大。

下列指令會將邏輯卷冊 **/dev/myvg/homevol** 延伸為 12 GB。

```
# 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
```

下列指令會增加額外的 1GB 至邏輯卷冊 **/dev/myvg/homevol**。

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



就和 **lvcreate** 指令相同，您可使用 **lvextend** 指令的 **-l** 引數來指定扇區的數量，並遞增邏輯卷冊的大小。您亦可使用此引數來指定卷冊群組的百分比，或是卷冊群組剩下可用空間的百分比。下列指令將會延伸一個名為 **testlv** 的邏輯卷冊，並填滿卷冊群組 **myvg** 中所有未分配的空間。

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

在您延伸了邏輯卷冊後，您需要增加檔案系統大小來進行匹配。

就預設值來講，大部分用來重設檔案系統大小的工具都會將檔案系統的大小遞增為基本邏輯卷冊的大小，如此一來您便無須擔心得為這兩項指令指定相同的大小。

#### 4.4.12.1. 延伸等量的卷冊

若要增加等量邏輯卷冊的大小，用來製作支援 **stripe** 的卷冊群組的基本實體卷冊上就必須要有足夠的空間。比方說，若您有個雙向的 **stripe** 而它使用了所有的卷冊群組，新增一個單獨的實體卷冊至卷冊群組將無法讓您延伸該 **stripe**。您必須要新增至少兩個實體卷冊至卷冊群組中。

比方說，請參考一個包含著兩個基本實體卷冊的卷冊群組 **vg**，如下列透過 **vgs** 指令所示。

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

您可透過使用卷冊群組中的所有空間來建立磁條。

```
# 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)
```

請注意，卷冊群組現在已無可用空間。

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

下列指令會增加額外的實體卷冊至卷冊群組，如此一來它便會含有 135G 的額外空間。

```
# 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
```

在此情況下，您無法將等量邏輯卷冊延伸為整個卷冊群組的大小，因為需要兩個基本的裝置才可 **stripe** 資料。

```
# 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
```

若要延伸等量邏輯卷冊，請新增另一實體卷冊然後再延伸邏輯卷冊。在此範例中，新增了兩個實體卷冊至卷冊群組之後，我們便可將邏輯卷冊延伸為卷冊群組的完整大小。

```
# 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
```

在您沒有足夠的基本實體裝置來延伸等量邏輯卷冊的情況下，若延伸沒有被 **stripe** 無所謂的話，您依然還是能夠延伸該卷冊，而這將有可能造成效能上的不一致。當增加邏輯卷冊的空間時，預設的作業就是使用和現有邏輯卷冊的最後一個區段相同的 **striping** 參數，不過您可將這些參數置換掉。下列範例在初始的 **lvextend** 指令失敗後延伸了現有的等量邏輯卷冊來使用剩下的可用空間。

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

#### 4.4.12.2. 延伸鏡像卷冊

由 RHEL 6.3 發行版起，您可透過 **lvextend** 指令來擴展鏡像邏輯卷冊，而無需同步新的鏡像區域。

若您在透過 **lvcreate** 指令建立鏡像邏輯卷冊時，使用了 **--nosync** 選項的話，當鏡像被建立時，鏡像區域將不會被同步（如 [節 4.4.3, “建立鏡像卷冊”](#) 中所述）。若您稍後以 **--nosync** 選項來延伸您所建立的鏡像，延伸的鏡像也不會當下被同步。

您可透過使用 **lvs** 指令來顯示卷冊屬性，以判斷既有的邏輯卷冊是否是透過 **--nosync** 選項所建立的。若鏡像卷冊是在未初始同步的情況下建立的，邏輯卷冊的屬性 bit 1 將會是 "M"，若透過初始同步所建立，則屬性 bit 1 將會是 "m"。

下列指令顯示了名為 **lv** 的鏡像邏輯卷冊之屬性，此鏡像邏輯卷冊乃在未同步的情況下建立的，因此屬性 bit 1 顯示為 "M"。屬性 bit 7 為 "m"，代表目標類型為 **mirror**。欲取得屬性 bit 意義上的相關資訊，請參閱 [表格 4.4, “lvs 顯示欄位”](#)。

```
# lvs vg
LV      VG      Attr      LSize Pool Origin Snap%  Move Log       Copy%  Convert
lv      vg      Mwi-a-m- 5.00g                               lv_mlog 100.00
```

若您透過 **lvextend** 指令來擴充此鏡像邏輯卷冊，鏡像延伸將不會被重新同步。

若您在未指定 **lvcreate** 指令的 **--nosync** 選項的情況下建立鏡像邏輯卷冊，您可透過 **lvextend** 指令的 **--nosync** 選項，來在不重新同步鏡像的情況下，擴展邏輯卷冊。

下列範例延伸了在未使用 **--nosync** 選項的情況下建立的邏輯卷冊，這代表該鏡像被建立時已同步。然而此範例指定了當在延伸卷冊時，不同步鏡像。請注意，卷冊的屬性為 "m"，不過在執行了 **lvextend** 指令並搭配 **--nosync** 選項時，卷冊的屬性則為 "M"。

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

若鏡像未啟用，它將不會在您延伸鏡像時，自動跳過同步化，儘管您透過了指定 **--nosync** 選項來建立鏡像。反之，您將會被提示是否要針對於邏輯卷冊延伸的部分，進行完整的重新同步。



#### 注意

當鏡像正在進行復原時，若您透過指定 **--nosync** 選項來建立或延伸了卷冊的話，您將無法延伸鏡像邏輯卷冊。然而，若您未指定 **--nosync** 選項的話，您便可在鏡像復原時將其延伸。

#### 4.4.12.3. 以 **cling** 分配政策延伸邏輯卷冊

當延伸 LVM 卷冊時，您可使用 **lvextend** 指令的 **--alloc cling** 選項，來指定 **cling** 定位政策。此政策將會選擇與既有邏輯卷冊的最後磁區相同的實體卷冊上的空間。若實體卷冊上的空間不足，並且有一列標籤定義於 **lvm.conf** 檔案中的話，LVM 將會檢查是否有任何標籤連接至實體卷冊，並嘗試比對既有扇區和新扇區之間的實體卷冊標籤。

比方說，若您在一個單獨的卷冊群組中的兩個區域之間，映射了邏輯卷冊的話，您可根據實體卷冊的所在地，來標記它們，您需將實體卷冊以 **@site1** 和 **@site2** 標籤標記，並在 **lvm.conf** 檔案中指定下列一行：

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

欲取得有關於標記實體卷冊上的相關資訊，請參閱〈[附錄 C, LVM 物件標籤 \(Object Tags\)](#)〉。

在下列範例中，**lvm.conf** 檔案已修改，並包含了以下一行：

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

此外，在此範例中，有個卷冊群組 **taft** 已被建立，並包含了實體卷冊 **/dev/sdb1**、**/dev/sdc1**、**/dev/sdd1**、**/dev/sde1**、**/dev/sdf1**、**/dev/sdg1**，以及 **/dev/sdh1**。這些實體卷冊已標記為 **A**、**B** 和 **C**。範例中未使用 **C** 標籤，不過這會顯示 LVM 使用了標籤，來選擇哪些實體卷冊將使用於 mirror leg。

```
# pvs -a -o +pv_tags /dev/sd[bcdefgh]1
```

```

PV          VG   Fmt  Attr PSize   PFree   PV Tags
/dev/sdb1   taft lvm2 a-    135.66g 135.66g A
/dev/sdc1   taft lvm2 a-    135.66g 135.66g B
/dev/sdd1   taft lvm2 a-    135.66g 135.66g B
/dev/sde1   taft lvm2 a-    135.66g 135.66g C
/dev/sdf1   taft lvm2 a-    135.66g 135.66g C
/dev/sdg1   taft lvm2 a-    135.66g 135.66g A
/dev/sdh1   taft lvm2 a-    135.66g 135.66g A

```

下列指令將會由 **taft** 卷冊群組中建立一個 100GB 的鏡像卷冊。

```
# lvcreate -m 1 -n mirror --nosync -L 100G taft
```

下列指令顯示了哪些裝置會被使用於 mirror leg 和 mirror log。

```

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

```

下列指令會延伸鏡像卷冊的大小，並使用 **cling** 分配政策來顯示 mirror leg 應使用相同標籤的實體卷冊來延伸。

```

# lvextend --alloc cling -L +100G taft/mirror
Extending 2 mirror images.
Extending logical volume mirror to 200.00 GiB
Logical volume mirror successfully resized

```

下列指令會顯示 mirror leg 已透過使用實體卷冊，以及與 leg 相同的標籤延伸。請注意，標籤為 **c** 的實體卷冊會被忽略。

```

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

```

#### 4.4.13. RAID 邏輯卷冊

由 RHEL 6.3 發行版起，LVM 開始支援 RAID4/5/6 以及新的鏡像實作。最新的鏡像實作與先前的鏡像實作有著以下的不同（記載於 [節 4.4.3, “建立鏡像卷冊”](#) 中）：

- 新的鏡像實作之磁區類型為 **raid1**。先前實作的磁區類型則為 **mirror**。
- 新的鏡射實作採用了 MD 軟體 RAID，RAID 4/5/6 實作亦然。
- 鏡像的實作為各個 mirror image 保有了完全冗余的 bitmap 區域，這會增強其錯誤處理的能力。這代表以此磁區類型建立的鏡像沒有 **--mirrorlog** 或是 **--corelog** 選項。
- 新的鏡像實作可處理暫時性的錯誤。
- 新的鏡射實作支援 snapshot（較高等級的 RAID 實作亦如此）。目前尚不支援針對於磁區類型為 **mirror** 的鏡像製作 snapshot（儘管可建立）。
- 鏡像映像檔能暫時性地由陣列中分割，並事後合併回陣列中。
- 新的 RAID 實作無法偵測到叢集。您無法在叢集卷冊群組中建立 LVM RAID 邏輯卷冊。

欲取得更多有關於 RAID 邏輯卷冊如何處理錯誤的相關資訊，請參閱 [節 4.4.13.8, “設定 RAID 錯誤政策”](#)。

此部分剩下的內容將詳述下列您可在 LVM RAID 裝置上進行的管理工作：

- [節 4.4.13.1, “建立 RAID 邏輯卷冊”](#)
- [節 4.4.13.2, “將線性裝置轉換為 RAID 裝置”](#)
- [節 4.4.13.3, “將 LVM RAID1 邏輯卷冊轉換為一個 LVM 線性邏輯卷冊”](#)
- [節 4.4.13.4, “將鏡像 LVM 裝置轉換為 RAID 1 裝置”](#)
- [節 4.4.13.5, “修改既有 RAID1 裝置中的映像檔數量”](#)
- [節 4.4.13.6, “將 RAID 映像檔分割為各別的邏輯卷冊”](#)
- [節 4.4.13.7, “分割與合併 RAID 映像檔”](#)
- [節 4.4.13.8, “設定 RAID 錯誤政策”](#)
- [節 4.4.13.9, “替換 RAID 裝置”](#)

#### 4.4.13.1. 建立 RAID 邏輯卷冊

若要建立 RAID 邏輯卷冊，您必須使用 **lvcreate** 指令的 **--type** 引數來指定 RAID 類型。一般當您透過 **lvcreate** 指令建立邏輯卷冊時，**--type** 會是隱藏的。比方說，當您指定 **-i stripes** 引數時，**lvcreate** 指令會假定設置 **--type stripe** 選項。當您指定 **-m mirrors** 引數時，**lvcreate** 指令則會假定設置 **--type mirror** 選項。然而當您建立 RAID 邏輯卷冊時，您必須明確指定您想要的磁區類型。可使用的 RAID 磁區類型詳述於 [表格 4.1, “RAID 磁區類型”](#) 中。

表格 4.1. RAID 磁區類型

磁區類型	描述
<b>raid1</b>	RAID1 鏡射

磁區類型	描述
<b>raid4</b>	RAID4 特屬的同位磁碟
<b>raid5</b>	和 <b>raid5_ls</b> 相同
<b>raid5_la</b>	<div>RAID5 向左對稱。</div> <div>循環 parity 0 並且資料將延續</div>
<b>raid5_ra</b>	<div>RAID5 向右對稱。</div> <div>循環 parity N 並且資料將延續</div>
<b>raid5_ls</b>	<div>RAID5 向左對稱。</div> <div>循環 parity 0 並且資料將重新啓用</div>
<b>raid5_rs</b>	<div>RAID5 向右對稱。</div> <div>循環 parity N 並且資料將重新啓用</div>
<b>raid6</b>	和 <b>raid6_zr</b> 相同
<b>raid6_zr</b>	<div>RAID6 零起始</div> <div>循環 parity zero（左到右）並且資料將重新啓用</div>
<b>raid6_nr</b>	<div>RAID6 N 重新啓用</div> <div>循環 parity N（左到右）並且資料將重新啓用</div>
<b>raid6_nc</b>	<div>RAID6 N 繼續</div> <div>循環 parity N（左到右）並且資料將延續</div>

對於大部份使用者來說，指定主要的可用類型之一（**raid[1456]**）應已足夠。欲取得更多有關於 RAID 5/6 所使用之不同演算法則上的相關資訊，請參閱《常用的 RAID 磁碟資料格式規格》的第四章節，位於 [http://www.snia.org/sites/default/files/SNIA\\_DDF\\_Technical\\_Position\\_v2.0.pdf](http://www.snia.org/sites/default/files/SNIA_DDF_Technical_Position_v2.0.pdf)。

當您建立 RAID 邏輯卷冊時，LVM 會建立一個 metadata 子卷冊，並且陣列中各資料或同位子卷冊的大小皆為一個扇區。比方說，建立一個雙向的 RAID1 陣列時，會產生兩個 metadata 子卷冊（**lv\_rmeta\_0** 與 **lv\_rmeta\_1**），以及兩個資料子卷冊（**lv\_rimage\_0** 與 **lv\_rimage\_1**）。相同的，當建立一個三向的 stripe（加上一個隱藏的同位裝置）RAID4 時，將會產生四個 metadata 子卷冊（**lv\_rmeta\_0**、**lv\_rmeta\_1**、**lv\_rmeta\_2** 以及 **lv\_rmeta\_3**）和四個資料子卷冊（**lv\_rimage\_0**、**lv\_rimage\_1**、**lv\_rimage\_2** 以及 **lv\_rimage\_3**）。

下列指令將會在卷冊群組 **my\_vg** 中，建立一個大小為 1G 的雙向 RAID1 陣列，名為 **my\_lv**。

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

您可根據您為 **-m** 引數所指定的值，來建立數個 RAID1 陣列。儘管 **-m** 引數與先前的鏡像實作中，使用來指定數量的引數相同，在此情況下，您會藉由明確將磁區類型設為 **raid1**，以置換預設磁區類型 **mirror**。相同地，您亦可透過熟悉的 **-i argument**，指定 RAID 4/5/6 邏輯卷冊的 stripe 數量，將預設的磁區類型置換為您想要的 RAID 類型。您亦可透過 **-I** 引數來指定 stripe 的大小。



#### 注意

您可藉由更改 **lvm.conf** 檔案中的 **mirror\_segtype\_default**，來將預設的鏡像磁區類型設為 **raid1**。

下列指令將會在 **my\_vg** 卷冊群組中建立一個 RAID5 陣列（三個 stripe 加上一個隱藏同位磁碟），名為 **my\_lv** 並且大小為 1G。請注意，您必須如同為 LVM stripe 卷冊一般的指定 stripe 數量；正確的同位磁碟數量將會被自動加入。

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

下列指令將會在 **my\_vg** 卷冊群組中，建立一個 RAID6 陣列（三個 stripe 加上兩個隱藏同位磁碟），名為 **my\_lv** 並且大小為 1G。

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

在您以 LVM 建立了 RAID 邏輯卷冊後，您可啓用、更改、移除、顯示以及使用卷冊，就像是任何其它 LVM 邏輯卷冊一樣。

#### 4.4.13.2. 將線性裝置轉換為 RAID 裝置

您可藉由使用 **lvconvert** 指令的 **--type** 引數，來將既有的線性邏輯卷冊轉換為 RAID 裝置。

下列指令會將卷冊群組 **my\_vg** 中的線性邏輯卷冊 **my\_lv** 轉換為一個雙向的 RAID1 陣列。

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

因為 RAID 邏輯卷冊是由 metadata 和資料子卷冊配對所構成的，因此當您將線性裝置轉換為 RAID1 陣列時，會有個新的 metadata 子卷冊被建立，並與該線性卷冊位於的（其中一個）相同實體卷冊上的原始邏輯卷冊相聯。額外的映像檔將會被新增至 metadata/data 子卷冊配對中。比方說，若原始裝置如下：

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

在轉換至雙向 RAID1 陣列之後，裝置將會包含下列資料與 metadata 子卷冊配對：



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

若與原始邏輯卷冊配對的 metadata 映像檔無法放置在相同的實體卷冊上，則 **lvconvert** 將會失敗。

#### 4.4.13.3. 將 LVM RAID1 邏輯卷冊轉換為一個 LVM 線性邏輯卷冊

您可透過使用 **lvconvert** 指令，並指定 **-m0** 引數來將既有的 RAID1 LVM 邏輯卷冊轉換為一個 LVM 線性邏輯卷冊。這將會移除構成了 RAID 陣列的所有 RAID 資料子卷冊，以及所有 RAID metadata 子卷冊，並使最上層的 RAID1 映像檔成為線性邏輯卷冊。

下列範例顯示了一個既有的 LVM RAID1 邏輯卷冊。

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

下列指令會將 LVM RAID1 邏輯卷冊 **my\_vg/my\_lv** 轉換為一個 LVM 線性裝置。

```
# lvconvert -m0 my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       /dev/sde1(1)
```

當您要將一個 LVM RAID1 邏輯卷冊轉換為一個 LVM 線性卷冊時，您可指定欲移除哪些實體卷冊。下列範例顯示了一個 LVM RAID1 邏輯卷冊的格式，此乃兩個映像檔所構成：**/dev/sda1** 和 **/dev/sda2**。在此範例中，**lvconvert** 指令將移除 **/dev/sda1**，並將 **/dev/sdb1** 保留為構成線性裝置的實體卷冊。

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

#### 4.4.13.4. 將鏡像 LVM 裝置轉換為 RAID 1 裝置

您可使用 **lvconvert** 指令，透過指定 **--type raid1** 引數來將既有的 LVM 裝置轉換為一個 RAID1

LVM 裝置。這會將鏡像的子卷冊 (**\*\_mimage\_\***) 重新命名為 RAID 子卷冊 (**\*\_rimage\_\***)。此外 mirror log 將會被移除，並且將在與相應資料子卷冊相同的實體卷冊上，為資料子卷冊建立 metadata 子卷冊 (**\*\_rmeta\_\***)。

下列範例顯示了鏡像邏輯卷冊 **my\_vg/my\_lv** 的格式。

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

下列指令會將鏡像邏輯卷冊 **my\_vg/my\_lv** 轉換為一個 RAID1 邏輯卷冊。

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

#### 4.4.13.5. 修改既有 RAID1 裝置中的映像檔數量

您能如在先前的 LVM 鏡射實作中一般，更改既有 RAID1 陣列中的映像檔數量，您必須藉由使用 **lvconvert** 指令來指定額外欲新增或移除的 metadata/data 子卷冊配對。欲取得在較早的 LVM 鏡射實作中修改卷冊配置上的相關資訊，請參閱 [節 4.4.3.4, “更改鏡像卷冊配置”](#)。

當您透過 **lvconvert** 指令來新增映像檔至 RAID1 裝置時，您可指定最終產生之裝置的映像檔總數，或是指定欲新增至該裝置的映像檔數量。您亦可選用性地指定欲在哪個實體裝置上放置新的 metadata/data 映像檔配對。

Metadata 子卷冊 (名為 **\*\_rmeta\_\***) 總是會位於與其資料子卷冊配對 (**\*\_rimage\_\***) 相同的實體裝置上。Metadata/data 子卷冊配對將不會如同其它 RAID 陣列中的 metadata/data 子卷冊一般，建立在相同的實體卷冊上 (除非您指定 **--alloc anywhere**)。

新增映像檔至 RAID1 卷冊的指令格式如下：

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

比方說，以下顯示了一個雙向 RAID1 陣列的 LVM 裝置 **my\_vg/my\_lv**：

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

下列指令會將雙向的 RAID1 裝置轉換為 **my\_vg/my\_lv** 一個三向的 RAID1 裝置：

```
# lvconvert -m 2 my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       6.25
my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]      /dev/sde1(0)
[my_lv_rimage_1]      /dev/sdf1(1)
[my_lv_rimage_2]      /dev/sdg1(1)
[my_lv_rmeta_0]        /dev/sde1(256)
[my_lv_rmeta_1]        /dev/sdf1(0)
[my_lv_rmeta_2]        /dev/sdg1(0)
```

當您新增了一個映像檔至 RAID1 陣列時，您可為映像檔指定實體卷冊。下列指令會將雙向 RAID1 裝置 **my\_vg/my\_lv** 轉換為一個三向的 RAID1 裝置，並指定陣列將使用實體卷冊 **/dev/sdd1**：

