



# Red Hat Enterprise Linux 7

## 論理ボリュームマネージャーの管理

LVM 管理者ガイド



## LVM 管理者ガイド

Steven Levine

Red Hat Customer Content Services

slevine@redhat.com

## 法律上の通知

Copyright © 2017 Red Hat, Inc.

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.

## 概要

本書では、クラスター環境における LVM の実行に関する情報など、LVM 論理ボリュームマネージャーについて説明します。

## 目次

<b>第1章 LVM 論理ボリュームマネージャー</b>	<b>4</b>
1.1. 新機能および変更された機能	4
1.2. 論理ボリューム	5
1.3. LVM アーキテクチャーの概要	6
1.4. クラスター化論理ボリュームマネージャー (CLVM)	7
1.5. ドキュメントの概要	8
<b>第2章 LVM コンポーネント</b>	<b>10</b>
2.1. 物理ボリューム	10
2.2. ボリュームグループ	11
2.3. LVM 論理ボリューム	12
<b>第3章 LVM 管理の概要</b>	<b>19</b>
3.1. クラスター内での LVM ボリューム作成	19
3.2. 論理ボリューム作成の概要	20
3.3. 論理ボリュームのファイルシステムの拡張	21
3.4. 論理ボリュームのバックアップ	21
3.5. ロギング	21
3.6. メタデータデーモン (LVMETAD)	22
3.7. LVM コマンドによる LVM 情報の表示	23
<b>第4章 CLI コマンドでの LVM 管理</b>	<b>24</b>
4.1. CLI コマンドの使用	24
4.2. 物理ボリュームの管理	25
4.3. ボリュームグループの管理	28
4.4. 論理ボリュームの管理	36
4.5. フィルターを使用した LVM デバイススキャンの制御	84
4.6. オンラインデータ移動	85
4.7. クラスター内の個別ノードでの論理ボリュームのアクティブ化	86
4.8. LVM 用のカスタム報告	86
<b>第5章 LVM 設定の例</b>	<b>101</b>
5.1. 3つのディスク上での LVM 論理ボリューム作成	101
5.2. ストライプ化論理ボリュームの作成	102
5.3. ボリュームグループの分割	103
5.4. 論理ボリュームからディスクを削除する	105
5.5. クラスター内でのミラー化 LVM 論理ボリュームの作成	107
<b>第6章 LVM トラブルシューティング</b>	<b>111</b>
6.1. トラブルシューティング診断	111
6.2. 障害の発生したデバイスの情報を表示	111
6.3. LVM ミラー障害からの回復	112
6.4. 物理ボリュームメタデータの復元	115
6.5. 紛失した物理ボリュームの入れ替え	117
6.6. 紛失した物理ボリュームのボリュームグループからの削除	117
6.7. 論理ボリュームでの不十分な空きエクステンツ	117
6.8. マルチパスデバイスに対する重複した PV 警告	118
<b>付録A デバイスマッパー</b>	<b>121</b>
A.1. デバイステーブルのマッピング	121
A.2. DMSETUP コマンド	132
A.3. デバイスマッパーの UDEV デバイスマネージャサポート	136

<b>付録B LVM 設定ファイル</b> .....	<b>141</b>
B.1. LVM 設定ファイル	141
B.2. LVMCONFIG コマンド	141
B.3. LVM プロファイル	142
B.4. サンプル LVM.CONF ファイル	143
<b>付録C LVM 選択基準</b> .....	<b>182</b>
C.1. 選択基準フィールドタイプ	182
C.2. 選択基準演算子	183
C.3. 選択基準フィールド	185
C.4. 時間値の指定	198
C.5. 選択基準表示の例	199
C.6. 選択基準処理の例	202
<b>付録D LVM オブジェクトタグ</b> .....	<b>205</b>
D.1. オブジェクトタグの追加と削除	205
D.2. ホストタグ	205
D.3. タグを使用したアクティブ化の制御	206
<b>付録E LVM ボリュームグループメタデータ</b> .....	<b>207</b>
E.1. 物理ボリュームラベル	207
E.2. メタデータの内容	207
E.3. サンプルのメタデータ	208
<b>付録F 改訂履歴</b> .....	<b>211</b>
<b>索引</b> .....	<b>212</b>



## 第1章 LVM 論理ボリュームマネージャー

この章では、Red Hat Enterprise Linux 7 の初期リリース以降のリリースに新たに組み込まれている LVM 論理ボリュームマネージャーの機能についてまとめています。さらに、論理ボリュームマネージャー (LVM) のコンポーネントの概要を説明します。

### 1.1. 新機能および変更された機能

このセクションでは、Red Hat Enterprise Linux 7 の初期リリース以降に加えられた LVM 論理ボリュームマネージャーの機能の一覧を提供します。

#### 1.1.1. Red Hat Enterprise Linux 7.1 の新機能および変更された機能

Red Hat Enterprise Linux 7.1 ではドキュメントと機能が以下のように更新、変更されています。

- シンプロビジョニングされたボリュームと、シンプロビジョニングされたスナップショットに関する記載がわかりやすくなりました。LVM シンプロビジョニングに関する追加情報は、**lvmthin(7)** の man ページに記載されています。シンプロビジョニングされた論理ボリュームに関する全般的な情報は、「[シンプロビジョニングされた論理ボリューム \(シンボリック\)](#)」を参照してください。シンプロビジョニングされたスナップショットボリュームに関する情報は、「[シンプロビジョニングされたスナップショットボリューム](#)」を参照してください。
- 本書では、**lvm dumpconfig** コマンドについて「[lvmconfig コマンド](#)」で説明しています。Red Hat Enterprise Linux 7.2 リリースで、このコマンドの名前が **lvmconfig** に変更になりました。ただし、以前の書式は引き続き利用できます。
- 本書では、LVM プロファイルの説明を「[LVM プロファイル](#)」に追加しました。
- 本書では、**lvm** コマンドの説明を「[lvm コマンドによる LVM 情報の表示](#)」に追加しました。
- Red Hat Enterprise Linux 7.1 リリースでは、「[論理ボリュームのアクティブ化の制御](#)」に説明されているように、**lvcreate** および **lvchange** コマンドで **-k** と **-K** オプションを使用して、シンプルスナップショットのアクティブ化を制御できます。
- 本書では、**vgimport** コマンドの **--force** 引数について説明します。この引数を使用すると、物理ボリュームがないボリュームグループをインポートし、その後に **vgreduce --removemissing** コマンドを実行することが可能になります。**vgimport** コマンドの詳細は、「[ボリュームグループの別のシステムへの移動](#)」を参照してください。
- 本書では、**vgreduce** コマンドの **--mirroronly** 引数について説明します。この引数を使用すると、障害が発生した物理ボリュームのミラーイメージである論理ボリュームのみを削除することができます。使用方法は、「[ボリュームグループの別のシステムへの移動](#)」を参照してください。

さらに、ドキュメント全体にわたり、技術的な内容の若干の修正と明確化を行いました。

#### 1.1.2. Red Hat Enterprise Linux 7.2 の新機能および変更された機能

Red Hat Enterprise Linux 7.2 ではドキュメントと機能が以下のように更新、変更されています。

- 多くの LVM 処理コマンドで、これらのコマンドの選択基準を定義する **-s** または **--select** オプションを指定できるようになりました。LVM の選択基準は、新たに追加された付録「[付録 C LVM 選択基準](#)」に記載されています。