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

若要由 RAID1 陣列中移除映像檔，請使用下列指令。當您透過 **lvconvert** 指令，由 RAID1 裝置移除映像檔時，您可為最終產生的裝置指定映像檔總數，或是指定欲由裝置上移除多少映像檔。您亦可選用性的指定欲由哪些實體卷冊移除裝置。

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

此外，當有個與 metadata 子卷冊相聯的映像檔被移除時，任何編號較高的映像檔皆會被往下移，以填入空位。若您由一個包含了 **lv\_rimage\_0**、**lv\_rimage\_1** 和 **lv\_rimage\_2** 的三向 RAID1 陣列中移除了 **lv\_rimage\_1**，這將會產生一個包含 **lv\_rimage\_0** 和 **lv\_rimage\_1** 的 RAID1 陣列。子卷冊 **lv\_rimage\_2** 將會被重新命名，並置於空位中成為 **lv\_rimage\_1**。

下列範例顯示了一個三向 RAID 1 邏輯卷冊 **my\_vg/my\_lv** 的格式。

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

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

下列指令會將三向的 RAID1 邏輯卷冊轉換為一個雙向的 RAID1 邏輯卷冊。

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

下列指令會將三向的 RAID1 邏輯卷冊轉換為一個雙向的 RAID1 邏輯卷冊，並將包含了欲移除之映像檔的實體卷冊指定為 **/dev/sde1**。

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

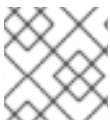
#### 4.4.13.6. 將 RAID 映像檔分割為各別的邏輯卷冊

您可分割 RAID 邏輯卷冊的映像檔，以形成新的邏輯卷冊。分割 RAID 映像檔的程序，與分割鏡像邏輯卷冊之冗余映像檔的程序相同，如 [節 4.4.3.2, “由鏡像邏輯卷冊切割出冗余映像”](#) 中所述。

分割 RAID 映像檔的指令格式如下：

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

就跟從現有的 RAID1 邏輯卷冊移除 RAID 映像檔一樣（如 [節 4.4.13.5, “修改既有 RAID1 裝置中的映像檔數量”](#)）所示）當您從裝置中間移除 RAID 資料的子卷冊（與其相關的 metadata 子卷冊）時，任何擁有更高數目的映像檔都會往下移，以填補空缺。這樣組成 RAID 陣列的邏輯卷冊索引編號才會保持連貫性。



#### 注意

如果 RAID1 陣列尚未同步，就不能分割 RAID 映像檔。

以下範例會將二個磁碟的 RAID1 邏輯卷冊 **my\_lv** 分割為兩個線性的邏輯卷冊：**my\_lv** 與 **new**。

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

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

以下範例會將三個磁碟的 RAID1 邏輯卷冊 **my\_lv** 分割為一個線性的雙磁碟 RAID1 邏輯卷冊：**my\_lv**，以及一個線性邏輯卷冊 **new**。

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

#### 4.4.13.7. 分割與合併 RAID 映像檔

您可暫時性地將 RAID1 陣列的映像檔分割為唯讀，並透過使用 **lvconvert** 指令，藉由 **--trackchanges** 引數搭配 **--splitmirrors** 引數來追蹤任何變更。這能讓您稍後將映像檔重新合併回陣列中，並僅只同步當映像檔被分割時，陣列所遭到更改的部分。

分割 RAID 映像檔的 **lvconvert** 指令之格式如下。

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

當您以 **--trackchanges** 引數來分割 RAID 映像檔時，您可指定欲分割哪個映像檔，不過您不可更改欲分割之卷冊的名稱。此外，最後產生的卷冊將會含有下列限制。

- 您新建的卷冊將會是唯讀。
- 您無法重新設定新卷冊的大小。
- 您無法將剩餘的陣列重新命名。
- 您無法重設剩餘陣列的大小。
- 您可獨立啓用新卷冊與剩餘的陣列。

您可藉由執行 **lvconvert** 搭配 **--merge** 引數來合併透過了 **--trackchanges** 引數分割的映像檔。當您合併映像檔時，僅有在映像檔分割後，遭到變更的陣列部分才會被重新同步。

合併 RAID 映像檔的 **lvconvert** 指令之格式如下。

```
lvconvert --merge raid_image
```

下列範例將建立一個 RAID1 邏輯卷冊，並由該卷冊分割一個映像檔，同時追蹤剩餘陣列的變更。

```
# lvcreate --type raid1 -m2 -L1G -n my_lv .vg
Logical volume "my_lv" created
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%   Devices
my_lv       100.00
my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]      /dev/sdb1(1)
[my_lv_rimage_1]      /dev/sdc1(1)
[my_lv_rimage_2]      /dev/sdd1(1)
[my_lv_rmeta_0]       /dev/sdb1(0)
[my_lv_rmeta_1]       /dev/sdc1(0)
[my_lv_rmeta_2]       /dev/sdd1(0)
# lvconvert --splitmirrors 1 --trackchanges my_vg/my_lv
my_lv_rimage_2 split from my_lv for read-only purposes.
Use 'lvconvert --merge my_vg/my_lv_rimage_2' to merge back into my_lv
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%   Devices
my_lv       100.00
my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]      /dev/sdb1(1)
[my_lv_rimage_1]      /dev/sdc1(1)
my_lv_rimage_2        /dev/sdd1(1)
[my_lv_rmeta_0]       /dev/sdb1(0)
[my_lv_rmeta_1]       /dev/sdc1(0)
[my_lv_rmeta_2]       /dev/sdd1(0)
```

下列範例將由 RAID1 卷冊分割一個映像檔，同時追蹤剩餘陣列的變更，然後再將卷冊重新合併回陣列中。

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

當您由 RAID1 卷冊分割映像檔後，您可藉由發出第二項 **lvconvert --splitmirrors** 指令、重複執行在未指定 **--trackchanges** 引數的情況下，分割映像檔的初始 **lvconvert** 指令，來使分割成為永久性。這將移除 **--trackchanges** 引數所建立的連結。

在您透過 **--trackchanges** 引數分割了映像檔之後，您無法在該陣列上發送其後的 **lvconvert --splitmirrors** 指令，除非您的原意乃將被追蹤的映像檔永久分割。

下列指令序列會將映像檔分割、追蹤映像檔並永久性地分割追蹤的映像檔。

```
# lvconvert --splitmirrors 1 --trackchanges my_vg/my_lv
my_lv_rimage_1 split from my_lv for read-only purposes.
Use 'lvconvert --merge my_vg/my_lv_rimage_1' to merge back into my_lv
# lvconvert --splitmirrors 1 -n new my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
LV      Copy%  Devices
my_lv    100.00 /dev/sdc1(1)
new      100.00 /dev/sdd1(1)
```

然而請注意，下列指令序列將會失敗。

```
# lvconvert --splitmirrors 1 --trackchanges my_vg/my_lv
my_lv_rimage_1 split from my_lv for read-only purposes.
Use 'lvconvert --merge my_vg/my_lv_rimage_1' to merge back into my_lv
# lvconvert --splitmirrors 1 --trackchanges my_vg/my_lv
Cannot track more than one split image at a time
```

相似地，下列指令序列也將會失敗，因為分割的映像檔並非被追蹤的映像檔。

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

#### 4.4.13.8. 設定 RAID 錯誤政策

LVM RAID 會根據 **lv.conf** 檔案中的 **raid\_fault\_policy** 欄位所定義的偏好設定，來以自動的方式處理裝置錯誤。

- 若 **raid\_fault\_policy** 欄位設為了 **allocate**，系統將會嘗試以來自卷冊群組中的可用裝置，來替換失效的裝置。若沒有可用裝置，此情況將會回報給系統日誌。
- 若 **raid\_fault\_policy** 欄位設為了 **warn**，系統將會產生一則警告，並且日誌將會顯示裝置已失效。這能讓使用者判斷應進行什麼樣的動作。

只要有可用裝置足以支援使用性，RAID 邏輯卷冊便會繼續進行作業。



#### 4.4.13.8.1. 「allocate」 RAID 錯誤政策

在下列範例中，`lvm.conf` 檔案中的 `raid_fault_policy` 欄位已設為 `allocate`。RAID 邏輯卷冊的格式如下。

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

若 `/dev/sde` 裝置失效，系統日誌將會顯示錯誤訊息。

```
# grep lvm /var/log/messages
Jan 17 15:57:18 bp-01 lvm[8599]: Device #0 of raid1 array, my_vg-my_lv,
has failed.
Jan 17 15:57:18 bp-01 lvm[8599]: /dev/sde1: read failed after 0 of 2048 at
250994294784: Input/output error
Jan 17 15:57:18 bp-01 lvm[8599]: /dev/sde1: read failed after 0 of 2048 at
250994376704: Input/output error
Jan 17 15:57:18 bp-01 lvm[8599]: /dev/sde1: read failed after 0 of 2048 at
0:
Input/output error
Jan 17 15:57:18 bp-01 lvm[8599]: /dev/sde1: read failed after 0 of 2048 at
4096: Input/output error
Jan 17 15:57:19 bp-01 lvm[8599]: Couldn't find device with uuid
3lugiV-3eSP-AFAR-sdrP-H200-wM2M-qdMANY.
Jan 17 15:57:27 bp-01 lvm[8599]: raid1 array, my_vg-my_lv, is not in-sync.
Jan 17 15:57:36 bp-01 lvm[8599]: raid1 array, my_vg-my_lv, is now in-sync.
```

因為 `raid_fault_policy` 欄位已設為 `allocate`，因此失效的裝置將會被替換為來自於卷冊群組中的新裝置。

```
# lvs -a -o name,copy_percent,devices vg
Couldn't find device with uuid 3lugiV-3eSP-AFAR-sdrP-H200-wM2M-qdMANY.
LV          Copy%   Devices
lv          100.00  lv_rimage_0(0),lv_rimage_1(0),lv_rimage_2(0)
[lv_rimage_0]          /dev/sdh1(1)
[lv_rimage_1]          /dev/sdf1(1)
[lv_rimage_2]          /dev/sdg1(1)
[lv_rmeta_0]           /dev/sdh1(0)
[lv_rmeta_1]           /dev/sdf1(0)
[lv_rmeta_2]           /dev/sdg1(0)
```

請注意，即使故障的裝置已經被取代，以上訊息依舊顯示 LVM 無法找到故障的裝置。這是因為雖然故障裝置已經從 RAID 邏輯卷冊上移除，並不表示故障裝置已經從卷冊群組中移除。要從卷冊群組上移除故障的裝置，您可以執行 `vgreduce --removemissing VG`。

若 **raid\_fault\_policy** 已設為 **allocate**，不過卻無可用裝置的話，配置將會失敗，使邏輯卷冊維持現狀。若配置失敗，您可選擇修復磁碟，然後如 節 4.4.13.8.2, “「warn」RAID 錯誤政策 (Fault Policy)” 中所述地停用邏輯卷冊。此外，您亦可將失效的裝置替換掉，如 節 4.4.13.9, “替換 RAID 裝置” 中所述。

#### 4.4.13.8.2. 「warn」RAID 錯誤政策 (Fault Policy)

在下列範例中，**lvm.conf** 檔案中的 **raid\_fault\_policy** 欄位已設為 **warn**。RAID 邏輯卷冊的格式如下。

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

若 **/dev/sdh** 裝置失效，系統日誌將會顯示錯誤訊息。然而在此情況下，LVM 將不會自動透過替換映像檔，以嘗試修復 RAID 裝置。反之，若裝置失效，您可如下透過 **lvconvert** 指令的 **--repair** 引數來替換裝置。

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

# lvs -a -o name,copy_percent,devices my_vg
Couldn't find device with uuid fbI0Y0-GX7x-firU-Vy5o-vzwx-vAKZ-feRxFF.
LV          Copy%  Devices
my_lv       64.00
my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]      /dev/sde1(1)
[my_lv_rimage_1]      /dev/sdf1(1)
[my_lv_rimage_2]      /dev/sdg1(1)
[my_lv_rmeta_0]       /dev/sde1(0)
[my_lv_rmeta_1]       /dev/sdf1(0)
[my_lv_rmeta_2]       /dev/sdg1(0)
```

請注意，即使故障的裝置已經被取代，以上訊息依舊顯示 LVM 無法找到故障的裝置。這是因為雖然故障裝置已經從 RAID 邏輯卷冊上移除，並不表示故障裝置已經從卷冊群組中移除。要從卷冊群組上移除故障的裝置，您可以執行 **vgreduce --removemissing VG**。

如果裝置故障只是暫時的，或是您可以修復故障裝置，那麼您可以停用、然後啓用此邏輯卷冊，這樣裝置就不會被視為故障，如以下指令所示。

```
# lvchange -an my_vg/my_lv
```

```
# lvchange -ay my_vg/my_lv
```

一旦磁碟再次同步，就會被視為啟動。

#### 4.4.13.9. 替換 RAID 裝置

RAID 並不像傳統的 LVM 鏡射。LVM 鏡射需要移除故障的裝置，否則鏡射的邏輯卷冊就會當掉。RAID 陣列能在裝置故障上繼續運作。事實上，除了 RAID1 以外的 RAID，移除裝置表示會轉換到更低階的 RAID 等級（例如從 RAID6 變為 RAID5，或從 RAID4/RAID5 降到 RAID0）。因此，與其無條件移除故障裝置，並以新的裝置取代，LVM 能讓您透過使用 **lvconvert** 指令的 **--replace** 引數這單一步驟，取代 RAID 卷冊中的裝置。

**lvconvert --replace** 的格式如下。

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

以下範例會建立 RAID1 邏輯卷冊，然後取代該卷冊中的裝置。

```
# lvcreate --type raid1 -m2 -L 1G -n my_lv my_vg
Logical volume "my_lv" created
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       100.00  my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]      /dev/sdb1(1)
[my_lv_rimage_1]      /dev/sdb2(1)
[my_lv_rimage_2]      /dev/sdc1(1)
[my_lv_rmeta_0]        /dev/sdb1(0)
[my_lv_rmeta_1]        /dev/sdb2(0)
[my_lv_rmeta_2]        /dev/sdc1(0)
# lvconvert --replace /dev/sdb2 my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       37.50  my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]      /dev/sdb1(1)
[my_lv_rimage_1]      /dev/sdc2(1)
[my_lv_rimage_2]      /dev/sdc1(1)
[my_lv_rmeta_0]        /dev/sdb1(0)
[my_lv_rmeta_1]        /dev/sdc2(0)
[my_lv_rmeta_2]        /dev/sdc1(0)
```

以下範例會建立 RAID1 邏輯卷冊，然後取代該卷冊中的裝置、指定要使用哪個實體卷冊來取代。

```
# lvcreate --type raid1 -m1 -L 100 -n my_lv my_vg
Logical volume "my_lv" created
# lvs -a -o name,copy_percent,devices my_vg
LV          Copy%  Devices
my_lv       100.00  my_lv_rimage_0(0),my_lv_rimage_1(0)
[my_lv_rimage_0]      /dev/sda1(1)
[my_lv_rimage_1]      /dev/sdb1(1)
[my_lv_rmeta_0]        /dev/sda1(0)
[my_lv_rmeta_1]        /dev/sdb1(0)
# pvs
PV          VG          Fmt  Attr PSize  PFree
```

```

/dev/sda1    my_vg    lvm2 a--  1020.00m  916.00m
/dev/sdb1    my_vg    lvm2 a--  1020.00m  916.00m
/dev/sdc1    my_vg    lvm2 a--  1020.00m 1020.00m
/dev/sdd1    my_vg    lvm2 a--  1020.00m 1020.00m
# lvconvert --replace /dev/sdb1 my_vg/my_lv /dev/sdd1
# lvs -a -o name,copy_percent,devices my_vg
LV
my_lv          28.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
[my_lv_rimage_0]      /dev/sda1(1)
[my_lv_rimage_1]      /dev/sdd1(1)
[my_lv_rmeta_0]       /dev/sda1(0)
[my_lv_rmeta_1]       /dev/sdd1(0)

```

您可以透過指定多個 **replace** 引數，一次取代超過一組 RAID 裝置，如以下範例所示：

```

# lvcreate --type raid1 -m 2 -L 100 -n my_lv my_vg
Logical volume "my_lv" created
# lvs -a -o name,copy_percent,devices my_vg
LV
my_lv          100.00
my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]      /dev/sda1(1)
[my_lv_rimage_1]      /dev/sdb1(1)
[my_lv_rimage_2]      /dev/sdc1(1)
[my_lv_rmeta_0]       /dev/sda1(0)
[my_lv_rmeta_1]       /dev/sdb1(0)
[my_lv_rmeta_2]       /dev/sdc1(0)
# lvconvert --replace /dev/sdb1 --replace /dev/sdc1 my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
LV
my_lv          60.00
my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0]      /dev/sda1(1)
[my_lv_rimage_1]      /dev/sdd1(1)
[my_lv_rimage_2]      /dev/sde1(1)
[my_lv_rmeta_0]       /dev/sda1(0)
[my_lv_rmeta_1]       /dev/sdd1(0)
[my_lv_rmeta_2]       /dev/sde1(0)

```



### 注意

當您使用 **lvconvert --replace** 指令來指定取代用的磁碟時，取代磁碟不該從陣列中已使用的磁碟空間中分配。舉例來說，**lv\_rimage\_0** 與 **lv\_rimage\_1** 就不該位於同樣的實體卷冊上。

#### 4.4.14. 縮減邏輯卷冊

若要縮減邏輯卷冊的大小，首先請將檔案系統卸載。接著您便可使用 **lvreduce** 指令來縮減卷冊。將卷冊縮減過後，請將檔案系統重新掛載。



### 警告

有一點相當重要，那就是您在縮減卷冊本身時，您一定要先將檔案系統大小或是保存在卷冊中的一切都先縮小，否則您將面臨資料遺失的危機。

縮減邏輯卷冊可空出一些卷冊群組空間來分配給該卷冊群組中的其它邏輯卷冊。

下列範例縮減了位於 **vg00** 卷冊群組中的邏輯卷冊 **lv011** 的 3 個邏輯扇區。

```
# lvreduce -l -3 vg00/lv011
```

## 4.5. 透過過濾器來控制 LVM 裝置掃描 (LVM DEVICE SCANS)

在開機時，**vgscan** 指令會被執行來掃描系統上的區塊裝置以搜尋 LVM 標籤，判斷出它們哪一個才是實體卷冊，並讀取 metadata 來建立一系列卷冊群組清單。實體卷冊的名稱儲存在系統中各個節點的 cache 檔案中（**/etc/lvm/cache/.cache**）。後續的指令可讀取該檔案來避免重新掃描（rescanning）。

您可藉由在 **lvm.conf** 配置檔案中設定過濾器，以控制 LVM 該掃描哪些裝置。這些位於 **lvm.conf** 中的過濾器包含著一系列套用至 **/dev** 目錄中的裝置名稱的基本正規表示式，以決定是否要接受或拒絕找到的各個區塊裝置。

下列範例顯示了如何使用此過濾器，以控制 LVM 掃描哪些裝置。請注意，以下有些範例並不全然代表最佳作法，因為正規表示式被自由地拿來和完整的路徑名稱作比較。比方說，**a/loop/** 相當於 **a/\*.loop.\*** 並且將會與 **/dev/solooperation/lv011** 相符。

下列過濾器新增了所有被發現的裝置，這是預設的特性，因為在配置檔案中未配置過濾器：

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

下列過濾器一除了 **cdrom** 裝置以避免在光碟機中沒有光碟時所造成的延緩：

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

下列過濾器新增了所有的 **loop** 並移除了所有其它的區塊裝置：

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

下列過濾器新增了所有 **loop** 和 **IDE** 並移除了所有其它的區塊裝置：

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

下列過濾器只在第一個 IDE drive 上新增了分割區 8 並移除了所有其它的區塊裝置：

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

欲取得更多有關於 **lvm.conf** 檔案的相關資訊，請參閱〈[附錄 B, LVM 配置檔案](#)〉和 **lvm.conf(5)** man page。

## 4.6. 線上資料重置 (ONLINE DATA RELOCATION)

您可透過使用 **pvmove** 指令來在系統使用中的時候移動資料。

**pvmove** 這項指令會將要移至不同部份中的資料分開並建立一個暫時性的鏡像來移動各個部份。如欲取得更多有關於 **pvmove** 指令作業上的相關資訊，請查看 **pvmove(8) man page**。



### 注意

為了要進行叢集中的 **pvmove** 作業，您應該確定已安裝 **cmirror** 與 **cmirror-kmod** 套件，同時 **cmirror** 服務處於執行狀態。必須安裝的 **cmirror-kmod** 套件端視執行中的 kernel 核心而定。舉例來說，如果執行中的 kernel 是 **kernel-largesmp**，那就需要有 **cmirror-kmod-largesmp** 以對應 kernel 版本。

下列指令會將所有經過分配的空間由實體卷冊 **/dev/sdc1** 上移至卷冊群組中其它可使用的實體卷冊上：

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

下列指令只會移動邏輯卷冊 **MyLV** 的扇區。

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

因為 **pvmove** 指令的執行可能會花上一段時間，我們建議您在背景環境 (background) 中執行這項指令來避免完成度更新顯示在前景環境 (foreground) 中。下列指令會將所有分配至實體卷冊 **/dev/sdc1** 的扇區移至背景環境中的 **/dev/sdf1**。

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

下列指令會以每五秒間隔和百分比的方式來回報移動的完成度。

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

## 4.7. 在叢集中啟用各別節點上的邏輯卷冊

若您在一個叢集環境下安裝了 LVM 的話，您可能有時需要特別地在某個節點上啟用邏輯卷冊。

若要特別地在某個節點上啟用邏輯卷冊，請使用 **lvchange -aey** 這項指令。另外，您亦可使用 **lvchange -aly** 指令來只在本機節點上啟用邏輯卷冊，而非特別地啟用。您之後可同時地在額外的節點上啟用它們。

您亦能透過使用 LVM 標籤，來在各別節點上啟用邏輯卷冊，LVM 標籤的相關資訊位於〈[附錄 C, LVM 物件標籤 \(Object Tags\)](#)〉中。您亦可在配置檔案中指定節點的啟用，該配置檔案的相關資訊位於〈[附錄 B, LVM 配置檔案](#)〉中。

## 4.8. LVM 的自訂化回報

您可透過使用 **pvs**、**lvs** 和 **vgs** 指令來產生出簡明與可自訂化的 LVM 物件報告。這些指令所產生的報告包含著各個物件的一行輸出。各個行列都包含著一列和物件相關、經過排序的屬性的欄位。有五種方式可選擇欲回報的物件：藉由實體卷冊、卷冊群組、邏輯卷冊、實體卷冊區段，以及邏輯卷冊區段。

下列部份提供了：

- 您可使用來控制產生出的報告格式的指令引數之摘要。
- 您能為 LVM 物件選擇的欄位之清單。
- 您可使用來排序產生出的報告的指令引數之摘要。
- 指定回報輸出單位的指示。

#### 4.8.1. 格式控制

無論您是使用 **pvs**、**lvs** 或 **vgs** 指令都能看見顯示出的預設欄位和排序順序。您可藉由下列引數來控制這些指令的輸出：

- 您可藉由使用 **-o** 引數來改變預設顯示的欄位。比方說，下列輸出為 **pvs** 指令的預設輸出（它顯示出了有關於實體卷冊的相關資訊）。

```
# 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
```

下列指令只會顯示實體卷冊的名稱和大小。

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

- 您可透過一個加號 (+) 來將一個欄位附加至輸出，這能夠和 **-o** 引數一起組合使用。

下列範例除了會顯示預設的欄位還會顯示實體卷冊的 UUID。

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

- 新增 **-v** 引數至一項指令可增加額外的欄位。比方說，輸入 **pvs -v** 指令的話將會顯示出預設欄位以及 **DevSize** 和 **PV UUID** 這兩個額外的欄位。

```
# pvs -v
Scanning for physical volume names
PV          VG      Fmt  Attr PSize  PFree  DevSize PV UUID
/dev/sdb1   new_vg  lvm2 a-   17.14G 17.14G 17.14G onFF2w-1fLC-ughJ-D9eB-M7iv-6XqA-dqGeXY
/dev/sdc1   new_vg  lvm2 a-   17.14G 17.09G 17.14G Joqlch-yWSj-
```



```
kuEn-IdwM-01S9-X08M-mcpsVe
/dev/sdd1 new_vg lvm2 a- 17.14G 17.14G 17.14G yvfvZK-Cf31-
j75k-dECm-0RZ3-0dGW-tUqkCS
```

- **--noheadings** 引數會將標題的行列抑制住。這對於編寫 script 相當有幫助。

下列範例合併使用了 **--noheadings** 和 **pv\_name** 引數來產生了一列包含著所有實體卷冊的清單。

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

- **--separator 分隔符號** 這個引數使用了 **分隔符號** 來區隔了各個欄位。

下列範例透過了一個等於符號 (=) 來區分了 **pvs** 的預設輸出欄位。

```
# 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
```

若要在使用 **separator** 引數時讓欄位對稱的話，請合併使用 **separator** 引數以及 **--aligned** 引數。

```
# 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
```

您可透過使用 **lvs** 或 **vgs** 指令的 **-P** 引數，來顯示原本不會出現在輸出中的失敗卷冊相關資訊。欲取得該引數所產生之輸出的相關資訊，請查看 [〈節 6.2, “顯示錯誤裝置的相關資訊”〉](#)。

如欲取得顯示引數的完整清單，請查看 **pvs(8)**、**vgs(8)** 以及 **lvs(8)** man page。

卷冊群組欄位能與實體卷冊（和實體卷冊區段）欄位或邏輯卷冊（和邏輯卷冊區段）欄位混合在一起，不過實體卷冊與邏輯卷冊欄位則無法混合在一起。比方說，下列指令將會顯示各個實體卷冊的一行輸出。

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

#### 4.8.2. 物件選擇

此部份提供了一系列的表格，這些表格列出了您可透過使用 **pvs**、**vgs** 和 **lvs** 來顯示出有關於 LVM 物件的相關資訊。

為求方便，欄位名稱字首若符合指令的預設值的話便可被省略掉。比方說 **pvs** 指令，**name** 代表 **pv\_name**，不過對 **vgs** 指令來講，**name** 會被解譯為 **vg\_name**。

執行下列指令相當於執行 **pvs -o pv\_free**。

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

#### 4.8.2.1. pvs 指令

〈[表格 4.2, “pvs 顯示欄位”](#)〉列出了 **pvs** 指令的顯示引數，以及出現在標頭顯示中的欄位名稱與欄位描述。

表格 4.2. pvs 顯示欄位

引數	標頭	描述
<b>dev_size</b>	DevSize	實體卷冊所建立於的基本裝置上的大小
<b>pe_start</b>	第一個 PE	設為基本裝置中第一個實體扇區的起始的偏差值
<b>pv_attr</b>	Attr	實體卷冊狀態：(a)lllocatable 或 e(x)ported。
<b>pv_fmt</b>	Fmt	實體卷冊的 metadata 格式 ( <b>lvm2</b> 或 <b>lvm1</b> )
<b>pv_free</b>	PFree	實體卷冊上剩下的可用空間
<b>pv_name</b>	PV	實體卷冊名稱
<b>pv_pe_alloc_count</b>	Alloc	已使用的實體扇區數量
<b>pv_pe_count</b>	PE	實體扇區數量
<b>pvseg_size</b>	SSize	實體卷冊的區段大小
<b>pvseg_start</b>	起始	實體卷冊區段的起始實體扇區
<b>pv_size</b>	PSize	實體卷冊的大小
<b>pv_tags</b>	PV Tag	連接至實體卷冊的 LVM 標籤
<b>pv_used</b>	已使用	實體卷冊上目前已使用的空間
<b>pv_uuid</b>	PV UUID	實體卷冊的 UUID

**pvs** 指令就預設值會顯示下列欄

位：**pv\_name**、**vg\_name**、**pv\_fmt**、**pv\_attr**、**pv\_size**、**pv\_free**。它們是藉由 **pv\_name** 來排序的。

```
# 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 **-v** 引數以及 **pvs** 指令可將下列欄位附加至預設的輸出：**dev\_size**、**pv\_uuid**。

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

您可使用 **pvs** 指令的 **--segments** 引數來顯示有關於各個實體卷冊區段的相關資訊。區段相當於扇區的群組。區段視點 (segment view) 可有效幫助您查看您的邏輯卷冊是否已分散掉。

**pvs --segments** 指令就預設值會顯示下列欄

位：**pv\_name**、**vg\_name**、**pv\_fmt**、**pv\_attr**、**pv\_size**、**pv\_free**、**pvseg\_start**、**pvseg\_size**。它們是藉由實體卷冊中的 **pv\_name** 和 **pvseg\_size** 來排序的。

```
# 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
```

您可使用 **pvs -a** 指令來查看 LVM 所偵測到不過尚未被初始化為 LVM 實體卷冊的裝置。

```
# 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

```

#### 4.8.2.2. vgs 指令

〈表格 4.3, “vgs 顯示欄位”〉列出了 **vgs** 指令的顯示引數、出現在標頭顯示中的欄位名稱，以及欄位的描述。

表格 4.3. vgs 顯示欄位

引數	標頭	描述
<b>lv_count</b>	#LV	卷冊群組中所包含的邏輯卷冊數量
<b>max_lv</b>	MaxLV	卷冊群組中可允許的最大邏輯卷冊數量（若無限制的話便是 0）
<b>max_pv</b>	MaxPV	卷冊群組中可允許的最大實體卷冊數量（若無限制的話便是 0）
<b>pv_count</b>	#PV	定義卷冊群組的實體卷冊數量
<b>snap_count</b>	#SN	卷冊群組所包含的 snapshot 數量
<b>vg_attr</b>	Attr	卷冊群組的狀態：可寫入（w）、唯讀（r）、可重設大小（z）、已匯出（x）、部分的（p）以及已聚成叢集（c）。
<b>vg_extent_count</b>	#Ext	卷冊群組中的實體扇區數量
<b>vg_extent_size</b>	Ext	卷冊群組中的實體扇區大小
<b>vg_fmt</b>	Fmt	卷冊群組的 metadata 格式（ <b>lvm2</b> 或 <b>lvm1</b> ）
<b>vg_free</b>	VFree	卷冊群組中剩下的可用空間大小
<b>vg_free_count</b>	Free	卷冊群組中可使用的實體扇區數量

引數	標頭	描述
<b>vg_name</b>	VG	卷冊群組名稱
<b>vg_seqno</b>	Seq	表示卷冊群組修訂版本的號碼
<b>vg_size</b>	VSize	卷冊群組的大小
<b>vg_sysid</b>	SYS ID	LVM1 System ID
<b>vg_tags</b>	VG Tags	連接至卷冊群組的 LVM 標籤
<b>vg_uuid</b>	VG UUID	卷冊群組的 UUID

**vgs** 指令就預設值會顯示下列欄

位：**vg\_name**、**pv\_count**、**lv\_count**、**snap\_count**、**vg\_attr**、**vg\_size**、**vg\_free**。它們是藉由 **vg\_name** 來排序的。

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

使用 **vgs** 指令和 **-v** 引數會將下列欄位新增至預設的畫面：**vg\_extent\_size**、**vg\_uuid**。

```
# vgs -v
Finding all volume groups
Finding volume group "new_vg"
VG      Attr   Ext   #PV #LV #SN VSize  VFree  VG UUID
new_vg wz--n- 4.00M   3   1   1 51.42G 51.36G jxQJ0a-ZKk0-0pM0-0118-
nlw0-wwqd-fD5D32
```

#### 4.8.2.3. lvs 指令

〈表格 4.4, “lvs 顯示欄位”〉列出了 **lvs** 指令的顯示引數，以及它所出現在標頭顯示中的欄位名稱和該欄位的描述。

表格 4.4. lvs 顯示欄位

引數	標頭	描述
<b>chunksize</b> <b>chunk_size</b>	Chunk	Snapshot 卷冊中的單位大小
<b>copy_percent</b>	Copy%	鏡像卷冊的同步化百分比；在實體卷冊被透過 <b>pv_move</b> 指令移動時也會使用到

引數	標頭	描述
<b>devices</b>	裝置	構成邏輯卷冊的基本裝置：實體卷冊、邏輯卷冊，以及起始實體扇區和邏輯扇區
<b>lv_attr</b>	Attr	<p>邏輯卷冊的狀態。邏輯卷冊的 attribute bit 如下：</p> <div> <p>Bit 1：卷冊類型：(m)irrored、(M)irrored 並無初始同步、(o)rigin、(O)rigin 並含有合併的 snapshot、(r)aid、(R)aid 並無初始同步、(s)napshot、合併 (S)napshot、(p)vmove、(v)irtual、鏡像或 raid (i)mage、鏡像或 raid (l)mage 未同步、mirror (l)og 裝置、(c)onversion 中、精簡 (V)olume、(t)hin 集區、(T)hin 集區資料、raid 或是精簡集區 m(e)tadata</p> <p>Bit 2：權限：(w)riteable、(r)ead-only、(R)ead-only 啓用非唯讀卷冊的唯讀權限</p> <p>Bit 3：配置政策：(c)ontiguous、c(l)ing、(n)ormal、(a)nywhere、(i)nherited。若卷冊目前被鎖定而不可進行配置變更，這些字母將會是大寫字母，比方說正在執行 <b>pvmove</b> 指令時。</p> <p>Bit 4：固定的 (m)inor</p> <p>Bit 5：狀態：(a)ctive、(s)uspended、(l)invalid snapshot、無效的 (S)uspended snapshot、snapshot (m)erge 失敗、已中止的 snapshot (M)erge 失敗、對映的 (d)evice 存在卻無表格、對映的裝置存在而表格乃 (i)nactive</p> <p>Bit 6：裝置 (o)pen</p> <p>Bit 7：目標類型：(m)irror、(r)aid、(s)napshot、(t)hin、(u)nknown、(v)irtual。這會將與相同 kernel 目標的邏輯卷冊分組在一起。因此，例如 mirror image、mirror log 以及 mirror 本身，因使用了原始的 device-mapper 鏡像 kernel 驅動程式而顯示為 (m)，而使用了 md raid kernel 驅動程式的 raid 相等的項目則顯示為 (r)。使用原始 device-mapper 驅動程式的 snapshot 會顯示為 (s)，而使用精簡佈建 (thin provisioning) 驅動程式的 thin 卷冊則會顯示為 (t)。</p> <p>Bit 8：新分配的資料區塊會在使用前，被充滿「零」(z) 的區塊覆蓋過去。</p> </div>
<b>lv_kernel_major</b>	KMaj	邏輯卷冊的實際主要裝置代碼（若未啓用的話就是 -1）
<b>lv_kernel_minor</b>	KMIN	邏輯卷冊的實際鏡像裝置代碼（若未啓用的話就是 -1）

引數	標頭	描述
<b>lv_major</b>	Maj	邏輯卷冊的持續性主要裝置代碼（若未指定的話就是 -1）
<b>lv_minor</b>	Min	邏輯卷冊的持續性次要裝置代碼（若未指定的話就是 -1）
<b>lv_name</b>	LV	邏輯卷冊名稱
<b>lv_size</b>	LSize	邏輯卷冊大小
<b>lv_tags</b>	LV Tags	連接至邏輯卷冊的 LVM 標籤
<b>lv_uuid</b>	LV UUID	邏輯卷冊的 UUID。
<b>mirror_log</b>	Log	鏡像 log 所位於的裝置
<b>modules</b>	模組	使用此邏輯卷冊所需要的相應 kernel 裝置映射（device-mapper）目標
<b>move_pv</b>	移動	提供一個透過 <b>pvmove</b> 指令來建立的暫時性邏輯卷冊的實體卷冊
<b>origin</b>	起源	snapshot 卷冊的起源裝置
<div>regionsize</div> <div>region_size</div>	範圍	鏡像邏輯卷冊的單位大小
<b>seg_count</b>	#Seg	邏輯卷冊中的區段數量
<b>seg_size</b>	SSize	邏輯卷冊中的區段大小
<b>seg_start</b>	起始	邏輯卷冊中的區段偏差值
<b>seg_tags</b>	Seg Tags	連接至邏輯卷冊之區段的 LVM 標籤
<b>segtype</b>	類型	邏輯卷冊的區段類型（例如：mirror、striped、linear）
<b>snap_percent</b>	Snap%	Snapshot 目前的使用量百分比
<b>stripes</b>	#Str	邏輯卷冊中的磁帶或鏡像數量
<div>stripesize</div> <div>stripe_size</div>	磁帶	等量邏輯卷冊中的磁條單位大小



**lvs** 指令就預設值會顯示下列欄

位：**lv\_name**、**vg\_name**、**lv\_attr**、**lv\_size**、**origin**、**snap\_percent**、**move\_pv**、**mirror\_log**、**copy\_percent**、**convert\_lv**。它們就預設值會被藉由卷冊群組中的 **vg\_name** 和 **lv\_name** 來排序。

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

使用 **lvs** 指令以及 **-v** 引數便能將下列欄位新增至預設的畫面

中：**seg\_count**、**lv\_major**、**lv\_minor**、**lv\_kernel\_major**、**lv\_kernel\_minor**、**lv\_uuid**。

```
# lvs -v
Finding all logical volumes
LV          VG      #Seg Attr   LSize  Maj  Min  KMaj  KMin  Origin Snap%
Move Copy%  Log Convert LV UUID
lvol0       new_vg    1 owi-a- 52.00M  -1  -1  253   3
LBy1Tz-sr23-0jsI-LT03-nHLC-y8XW-EhCl78
newvgsnap1  new_vg    1 swi-a- 8.00M  -1  -1  253   5   lvol0    0.20
1ye10U-1cIu-o79k-20h2-ZGF0-qCJm-CfbsIx
```

您可使用 **lvs** 指令的 **--segments** 引數來顯示含有強調區段資訊的預設欄位的相關資訊。當您使用

**segments** 引數時，**seg** 前綴字元為非必要的。**lvs --segments** 指令就預設值會顯示下列欄

位：**lv\_name**、**vg\_name**、**lv\_attr**、**stripes**、**segtype**、**seg\_size**。畫面就預設值會藉由卷冊群組中的 **vg\_name**、**lv\_name** 以及邏輯卷冊中的 **seg\_start** 來排序。若邏輯卷冊被分散的話，這項指令的輸出便會將其顯示出來。

```
# 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
```

若使用 **lvs --segments** 指令以及 **-v** 引數便可新增下列欄位至預設的畫面

中：**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
```

下列範例顯示了 **lvs** 指令在一部配置了一個邏輯卷冊的系統上的預設輸出，以及 **lvs** 指令和 **segments** 引數的預設輸出。

```
# 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
lvvol0  new_vg  -wi-a-    1 linear 52.00M
```

### 4.8.3. 排序 LVM 報告

通常 **lvs**、**vgs** 或 **pvs** 指令的所有輸出都要先被產生並儲存於內部之後才可進行排序並且行列才可正確對稱。您可指定 **--unbuffered** 這個引數來在未排序的輸出一被產生時即刻將它顯示出。

若要指定另一列行列來進行排序的話，請使用任何回報指令的 **-o** 引數。無須在輸出中包含這些欄位。

下列範例顯示了 **pvs** 指令的輸出，它顯示了實體卷冊的名稱、大小以及可用空間。

```
# 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
```

下列範例顯示了相同的輸出，並藉由可用空間的欄位來排序。

```
# 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
```

下列範例顯示了您無須顯示您正在排序的欄位。

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

若要顯示反向的排序，在 **-O** 引數之後，於您所指定的欄位之前前置 **-** 這個字元。

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

### 4.8.4. 指定單位

若要為 LVM 回報顯示指定單位，請使用回報指令的 **--units** 引數。您可指定 (b)ytes、(k)ilobytes、(m)egabytes、(g)igabytes、(t)erabytes、(e)xabytes、(p)etabytes、以及 (h)uman-readable。預設的顯示為 human-readable。您可透過在 **lvm.conf** 檔案的 **global** 部份中設定 **units** 這個參數來置換預設值。

下列範例指定了 **pvs** 指令的輸出為 MB 而非預設的 GB。

```
# pvs --units m
```

```

PV          VG      Fmt  Attr PSize      PFree
/dev/sda1    VG      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

```

就預設值，單位會以 2 的因子（1024 的倍數）來顯示出。您可藉由將單位規格（B、K、M、G、T、H）大寫化來將單位指定為 1000 的倍數。

下列指令會將輸出顯示為 1024 的倍數，這是預設的特性。

```

# 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

```

下列指令會將輸出顯示為 1000 的倍數。

```

# 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

```

您亦可指定區段（s）（定義為 512 位元組）或是自訂單位。

下列範例將 **pvs** 指令的輸出顯示成了幾個磁區。

```

# 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

```

下列範例會將 **pvs** 指令的輸出以 4 MB 為單位的方式顯示出。

```

# 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

```

## 章 5. LVM 配置範例

本章節提供了一些基本的 LVM 配置範例。

### 5.1. 在三個磁碟上建立一個 LVM 邏輯卷冊

此範例建立了一個稱為 **new\_logical\_volume** 的 LVM 邏輯卷冊，它包含著位於 **/dev/sda1**、**/dev/sdb1** 以及 **/dev/sdc1** 的磁碟。

#### 5.1.1. 建立實體卷冊（Physical Volumes）

若要使用卷冊群組中的磁碟，您必須將它們標記為 LVM 實體卷冊。



#### 警告

這項指令會將 **/dev/sda1**、**/dev/sdb1** 以及 **/dev/sdc1** 上的所有資料都損毀。