- 本書では、キャッシュ論理ボリュームの作成に関する基本的な手順について「[LVM 論理ボリュームの作成](#)」で説明します。
- 本書のトラブルシューティングに関する章に、新しいセクション「[マルチパスデバイスに対する重複した PV 警告](#)」が追加されました。
- Red Hat Enterprise Linux 7.2 リリースで、`lvm dumpconfig` コマンドの名前が `lvmconfig` に変更になりました。ただし、以前の書式は引き続き利用できます。この変更は本書全体に反映されています。

さらに、ドキュメント全体にわたり、技術的な内容の若干の修正と明確化を行いました。

### 1.1.3. Red Hat Enterprise Linux 7.3 の新機能および変更された機能

Red Hat Enterprise Linux 7.3 ではドキュメントと機能が以下のように更新、変更されています。

- LVM は RAID0 セグメントタイプをサポートします。RAID0 では、ストライプサイズを単位として、複数のデータサブボリュームに論理ボリュームデータが分散されます。RAID0 ボリュームの作成については、「[RAID0 ボリュームの作成 \(Red Hat Enterprise Linux 7.3 以降\)](#)」を参照してください。
- `lvm fullreport` コマンドを使用して、物理ボリューム、ボリュームグループ、論理ボリューム、物理ボリュームセグメント、および論理ボリュームセグメントに関する情報を一度に報告できます。このコマンドとその機能については、`lvm-fullreport(8)` の man ページを参照してください。
- LVM は、LVM コマンドの実行中に収集された操作、メッセージ、およびオブジェクトごとのステータス (完全なオブジェクト ID 付き) が含まれるメッセージログレポートをサポートします。LVM ログレポートの例は、「[コマンドログレポート \(Red Hat Enterprise Linux 7.3 以降\)](#)」を参照してください。LVM ログレポートの詳細は、`lvmreport(7)` の man ページを参照してください。
- LVM 表示コマンドで `--reportformat` オプションを使用して JSON 形式で出力を表示できます。JSON 形式で表示された出力例は、「[JSON 形式の出力 \(Red Hat Enterprise Linux 7.3 以降\)](#)」を参照してください。
- `lvm.conf` 設定ファイルで `record_lvs_history` メタデータオプションを有効にすることで、削除されたシンスナップショットとシン論理ボリュームの追跡を設定できるようになりました。これにより、元の依存関係チェーンから削除され、過去の論理ボリュームになった論理ボリュームを含む、完全なシンスナップショット依存関係チェーンを表示できます。過去の論理ボリュームについては、「[過去の論理ボリュームの追跡および表示 \(Red Hat Enterprise Linux 7.3 以降\)](#)」を参照してください。

さらに、ドキュメント全体にわたり、技術的な内容の若干の修正と明確化を行いました。

### 1.1.4. Red Hat Enterprise Linux 7.4 の新機能および変更された機能

Red Hat Enterprise Linux 7.4 では以下のドキュメントと機能が以下のように更新、変更されています。

- Red Hat Enterprise Linux 7.4 では、RAID テイクオーバーおよび RAID の再成形 (reshaping) に対応します。機能の概要は、「[RAID テイクオーバー \(Red Hat Enterprise Linux 7.4 以降\)](#)」と「[RAID 論理ボリュームの再成形 \(Red Hat Enterprise Linux 7.4 以降\)](#)」を参照してください。

## 1.2. 論理ボリューム

ボリューム管理によって、物理ストレージ上に抽象化レイヤーが作成され、論理ストレージボリュームを作成できるようになります。論理ストレージボリュームは、様々な面で、物理ストレージディレクトリよりも柔軟性が高くなります。論理ボリュームを作成すると、物理ディスクのサイズに制限されなくなります。また、ハードウェアストレージ設定はソフトウェアには表示されないの、アプリケーションを停止したりファイルシステムをアンマウントしたりせずに、サイズ変更や移動が可能になります。したがって、運用コストが削減できます。

論理ボリュームは、物理ストレージを直接使用する場合と比較して、以下のような利点があります。

- 容量の柔軟性

論理ボリュームを使用すると、ディスクとパーティションを1つの論理ボリュームに集約できるため、ファイルシステムを複数のディスクに渡って拡張できます。

- サイズ変更可能なストレージプール

基礎となるディスクデバイスを再フォーマットしたり、パーティションを再作成したりせずに、簡単なソフトウェアコマンドを使用して論理ボリュームのサイズを拡大または縮小できます。

- オンラインデータ移動

より新しく、迅速で、障害耐性の高いストレージサブシステムを導入するために、システムがアクティブな状態でもデータを移動できます。データは、ディスクが使用中の場合でもディスクに再配置できます。たとえば、ホットスワップ可能なディスクを削除する前に空にすることができます。

- 便宜的なデバイスの命名

論理ストレージボリュームは、ユーザー定義のグループで管理することができ、便宜上命名できます。

- ディスクのストライピング

2つ以上のディスクにまたがってデータをストライプ化する論理ボリュームを作成できます。これにより、スループットが大幅に向上します。

- ボリュームのミラーリング

論理ボリュームは、データのミラーを設定する際に便利な方法を提供します。

- ボリュームスナップショット

論理ボリュームを使用すると、一貫したバックアップが可能なデバイススナップショットを取ったり、実際のデータに影響を及ぼすことなく変更の影響をテストしたりできます。

LVM にこれらの機能を実装する方法は、後の章で説明します。

## 1.3. LVM アーキテクチャーの概要



### 注記

LVM2 には LVM1 との下位互換性があります。ただし、スナップショットとクラスターサポートは例外となっています。ボリュームグループは、**vgconvert** コマンドを使用して LVM1 形式から LVM2 形式に変換することができます。LVM メタデータ形式の変換に関する情報は、**vgconvert(8)** の man ページを参照してください。

LVM 論理ボリュームの基礎となる物理ストレージユニットは、パーティション、ディスク全体などのブロックデバイスです。このデバイスは、LVM **物理ボリューム** (Physical Volume: PV) として初期化されます。

LVM 論理ボリュームを作成するために、物理ボリュームを**ボリュームグループ** (Volume Group: VG) に統合します。これによってディスク領域のプールが作成され、そこから LVM 論理ボリューム (Logical Volume: LV) を割り当てます。このプロセスは、ディスクをパーティションに分割する方法に類似しています。論理ボリュームは、ファイルシステムやアプリケーション (データベースなど) に使用されます。