```
# 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. 建立卷冊群組

下列指令將會建立卷冊群組 **new\_vol\_group**。

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

您可使用 **vgs** 指令來顯示新卷冊群組的屬性。

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

#### 5.1.3. 建立邏輯卷冊

下列指令會由 **new\_vol\_group** 卷冊群組建立邏輯卷冊 **new\_logical\_volume**。此範例會建立一個使用卷冊群組 2GB 的邏輯卷冊。

```
# lvcreate -L2G -n new_logical_volume new_vol_group
Logical volume "new_logical_volume" created
```

#### 5.1.4. 建立檔案系統

下列指令可在邏輯卷冊上建立一個 GFS2 檔案系統。

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

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

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

Syncing...
All Done
```

下列指令會將邏輯卷冊掛載並回報檔案系統磁碟空間上的使用量。

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

## 5.2. 建立一個磁條邏輯卷冊 (STRIPED LOGICAL VOLUME)

此範例建立了一個稱為 **striped\_logical\_volume** 的 LVM 磁條邏輯卷冊 (LVM striped logical volume)，它會將資料分散到 **/dev/sda1**、**/dev/sdb1** 和 **/dev/sdc1** 磁碟上去。

### 5.2.1. 建立實體卷冊 (Physical Volumes)

標記您將會在卷冊群組中使用來作為 LVM 實體卷冊的磁碟。



#### 警告

這項指令會將 **/dev/sda1**、**/dev/sdb1** 以及 **/dev/sdc1** 上的所有資料都損毀。

```
# 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. 建立卷冊群組

下列指令會建立卷冊群組 **volgroup01**。

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

您可使用 **vgs** 指令來顯示新卷冊群組的屬性。

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

### 5.2.3. 建立邏輯卷冊

下列指令會由 **volgroup01** 這個卷冊群組建立磁條邏輯卷冊 **striped\_logical\_volume**。此範例會建立一個大小為 2GB 的邏輯卷冊，它將含有三個磁條，並且每個磁條大小為 4 KB。

```
# lvcreate -i3 -l4 -L2G -nstriped_logical_volume volgroup01
Rounding size (512 extents) up to stripe boundary size (513 extents)
Logical volume "striped_logical_volume" created
```

### 5.2.4. 建立檔案系統

下列指令可在邏輯卷冊上建立一個 GFS2 檔案系統。

```
# mkfs.gfs2 -plock_nolock -j 1 /dev/volgroup01/striped_logical_volume
This will destroy any data on /dev/volgroup01/striped_logical_volume.
```

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

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

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

下列指令會將邏輯卷冊掛載並回報檔案系統磁碟空間上的使用量。

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

## 5.3. 分割卷冊群組

在此範例中，有個現有的卷冊群組包含了三個實體卷冊。實體卷冊上若有足夠的未使用空間的話，您便可在不新增磁碟的情況下新建卷冊群組。

在初始設定中，邏輯卷冊 **mylv** 是由 **myvol** 這個卷冊群組所分割的，並且它包含著 **/dev/sda1**、**/dev/sdb1** 以及 **/dev/sdc1** 這三個實體卷冊。

完成了這項程序後，**myvg** 這個卷冊群組便會包含著 **/dev/sda1** 和 **/dev/sdb1**。第二個邏輯群組 **yourvg** 則會包含著 **/dev/sdc1**。

### 5.3.1. 判斷可用空間

您可透過使用 **pvscan** 指令來判斷出卷冊群組中目前有多少可用空間。

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

### 5.3.2. 移動資料

您可透過使用 **pvmove** 指令來將 **/dev/sdc1** 中所有已使用的實體扇區移至 **/dev/sdb1**。**pvmove** 指令的執行可能會花上一段時間。

```
# pvmove /dev/sdc1 /dev/sdb1
/dev/sdc1: Moved: 14.7%
/dev/sdc1: Moved: 30.3%
/dev/sdc1: Moved: 45.7%
/dev/sdc1: Moved: 61.0%
/dev/sdc1: Moved: 76.6%
/dev/sdc1: Moved: 92.2%
/dev/sdc1: Moved: 100.0%
```

移動了資料之後，您便會看見 **/dev/sdc1** 上的所有空間都可使用。

```
# 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. 分割卷冊群組

若要建立新的卷冊群組 **yourvg**，請使用 **vgsplit** 指令來分割卷冊群組 **myvg**。

在您能夠分割卷冊群組之前，邏輯卷冊必須被停用。若檔案系統已被掛載，您必須在停用邏輯卷冊前先將檔案系統卸載。

您可透過使用 **lvchange** 指令或是 **vgchange** 指令來停用邏輯卷冊。下列指令可停用邏輯卷冊 **mylv** 然後由 **myvg** 卷冊群組分割 **yourvg** 卷冊群組，然後將實體卷冊 **/dev/sdc1** 移至新的卷冊群組 **yourvg** 中。



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

您可使用 **vgs** 指令來查看這兩個卷冊群組的屬性。

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

### 5.3.4. 建立新的邏輯卷冊

在建立了新的卷冊群組之後，您便可新建邏輯卷冊 **yourlv**。

```
# lvcreate -L5G -n yourlv yourvg
Logical volume "yourlv" created
```

### 5.3.5. 製作檔案系統和掛載新的邏輯卷冊

您可在新的邏輯卷冊上建立檔案系統然後將它掛載。

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

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

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

Syncing...
All Done

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

### 5.3.6. 啓用和掛載原本的邏輯卷冊

因為您必須停用邏輯卷冊 **mylv**，因此在您掛載它之前您必須再次啓用它。

```
# 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. 由邏輯卷冊中移除磁碟

此範例顯示了如何將磁碟由現有的邏輯卷冊中移除（為了更換磁碟或是使用該磁碟來作為不同卷冊的一部分）。若要移除磁碟，您首先必須將 LVM 實體卷冊上的扇區移至一個或一組不同的磁碟中。

### 5.4.1. 將扇區移至現有的實體卷冊中

在此範例中，邏輯卷冊被分配至 **myvg** 卷冊群組中的四個實體卷冊上。

```
# 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
```

我們希望將 **/dev/sdb1** 的扇區移除，如此一來我們才能將該磁碟由卷冊群組中移除。

若卷冊群組中的其它實體卷冊上有足夠的可用扇區的話，您可針對於您希望移除的裝置執行 **pvmove** 指令並且不使用其它選項，如此一來扇區便會被分配至其它裝置上。

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

當 **pvmove** 指令執行完成後，扇區的分配會如下所示：

```
# 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
```

請使用 **vgreduce** 指令來將 **/dev/sdb1** 實體卷冊由卷冊群組中移除。

```
# 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
```

磁碟現在可被實體移除或分配給其他用戶。

### 5.4.2. 將扇區移至新磁碟上

在此範例中，邏輯卷冊分配在 **myvg** 卷冊群組中的三個實體卷冊上：

```
# 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
```

我們希望將 **/dev/sdb1** 的扇區移至新的 **/dev/sdd1** 裝置上。

#### 5.4.2.1. 新建實體卷冊

由 **/dev/sdd1** 建立新的實體卷冊。

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

#### 5.4.2.2. 新增實體卷冊至卷冊群組中

新增 **/dev/sdd1** 至現有的卷冊群組 **myvg** 中。

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

#### 5.4.2.3. 移動資料

使用 **pvmove** 指令來將資料由 **/dev/sdb1** 移至 **/dev/sdd1**。

```
# 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. 將舊的實體卷冊由卷冊群組中移除

當您將資料由 **/dev/sdb1** 中移走後，您便可將它由卷冊群組中移除。

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

您現在已能將磁碟重新分配至另一個卷冊群組或將磁碟由系統中移除。

## 5.5. 在叢集中建立鏡像 LVM 邏輯卷冊

若在叢集中建立鏡像 LVM 邏輯卷冊，需使用與在單獨節點上建立鏡像 LVM 邏輯卷冊相同的指令和程序。然而，若要在叢集中建立鏡像 LVM 卷冊，該叢集和叢集鏡像架構必須要執行，叢集必須要 quorate，並且 **lvm.conf** 檔案中的鎖定類型並需正確設定，以允許直接或透過 **lvmconf** 指令啟用叢集鎖定（如 [節 3.1](#)，“在叢集中建立 LVM 卷冊”中所描述）。

下列程序將在叢集中建立鏡像邏輯卷冊。首先，程序會先查看叢集服務是否已安裝並在執行，接著程序便會建立鏡像卷冊。

1. 若要建立一個讓叢集中所有節點皆能共享的鏡像邏輯卷冊，鎖定類型必須正確設定於叢集裡各個節點上的 **lvm.conf** 檔案中。就預設值，鎖定類型會被設為本機。若要進行變更，請在叢集中的各個節點上執行下列指令，以啟用叢集鎖定：

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

2. 若要建立叢集化的邏輯卷冊，叢集架構必須在叢集中的各個節點上啟用並運作。下列範例驗證了 **clvmd** daemon 正在啟用它的節點上運作：

```
ps auxw | grep clvmd
root      17642  0.0  0.1 32164 1072 ?        Ssl  Apr06   0:00
clvmd -T20 -t 90
```

下列指令將顯示叢集狀態的本機檢視：

```
# cman_tool services
fence domain
member count 3
victim count 0
victim now 0
master nodeid 2
wait state none
members 1 2 3

dlm lockspaces
name clvmd
id 0x4104eeefa
flags 0x00000000
change member 3 joined 1 remove 0 failed 0 seq 1,1
members 1 2 3
```

3. 確認 **cmirror** 套件已安裝。

4. 啟用 **cmirrord** 服務

```
# service cmirrord start
Starting cmirrord: [ OK
]
```

5. 建立鏡像。第一個步驟就是建立實體卷冊。下列指令將建立三個實體卷冊。兩個實體卷冊將使用於鏡像 **leg**，並且第三個實體卷冊將會包含鏡像 **log**。

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

6. 建立卷冊群組。此範例將建立包含了先前步驟所建立的三個實體卷冊的卷冊群組 **vg001**。

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

請注意 **vgcreate** 指令的輸出顯示了卷冊群組已叢集化。您可藉由 **vgs** 指令來驗證卷冊群組是否已叢集化，這將會顯示卷冊群組的屬性。若卷冊群組已叢集化，它將會顯示 **c** 屬性。

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

7. 建立鏡像邏輯卷冊。此範例將由卷冊群組 **vg001** 建立邏輯卷冊 **mirrorlv**。此卷冊含有一個鏡像 leg。此範例指定了實體卷冊的哪個磁區將會被使用來作為邏輯卷冊。

```
# lvcreate -l 1000 -m1 vg001 -n mirrorlv /dev/xvdb1:1-1000
/dev/xvdb2:1-1000 /dev/xvdc1:0
Logical volume "mirrorlv" created
```

您可使用 **lvs** 指令來顯示鏡像建立的進度。下列範例顯示了鏡像已同步化 47%、91%，並且當鏡像完成後便成為 100% 同步化。

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

鏡像的完成會記載於系統日誌中：

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

8. 您可搭配使用 **lvs** 與 **-o +devices** 選項來顯示鏡像的配置，包括構成鏡像 leg 的裝置有哪些。您可看見此範例中的邏輯卷冊是以兩個線性映像和一個 log 所組成的。

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

您可使用 **lvs** 指令的 **seg\_pe\_ranges** 選項來顯示資料配置。您可使用此選項來驗證您的配置是否有正確冗餘。這項指令的輸出會顯示 PE 範圍，並且格式與 **lvcreate** 和 **lvresize** 指令的輸入相同。

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



### 注意

欲取得由失效的 LVM 鏡像卷冊之 leg 進行復原的相關資訊，請參閱 [節 6.3, “由 LVM 鏡像錯誤中復原”](#)。

## 章 6. LVM 疑難排解

此章節提供了針對於各種 LVM 問題進行疑難排解的指南。

### 6.1. 疑難排解診斷結果

若指令不如預期地運作的話，您可透過下列方式來取得診斷結果：

- 使用任何指令的 **-v**、**-vv**、**-vvv** 或 **-vvvv** 引數來取得更加詳細的輸出。
- 若問題和邏輯卷冊的啟動相關的話，請在配置檔案的「log」部份中設置「activation = 1」然後以 **-vvvv** 引數來執行指令。當您檢查過了這些輸出時，請記得將這個參數重設為 0 以避免發生機器在記憶體不足的情況下將機器鎖住的問題。
- 請執行 **lvmdump** 指令，它提供了用來作為診斷用的資訊傾印。如欲取得更多相關資訊，請參閱 **lvmdump(8)** man page。
- 您可執行 **lvs -v**、**pvs -a** 或 **dmsetup info -c** 指令來取得額外的系統資訊。
- 在 **/etc/lvm/backup** 檔案中（或於 **/etc/lvm/archive** 檔案中的壓縮版本）檢查 metadata 的最後一個備份。
- 透過執行 **lvm dumpconfig** 指令來檢查目前的配置資訊。
- 檢查 **/etc/lvm** 目錄中的 **.cache** 檔案來取得有關於哪個裝置上含有實體卷冊的相關記錄。

### 6.2. 顯示錯誤裝置的相關資訊

您可使用 **lvs** 或 **vgs** 指令的 **-P** 引數來顯示原本不會出現在輸出中，有關於錯誤卷冊的相關資訊。此引數允許某些作業的進行，儘管內部的 metadata 並不完全地一致。比方說，若構成卷冊群組 **vg** 的其中一個裝置發生錯誤的話，**vgs** 指令可能會顯示下列輸出。

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

若您指定了 **vgs** 指令的 **-P** 引數的話，卷冊群組還是無法使用，不過您可看見更多有關於錯誤裝置的相關資訊。

```
# 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)
```

在此範例中，錯誤的裝置會造成卷冊群組中的線性和等量邏輯卷冊發生錯誤。缺少了 **-P** 引數的 **lvs** 指令會顯示下列輸出。

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

使用 **-P** 引數來顯示錯誤的邏輯卷冊。

```
# 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)
```

下列範例顯示了當某個鏡像邏輯卷冊的 `leg` 發生錯誤時而使用了 `-P` 引數的 `pvs` 和 `lvs` 指令的輸出。

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

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

### 6.3. 由 LVM 鏡像錯誤中復原

本節提供的範例闡述了 LVM 鏡射卷冊之一因為下方的實體卷冊失效，且 `mirror_log_fault_policy` 參數設為 `remove`，需要您手動重建鏡射之復原範例。欲知設定 `mirror_log_fault_policy` 參數的方法，請參閱〈[節 6.3, “由 LVM 鏡像錯誤中復原”](#)〉。

當一個鏡像 `leg` 發生錯誤時，LVM 便會將鏡像卷冊轉換為線性卷冊，它會和先前一樣地繼續進行作業，不過缺少鏡像的重複。在此情況下，您可新增一個新的磁碟裝置至系統，以使用它來取代實體裝置，並重建鏡像。

下列指令建立了將會使用於鏡像的實體卷冊。

```
# 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
```

下列指令建立了卷冊群組 **vg** 和鏡像卷冊 **groupfs**。

```
# 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
```

您可使用 **lvs** 指令來驗證 mirror leg 和 mirror log 的基本裝置和鏡像卷冊的格式。請注意，在第一個範例中，鏡像還未完整同步化；您應等到 **Copy%** 這個欄位顯示了 100.00 之後才繼續進行。

```
# 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)
```

在此範例中，**/dev/sda1** 這個鏡像的主要 leg 發生了錯誤。任何寫入至鏡像卷冊的動作都會造成 LVM 偵測到錯誤的鏡像。當這情況發生時，LVM 會將鏡像轉換為單獨的線性卷冊。在此情況下，若要進行轉換的話，請執行 **dd** 這項指令

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

您可使用 **lvs** 指令來驗證裝置現在是否是個 linear 裝置。因為磁碟發生錯誤，因而產生了 I/O 錯誤。

```
# 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)

```

在此情況下，您還是應該能使用邏輯卷冊，不過將不會有鏡像重複。

若要重建鏡像卷冊，請替換掉損壞的磁碟，並重建實體卷冊。若您使用的是相同的磁碟，而沒有將它替換為新的磁碟的話，那麼當您執行 **pvcreate** 指令時，您將會看見「inconsistent (不一致)」的警告。您可藉由執行 **vgreduce --removemissing** 指令，來避免這項警告出現。

```

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

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

```

接下來請透過新的實體卷冊來延伸原始的卷冊群組。

```

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

# pvscan
PV /dev/sdb1    VG vg      lvm2 [67.83 GB / 67.10 GB free]
PV /dev/sdb2    VG vg      lvm2 [67.83 GB / 67.83 GB free]
PV /dev/sdc1    VG vg      lvm2 [67.83 GB / 67.83 GB free]
PV /dev/sdc2    VG vg      lvm2 [67.83 GB / 67.83 GB free]
PV /dev/sdd1    VG vg      lvm2 [67.83 GB / 67.83 GB free]
PV /dev/sdd2    VG vg      lvm2 [67.83 GB / 67.83 GB free]
PV /dev/sde1    VG vg      lvm2 [67.83 GB / 67.83 GB free]
PV /dev/sde2    VG vg      lvm2 [67.83 GB / 67.83 GB free]
PV /dev/sdf1    VG vg      lvm2 [67.83 GB / 67.83 GB free]
PV /dev/sdf2    VG vg      lvm2 [67.83 GB / 67.83 GB free]
PV /dev/sdg1    VG vg      lvm2 [67.83 GB / 67.83 GB free]
PV /dev/sdg2    VG vg      lvm2 [67.83 GB / 67.83 GB free]
PV /dev/sdh1    VG vg      lvm2 [67.83 GB / 67.83 GB free]
PV /dev/sdh2    VG vg      lvm2 [67.83 GB / 67.83 GB free]

```

```
PV /dev/sdi1   VG vg    lvm2 [603.93 GB / 603.93 GB free]
PV /dev/sdi2   VG vg    lvm2 [603.93 GB / 603.93 GB free]
Total: 16 [2.11 TB] / in use: 16 [2.11 TB] / in no VG: 0 [0  ]
```

將 linear 卷冊轉換回它原始的鏡像狀態。

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

您可使用 **lvs** 指令來驗證鏡像是否已被儲存。

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

## 6.4. 復原實體卷冊的 METADATA

若實體卷冊的卷冊群組 metadata 區域不小心被覆寫或損毀的話，您將會看見一則顯示 metadata 區域不正確，或是系統無法找到含有某個 UUID 的實體卷冊的錯誤訊息。您可藉由在實體卷冊上編寫新的 metadata 區域，指定和遺失的 metadata 相同的 UUID 來恢復實體卷冊的資料。



### 警告

您不該以一個可運作的 LVM 邏輯卷冊來嘗試這項程序。若您指定了錯誤的 UUID，您將會遺失您的資料。

下列範例顯示了當 metadata 遺失或損毀時您所可能會看見的輸出。

```
# 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.
...
```

您可能能夠透過查看 **/etc/lvm/archive** 目錄來找尋被覆寫的實體卷冊的 UUID。若要找尋特定卷冊群組最後一個已知、有效、壓縮過的 LVM metadata，請查看 **VolumeGroupName\_xxxx.vg** 檔案。

此外，您可能會發現停用卷冊並設置 **partial (-P)** 引數能讓您找到遺失或損毀的實體卷冊的 UUID。

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

您可使用 **pvccreate** 指令的 **--uuid** 和 **--restorefile** 引數來復原實體卷冊。下列範例已將 **/dev/sdh1** 裝置標記為實體卷冊和以上所顯示的 UUID (**FmGRh3-zhok-iVI8-7qTD-S5BI-MAEN-NYM5Sk**)。這項指令會透過包含在 **VG\_00050.vg** 中的 metadata 資訊 (卷冊群組最近期良好封存的 metadata) 來將實體卷冊的標籤恢復。**restorefile** 引數會指示 **pvccreate** 指令使新的實體卷冊能和卷冊群組上的舊實體卷冊相容，並確保新的 metadata 不會被放置在舊實體卷冊包含著資料的位置上 (這是有可能會發生的，比方說在原始的 **pvccreate** 指令有使用控制 metadata 定位的指令列引數的情況下，或是實體卷冊原本是透過使用不同預設值的不同版本軟體來建立的情況下)。**pvccreate** 指令只會將 LVM metadata 區域覆寫，而不會影響到現有的資料區域。

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

接著您可使用 **vgcfgrestore** 指令來儲存卷冊群組的 metadata。

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

您現在能夠顯示邏輯卷冊了。

```
# 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)
```

下列指令會啟用卷冊並顯示啟用的卷冊。

```
# 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)
```

若 on-disk 的 LVM metadata 所需容量和將它覆寫的資料相同，這項指令便可將實體卷冊復原。若將 metadata 覆寫的資料超過了 metadata 範圍，那麼卷冊上的資料就可能會被影響到。您可能能夠透過使用 **fsck** 指令來將資料復原。

## 6.5. 替換一個遺失的實體卷冊

若實體卷冊發生錯誤或是需要被置換的話，您可透過和您用來復原實體卷冊 metadata 所進行的相同程序，來標記一個新的實體卷冊並替換掉在現有卷冊群組中遺失的實體卷冊 (描述於 [〈節 6.4, “復原實體卷冊的 Metadata”](#)〉之中)。您可使用 **vgdisplay** 指令的 **--partial** 和 **--verbose** 引數來顯示任何已

不存在的實體卷冊的 UUID 和大小。若您希望取代另一個相同大小的實體卷冊，您可使用 **pvccreate** 指令以及 **--restorefile** 和 **--uuid** 引數，以初始化一個和遺失的實體卷冊相同 UUID 的新裝置。接著您便可使用 **vgcfgrestore** 指令來復原卷冊群組的 metadata。

## 6.6. 將遺失的實體卷冊由一個卷冊群組中移除掉

若您遺失了一個實體卷冊，您可透過 **vgchange** 指令的 **--partial** 引數來啟用卷冊群組中剩下的實體卷冊。您可透過 **vgreduce** 指令的 **--removemissing** 引數來將所有使用了該實體卷冊的邏輯卷冊由卷冊群組中移除。

我們建議您執行 **vgreduce** 指令和 **--test** 引數來驗證您所要銷毀的邏輯卷冊。

就和大部分的 LVM 作業一樣，您可透過即刻地使用 **vgcfgrestore** 指令來將卷冊群組的 metadata 復原為先前的狀態以撤消 **vgreduce** 指令。比方說，若您使用了 **vgreduce** 指令的 **--removemissing** 引數而不使用 **--test** 引數並且發現您移除了您想要保留的邏輯卷冊，您還是能夠將實體卷冊替換掉並使用另一項 **vgcfgrestore** 指令來將卷冊群組復原為它先前的狀態。

## 6.7. 邏輯卷冊的可用扇區不足

當您根據 **vgdisplay** 或 **vgs** 指令的輸出來假設您有足夠的扇區並建立一個邏輯卷冊時，您可能會看見「Insufficient free extents（可用扇區不足）」這則錯誤訊息。這是因為這些指令會將數字的兩個小數位數四捨五入以便提供易於讀取的輸出。若要指定確切的大小，請使用可用實體扇區的計算而不是使用一些位元組來斷定邏輯卷冊的大小。

**vgdisplay** 指令就預設值會包含下列顯示了可用實體扇區的輸出。

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

此外，您亦可使用 **vgs** 指令的 **vg\_free\_count** 和 **vg\_extent\_count** 引數來顯示可用的扇區和扇區的總數。

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

有了 8780 的可用扇區，您便可執行下列指令，並使用小寫的 **l** 引數來使用扇區並取代位元組：

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

這使用了卷冊群組中所有的可用扇區。

```
# 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
```

此外，您亦可透過 **lvcreate** 指令的 **-l** 引數來延伸邏輯卷冊，以使用卷冊群組中剩下可用空間的一個特定百分比。如欲取得更多資訊，請參閱〈[節 4.4.1, “建立線性邏輯卷冊”](#)〉。

## 章 7. 利用 LVM GUI 來進行 LVM 管理

除了指令列介面（Command Line Interface, CLI），LVM 還提供了一個圖形化使用者介面（GUI），您可使用該圖形化介面來配置 LVM 邏輯卷冊。您可藉由輸入 **system-config-lvm** 來啓動此工具程式。*儲存裝置管理指南* 的 LVM 一章提供了如何逐步地使用此工具，來配置 LVM 邏輯卷冊的相關指南。



## 附錄 A. 裝置映射 (DEVICE MAPPER) 設備

Device Mapper 是一個提供了卷冊管理架構的 kernel 驅動程式。它提供了建立映射裝置的一般方式，並且該映射裝置可被用來作為邏輯卷冊。它並不確切地清楚卷冊群組或 metadata 的格式為何。

Device Mapper 提供了幾個高層級技術上的基礎。除了 LVM 之外，Device-Mapper multipath 和 **dmraid** 指令亦使用 Device Mapper。Device Mapper 的應用程式介面為 **ioctl** system call。**dmsetup** 指令則為用戶介面。

LVM 邏輯卷冊可藉由使用 Device Mapper 啟用。所有邏輯卷冊皆會被轉換為映射裝置。所有區段皆會被轉換為一系列位於描述裝置之映射表格當中的行列。Device Mapper 支援各種映射目標，其中包含了 linear mapping、striped mapping，以及 error mapping。比方說，兩個磁碟可透過一對線性映射（一個磁碟一個）來被序連為一個單獨的邏輯卷冊。當 LVM 建立了一個卷冊時，它會建立一個可透過 **dmsetup** 指令來查詢的基本 device-mapper 裝置。欲取得更多有關於映射表格當中的裝置格式上的相關資訊，請參閱〈節 A.1, “裝置表格映射”〉。如需取得使用 **dmsetup** 指令來查詢某個裝置上的相關資訊，請參閱〈節 A.2, “dmsetup 指令”〉。

### A.1. 裝置表格映射

映射裝置是由一個透過受支援的裝置表格 (Device Table) 映射，來指定如何映射邏輯磁區 (logical sector) 的各個範圍的表格來定義的。映射裝置的表格是由一系列具有下列格式的行列所構成的：

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

在 Device Mapper 表格的第一個行列中，**start** 這個參數必須等於 0。一個行列上的 **start + length** 參數必須等於下個行列上的 **start**。指定於映射表格行列中的映射參數，取決於該行列上所指定的 **mapping**。

Device Mapper 中的大小總是以磁區來指定 (512 位元組)。

當某個裝置被指定為 Device Mapper 中的映射參數時，它可被檔案系統中的裝置名稱 (例如 **/dev/hda**) 參照，或是被格式為 **major:minor** 的 major 和 minor 數字參照。我們建議使用 major:minor 這個格式因為它避免了路徑名稱搜尋。

以下顯示了一個裝置的映射表格範例。在此表格中有四個 linear 目標：

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

各個行列的前兩個參數都是區段的起始區塊以及區塊的長度。下個關鍵字為映射目標，在此範例中的所有情況下都是 **linear**。行列剩下的部份則包含著 **linear** 目標的參數。

下列部份中描述了以下映射的格式：

- linear
- striped
- mirror
- snapshot 和 snapshot-origin
- error

- zero
- multipath
- crypt

### A.1.1. Linear 映射目標

Linear 映射目標會將一個連續範圍的區塊映射至另一個區塊裝置上。Linear 目標的格式如下：

```
start length linear device offset
```

#### ***start***

在虛擬裝置中啟用區塊

#### ***length***

此區段的長度

#### ***device***

區塊裝置，被檔案系統中的裝置名稱所參照，或是被格式為 *major:minor* 的 major 和 minor 數字所參照

#### ***offset***

在裝置上啟用映射的偏差值 (offset)

下列範例顯示了一個起始區塊於虛擬裝置 0 中、區段長度為 1638400、major:minor 數字配對為 8:2，以及裝置起始偏差值為 41146992 的 linear 目標。

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

下列範例顯示了一個裝置參數指定為 */dev/hda* 這個裝置的 linear 目標。

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

### A.1.2. 等量映射目標

等量映射目標 (striped Mapping Target) 支援實體裝置上的 striping。它會取等量磁碟數量和 striping chunk size 為引數以及一系列裝置名稱與磁區的配對。等量目標的格式如下

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

各個等量磁碟都有一組 ***device*** 和 ***offset*** 參數。

#### ***start***

在虛擬裝置中啟用區塊

#### ***length***

此區段的長度

**#stripes**

虛擬裝置的等量磁碟數量

**chunk\_size**

切換至下一個等量磁碟之前可寫至各個等量磁碟的磁區數量；必須至少和 kernel page 大小的 2 的 n 次方一樣大

**device**

區塊裝置，被檔案系統中的裝置名稱所參照，或是被格式為 *major:minor* 的 major 和 minor 數字所參照。

**offset**

在裝置上啟用映射的偏差值 (offset)

下列範例顯示了一個等量目標以及三個等量磁碟和大小為 128 的 chunk size：

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

**0**

在虛擬裝置中啟用區塊

**73728**

此區段的長度

**striped 3 128**

跨越了三個裝置的 stripe 以及大小為 128 block 的 chunk size

**8:9**

第一個裝置的 major:minor 數字

**384**

第一個裝置上的映射的起始偏差值

**8:8**

第二個裝置上的 major:minor 數字

**384**

第二個裝置上的映射的起始偏差值

**8:7**

第三個裝置上的 major:minor 數字

**9789824**

第三個裝置上的映射的起始偏差值

下列範例顯示了一個擁有兩個 256 KiB chunk 的等量磁碟的等量目標，並且裝置參數是由檔案系統中的裝置名稱來指定的，而不是以 major 和 minor 數字來指定。

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

### A.1.3. 鏡像映射目標

鏡像映射目標（mirror mapping target）支援鏡像邏輯卷冊的映射。鏡像目標的格式如下：

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

#### **start**

在虛擬裝置中啟用區塊

#### **length**

此區段的長度

#### **log\_type**

各種可能的日誌類型與它們的引數如下：

##### **core**

鏡像位於本機，並且鏡像日誌（mirror log）存放在核心記憶體（core memory）中。此日誌類型接受 1 到 3 個引數：

```
regionsize [[no]sync] [block_on_error]
```

##### **disk**

鏡像位於本機，並且鏡像日誌存放在磁碟上。此日誌類型接受 2 到 4 個引數：

```
logdevice regionsize [[no]sync] [block_on_error]
```

##### **clustered\_core**

鏡像已被叢集連結，並且鏡像日誌存放在核心記憶體中。此日誌類型接受 2 到 4 個引數：

```
regionsize UUID [[no]sync] [block_on_error]
```

##### **clustered\_disk**

鏡像已被叢集連結，並且鏡像日誌存放在磁碟上。此日誌類型接受 3 到 5 個引數：

```
logdevice regionsize UUID [[no]sync] [block_on_error]
```

LVM 會保存一個小型的日誌，它會使用該日誌來追蹤哪些區域已和鏡像同步化。*regionsize* 這個引數能指定這些區域的大小。

在一個叢集環境中，*UUID* 這個引數是個和鏡像日誌裝置相聯的唯一識別碼（unique identifier），日誌狀態可從而在叢集環境下被保留。

**[no]sync** 這個可選的引數可被用來將鏡像指定為「in-sync」或是「out-of-sync」。 **block\_on\_error** 這個引數可用來使鏡像針對於錯誤進行回應而不是將它們忽略掉。

**#log\_args**

將會被指定在映射中的日誌引數數量

**logargs**

鏡像的日誌引數；日誌引數的數量是透過 **#log-args** 參數來指定的，並且有效的日誌引數則是透過 **log\_type** 參數來判斷出的。

**#devs**

鏡像中的 leg 數量；各個 leg 皆被指定了一個裝置和偏差值。

**device**

各個鏡像 leg 的區塊裝置，由檔案系統中的裝置名稱參照，或由格式為 **major:minor** 的 major 和 minor 數字參照。如 **#devs** 參數所顯示，各個鏡像 leg 都會被指定一個區塊裝置和偏差值。

**offset**

裝置上的映射的起始偏差值。如 **#devs** 參數所顯示，各個鏡像 leg 都會被指定一個區塊裝置和偏差值。

下列範例顯示了一個叢集鏡像的鏡像映射目標，並且該叢集鏡像的鏡像日誌被存放在磁碟上。

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

**0**

在虛擬裝置中啟用區塊

**52428800**

此區段的長度

**mirror clustered\_disk**

含有一個指定了鏡像已被叢集連結並且鏡像日誌存放在磁碟上的鏡像目標

**4**

會有四個鏡像日誌

**253:2**

日誌裝置的 major:minor 數字

**1024**

鏡像日誌使用來追蹤已同步化之鏡像的區域大小

**UUID**

用來在叢集環境下保留日誌資訊的鏡像日誌裝置 UUID

**block\_on\_error**

鏡像應針對於錯誤做出回應

## 3

鏡像中的 leg 數量

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

構成各個鏡像 leg 的裝置的 major:minor 數字和偏差值

#### A.1.4. 快照和 **snapshot-origin** 映射目標

當您建立了卷冊的第一個 LVM 快照時，會有四個 Device Mapper 裝置被使用到：

1. 一個含有 **linear** 映射的裝置，並且該映射包含著來源卷冊的原始映射表格。
2. 一個含有 **linear** 映射的裝置，它會被用來作為來源卷冊的寫入即複製（copy-on-write, COW）的裝置；針對於各個寫入動作，原始資料都會被儲存在各個 snapshot 的 COW 裝置中，如此一來它的可見內容便不會遭到更改（直到 COW 裝置滿出）。
3. 一個含有 **snapshot** 映射的裝置，它結合了 #1 與 #2，也就是可見的快照卷冊。
4. 「原始」卷冊（它使用了原始來源卷冊所使用的裝置號碼），它的表格已被替換為一個來自於裝置 #1 的「snapshot-origin」映射。

有個使用來建立這些裝置的固定命名方案。比方說，您可能會使用下列指令來建立一個名為 **base** 的 LVM 卷冊以及一個基於該卷冊、名為 **snap** 的快照卷冊。

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

這會產生四個裝置，並且您可透過下列指令來檢視這些裝置：

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

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

**snapshot-origin** 目標的格式如下：

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

**start**

在虛擬裝置中啟用區塊

**length**

此區段的長度

### ***origin***

快照的基本卷冊

**snapshot-origin** 通常會有一個或是更多個基於它的快照。讀取動作會直接被映射至 backing device。針對於各個寫入動作，原始資料都會被儲存在各個快照的 COW 裝置中來保存它的可見內容直到 COW 裝置填滿。

**snapshot** 目標的格式如下：

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

### ***start***

在虛擬裝置中啓用區塊

### ***length***

此區段的長度

### ***origin***

快照的基本卷冊

### ***COW-device***

遭到更改的資料區塊所被儲存至的裝置

### ***P|N***

P (Persistent [一致性]) 或是 N (Not persistent [非一致性])；顯示了快照在系統重新啓動後是否還會存在。針對於暫時性的快照 (N)，磁碟上必須要儲存較少 metadata；它們可被 kernel 保存在記憶體中。

### ***chunksize***

被存放在 COW 裝置上、遭到更改的資料區塊大小 (以磁區為單位)

下列範例顯示了一個原始裝置為 254:11 的 **snapshot-origin** 目標。

```
0 2097152 snapshot-origin 254:11
```

下列範例顯示了一個原始裝置為 254:11 並且 COW 裝置為 254:12 的 **snapshot** 目標。這個快照裝置經過了系統重新啓動後將依然有效，並且儲存在 COW 裝置上的資料的區塊大小為 16 個磁區。

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

## **A.1.5. error 映射目標**

若使用了 error 映射目標的話，任何針對於映射的磁區的 I/O 作業都會失敗。

error 映射目標可用來進行測試。若要測試某個裝置在錯誤的情況下會有什麼特性，您可建立一個裝置映射，並且在該裝置中間含有個錯誤的磁區，或是您可將鏡像的一個 leg 替換為另一個 error 目標。

一個 **error** 目標可被用來取代一個發生錯誤的裝置，這是個避免在實際的裝置上逾時和進行重新嘗試的方式。它可被用來作為一個當您在發生錯誤，而進行 LVM metadata 重整時的媒介目標。

**error** 映射目標除了 *start* 和 *length* 這兩個參數之外不接受額外的參數。

下列範例顯示了一個 **error** 目標。

```
0 65536 error
```

### A.1.6. zero 映射目標

**zero** 映射目標是個相當於 `/dev/zero` 的區塊裝置。對於此映射所進行的讀取作業會回傳一些零的區塊。寫至此映射的資料將會被丟棄，不過寫入作業會成功。**zero** 映射目標不接受 *start* 和 *length* 參數以外的額外參數。

下列範例顯示了一個 16Tb 裝置的 **zero** 目標。

```
0 65536 zero
```

### A.1.7. multipath 映射目標

**multipath** 映射目標支援多路徑裝置的映射。**multipath** 目標的格式如下：

```
start length multipath #features [feature1 ... featureN] #handlerargs
[handlerarg1 ... handlerargN] #pathgroups pathgroup pathgroupargs1 ...
pathgroupargsN
```

各個路徑群組都有一組 **pathgroupargs** 參數。

#### **start**

在虛擬裝置中啟用區塊

#### **length**

此區段的長度

#### **#features**

**multipath** 功能的數量以及它們的功能。若此參數為零的話，那麼就不會有 **feature** 這個參數並且下一個裝置映射參數會是 **#handlerargs**。  
目前只有一個受支援的 **multipath** 功能，**queue\_if\_no\_path**，可以在 **multipath.conf** 檔案中的 **features** 屬性中設定。這表示此多路徑裝置目前已設為若沒有可用路徑的話便會將 I/O 作業置於佇列中。

在以下例子裡，**multipath.conf** 檔案中的 **no\_path\_retry** 屬性只有在嘗試路徑數次且失敗後，並被標示為失敗時，才會被設定到佇列 I/O 操作中。在這情況下，映射就會以下列格式來顯示出，直到所有路徑檢查程式所被指定的檢查次數都已完成後。

```
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
```

當所有路徑檢查程式所被指定的檢查次數都已完成後，映射就會以下列格式顯示出。



```
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**

硬體處理程式引數的數量以及那些引數。硬體處理程式會在切換路徑群組或是處理 I/O 錯誤時指定一個用來執行硬體特屬之動作的模組。若這被設為 0 的話，那麼下個參數便是 **#pathgroups**。

### **#pathgroups**

路徑群組的數量。路徑群組 (path group) 代表一組路徑，並且多路徑的裝置將會在這些路徑上進行負載平衡。各個路徑群組都有一組 **pathgroupargs** 參數。

### **pathgroup**

下一個嘗試的路徑群組。

### **pathgroupsargs**

各個路徑群組都包含著下列引數：

```
pathselector #selectorargs #paths #pathargs device1 ioreqs1 ... deviceN
ioreqsN
```

路徑群組中的各個路徑都都有一組路徑引數。

### **pathselector**

可指定使用中的演算法來判斷下個 I/O 作業該使用此路徑群組中的哪個路徑。

### **#selectorargs**

允許此引數使用於 multipath 映射的路徑選擇器引數數量。目前，這個引數的值總會是 0。

### **#paths**

此路徑群組中的路徑數量。

### **#pathargs**

為此群組中各個路徑所指定的路徑引數數量。目前，這個數字總會是 1，也就是 **ioreqs** 引數。

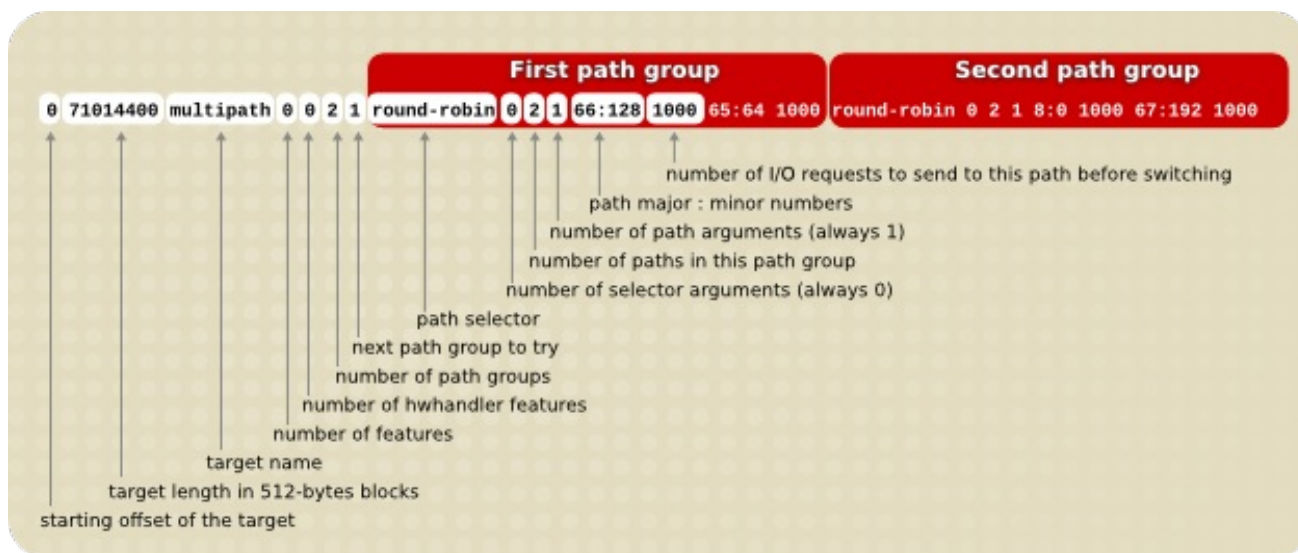
### **device**

路徑的區塊裝置號碼，以格式為 **major:minor** 的 major 和 minor 數字來參照

### **ioreqs**

要切換至目前群組中的下個路徑前所需要用來 route 至此路徑的 I/O 請求數量。

〈圖形 A.1, “Multipath 映射目標”〉顯示了含有兩個路徑群組的 multipath 目標的格式。



圖形 A.1. Multipath 映射目標

下列範例顯示了相同 multipath 裝置的純容錯（failover）目標定義。在此目標中有四個路徑群組，並且一個路徑群組只有一個開放的路徑，這樣一來 multipath 裝置便只會一次使用一個路徑。

```
0 71014400 multipath 0 0 4 1 round-robin 0 1 1 66:112 1000 \
round-robin 0 1 1 67:176 1000 round-robin 0 1 1 68:240 1000 \
round-robin 0 1 1 65:48 1000
```