図1.1「LVM 論理ボリュームのコンポーネント」は、LVM 論理ボリュームのコンポーネントを簡略に示しています。

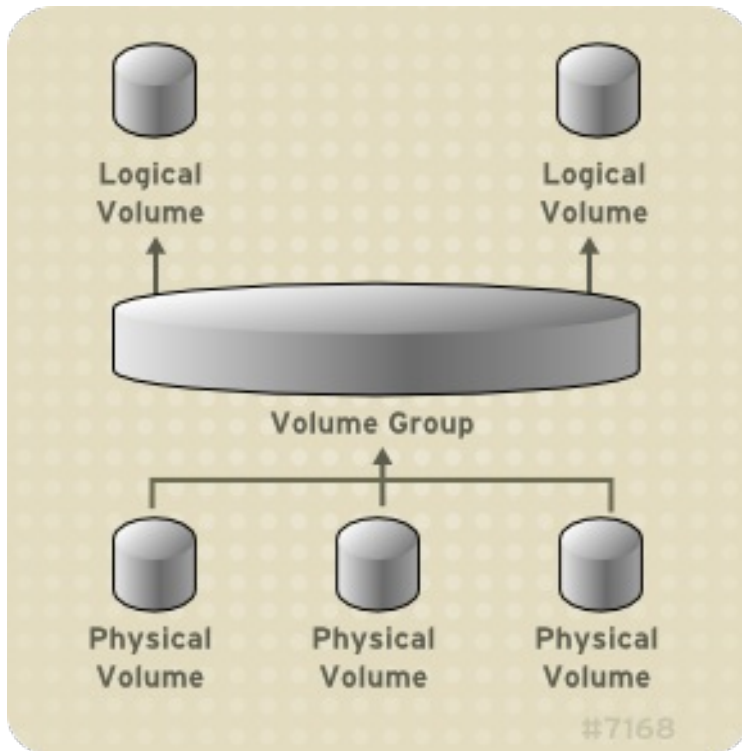


図1.1 LVM 論理ボリュームのコンポーネント

LVM 論理ボリュームのコンポーネントの詳細は、「[2章LVM コンポーネント](#)」を参照してください。

## 1.4. クラスター化論理ボリュームマネージャー (CLVM)

Clustered Logical Volume Manager (CLVM) は、LVM のクラスタリング拡張機能セットです。この拡張機能により、LVM を使用すれば、コンピューターのクラスターで、LVM を使用した共有ストレージ (例: SAN 上のストレージ) を管理できるようになります。CLVM は、Resilient Storage Add-On の一部です。

CLVM を使用すべきかどうかは、システム要件によって異なります。

- ご使用のシステムで、1つのノードのみが、論理ボリュームとして設定したストレージへのアクセスを必要とする場合は、CLVM 拡張機能を使用せずに LVM を使用することがあります。このとき、そのノードで作成される論理ボリュームは、そのノードに対してすべてローカルとなります。さらに、フェイルオーバー用にクラスター化したシステムを使用し、ストレージにアクセスする1つのノードのみが常にアクティブである場合も、CLVM 拡張機能を使用せずに LVM を使用することができます。CLVM 拡張機能が不要なクラスターで論理ボリュームを設定

する場合は、**LVM high availability** リソースエージェントでシステムを設定します。クラスター内でリソースを設定する方法は、『High Availability Add-On Reference』を参照してください。

- ご使用のクラスターで複数のノードが共有ストレージへのアクセスを必要とし、そのストレージがアクティブなノード間で共有される場合は、CLVM を使用する必要があります。CLVM の使用して、論理ボリュームの設定時に物理ストレージへのアクセスをロックすることで、共有ストレージに論理ボリュームを設定することができるようになります。LVM は、クラスター ロッキングサービスを使用して共有ストレージを管理します。CLVM 拡張を必要とするクラスターで論理ボリュームを設定する場合は、**clvm** リソースエージェントを使用してシステムを設定します。クラスター内でリソースを設定する方法は、『High Availability Add-On Reference』を参照してください。

CLVM を使用するには、**clvmd** デーモンが含まれる High Availability Add-On および Resilient Storage Add-On のソフトウェアが稼働していなければなりません。**clvmd** デーモンは、LVM における主要なクラスタリング拡張機能です。**clvmd** デーモンは、各クラスターコンピューターで稼働し、クラスターで LVM メタデータの更新を配布して、論理ボリュームの同一ビューを各クラスターコンピューターに提供します。

図1.2「CLVM の概観」は、クラスターにおける CLVM の概観を示しています。

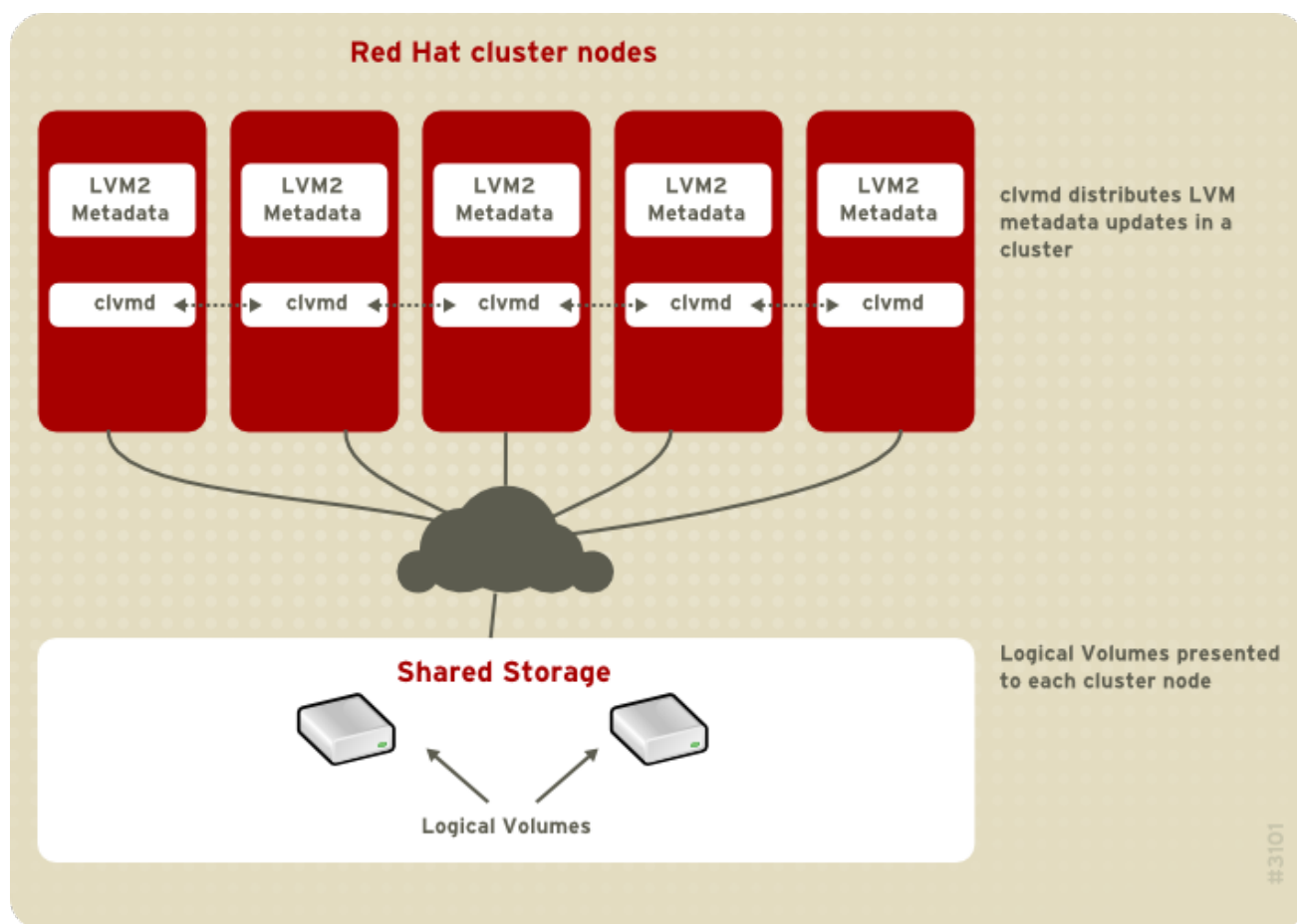


図1.2 CLVM の概観

Red Hat Enterprise Linux 7 では、クラスターは Pacemaker で管理されます。クラスター化された LVM 論理ボリュームは Pacemaker クラスターと併用される場合のみサポートされ、クラスターリソースとして設定する必要があります。クラスターで LVM ボリュームを設定する方法についての情報は、『[クラスター内での LVM ボリューム作成](#)』を参照してください。