下列範例顯示了相同 multipath 裝置的一個多重匯流排（multibus）目標定義。在此目標中只有一個路徑群組，並且它包含了所有的路徑。在此設定中，multipath 會將負載平衡地分配至所有路徑中。

```
0 71014400 multipath 0 0 1 1 round-robin 0 4 1 66:112 1000 \
67:176 1000 68:240 1000 65:48 1000
```

如欲取得更多有關於進行 multipath 上的相關資訊，請參閱 *Using Device Mapper Multipath* 文件。

### A.1.8. crypt 映射目標

**crypt** 目標會將通過指定裝置的資料加密。它使用了 kernel 的 Crypto API。

**crypt** 目標的格式如下：

```
start length crypt cipher key IV-offset device offset
```

#### **start**

在虛擬裝置中啟用區塊

#### **length**

此區段的長度

#### **cipher**

Cipher 包含著 ***cipher[-chainmode]-ivmode[:iv options]***。

#### **cipher**

可用的 Cipher 列在 `/proc/crypto` 中（比方說 **aes**）。

**chainmode**

總是使用 **cbc**。請勿使用 **ebc**；它並不使用初始向量 (IV)。

**ivmode[:iv options]**

IV 是個用來改變加密的初始向量。IV 模式為 **plain** 或 **essiv:hash**。**-plain** 的 **ivmode** 使用了磁區編號（加上 IV 偏差值）來作為 IV。**-essiv** 的 **ivmode** 是為了避免 watermark 弱點用的。

**key**

加密金鑰、以十六進位提供

**IV-offset**

初始向量 (Initial Vector, IV) 偏差值

**device**

區塊裝置，被檔案系統中的裝置名稱所參照，或是被格式為 **major:minor** 的 major 和 minor 數字所參照

**offset**

在裝置上啟用映射的偏差值 (offset)

下列為 **crypt** 目標的範例。

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

**A.2. DMSETUP 指令**

**dmsetup** 指令是一項用來和 Device Mapper 進行通訊的指令列 wrapper。如欲取得有關於 LVM 裝置的一般系統資訊，您可使用描述於下列部份的 **dmsetup** 指令的 **info**、**ls**、**status** 和 **deps** 選項。

如欲取得有關於 **dmsetup** 指令的額外選項和功能上的相關資訊，請查看 **dmsetup(8)** man page。

**A.2.1. dmsetup info 指令**

**dmsetup info device** 這項指令提供了有關於 Device Mapper 裝置的總覽資訊。若您不指定一個裝置名稱的話，輸出便會是有關於目前所有已配置的 Device Mapper 裝置的相關資訊。若您指定了一個裝置，那麼這項指令便只會產生該裝置的相關資訊。

**dmsetup info** 指令提供了下列種類的資訊：

**Name**

裝置的名稱。LVM 裝置是以卷冊群組名稱和邏輯卷冊名稱來表示並以連字符號來區隔開。原始名稱中的連字符號會被轉譯為兩個連字符號。

**State**

可能的裝置狀態有 **SUSPENDED**、**ACTIVE** 和 **READ-ONLY**。**dmsetup suspend** 指令會將裝置狀態設為 **SUSPENDED**。當裝置休眠 (suspend) 時，該裝置的所有 I/O 作業都會停下。**dmsetup resume** 指令則會將裝置狀態恢復為 **ACTIVE**。

## Read Ahead

系統將為讀取作業繼續進行中的任何已開啓的檔案所預讀的資料區塊數量。就預設值，kernel 會自動地選擇一個適當的值。您可透過 **--readahead** option of the **dmsetup** 指令來更改這個值。

## Tables present

此類型的可能狀態為 **LIVE** 和 **INACTIVE**。**INACTIVE** 這個狀態顯示了有個表格已被載入並且該表格會在 **dmsetup resume** 指令將裝置狀態恢復為 **ACTIVE** 時被換入。此時，該表格的狀態會成為 **LIVE**。如欲取得相關資訊，請參閱 **dmsetup** man page。

## Open count

Open reference count 表示了裝置被開啓了多少次。**mount** 指令可將裝置開啓。

## Event number

目前取得的事件數量。輸入 **dmsetup wait n** 這項指令能讓用戶等待第 n 項事件，並在取得該項事件之前阻擋調用。

## Major, minor

Major 和 minor

## Number of targets

構成一個裝置的片段數量。比方說一個跨距了三個磁碟的 linear 裝置將會有三個目標。一個由磁碟起始和結尾（少了中間）所構成的 linear 裝置將會有兩個目標。

## UUID

裝置的 UUID。

下列範例顯示了 **dmsetup info** 指令的部份輸出。

```
# 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-t0cS1kqFV9drb0X1Vr8sxeYP0tqcrpdegyqj5lZxe45JMGlmvtqLmbLpBcenh2L3
```

### A.2.2. dmsetup ls 指令

您可透過 **dmsetup ls** 指令來列出映射裝置的裝置名稱。您可透過使用 **dmsetup ls --target target\_type** 指令來列出擁有至少一個指定類型的目標的裝置。如欲取得其它 **dmsetup ls** 的選項的相關資訊，請參閱 **dmsetup man page**。

下列範例顯示了用來列出目前已配置的映射裝置之裝置名稱的指令。

```
# dmsetup ls
testgfsvg-testgfslv3      (253:4)
testgfsvg-testgfslv2      (253:3)
testgfsvg-testgfslv1      (253:2)
VolGroup00-LogVol01       (253:1)
VolGroup00-LogVol00       (253:0)
```

下列範例顯示了用來列出目前已配置的鏡像映射之裝置名稱的指令。

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

堆在 multipath 或其它 device mapper 裝置上的 LVM 配置可能非常複雜。**dmsetup ls** 指令提供了一項 **--tree** 選項，以將裝置之間的相依性，如下列範例以樹狀顯示。

```
# dmsetup ls --tree
vgtest-lvmir (253:13)
├─vgtest-lvmir_mimage_1 (253:12)
│   └─mpathp1 (253:8)
│       └─mpathe (253:5)
│           ├── (8:112)
│           └─ (8:64)
├─vgtest-lvmir_mimage_0 (253:11)
│   └─mpathcp1 (253:3)
│       └─mpathc (253:2)
│           ├── (8:32)
│           └─ (8:16)
└─vgtest-lvmir_mlog (253:4)
    └─mpathfp1 (253:10)
        └─mpathf (253:6)
            ├── (8:128)
            └─ (8:80)
```

### A.2.3. dmsetup status 指令

**dmsetup status device** 這項指令提供了指定裝置中的各個目標的狀態資訊。若您不指定裝置名稱的話，輸出便會是有關於所有目前已配置的 Device Mapper 裝置的相關資訊。您只可透過 **dmsetup status --target target\_type** 指令來將擁有至少一個指定類型的目標的裝置狀態列出。

下列範例顯示了用來列出目前已配置的映射裝置中的所有目標狀態的指令。

```
# dmsetup status
testgfsvg-testgfs1v3: 0 312352768 linear
testgfsvg-testgfs1v2: 0 312352768 linear
testgfsvg-testgfs1v1: 0 312352768 linear
testgfsvg-testgfs1v1: 312352768 50331648 linear
VolGroup00-LogVol01: 0 4063232 linear
VolGroup00-LogVol00: 0 151912448 linear
```

#### A.2.4. dmsetup deps 指令

**dmsetup deps device** 指令會提供一系列裝置配對 (major、minor)，並且這些裝置配對已被指定裝置的映射表格所參照。若您不指定裝置名稱的話，輸出將會是有關於所有目前已配置的 Device Mapper 裝置的相關資訊。

下列範例顯示了用來列出目前已配置的映射裝置的所有相依的指令。

```
# dmsetup deps
testgfsvg-testgfs1v3: 1 dependencies : (8, 16)
testgfsvg-testgfs1v2: 1 dependencies : (8, 16)
testgfsvg-testgfs1v1: 1 dependencies : (8, 16)
VolGroup00-LogVol01: 1 dependencies : (8, 2)
VolGroup00-LogVol00: 1 dependencies : (8, 2)
```

下列範例顯示了只用來列出 **lock\_stress-grant--02.1722** 裝置之相依的指令：

```
# dmsetup deps lock_stress-grant--02.1722
3 dependencies : (253, 33) (253, 32) (253, 31)
```

### A.3. UDEV DEVICE MANAGER (裝置管理員) 的 DEVICE MAPPER 支援

**udev** 裝置管理員的主要用途就是在 **/dev** 目錄中提供動態式的節點設定方式。這些節點的建置是以使用者空間中的 **udev** 應用程式規則來指定的。這些規則會直接在來自於 kernel 的 **udev** 事件上進行處理，以完成特定裝置的新增、移除，或變更。這為熱插拔支援提供了方便和中央化的機制。

除了建立實際的節點之外，**udev** 裝置管理員亦可以自己的名稱建立符號連結。這提供使用者視需求選擇自訂化命名，以及選擇 **/dev** 目錄中之目錄結構的自由。

各個 **udev** 事件皆包含了有關於被裝置的裝置的基本資訊，例如它的名、它所屬之子系統、裝置類型、所使用的 major 與 minor 號碼，以及事件的類型。除了能存取 **/sys** 目錄中所找到，並且 **udev** 規則中亦可存取的所有資訊，使用者亦能根據這項資訊，並條件性地執行規則，利用基本的過濾器。

**udev** 裝置管理員還提供了中央化的節點權限設定。使用者能輕易地新增一組自訂的規則，以定義任何在處理事件時，所能使用的任何資訊所指定的裝置。

您亦可在 **udev** 規則中直接新增 program hook。**udev** 裝置管理員可調用這些程式，以提供處理事件所需的額外處理。並且，程式亦可透過這項處理來匯出環境變數。任何得到的結果，皆可使用於規則中作為補充的資訊來源。

任何使用 **udev** 函式庫的軟體皆能取得和處理 **udev** 事件，以及所有資訊，因此處理程序便不僅限於 **udev** daemon。

### A.3.1. udev 與裝置管理員的整合

在 RHEL 6 中，Device Mapper 提供了直接的 **udev** 整合支援。這可同步化 Device Mapper 以及所有與 Device Mapper 裝置相關的 **udev** 程序，包括 LVM 裝置。同步化是必要的，因為 **udev** daemon 中的規則應用程式格式，是與裝置變更來源之程式的平行處理。（例如 **dmsetup** 和 LVM）。少了這項支援，將造成一項以往使用者嘗試移除一個依然開啓，並由 **udev** 規則所處理的裝置，而造成了先前變更事件上的問題；這特別是在裝置變更之間的時間非常短的情況下，經常發生。

RHEL 6 發行版提供了一般 Device Mapper 裝置與 LVM 的正式 **udev** 規則支援。〈[表格 A.1, “Device-Mapper 裝置的 udev 規則”](#)〉概述了這些規則，並且這些規則安裝在 `/lib/udev/rules.d` 中。

表格 A.1. Device-Mapper 裝置的 udev 規則

檔案名稱	描述
<b>10-dm.rules</b>	<p>包含了基本/一般的 Device Mapper 規則，並在 <code>/dev/mapper</code> 中建立了 symlink，它包含了一個 <code>/dev/dm-N</code> target，並且 N 代表一個 kernel 動態指定給裝置的數字（<code>/dev/dm-N</code> 是個節點）。</p> <p>請注意：<code>/dev/dm-N</code> 節點絕不該使用於 script 中來存取裝置，因為 N 數字是動態式指定的，而會隨著裝置的啓用循序而改變。因此，應使用 <code>/dev/mapper</code> 目錄中的真實名稱。此格式是用來支援建立節點/symlink 時的 <b>udev</b> 需求。</p>
<b>11-dm-lvm.rules</b>	<p>包含了 LVM 裝置所套用的規則，並為卷冊群組的邏輯卷冊建立 symlink。Symlink 會被建立在 <code>/dev/vgname</code> 目錄中，並含有一個 <code>/dev/dm-N</code> 目標。</p> <p>請注意：若要 Device Mapper 子系統未來所有的規則標準皆能一致，udev 規則應使用 <b>11-dm-subsystem_name.rules</b> 這種格式。任何提供 <b>udev</b> 規則的 <b>libdevmapper</b> 使用者也應遵照此標準。</p>
<b>13-dm-disk.rules</b>	<p>包含了一般可套用至所有 Device Mapper 裝置的規則，並在 <code>/dev/disk/by-id</code>、<code>/dev/disk/by-uuid</code> 以及 <code>/dev/disk/by-uuid</code> 目錄中建立符號連結。</p>
<b>95-dm-notify.rules</b>	<p>包含了通知使用 <b>libdevmapper</b>（就如同 LVM 和 <b>dmsetup</b>）時的等待時間的規則。通知會在所有先前規則被套用後才會完成，以確保所有 <b>udev</b> 處理程序皆已完成。之後被通知的程序便會復原。</p>

您可藉由 **12-dm-permissions.rules** 檔案來新增額外的自訂權限規則。此檔案並非安裝在 `/lib/udev/rules` 目錄中；它位於 `/usr/share/doc/device-mapper-版本` 目錄中。**12-dm-permissions.rules** 檔案是個範本，它包含了有關於如何根據一些相符規則，以設置權限的相關提示；此檔案包含了一些常見情況下的範例。您可編輯此檔案，並手動式地將它放置在 `/etc/udev/rules.d` 目錄中，如此一來當您進行更新時，這些設定皆能被保留住。

這些規則設置了所有當處理事件時，任何其它規則亦可使用的基本變數。

下列變數設置於 10-dm.rules 中：

- **DM\_NAME** : Device Mapper 裝置名稱
- **DM\_UUID** : Device Mapper 裝置 UUID
- **DM\_SUSPENDED** : Device Mapper 裝置的暫停狀態
- **DM\_UDEV\_RULES\_VSN** : **udev** 規則版本（這主要是用來讓所有其它規則檢查，先前所提到的變數是否透過正式的 Device Mapper 規則所設置的）

下列變數設置於 **11-dm-lvm.rules** 中：

- **DM\_UUID** : Device Mapper 裝置 UUID
- **DM\_VG\_NAME** : 卷冊群組名稱
- **DM\_LV\_LAYER** : LVM 層名

所有的這些變數皆可使用於 **12-dm-permissions.rules** 檔案中，以為特定 Device Mapper 裝置定義權限，如 **12-dm-permissions.rules** 檔案中所記載。

### A.3.2. 支援 udev 的指令與介面

〈[表格 A.2, “支援 udev 的 dmsetup 指令”](#)〉概述了支援 **udev** 整合的 **dmsetup** 指令。

表格 A.2. 支援 udev 的 dmsetup 指令

指令	描述
<b>dmsetup udevcomplete</b>	使用來通知 <b>udev</b> 已完成了處理規則，並解除了等待程序上的鎖定（由 <b>95-dm-notify.rules</b> 中的 <b>udev</b> 規則調用）。
<b>dmsetup udevcomplete_all</b>	使用於除錯用途，以手動式地解除所有等待程序上的鎖定。
<b>dmsetup udevcookies</b>	使用於除錯用途，以顯示所有既有的 cookies（系統全域的旗號）。
<b>dmsetup udevcreatecookie</b>	使用於手動式建立 cookie（旗號）。這對於在一個同步化的資源下執行多項程序來說，相當有幫助。
<b>dmsetup udevreleasecookie</b>	使用來等待全部與所有單一同步化 cookie 的程序相關的 <b>udev</b> 處理程序。

支援 **udev** 整合的 **dmsetup** 選項如下。

#### --udevcookie

需為我們所希望加入 **udev** 交易中的所有 **dmsetup** 程序定義。它可與 **udevcreatecookie** 和 **udevreleasecookie** 搭配使用：

```
COOKIE=$(dmsetup udevcreatecookie)
dmsetup command --udevcookie $COOKIE ....
dmsetup command --udevcookie $COOKIE ....
```



```
....
dmsetup command --udevcookie $COOKIE ....
dmsetup udevreleasecookie --udevcookie $COOKIE
```

除了使用 **--udevcookie** 選項之外，您可直接將變數匯出至程序的環境中：

```
export DM_UDEV_COOKIE=$(dmsetup udevcreatecookie)
dmsetup command ...
dmsetup command ...
...
dmsetup command ...
```

### **--noudevrules**

停用 udev 規則。節點/符號連結將會由 **libdevmapper** 本身所建立。此選項的用途乃為了在 **udev** 無法正常運作的情況下進行除錯。

### **--noudevsync**

停用 **udev** 同步化。這也是為了進行除錯。

欲取得更多有關於 **dmsetup** 與其選項上的相關資訊，請查看 **dmsetup(8)** man page。

LVM 指令支援下列支援 **udev** 整合的選項：

- **--noudevrules**：就如與 **dmsetup** 指令搭配使用一般，將會停用 **udev** 規則。
- **--noudevsync**：就如與 **dmsetup** 指令搭配使用一般，將會停用 **udev** 同步化。

**lvm.conf** 檔案包含了下列支援 **udev** 整合的選項：

- **udev\_rules**：全域啟用/停用所有 LVM2 指令的 **udev\_rules**。
- **udev\_sync**：全域啟用/停用所有 LVM 指令的 **udev** 同步化。

欲取得更多有關於 **lvm.conf** 檔案選項上的相關資訊，請參閱 **lvm.conf** 檔案中的內嵌註解。

## 附錄 B. LVM 配置檔案

LVM 支援多重配置檔案。當系統啟動時，`lvm.conf` 配置檔案會被由 `LVM_SYSTEM_DIR` 這個環境變數所指定的目錄載入，預設值會是 `/etc/lvm`。

`lvm.conf` 檔案可指定所要載入的額外配置檔案。檔案中之後的設定將會覆蓋先前的設定。若要在載入所有配置檔案之後顯示使用中的設定，請執行 `lvm dumpconfig` 指令。

欲取得有關於載入額外配置檔案的相關資訊，請參閱〈[節 C.2, “主機標籤 \(Host Tags\)”](#)〉。

### B.1. LVM 配置檔案

下列檔案為 LVM 的相關配置檔案：

#### `/etc/lvm/lvm.conf`

工具所讀取的中央配置檔案。

#### `etc/lvm/lvm_hosttag.conf`

額外配置檔案會針對於存在的主機標記 (host tag) 被讀取：`lvm_hosttag.conf`。若該檔案定義了新的標記的話，那麼額外的配置檔案便會被附加至即將被讀取的清單中。欲取得更多有關於主機標記的相關資訊，請參閱〈[節 C.2, “主機標籤 \(Host Tags\)”](#)〉。

除了 LVM 配置檔案以外，執行 LVM 的系統還包含了下列可影響 LVM 系統設定的檔案：

#### `/etc/lvm/cache/.cache`

裝置名稱過濾快取檔案（可配置）。

#### `/etc/lvm/backup/`

自動卷冊群組 metadata 備份的目錄（可配置）。

#### `/etc/lvm/archive/`

自動卷冊群組 metadata 壓縮檔的目錄（可藉由目錄路徑和封存紀錄深度來進行配置）。

#### `/var/lock/lvm/`

在單主機的配置中，檔案會被鎖定來預防 parallel tool 執行並損壞 metadata；在叢集中，叢集全域的 DLM 會被使用。

### B.2. 範例 LVM.CONF 檔案

以下為 `lvm.conf` 配置檔案的範本。您的配置檔案可能會與此檔案有所不同。

```
# This is an example configuration file for the LVM2 system.
# It contains the default settings that would be used if there was no
# /etc/lvm/lvm.conf file.
#
# Refer to 'man lvm.conf' for further information including the file
# layout.
#
# To put this file in a different directory and override /etc/lvm set
```

```

# the environment variable LVM_SYSTEM_DIR before running the tools.
#
# N.B. Take care that each setting only appears once if uncommenting
# example settings in this file.

# This section allows you to configure which block devices should
# be used by the LVM system.
devices {

    # Where do you want your volume groups to appear ?
    dir = "/dev"

    # An array of directories that contain the device nodes you wish
    # to use with LVM2.
    scan = [ "/dev" ]

    # If set, the cache of block device nodes with all associated symlinks
    # will be constructed out of the existing udev database content.
    # This avoids using and opening any inapplicable non-block devices or
    # subdirectories found in the device directory. This setting is
    applied
    # to udev-managed device directory only, other directories will be
    scanned
    # fully. LVM2 needs to be compiled with udev support for this setting
    to
    # take effect. N.B. Any device node or symlink not managed by udev in
    # udev directory will be ignored with this setting on.
    obtain_device_list_from_udev = 1

    # If several entries in the scanned directories correspond to the
    # same block device and the tools need to display a name for device,
    # all the pathnames are matched against each item in the following
    # list of regular expressions in turn and the first match is used.
    # preferred_names = [ ]

    # Try to avoid using un-descriptive /dev/dm-N names, if present.
    preferred_names = [ "^/dev/mpath/", "^/dev/mapper/mpath",
    "^/dev/[hs]d" ]

    # A filter that tells LVM2 to only use a restricted set of devices.
    # The filter consists of an array of regular expressions. These
    # expressions can be delimited by a character of your choice, and
    # prefixed with either an 'a' (for accept) or 'r' (for reject).
    # The first expression found to match a device name determines if
    # the device will be accepted or rejected (ignored). Devices that
    # don't match any patterns are accepted.

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

```

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

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

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

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

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

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

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

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

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

# Advanced settings.

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

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

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

# By default, if a PV is placed directly upon an md device, LVM2
# will align its data blocks with the md device's stripe-width.
```

```

# 1 enables; 0 disables.
md_chunk_alignment = 1

# Default alignment of the start of a data area in MB. If set to 0,
# a value of 64KB will be used. Set to 1 for 1MiB, 2 for 2MiB, etc.
# default_data_alignment = 1

# By default, the start of a PV's data area will be a multiple of
# the 'minimum_io_size' or 'optimal_io_size' exposed in sysfs.
# - minimum_io_size - the smallest request the device can perform
#   w/o incurring a read-modify-write penalty (e.g. MD's chunk size)
# - optimal_io_size - the device's preferred unit of receiving I/O
#   (e.g. MD's stripe width)
# minimum_io_size is used if optimal_io_size is undefined (0).
# If md_chunk_alignment is enabled, that detects the optimal_io_size.
# This setting takes precedence over md_chunk_alignment.
# 1 enables; 0 disables.
data_alignment_detection = 1

# Alignment (in KB) of start of data area when creating a new PV.
# md_chunk_alignment and data_alignment_detection are disabled if set.
# Set to 0 for the default alignment (see: data_alignment_default)
# or page size, if larger.
data_alignment = 0

# By default, the start of the PV's aligned data area will be shifted
by
# the 'alignment_offset' exposed in sysfs. This offset is often 0 but
# may be non-zero; e.g.: certain 4KB sector drives that compensate for
# windows partitioning will have an alignment_offset of 3584 bytes
# (sector 7 is the lowest aligned logical block, the 4KB sectors start
# at LBA -1, and consequently sector 63 is aligned on a 4KB boundary).
# But note that pvcreate --dataalignmentoffset will skip this
detection.
# 1 enables; 0 disables.
data_alignment_offset_detection = 1

# If, while scanning the system for PVs, LVM2 encounters a device-
mapper
# device that has its I/O suspended, it waits for it to become
accessible.
# Set this to 1 to skip such devices. This should only be needed
# in recovery situations.
ignore_suspended_devices = 0

# During each LVM operation errors received from each device are
counted.
# If the counter of a particular device exceeds the limit set here, no
# further I/O is sent to that device for the remainder of the
respective
# operation. Setting the parameter to 0 disables the counters
altogether.
disable_after_error_count = 0

# Allow use of pvcreate --uuid without requiring --restorefile.
require_restorefile_with_uuid = 1

```

```

# Minimum size (in KB) of block devices which can be used as PVs.
# In a clustered environment all nodes must use the same value.
# Any value smaller than 512KB is ignored.