## 1.5. ドキュメントの概要

本書には以下の章が含まれます。

- 「[2章LVM コンポーネント](#)」では、LVM 論理ボリュームを構成するコンポーネントについて説明します。
- 「[3章LVM 管理の概要](#)」では、LVM 論理ボリュームを設定するにあたって実行する基本手順の概要を提供します。
- 「[4章CLI コマンドでの LVM 管理](#)」では、論理ボリュームの作成と保守を行うために LVM CLI コマンドで実行できる個別の管理タスクをまとめています。
- 「[5章LVM 設定の例](#)」では、LVM の各種設定を示した例を記載します。
- 「[6章LVM トラブルシューティング](#)」では、LVM の各種問題のトラブルシューティングの手順について説明します。
- 「[付録A デバイスマッパー](#)」では、論理ボリュームと物理ボリュームをマップするために LVM が使用するデバイスマッパーについて説明します。
- 「[付録B LVM 設定ファイル](#)」では、LVM 設定ファイルについて説明します。
- 「[付録D LVM オブジェクトタグ](#)」では、LVM オブジェクトタグとホストタグについて説明します。
- 「[付録E LVM ボリュームグループメタデータ](#)」では、LVM ボリュームグループのメタデータと LVM ボリュームグループのメタデータのサンプルコピーを記載します。

## 第2章 LVM コンポーネント

この章では、LVM 論理ボリュームのコンポーネントについて説明します。

### 2.1. 物理ボリューム

LVM 論理ボリュームの基礎となる物理ストレージユニットは、パーティションやディスク全体のようなブロックデバイスです。LVM 論理ボリューム用にデバイスを使用するには、デバイスを物理ボリューム (PV) として初期化する必要があります。ブロックデバイスを物理ボリュームとして初期化すると、デバイスの先頭位置にラベルが付けられます。

LVM ラベルは、デフォルトでは 2 番目の 512 バイトセクターに配置されます。物理ボリュームを作成する場合は、先頭の 4 つのセクターのいずれかにラベルを配置することにより、このデフォルト設定を書き換えることができます。これにより、必要であれば LVM ボリュームを、これらのセクターの他のユーザーと共存できるようになります。

デバイスは、システムの起動時に任意の順序で立ち上がるため、LVM ラベルで、物理デバイスに正確な ID とデバイスの順序を指定します。LVM ラベルは、再起動してもクラスター全体で持続します。

LVM ラベルは、デバイスを LVM 物理ボリュームとして識別するものです。これには、物理ボリューム用のランダムな一意識別子 (UUID) が含まれます。また、ブロックデバイスのサイズもバイト単位で保存し、LVM メタデータがデバイスのどこに保存されているかも記録します。

LVM メタデータには、システムにある LVM ボリュームグループの設定詳細が含まれています。デフォルトでは、メタデータの複製コピーが、ボリュームグループ内で、すべての物理ボリュームの、すべてのメタデータ領域に保存されています。LVM メタデータのサイズは小さく、ASCII 形式が使用されます。

現在、LVM では、各物理ボリュームにメタデータのコピーを 1 つまたは 2 つ保存できます。デフォルトでは 1 つ保存されます。物理ボリューム上に保存するメタデータのコピー数を一度設定したら、その数を後で変更することはできません。最初のコピーはデバイスの先頭にあるラベルの後に保存されます。2 つ目のコピーがある場合は、デバイスの最後に配置されます。別のディスクに誤って書き込みを行い、ディスクの先頭領域を上書きしてしまった場合でも、デバイス後部にある 2 つ目のコピーでメタデータを復元できます。

LVM メタデータとメタデータパラメーターの変更に関する詳細は、「[付録E LVM ボリュームグループメタデータ](#)」を参照してください。

#### 2.1.1. LVM 物理ボリュームのレイアウト

[図2.1「物理ボリュームレイアウト」](#) は、LVM 物理ボリュームのレイアウトを示しています。LVM ラベルが 2 番目のセクターにあり、その後にメタデータ領域、使用可能なデバイス領域と続きます。



#### 注記

Linux カーネル (および本書全体) では、セクターのサイズは 512 バイトとされています。



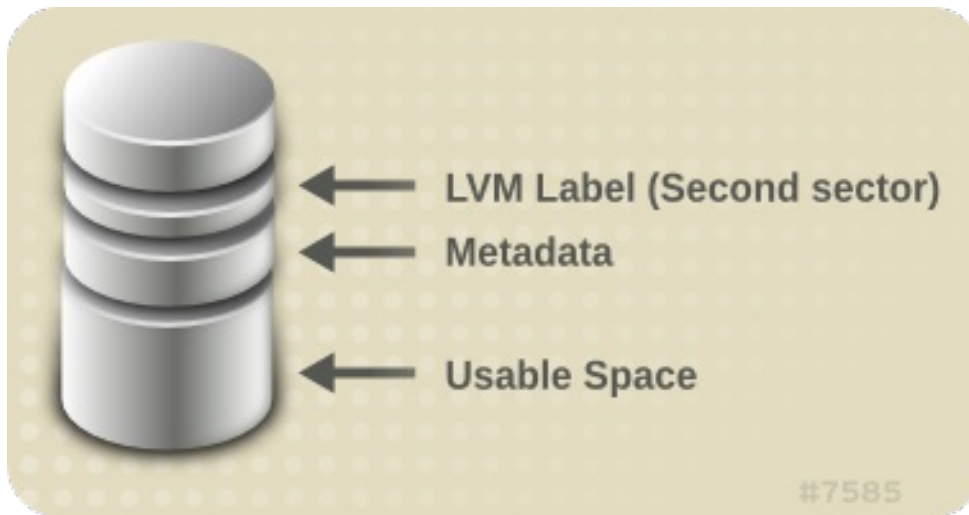


図2.1 物理ボリュームレイアウト

### 2.1.2. ディスク上の複数パーティション

LVM の使用により、ディスクパーティションから物理ボリュームを作成できます。Red Hat では、以下のような理由により、ディスク全体をカバーするパーティションを 1 つ作成し、1 つの LVM 物理ボリュームとしてラベルを付けることを推奨しています。

- 管理上の利便性

それぞれの実ディスクが一度だけ提示されると、システム内のハードウェアを追跡記録するのが簡単になります。これは、特にディスクに障害が発生した場合に役に立ちます。さらに、1 つのディスクに物理ボリュームが複数あると、起動時に、不明なパーティションに関するカーネルの警告が発生する可能性があります。

- ストライピングのパフォーマンス

LVM は、2 つの物理ボリュームが同じ物理ディスクにあるかどうかを認識しません。2 つの物理ボリュームが同じ物理ディスクにあるときに、ストライプ化された論理ボリュームを作成すると、作成されたボリュームのディスクは同じでも、パーティションは異なる可能性があります。このとき、パフォーマンスは向上せず、低下します。

1 つのディスクを、複数の LVM 物理ボリュームに分割しないといけない場合があります (推奨はされません)。たとえば、ディスクがほとんどないシステムで、既存システムを LVM ボリュームに移行する場合に、パーティション間でデータを移動しなければならない場合があります。さらに、大容量のディスクが存在し、管理目的で複数のボリュームグループを必要とする場合は、そのディスクにパーティションを設定する必要があります。ディスクに複数のパーティションがあり、そのパーティションがいずれも同じボリュームグループにある場合に、ストライプ化ボリュームを作成するには、論理ボリュームに追加するパーティションを注意して指定してください。

## 2.2. ボリュームグループ

物理ボリュームはボリュームグループ (VG) に統合されます。これにより、論理ボリュームに割り当てるためのディスク領域のプールが作成されます。

ボリュームグループ内で、割り当て可能なディスク領域は、エクステントと呼ばれる固定サイズの単位に分割されます。1 エクステントが、割り当て可能な領域の最小単位となります。物理ボリューム内では、エクステントは物理エクステントと呼ばれます。

論理ボリュームには、物理エクステントと同じサイズの論理エクステントが割り当てられます。そのため、エクステントのサイズは、ボリュームグループ内のすべての論理ボリュームで同じになります。ボリュームグループは、論理エクステントを物理エクステントにマッピングします。

## 2.3. LVM 論理ボリューム

LVM では、ボリュームグループは論理ボリュームに分割されます。以下のセクションでは、論理ボリュームのタイプを説明します。

### 2.3.1. リニアボリューム

リニアボリュームは、複数の物理ボリュームの領域を 1 つの論理ボリュームに統合します。たとえば、60GB ディスクが 2 つある場合は、120GB の論理ボリュームを作成できます。物理ストレージは連結されます。

リニアボリュームを作成すると、論理ボリュームの領域に、物理エクステントの範囲が順番に割り当てられます。たとえば、[図2.2「エクステントのマッピング」](#)にあるように、1 から 99 までの論理エクステントが 1 つ目の物理ボリュームにマッピングされ、100 から 198 までの論理エクステントが 2 つ目の物理ボリュームにマッピングされます。アプリケーションからは、サイズが 198 エクステントのデバイスが 1 つあるように見えます。

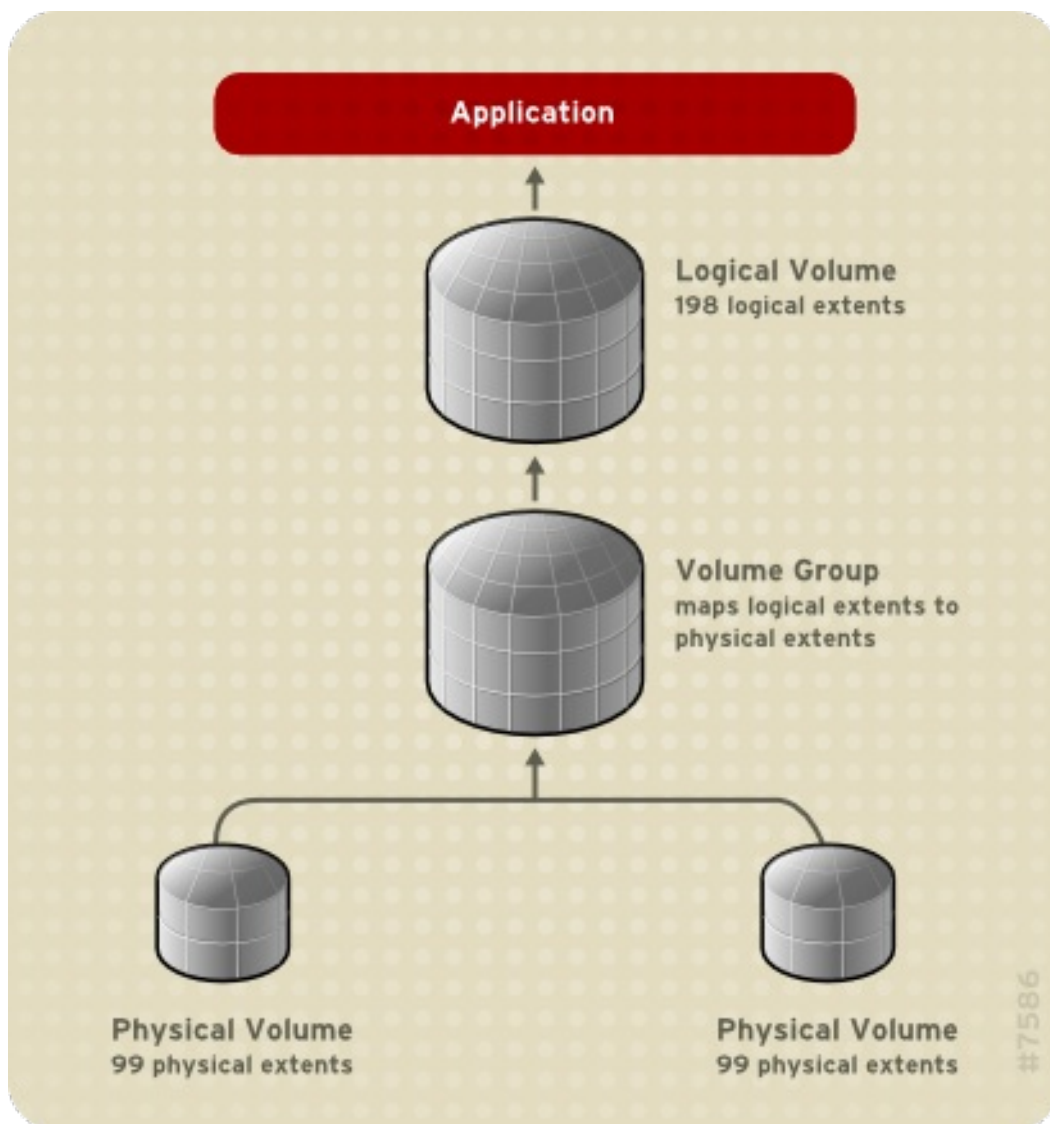


図2.2 エクステントのマッピング



論理ボリュームを構成している各物理ボリュームのサイズは、すべて同じである必要はありません。図2.3「サイズの異なる物理ボリュームを用いたリニアボリューム」は、物理エクステントサイズが4MBのボリュームグループ **VG1** を示しています。このボリュームグループには、**PV1** と **PV2** という2つの物理ボリュームがあります。1エクステントは4MBなので、物理ボリュームが分割される単位は4MBになります。この例では、**PV1** のエクステントサイズは200 (800MB) で、**PV2** のエクステントサイズは100 (400MB) です。リニアボリュームは、エクステントサイズ1から300 (4MBから1200MB)の間で作成できます。この例では、300エクステントのリニアボリューム **LV1** を作成しました。