# Ignore devices smaller than 2MB such as floppy drives.
pv_min_size = 2048

# The original built-in setting was 512 up to and including version
2.02.84.
# pv_min_size = 512

# Issue discards to a logical volumes's underlying physical volume(s)
when
# the logical volume is no longer using the physical volumes' space
(e.g.
# lvremove, lvreduce, etc). Discards inform the storage that a region
is
# no longer in use. Storage that supports discards advertise the
protocol
# specific way discards should be issued by the kernel (TRIM, UNMAP,
or
# WRITE SAME with UNMAP bit set). Not all storage will support or
benefit
# from discards but SSDs and thinly provisioned LUNs generally do. If
set
# to 1, discards will only be issued if both the storage and kernel
provide
# support.
# 1 enables; 0 disables.
issue_discards = 0
}

# This section allows you to configure the way in which LVM selects
# free space for its Logical Volumes.
#allocation {
#   When searching for free space to extend an LV, the "cling"
#   allocation policy will choose space on the same PVs as the last
#   segment of the existing LV. If there is insufficient space and a
#   list of tags is defined here, it will check whether any of them are
#   attached to the PVs concerned and then seek to match those PV tags
#   between existing extents and new extents.
#   Use the special tag "@" as a wildcard to match any PV tag.
#
#   Example: LVs are mirrored between two sites within a single VG.
#   PVs are tagged with either @site1 or @site2 to indicate where
#   they are situated.
#
#   cling_tag_list = [ "@site1", "@site2" ]
#   cling_tag_list = [ "@" ]
#
#   Changes made in version 2.02.85 extended the reach of the 'cling'
#   policies to detect more situations where data can be grouped
#   onto the same disks. Set this to 0 to revert to the previous
#   algorithm.
#
#

```

```

#   maximise_cling = 1
#
#   Set to 1 to guarantee that mirror logs will always be placed on
#   different PVs from the mirror images.  This was the default
#   until version 2.02.85.
#
#   mirror_logs_require_separate_pvs = 0
#}

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

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

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

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

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

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

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

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

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

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

```

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

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

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

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

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

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

    # What is the minimum number of archive files you wish to keep ?
    retain_min = 10

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

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

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

# Miscellaneous global LVM2 settings
global {

    # The file creation mask for any files and directories created.
    # Interpreted as octal if the first digit is zero.
    umask = 077

    # Allow other users to read the files
    #umask = 022

    # Enabling test mode means that no changes to the on disk metadata
    # will be made. Equivalent to having the -t option on every
```



```

# command. Defaults to off.
test = 0

# Default value for --units argument
units = "h"

# Since version 2.02.54, the tools distinguish between powers of
# 1024 bytes (e.g. KiB, MiB, GiB) and powers of 1000 bytes (e.g.
# KB, MB, GB).
# If you have scripts that depend on the old behaviour, set this to 0
# temporarily until you update them.
si_unit_consistency = 1

# Whether or not to communicate with the kernel device-mapper.
# Set to 0 if you want to use the tools to manipulate LVM metadata
# without activating any logical volumes.
# If the device-mapper kernel driver is not present in your kernel
# setting this to 0 should suppress the error messages.
activation = 1

# If we can't communicate with device-mapper, should we try running
# the LVM1 tools?
# This option only applies to 2.4 kernels and is provided to help you
# switch between device-mapper kernels and LVM1 kernels.
# The LVM1 tools need to be installed with .lvm1 suffices
# e.g. vgscan.lvm1 and they will stop working after you start using
# the new lvm2 on-disk metadata format.
# The default value is set when the tools are built.
# fallback_to_lvm1 = 0

# The default metadata format that commands should use - "lvm1" or
"lvm2".
# The command line override is -M1 or -M2.
# Defaults to "lvm2".
# format = "lvm2"

# Location of proc filesystem
proc = "/proc"

# Type of locking to use. Defaults to local file-based locking (1).
# Turn locking off by setting to 0 (dangerous: risks metadata
corruption
# if LVM2 commands get run concurrently).
# Type 2 uses the external shared library locking_library.
# Type 3 uses built-in clustered locking.
# Type 4 uses read-only locking which forbids any operations that
might
# change metadata.
locking_type = 1

# Set to 0 to fail when a lock request cannot be satisfied
immediately.
wait_for_locks = 1

# If using external locking (type 2) and initialisation fails,
# with this set to 1 an attempt will be made to use the built-in

```

```

# clustered locking.
# If you are using a customised locking_library you should set this to
0.
fallback_to_clustered_locking = 1

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

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

# Whenever there are competing read-only and read-write access
requests for
# a volume group's metadata, instead of always granting the read-only
# requests immediately, delay them to allow the read-write requests to
be
# serviced. Without this setting, write access may be stalled by a
high
# volume of read-only requests.
# NB. This option only affects locking_type = 1 viz. local file-based
# locking.
prioritise_write_locks = 1

# Other entries can go here to allow you to load shared libraries
# e.g. if support for LVM1 metadata was compiled as a shared library
use
#   format_libraries = "liblvm2format1.so"
# Full pathnames can be given.

# Search this directory first for shared libraries.
#   library_dir = "/lib"

# The external locking library to load if locking_type is set to 2.
#   locking_library = "liblvm2clusterlock.so"

# Treat any internal errors as fatal errors, aborting the process that
# encountered the internal error. Please only enable for debugging.
abort_on_internal_errors = 0

# Check whether CRC is matching when parsed VG is used multiple times.
# This is useful to catch unexpected internal cached volume group
# structure modification. Please only enable for debugging.
detect_internal_vg_cache_corruption = 0

# If set to 1, no operations that change on-disk metadata will be
permitted.

```

```

# Additionally, read-only commands that encounter metadata in need of
repair
# will still be allowed to proceed exactly as if the repair had been
# performed (except for the unchanged vg_seqno).
# Inappropriate use could mess up your system, so seek advice first!
metadata_read_only = 0

# 'mirror_segtype_default' defines which segtype will be used when the
# shorthand '-m' option is used for mirroring. The possible options
are:
#
# "mirror" - The original RAID1 implementation provided by LVM2/DM.
It is
#
# characterized by a flexible log solution (core, disk,
mirrored)
#
# and by the necessity to block I/O while reconfiguring in the
# event of a failure. Snapshots of this type of RAID1 can be
# problematic.
#
# "raid1" - This implementation leverages MD's RAID1 personality
through
#
# device-mapper. It is characterized by a lack of log
options.
#
# (A log is always allocated for every device and they are placed
# on the same device as the image - no separate devices are
# required.) This mirror implementation does not require I/O
# to be blocked in the kernel in the event of a failure.
#
# Specify the '--type <mirror|raid1>' option to override this default
# setting.
mirror_segtype_default = "mirror"
}

activation {
# Set to 1 to perform internal checks on the operations issued to
# libdevmapper. Useful for debugging problems with activation.
# Some of the checks may be expensive, so it's best to use this
# only when there seems to be a problem.
checks = 0

# Set to 0 to disable udev synchronisation (if compiled into the
binaries).
# Processes will not wait for notification from udev.
# They will continue irrespective of any possible udev processing
# in the background. You should only use this if udev is not running
# or has rules that ignore the devices LVM2 creates.
# The command line argument --nodevsysnc takes precedence over this
setting.
# If set to 1 when udev is not running, and there are LVM2 processes
# waiting for udev, run 'dmsetup udevcomplete_all' manually to wake
them up.
udev_sync = 1

# Set to 0 to disable the udev rules installed by LVM2 (if built with
# --enable-udev_rules). LVM2 will then manage the /dev nodes and
symlinks

```

```

# for active logical volumes directly itself.
# N.B. Manual intervention may be required if this setting is changed
# while any logical volumes are active.
udev_rules = 1

# Set to 1 for LVM2 to verify operations performed by udev. This turns
on
# additional checks (and if necessary, repairs) on entries in the
device
# directory after udev has completed processing its events.
# Useful for diagnosing problems with LVM2/udev interactions.
verify_udev_operations = 0

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

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

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

# Nice value used while devices suspended
process_priority = -18

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

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

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

# 'mirror_image_fault_policy' and 'mirror_log_fault_policy' define
# how a device failure affecting a mirror is handled.
# A mirror is composed of mirror images (copies) and a log.
# A disk log ensures that a mirror does not need to be re-synced
# (all copies made the same) every time a machine reboots or crashes.
#
# In the event of a failure, the specified policy will be used to

```

```

determine
    # what happens. This applies to automatic repairs (when the mirror is
being
    # monitored by dmeventd) and to manual lvconvert --repair when
    # --use-policies is given.
    #
    # "remove" - Simply remove the faulty device and run without it. If
    #             the log device fails, the mirror would convert to using
    #             an in-memory log. This means the mirror will not
    #             remember its sync status across crashes/reboots and
    #             the entire mirror will be re-synced. If a
    #             mirror image fails, the mirror will convert to a
    #             non-mirrored device if there is only one remaining good
    #             copy.
    #
    # "allocate" - Remove the faulty device and try to allocate space on
    #               a new device to be a replacement for the failed device.
    #               Using this policy for the log is fast and maintains the
    #               ability to remember sync state through crashes/reboots.
    #               Using this policy for a mirror device is slow, as it
    #               requires the mirror to resynchronize the devices, but it
    #               will preserve the mirror characteristic of the device.
    #               This policy acts like "remove" if no suitable device and
    #               space can be allocated for the replacement.
    #
    # "allocate_anywhere" - Not yet implemented. Useful to place the log
device
    #                       temporarily on same physical volume as one of the mirror
    #                       images. This policy is not recommended for mirror devices
    #                       since it would break the redundant nature of the mirror.
This
    #                       policy acts like "remove" if no suitable device and space
can
    #                       be allocated for the replacement.

    mirror_log_fault_policy = "allocate"
    mirror_image_fault_policy = "remove"

    # 'snapshot_autoextend_threshold' and 'snapshot_autoextend_percent'
define
    # how to handle automatic snapshot extension. The former defines when
the
    # snapshot should be extended: when its space usage exceeds this many
    # percent. The latter defines how much extra space should be allocated
for
    # the snapshot, in percent of its current size.
    #
    # For example, if you set snapshot_autoextend_threshold to 70 and
    # snapshot_autoextend_percent to 20, whenever a snapshot exceeds 70%
usage,
    # it will be extended by another 20%. For a 1G snapshot, using up 700M
will
    # trigger a resize to 1.2G. When the usage exceeds 840M, the snapshot
will
    # be extended to 1.44G, and so on.
    #

```

```

# Setting snapshot_autoextend_threshold to 100 disables automatic
# extensions. The minimum value is 50 (A setting below 50 will be
treated
# as 50).

snapshot_autoextend_threshold = 100
snapshot_autoextend_percent = 20

# While activating devices, I/O to devices being (re)configured is
# suspended, and as a precaution against deadlocks, LVM2 needs to pin
# any memory it is using so it is not paged out. Groups of pages that
# are known not to be accessed during activation need not be pinned
# into memory. Each string listed in this setting is compared against
# each line in /proc/self/maps, and the pages corresponding to any
# lines that match are not pinned. On some systems locale-archive was
# found to make up over 80% of the memory used by the process.
# mlock_filter = [ "locale/locale-archive", "gconv/gconv-
modules.cache" ]

# Set to 1 to revert to the default behaviour prior to version 2.02.62
# which used mlockall() to pin the whole process's memory while
activating
# devices.
use_mlockall = 0

# Monitoring is enabled by default when activating logical volumes.
# Set to 0 to disable monitoring or use the --ignoremonitoring option.
monitoring = 1

# When pvmove or lvconvert must wait for the kernel to finish
# synchronising or merging data, they check and report progress
# at intervals of this number of seconds. The default is 15 seconds.
# If this is set to 0 and there is only one thing to wait for, there
# are no progress reports, but the process is awoken immediately the
# operation is complete.
polling_interval = 15
}

#####
# Advanced section #
#####

# Metadata settings
#
# metadata {
#   # Default number of copies of metadata to hold on each PV. 0, 1 or 2.
#   # You might want to override it from the command line with 0
#   # when running pvcreate on new PVs which are to be added to large VGs.

#   pvmetadatascopies = 1

#   # Default number of copies of metadata to maintain for each VG.
#   # If set to a non-zero value, LVM automatically chooses which of
#   # the available metadata areas to use to achieve the requested
#   # number of copies of the VG metadata. If you set a value larger

```

```

# than the total number of metadata areas available then
# metadata is stored in them all.
# The default value of 0 ("unmanaged") disables this automatic
# management and allows you to control which metadata areas
# are used at the individual PV level using 'pvchange
# --metadataignore y/n'.

# vgmetadatasize = 0

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

# pvmetadatasize = 255

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

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

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

    mirror_library = "libdevmapper-event-lvm2mirror.so"

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

    snapshot_library = "libdevmapper-event-lvm2snapshot.so"

    # Full path of the dmeventd binary.

```

```
#  
# executable = "/sbin/dmeventd"  
}
```



## 附錄 C. LVM 物件標籤 (OBJECT TAGS)

LVM 標籤是個可用來將類型相同的 LVM2 物件組織在一起的字串。標籤可被連至像是實體卷冊、卷冊群組，以及邏輯卷冊的物件。標籤可在叢集配置中被連至主機。快照 (Snapshot) 無法被標記。

標籤可代替 PV、VG 或 LV 引數在指令列上提供。標籤應以一個 @ 來作為字首以避免意義不明確。各個標籤都是透過將它取代為持有該標籤、類型基於它在指令列上的位置來斷定的所有物件來擴充的。

由 Red Hat Enterprise Linux 6.1 發行版開始，LVM 標籤字串可達 1024 個字元（在較早的發行版中，最大限制為 128 個字元）。LVM 標籤不可以連字符號作為起始。

有效的標籤只可包含有限範圍的字元。RHEL 6.0 發行版上，允許的字元為[A-Za-z0-9\_+.-]。在 RHEL 6.1 發行版上，允許的字元已增加，並可包含「/」、「=」、「!」、「:」、「#」，以及「&」字元。

只有卷冊群組中的物件可被標記。實體卷冊若由卷冊群組中被移除掉的話，它們便會失去它們的標籤；這是因為標籤會被作為是卷冊群組 metadata 的一部分來儲存，因此當某個實體卷冊被移除時，它們也會跟著被刪除掉。快照無法被標記。

下列指令列出了所有標有著 **database** 標籤的邏輯卷冊。

```
lvs @database
```

### C.1. 新增和移除物件標籤

若要新增或移除實體卷冊的標籤，請使用 **pvchange** 指令的 **--addtag** 或 **--deltag** 選項。

若要新增或移除卷冊群組的標籤，請使用 **vgchange** 或 **vgcreate** 指令的 **--addtag** 或 **--deltag** 選項。

若要新增或移除邏輯卷冊的標籤，請使用 **lvchange** 或 **lvcreate** 指令的 **--addtag** 或 **--deltag** 選項。

由 RHEL 6.1 發行版起，您可在單獨的 **pvchange**、**vgchange** 或 **lvchange** 指令中指定多重的 **--addtag** 和 **--deltag** 引數。比方說，下列指令會將 **T9** 和 **T10** 標籤由 **grant** 卷冊群組中刪除，並新增 **T13** 和 **T14** 標籤。

```
vgchange --deltag T9 --deltag T10 --addtag T13 --addtag T14 grant
```

### C.2. 主機標籤 (HOST TAGS)

在叢集配置中，您可在配置檔案中定義主機標籤。若您在 **tags** 的部份中設置了 **hosttags = 1** 的話，主機標籤就會被透過使用機器的主機名稱來自動地定義。這能讓您使用通用的配置檔案並在您所有的機器上複製，如此一來它們便會持有相同的檔案副本，不過機器之間的特性可能會根據主機名稱而有所不同。

如欲取得更多有關於配置檔案的相關資訊，請參閱〈[附錄 B, LVM 配置檔案](#)〉。

針對於各個主機標籤，會有個額外的配置檔案被讀取（若它存在的話）：**lvm\_hosttag.conf**。若該檔案定義了新的標籤的話，那麼額外的配置檔案便會被附加至需要被讀取的檔案清單中。

比方說，每當主機名稱為 **host1** 時，下列配置檔案中的項目就會定義 **tag1** 以及 **tag2**。

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

### C.3. 利用標籤來控制啓用

您可在配置檔案中指定只有特定邏輯卷冊可在該主機上被啓用。比方說，下列項目被作為是一個啓用請求（例如 **vgchange -ay**）的過濾器，並且只會啓用 **vg1/lvol0** 和該主機上在 metadata 中含有 **database** 標籤的邏輯卷冊或卷冊群組。

```
activation { volume_list = ["vg1/lvol0", "@database" ] }
```

有個特殊的「@\*」match 只會在當任何 metadata 標籤符合該機器上的任何主機標籤時產生一個 match。

讓我們思考另一個當叢集中所有的機器在配置檔案中都含有下列項目的範例：

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

若您只希望在 **db2** 主機上啓用 **vg1/lvol2** 的話，請進行下列步驟：

1. 在叢集中的任何主機上執行 **lvchange --addtag @db2 vg1/lvol2**。
2. 執行 **lvchange -ay vg1/lvol2**。

此方法需將主機名稱儲存在卷冊群組的 metadata 中。

## 附錄 D. LVM 卷冊群組 METADATA

參照叢集群組的配置詳細資料作為 metadata。預設上，完全一樣的 metadata 會被保留在每個卷冊群組的每個實體卷冊之 metadata 區域。LVM 卷冊群組的 metadata 會以 ASCII 方式儲存。

若有個卷冊群組包含著許多實體卷冊，含有許多重複的 metadata 副本是非常沒有效率的。您可透過使用 **pvcreate** 指令的 **--metadatasize 0** 選項來建立一個實體卷冊並且不建立任何 metadata 的副本。一旦您選擇了實體卷冊將會包含的 metadata 副本數量後，您之後便無法再針對它進行變更。不過請注意，不管任何時候，每個卷冊群組都必須包含著至少一個實體卷冊以及一個 metadata 區域（除非您使用了一項能讓您將卷冊群組 metadata 儲存在檔案系統中的進階配置設定）。若您打算在未來將卷冊群組切割的話，所有卷冊群組就都需要至少一個 metadata 副本。