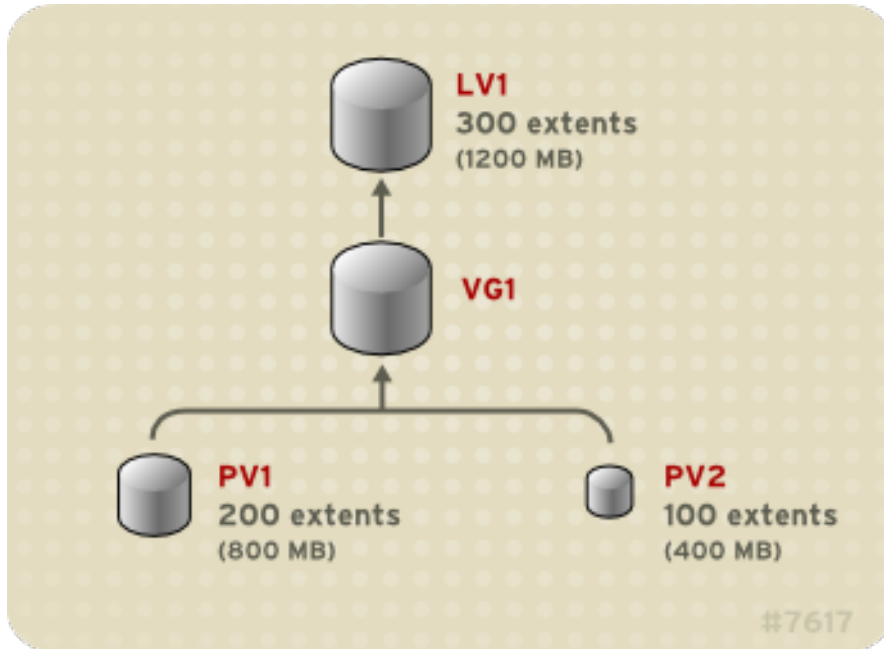


図2.3 サイズの異なる物理ボリュームを用いたリニアボリューム

物理エクステントのプールから、任意のサイズでリニア論理ボリュームを複数設定することができます。図2.4「複数の論理ボリューム」は、図2.3「サイズの異なる物理ボリュームを用いたリニアボリューム」と同じボリュームグループを示していますが、ここでは、そのボリュームグループから論理ボリュームを2つ作成しています。250エクステント (1000MB) の **LV1** と、50エクステント (200MB) の **LV2** です。

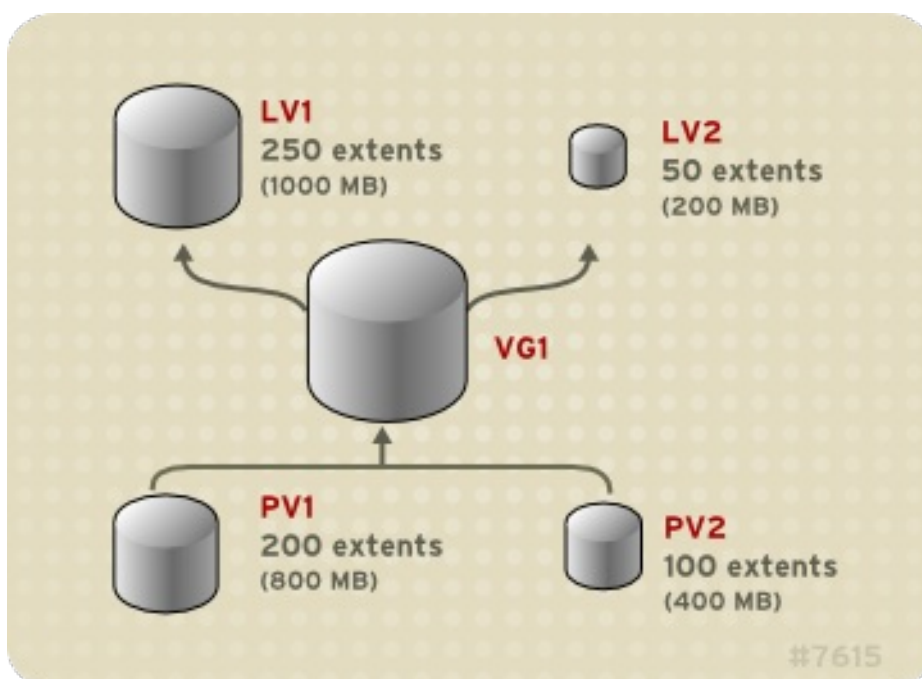


図2.4 複数の論理ボリューム

### 2.3.2. ストライプ化論理ボリューム

LVM 論理ボリュームにデータを書き込む際に、ファイルシステムは、基礎となる物理ボリューム全体にデータを分配します。このとき、ストライプ化論理ボリュームを作成すると、データを物理ボリュームに書き込む方法を制御することができます。順次読み取りと書き込みが大量に行われる場合には、これによりデータ I/O の効率を向上できます。

ストライピングは、ラウンドロビン式で、指定した数の物理ボリュームにデータを書き込んでいくことで、パフォーマンスを向上させます。I/O は、ストライピングでは並行して実行されます。これにより、ストライプで追加される各物理ボリュームでは、ほぼ直線的なパフォーマンスの向上が期待できます。

以下の図では、3 つの物理ボリュームにデータがストライプ化されている状態を示しています。

- データの 1 番目のストライプは、1 番目の物理ボリュームに書き込まれます。
- データの 2 番目のストライプは、2 番目の物理ボリュームに書き込まれます。
- データの 3 番目のストライプは、3 番目の物理ボリュームに書き込まれます。
- データの 4 番目のストライプは、1 番目の物理ボリュームに書き込まれます。

ストライプ化された論理ボリュームでは、ストライプのサイズは、エクステントのサイズより小さくなります。

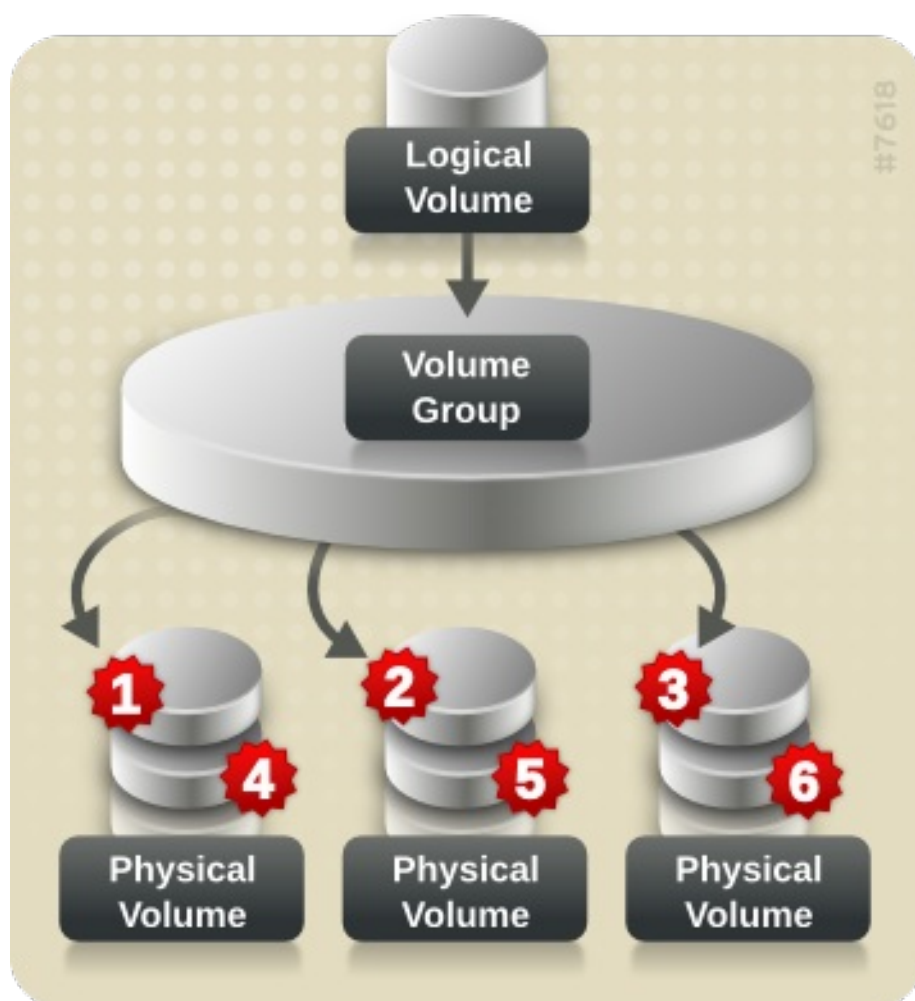


図2.5 3 つの PV にまたがるデータのストライピング

ストライプ化論理ボリュームは、別のデバイスセットを最初のセットの末端に連結すれば拡張できま

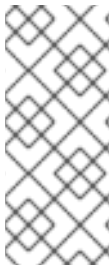
す。ストライプ化論理ボリュームを拡張するには、ストライプに対応するボリュームグループを構成する、基礎となる物理ボリュームセットに、十分な空き領域が必要です。たとえば、ボリュームグループ全域を使用している 2 方向ストライプがある場合は、そのボリュームグループに物理ボリュームを 1 つだけ追加しても、ストライプは拡張できません。ボリュームグループには物理ボリュームを 2 つ以上追加する必要があります。ストライプ化ボリュームの拡張の詳細は、「[ストライプ化ボリュームの拡張](#)」を参照してください。

### 2.3.3. RAID 論理ボリューム

LVM は RAID0/1/4/5/6/10 に対応します。LVM の RAID ボリュームには以下の特徴があります。

- LVM で作成および管理される RAID 論理ボリュームは、MD カーネルドライバを使用します。
- RAID1 イメージはアレイから一時的に切り離して、後でアレイにマージし直すことができます。
- LVM RAID ボリュームはスナップショットに対応します。

RAID 論理ボリュームの作成に関する情報は、「[RAID 論理ボリューム](#)」を参照してください。



#### 注記

RAID 論理ボリュームはクラスターには対応していません。RAID 論理ボリュームは 1 台のマシンに作成でき、かつ排他的にアクティブ化することができますが、複数のマシンで同時にアクティブにすることはできません。排他的ではなく、ミラー化されたボリュームが必要な場合は、「[ミラー化ボリュームの作成](#)」に説明されているように、セグメントタイプの **mirror** を指定してボリュームを作成する必要があります。

### 2.3.4. シンプロビジョニングされた論理ボリューム (シンボリューム)

論理ボリュームのシンプロビジョニングが可能になりました。これにより、利用可能なエクステントよりも大きい論理ボリュームを作成できます。シンプロビジョニングを使用すると、空き領域のストレージプール (シンプールと呼ばれる) を管理して、アプリケーションで必要なときに、任意の数のデバイスに割り当てることができます。その後、アプリケーションが実際に論理ボリュームに書き込むときに割り当てののに使用する、シンプールにバインド可能なデバイスを作成できます。シンプールは、高いストレージ領域をコスト効率よく割り当てるために、動的に拡張できます。



#### 注記

シンボリュームは、クラスターのノード間ではサポートされません。シンプールとそのすべてのシンボリュームは、1 つのクラスターノードでのみ排他的にアクティブ化する必要があります。

ストレージ管理者は、シンプロビジョニングを使用することで物理ストレージをオーバーコミットできるため、多くの場合、追加のストレージを購入する必要がなくなります。たとえば、10 人のユーザーから、各自のアプリケーションに使用するファイルシステムをそれぞれ 100GB 要求された場合、ストレージ管理者は各ユーザーに 100GB と示されるファイルシステムを作成します (ただし、実際には 100GB 未満のストレージが、必要に応じて使用されます)。シンプロビジョニングを使用する場合、ストレージ管理者がストレージプールを監視し、容量が一杯になり始めたら容量を追加することが重要です。

利用可能な領域をすべて使用できるようにするために、LVM はデータの破棄に対応します。これにより、破棄されたファイルや、その他のブロック範囲で以前に使用された領域を再利用できます。

シンボリックの作成の詳細は、「[シンプロビジョニングされた論理ボリュームの作成](#)」を参照してください。

シンボリックは、新たに実装されたコピーオンライト (COW) スナップショット論理ボリュームに対応します。これにより、多くの仮想デバイスでシンボリック内の同一データを共有できます。シンプロビジョニングされたスナップショットボリュームの詳細は、「[シンプロビジョニングされたスナップショットボリューム](#)」を参照してください。

### 2.3.5. スナップショットボリューム

LVM スナップショット機能により、サービスを中断せずに任意の時点でデバイスの仮想イメージを作成できます。スナップショットの取得後に複製元のデバイスに変更が加えられると、データが変更される前に、これから変更される部分のコピーがスナップショット機能により作成されるため、このコピーを使って、デバイスの状態を再構築できます。



#### 注記

LVM は、シンプロビジョニングされたスナップショットに対応します。シンプロビジョニングされたスナップショットボリュームの詳細については、「[シンプロビジョニングされたスナップショットボリューム](#)」を参照してください。



#### 注記

LVM スナップショットは、クラスター内のノード間ではサポートされていません。クラスター化されたボリュームグループ内ではスナップショットボリュームを作成できません。

スナップショットは、スナップショットが作成されてから変更されたデータ部分のみをコピーするため、スナップショット機能に必要なストレージの容量は最小限で済みます。たとえば、複製元の大部分が更新されない場合は、その容量の 3-5 % があればスナップショットを十分に維持することができます。



#### 注記

ファイルシステムのスナップショットコピーは仮想コピーであり、ファイルシステムのメディアバックアップを実際に作成するわけではありません。スナップショットは、バックアップの代替手順にはなりません。

複製元のボリュームへの変更を保管するために確保するスペース量は、スナップショットのサイズによって異なります。たとえば、スナップショットを作成してから複製元を完全に上書きした場合に、その変更を保管するのに必要なスナップショットのサイズは、複製元のボリュームと同じか、それ以上になります。スナップショットのサイズは、予想される変更レベルに応じて決定する必要があります。たとえば、`/usr` など、その大部分が読み取り用に使用されるボリュームの短期的なスナップショットに必要なスペースは、`/home` のように大量の書き込みが行われるボリュームの長期的なスナップショットに必要なスペースよりも小さくなります。