核心的 metadata 是以 ASCII 來儲存的。metadata 區域是個循環緩衝（circular buffer）。新的 metadata 會被附加至較舊的 metadata 然後指向它起始的指標（pointer）將會被更新。

您可以使用 **pvcreate** 指令的 **--metadatasize** 選項來指定 metadata 的大小。對於包含上百個實體卷冊與邏輯卷冊的卷冊群組來說，預設值可能會太小。

### D.1. 實體卷冊標籤（PHYSICAL VOLUME LABEL）

就預設值，**pvcreate** 指令會將實體卷冊標籤放置在第二個 512 位元組的磁區中。這個標籤亦可被選擇性地放置於前四個磁區中的任何一個，因為掃描實體卷冊標籤的 LVM 工具會檢查前四個磁區。實體卷冊的標籤是以 **LABELONE** 這個字串作為起始的。

實體卷冊標籤包含著：

- 實體卷冊的 UUID
- 區塊裝置的大小（以位元組為單位）
- 無終結（NULL-terminated）的資料區域位置清單
- 無終結的 metadata 區域位置清單

Metadata 的位置是以偏差值和大小（單位為位元組）來儲存的。標籤中可放置 15 個左右的位置，不過 LVM 工具目前只使用了 3 個：一個單獨的資料區域加上兩個 metadata 區域。

### D.2. METADATA 內容

卷冊群組 metadata 中包含著：

- 有關於它如何以及何時被建立的相關資訊
- 有關於卷冊群組的資訊

卷冊群組資訊包含著：

- 名稱和特殊 id
- 一個每當 metadata 被更新時便會跟著遞增的版本號碼
- 任何屬性：可讀取/寫入？可重設大小？
- 任何它所可能包含的實體卷冊和邏輯卷冊數量上的管理限制
- 扇區大小（以磁區〔sector〕為單位，定義為 512 個位元組）

- 一系列未經順序排序、構成卷冊群組的實體卷冊，各個都含有：
  - 它的 UUID，用來測定包含著它的區塊裝置
  - 任何屬性，例如實體卷冊是否可被分配
  - 實體卷冊中第一個扇區起始的 offset（單位為磁區〔sector〕）
  - 扇區數量
- 一系列未經順序排序的邏輯卷冊。各個都含有
  - 一系列經過順序排序的邏輯卷冊區段（logical volume segment）。Metadata 會針對於各個區段來包含一個映射以套用至經過排序的實體卷冊區段或是邏輯卷冊區段

### D.3. METADATA 範例

下列顯示了一個稱為 **myvg** 的卷冊群組的 LVM 卷冊群組 metadata 範例。

```
# 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-j06s-RAVx-UREW0G"
        device = "/dev/sdc"      # Hint only

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

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

        status = ["ALLOCATABLE"]
        dev_size = 35964301      # 17.1491 Gigabytes
        pe_start = 384
        pe_count = 4390 # 17.1484 Gigabytes
    }
}
logical_volumes {
    mylv {
        id = "GhUYSF-qVM3-rzQo-a6D2-o0aV-LQet-Ur90F9"
        status = ["READ", "WRITE", "VISIBLE"]
        segment_count = 2

        segment1 {
            start_extent = 0
            extent_count = 1280      # 5 Gigabytes

            type = "striped"
            stripe_count = 1          # linear

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

            type = "striped"
            stripe_count = 1          # linear

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

```

```
|      }
```

## 附錄 E. 修訂歷史

<b>修訂 4.0-2.2.400</b> Rebuild with publican 4.0.0	<b>2013-10-31</b>	<b>Rüdiger Landmann</b>
<b>修訂 4.0-2.2</b> RHEL 6.3 版翻譯完成。	<b>Tue Sep 25 2012</b>	<b>Chester Cheng</b>
<b>修訂 4.0-2.1</b> Translation files synchronised with XML sources 4.0-2	<b>Mon Aug 27 2012</b>	<b>Chester Cheng</b>
<b>修訂 4.0-2</b> 6.3 GA 發行版	<b>Fri Jun 15 2012</b>	<b>Steven Levine</b>
<b>修訂 4.0-1</b> 解決：#787018 撰寫 snapshot 的自動延展功能。  解決：#749932 撰寫 lvextend 的 --nosync 選項。  解決：#758695 讓範例中的提示全數一致。  解決：#729715 撰寫對 LVM RAID 的支援。	<b>Fri Apr 13 2012</b>	<b>Steven Levine</b>
<b>修訂 3.0-4</b> 解決：#755371, #755373, #755374 撰寫 QE 的檢視部份。	<b>Mon Nov 21 2011</b>	<b>Steven Levine</b>
<b>修訂 3.0-3</b> 解決：#749487 釐清 pvcreate 對整個磁碟的運作。	<b>Mon Nov 07 2011</b>	<b>Steven Levine</b>
<b>修訂 3.0-2</b> 解決：#744999 修正錯字。	<b>Wed Oct 12 2011</b>	<b>Steven Levine</b>
<b>修訂 3.0-1</b> RHEL 6.2 Beta 發行版的初始修訂  解決：#730788 撰寫文件，關於實體卷冊空間不再使用時，放棄邏輯卷冊下方實體卷冊空間的問題。  解決：#728361 移除參照到已淘汰文件的連結。  解決：#714579 修正了錯字。  解決：#664107 修正 <b>.cache</b> 檔案的位置。	<b>Mon Sep 19 2011</b>	<b>Steven Levine</b>
<b>修訂 2.0-1</b>	<b>Thu May 19 2011</b>	<b>Steven Levine</b>

Red Hat Enterprise Linux 6.1 初始發行版

Resolves: #694619

記載了延伸邏輯卷冊時的新 **cling** 分配政策。

Resolves: #682649

新增了有關於在叢集卷冊上執行多重鏡像建立指令時的警告。

Resolves: #674100

新增了 **dmsetup ls --tree** 指令的範例輸出。

Resolves: #694607

記載了在單獨指令列上包含多重 **--addtag** 和 **--deltag** 引數上的支援。

Resolves: #694604

記載了標籤中，已延伸並可使用的字元清單。

Resolves: #694611

撰寫對鏡射磁條的支援。

Resolves: #694616

記載了有關於支援鏡像卷冊 snapshot 上的相關資訊。

Resolves: #694618

記載了有關於支援單獨啓用的叢集卷冊 snapshot 上的相關資訊。

Resolves: #682648

記載了當 mirror leg 被重新定位後，mirror log 可能也會被移動。

Resolves: #661530

將範例 **cluster.conf** 更新為記載了最新功能的版本。

Resolves: #642400

新增了有關於叢集紀錄管理由最低叢集 ID 的叢集節點所維護的備註。

Resolves: #663462

移除了 Xen 虛擬機器監控程式過時的參照。

修訂 1.0-1

Wed Nov 10 2010

Steven Levine

Red Hat Enterprise Linux 6 初始發行版



# 索引

## 符號

[/lib/udev/rules.d](#) 目錄, [udev](#) 與裝置管理員的整合

一致的裝置號碼, [一致的裝置號碼](#)

停用卷冊群組, [啟用和停用卷冊群組](#)

單一節點上的唯一, [啟用和停用卷冊群組](#)

限本機節點, [啟用和停用卷冊群組](#)

## 備份

[metadata](#), [邏輯卷冊備份](#), [備份卷冊群組的 Metadata](#)

檔案, [邏輯卷冊備份](#)

## 分割區

多重, [磁碟上的多重分割區](#)

分割區類型, [設定](#), [設定分割區類型](#)

## 初始化

分割區, [初始化實體卷冊](#)

實體卷冊, [初始化實體卷冊](#)

## 剩餘的

[邏輯卷冊](#), [重新為邏輯卷冊命名](#)

## 區塊裝置

掃描, [掃描區塊裝置](#)

## 卷冊群組

[vgs](#) 顯示引數, [vgs 指令](#)

停用, [啟用和停用卷冊群組](#)

## 分割

範例程式, [分割卷冊群組](#)

劃分, [分割卷冊群組](#)

合併, [結合卷冊群組](#)

啟用, [啟用和停用卷冊群組](#)

在系統之間進行移動, [將卷冊群組移至另一部系統](#)

壓縮, [由卷冊群組中移除實體卷冊](#)

定義, [卷冊群組](#)

延伸, [新增實體卷冊至卷冊群組中](#)

建立, [建立卷冊群組](#)

建立於叢集中, [在叢集中建立卷冊群組](#)

擴充, [新增實體卷冊至卷冊群組中](#)

更改參數, [更改卷冊群組的參數](#)

移除, [移除卷冊群組](#)

管理, 一般, [卷冊群組管理](#)

組合, [結合卷冊群組](#)

縮減, [由卷冊群組中移除實體卷冊](#)

重新命名, [為卷冊群組重新命名](#)

顯示, [顯示卷冊群組](#), [LVM 的自訂化回報](#), [vgs 指令](#)

叢集環境, [叢集邏輯卷測管理員 \(Clustered Logical Volume Manager, CLVM\)](#), [在叢集中建立 LVM 卷冊](#)

可用扇區不足的訊息, [邏輯卷冊的可用扇區不足](#)

啟用卷冊群組, [啟用和停用卷冊群組](#)

個別節點, [啟用和停用卷冊群組](#)

限本機節點, [啟用和停用卷冊群組](#)

啟用邏輯卷冊

各別節點, [在叢集中啟用各別節點上的邏輯卷冊](#)

單位, 指令列, [使用 CLI 指令](#)

報告格式, LVM 裝置, [LVM 的自訂化回報](#)

失效的裝置

顯示, [顯示錯誤裝置的相關資訊](#)

實體卷冊

[pvs](#) 顯示引數, [pvs 指令](#)

初始化, [初始化實體卷冊](#)

定義, [實體卷冊](#)

建立, [建立實體卷冊](#)

復原, [替換一個遺失的實體卷冊](#)

復原遺失的卷冊, [將遺失的實體卷冊由一個卷冊群組中移除掉](#)

描述, [LVM 實體卷冊配置](#)

新增至卷冊群組, [新增實體卷冊至卷冊群組中](#)

由卷冊群組中移除, [由卷冊群組中移除實體卷冊](#)

移除, [移除實體卷冊](#)

管理, 一般, [實體卷冊管理](#)

配置, [LVM 實體卷冊配置](#)

重設大小, [重設實體卷冊的大小](#)

顯示, [顯示實體卷冊](#), [LVM 的自訂化回報](#), [pvs 指令](#)

實體扇區

防止配置, [避免配置於實體卷冊上](#)

封存檔案, [邏輯卷冊備份](#)

建立

LVM 卷冊於叢集中, [在叢集中建立 LVM 卷冊](#)

卷冊群組, [建立卷冊群組](#)

卷冊群組, 已叢集化, [在叢集中建立卷冊群組](#)

實體卷冊, [建立實體卷冊](#)

磁條邏輯卷冊, 範例, [建立一個磁條邏輯卷冊 \(Striped Logical Volume\)](#)

邏輯卷冊, [建立線性邏輯卷冊](#)

邏輯卷冊, 範例, [在三個磁碟上建立一個 LVM 邏輯卷冊](#)

## 建立 LVM 卷冊

總覽, [邏輯卷冊建立總覽](#)

## 快取檔案

建置, [掃描磁碟來找尋卷冊群組以便建立快取檔案](#)

## 意見

本指南的聯絡資訊, [我們需要您的意見！](#)

## 扇區

定義, [卷冊群組](#), [建立卷冊群組](#)

配置, [建立卷冊群組](#)

指令列單位, [使用 CLI 指令](#)

## 掃描

區塊裝置, [掃描區塊裝置](#)

掃描裝置, 過濾器, [透過過濾器來控制 LVM 裝置掃描 \(LVM Device Scans\)](#)

## 擴展檔案系統

邏輯卷冊, [在邏輯卷冊上遞增檔案系統](#)

新功能與遭到更改的功能, [新功能與遭到變更的功能](#)

## 檔案系統

在邏輯卷冊上擴展, [在邏輯卷冊上遞增檔案系統](#)

疑難排解, [LVM 疑難排解](#)

## 磁條邏輯卷冊

定義, [等量邏輯卷冊](#)

延伸, [延伸等量的卷冊](#)

建立範例, [建立一個磁條邏輯卷冊 \(Striped Logical Volume\)](#)

擴充, [延伸等量的卷冊](#)

## 移除

實體卷冊, [移除實體卷冊](#)

邏輯卷冊, [移除邏輯卷冊](#)

邏輯卷冊中的磁碟, [由邏輯卷冊中移除磁碟](#)

## 等量邏輯卷冊

建立, [建立等量卷冊](#)

管理程序, [LVM 管理總覽](#)

線上資料重定位, [線上資料重置 \(Online Data Relocation\)](#)

線信邏輯卷冊

轉換為鏡像, [更改鏡像卷冊配置](#)

線性邏輯卷冊

定義, [線性邏輯卷冊](#)

建立, [建立線性邏輯卷冊](#)

總覽

新功能與遭到變更的功能, [新功能與遭到變更的功能](#)

裝置大小, 最大值, [建立卷冊群組](#)

裝置掃描過濾器, [透過過濾器來控制 LVM 裝置掃描 \(LVM Device Scans\)](#)

裝置特殊檔案目錄, [建立卷冊群組](#)

裝置號碼

一致性, [一致的裝置號碼](#)

主要, [一致的裝置號碼](#)

次要, [一致的裝置號碼](#)

裝置路徑名稱, [使用 CLI 指令](#)

記錄, [記錄](#)

詳細輸出, [使用 CLI 指令](#)

說明畫面, [使用 CLI 指令](#)

資料重定位, 線上, [線上資料重置 \(Online Data Relocation\)](#)

路徑名稱, [使用 CLI 指令](#)

過濾器, [透過過濾器來控制 LVM 裝置掃描 \(LVM Device Scans\)](#)

邏輯卷冊

[lvs](#) 顯示引數, [lvs 指令](#)

snapshot, [建立快照卷冊 \(Snapshot Volumes\)](#)

定義, [邏輯卷冊](#), [LVM 邏輯卷冊](#)

延伸, [遞增邏輯卷冊](#)

建立, [建立線性邏輯卷冊](#)

建立範例, [在三個磁碟上建立一個 LVM 邏輯卷冊](#)

擴充, [遞增邏輯卷冊](#)

更改參數, [更改邏輯卷冊群組的參數](#)

本機存取, [在叢集中啓用各別節點上的邏輯卷冊](#)

獨占存取, [在叢集中啓用各別節點上的邏輯卷冊](#)

移除, [移除邏輯卷冊](#)

等量, [建立等量卷冊](#)

管理, 一般, [邏輯卷冊管理](#)

線性, [建立線性邏輯卷冊](#)

縮小, [縮減邏輯卷冊](#)

縮減, [縮減邏輯卷冊](#)

重新命名, [重新為邏輯卷冊命名](#)

重設大小, [重設邏輯卷冊大小](#)

鏡像, [建立鏡像卷冊](#)

顯示, [顯示邏輯卷冊](#), [LVM 的自訂化回報](#), [lvs 指令](#)

## 配置

政策, [建立卷冊群組](#)

防止, [避免配置於實體卷冊上](#)

配置範例, [LVM 配置範例](#)

## 重新命名

卷冊群組, [為卷冊群組重新命名](#)

## 重設大小

實體卷冊, [重設實體卷冊的大小](#)

邏輯卷冊, [重設邏輯卷冊大小](#)

## 鏡像邏輯卷冊

叢集化, [在叢集中建立鏡像 LVM 邏輯卷冊](#)

失效復原, [由 LVM 鏡像錯誤中復原](#)

失效政策, [鏡像邏輯卷冊失效政策](#)

定義, [鏡像邏輯卷冊](#)

延伸, [延伸鏡像卷冊](#)

建立, [建立鏡像卷冊](#)

擴充, [延伸鏡像卷冊](#)

轉換為線性, [更改鏡像卷冊配置](#)

重新配置, [更改鏡像卷冊配置](#)

## 顯示

卷冊群組, [顯示卷冊群組](#), [vgs 指令](#)

實體卷冊, [顯示實體卷冊](#), [pvs 指令](#)

排序輸出, [排序 LVM 報告](#)

邏輯卷冊, [顯示邏輯卷冊](#), [lvs 指令](#)

## A

archive 檔案, [備份卷冊群組的 Metadata](#)

## B

backup 檔案, [備份卷冊群組的 Metadata](#)

## C

### CLVM

定義, [叢集邏輯卷測管理員 \(Clustered Logical Volume Manager, CLVM\)](#)

clvmd daemon, [叢集邏輯卷冊管理員 \(Clustered Logical Volume Manager, CLVM\)](#)

## L

lvchange 指令, [更改邏輯卷冊群組的參數](#)

lvconvert 指令, [更改鏡像卷冊配置](#)

lvcreate 指令, [建立線性邏輯卷冊](#)

lvdisplay 指令, [顯示邏輯卷冊](#)

lvextend 指令, [遞增邏輯卷冊](#)

## LVM

元件, [LVM 架構總覽](#), [LVM 元件](#)

卷冊群組, [定義](#), [卷冊群組](#)

叢集化, [叢集邏輯卷冊管理員 \(Clustered Logical Volume Manager, CLVM\)](#)

實體卷冊管理, [實體卷冊管理](#)

實體卷冊, [定義](#), [實體卷冊](#)

架構總覽, [LVM 架構總覽](#)

標籤, [實體卷冊](#)

歷史, [LVM 架構總覽](#)

目錄結構, [建立卷冊群組](#)

紀錄, [記錄](#)

自訂報告格式, [LVM 的自訂化回報](#)

說明, [使用 CLI 指令](#)

邏輯卷冊管理, [邏輯卷冊管理](#)

LVM1, [LVM 架構總覽](#)

LVM2, [LVM 架構總覽](#)

lvmdiskscan 指令, [掃描區塊裝置](#)

lvreduce 指令, [重設邏輯卷冊大小](#), [縮減邏輯卷冊](#)

lvremove 指令, [移除邏輯卷冊](#)

lvrename 指令, [重新為邏輯卷冊命名](#)

lvs 指令, [LVM 的自訂化回報](#), [lvs 指令](#)

顯示引數, [lvs 指令](#)

lvscan 指令, [顯示邏輯卷冊](#)

## M

man page 畫面, [使用 CLI 指令](#)

## metadata

備份, [邏輯卷冊備份](#), [備份卷冊群組的 Metadata](#)

復原, [復原實體卷冊的 Metadata](#)

mirror\_image\_fault\_policy 配置參數, [鏡像邏輯卷冊失效政策](#)

mirror\_log\_fault\_policy 配置參數, [鏡像邏輯卷冊失效政策](#)

## P

**pvdisplay** 指令, [顯示實體卷冊](#)  
**pvmove** 指令, [線上資料重置 \(Online Data Relocation\)](#)  
**pvremove** 指令, [移除實體卷冊](#)  
**pvresize** 指令, [重設實體卷冊的大小](#)  
**pvs** 指令, [LVM 的自訂化回報](#)  
顯示引數, [pvs 指令](#)

**pvscan** 指令, [顯示實體卷冊](#)

## R

**rules.d** 目錄, [udev 與裝置管理員的整合](#)

## S

**snapshot** 卷冊  
定義, [Snapshot 卷冊](#)

**snapshot** 邏輯卷冊  
建立, [建立快照卷冊 \(Snapshot Volumes\)](#)

## U

**udev** 裝置管理員, [udev Device Manager \(裝置管理員\) 的 Device Mapper 支援](#)  
**udev** 規則, [udev 與裝置管理員的整合](#)

## V

**vgcfbackup** 指令, [備份卷冊群組的 Metadata](#)  
**vgcfrestore** 指令, [備份卷冊群組的 Metadata](#)  
**vgchange** 指令, [更改卷冊群組的參數](#)  
**vgcreate** 指令, [建立卷冊群組, 在叢集中建立卷冊群組](#)  
**vgdisplay** 指令, [顯示卷冊群組](#)  
**vgexport** 指令, [將卷冊群組移至另一部系統](#)  
**vgextend** 指令, [新增實體卷冊至卷冊群組中](#)  
**vgimport** 指令, [將卷冊群組移至另一部系統](#)  
**vgmerge** 指令, [結合卷冊群組](#)  
**vgmknodes** 指令, [重新建立一個卷冊群組目錄](#)  
**vgreduce** 指令, [由卷冊群組中移除實體卷冊](#)  
**vgrename** 指令, [為卷冊群組重新命名](#)  
**vgs** 指令, [LVM 的自訂化回報](#)  
顯示引數, [vgs 指令](#)

**vgscan** 指令, [掃描磁碟來找尋卷冊群組以便建立快取檔案](#)  
**vgsplit** 指令, [分割卷冊群組](#)