スナップショットが満杯になると、そのスナップショットは無効になります。複製元のボリューム上の変更をトラッキングできなくなるためです。スナップショットのサイズは常時監視する必要があります。ただし、スナップショットのサイズは完全に変更することが可能なため、ストレージに余裕があれば、スナップショットが停止しないように、ボリュームサイズを拡大することができます。逆に、スナップショットボリュームサイズが必要以上に大きければ、そのボリュームのサイズを縮小して、他の論理ボリュームで必要となる領域を確保することができます。



スナップショットのファイルシステムを作成すると、複製元への完全な読み取り/書き込みのアクセスがそのまま残ります。スナップショット上のチャンクを変更した場合は、そのチャンクにマークが付けられ、そこには複製元のボリュームのコピーは入りません。

スナップショット機能にはいくつかの用途があります。

- 最も一般的な用途は、継続的にデータを更新している稼働中のシステムを停止せずに、論理ボリューム上でバックアップを実行する必要がある場合のスナップショット作成です。
- スナップショットファイルシステムで **fsck** コマンドを実行し、ファイルシステムの整合性をチェックすれば、複製元のファイルシステムを修復する必要があるかどうかを判断できます。
- スナップショットは読み取り/書き込み用なので、スナップショットを取ってそのスナップショットに対してテストを実行することにより、実際のデータに触れることなく、実稼働データに対するアプリケーションのテストを実行できます。
- LVM ボリュームを作成して、Red Hat の仮想化と併用することが可能です。LVM スナップショットを使用して、仮想ゲストイメージのスナップショットを作成できます。このスナップショットは、最小限のストレージを使用して、既存のゲストの変更や新規ゲストの作成を行う上で利便性の高い方法を提供します。Red Hat Virtualization による LVM ベースのストレージプールの作成についての詳細は、『仮想化管理ガイド』を参照してください。

スナップショットボリュームの作成に関する情報は、「[ミラー化ボリュームの作成](#)」を参照してください。

**lvconvert** コマンドの **--merge** オプションを使用して、スナップショットを複製元のボリュームにマージすることが可能です。この機能の用途の 1 つとして挙げられるのはシステムロールバックの実行で、データやファイルを紛失した場合や、システムを以前の状態に復元する必要がある場合に行います。スナップショットボリュームのマージの結果作成される論理ボリュームには、複製元のボリューム名、マイナー番号、UUID が付けられ、マージされたスナップショットは削除されます。このオプションの使用方法は、「[スナップショットボリュームのマージ](#)」を参照してください。

### 2.3.6. シンプロビジョニングされたスナップショットボリューム

Red Hat Enterprise Linux は、シンプロビジョニングされたスナップショットボリュームに対応します。シンプロビジョニングされたスナップショットボリュームにより、多くの仮想デバイスを同じデータボリュームに格納することができます。これにより管理が簡略化され、スナップショットボリューム間のデータ共有が可能になります。

シンボリュームや、LVM スナップショットボリュームの場合、シンプロビジョニングされたスナップショットボリュームは、クラスター内のノード間でサポートされていません。スナップショットボリュームは、1 つのクラスターノードで排他的にアクティブ化する必要があります。

シンプロビジョニングされたスナップショットボリュームの利点は以下のとおりです。

- 同じボリュームからのスナップショットが複数ある場合に、シンプロビジョニングされたスナップショットボリュームを使えば、ディスクの使用量を減らすことができます。
- 複製元が同じスナップショットが複数ある場合は、複製元に 1 回書き込むことにより 1 回の COW 操作でデータを保存できます。複製元のスナップショットの数を増やしても、速度が大幅に低下することはありません。
- シンプロビジョニングされたスナップショットボリュームは、別のスナップショットの作成元の論理ボリュームとして使用できます。これにより、再帰的スナップショット (スナップショットのスナップショットのそのまたスナップショットなど) の深度を任意に決定できます。

- シン論理ボリュームのスナップショットにより、シン論理ボリュームを作成することもできます。COW 操作が必要になるまで、あるいはスナップショット自体が書き込まれるまで、データ領域は消費されません。
- シンスナップショットボリュームは、複製元とともにアクティブにしておく必要はありません。そのため、スナップショットボリュームが多数ある場合は、複製元のみをアクティブにし、スナップショットボリュームはアクティブにしないことができます。
- シンプロビジョニングされたスナップショットボリュームの複製元を削除すると、そのボリュームのスナップショットは、それぞれ独立したシンプロビジョニングボリュームになります。したがって、スナップショットとその複製元のボリュームをマージする代わりに、複製元のボリュームを削除し、その独立したボリュームを新たな複製元ボリュームにして、シンプロビジョニングされたスナップショットを新たに作成できます。

シンスナップショットボリュームを使用する利点は数多くありますが、古い LVM スナップショットボリューム機能の方がニーズに沿うケースもあります。

- シンプルのチャンクサイズは変更できません。シンプルのチャンクサイズが大きい場合 (1MB など) や、チャンクサイズが短時間のスナップショットには効率的でない場合は、代わりに以前のスナップショット機能を使用できます。
- シンスナップショットボリュームのサイズを制限することはできません。スナップショットは、必要な場合はシンプル内の全領域を使用するため、ニーズに沿っていない場合があります。

一般的には、使用するスナップショットの形式を決定する際に、使用しているサイトの特定要件を考慮に入れるようにしてください。

シンスナップショットボリュームの設定についての詳細は、[「シンプロビジョニングされたスナップショットボリュームの作成」](#) を参照してください。

### 2.3.7. キャッシュボリューム

Red Hat Enterprise Linux 7.1 リリースでは、LVM は高速ブロックデバイス (SSD ドライブなど) を、大規模な低速ブロックデバイスのライトバックまたはライトスルーキャッシュとして使用することをサポートします。既存の論理ボリュームのパフォーマンスを改善するためにキャッシュ論理ボリュームを作成したり、大規模で低速なデバイスと共に小規模で高速なデバイスで構成される新規のキャッシュ論理ボリュームを作成したりできます。

LVM キャッシュボリュームの作成については、[「LVM 論理ボリュームの作成」](#) を参照してください。

## 第3章 LVM 管理の概要

この章では、LVM 論理ボリュームを設定するのに使用する管理手順の概要を説明します。本章は、必要なステップについて全般的な理解を図ることを目的としています。一般的な LVM 設定手順の各具体例は、「[5章 LVM 設定の例](#)」を参照してください。

LVM 管理を実行するのに使用できる CLI コマンドの説明は、「[4章 CLI コマンドでの LVM 管理](#)」を参照してください。

### 3.1. クラスター内での LVM ボリューム作成

クラスター環境内で論理ボリュームを作成するには、CLVM (Clustered Logical Volume Manager) を使用します。これは LVM へのクラスタリング拡張のセットです。この拡張により、コンピューターのクラスターが LVM を使用して (SAN 上などの) 共有ストレージを管理できるようになります。

Red Hat Enterprise Linux 7 では、クラスターは Pacemaker で管理されます。クラスター化された LVM 論理ボリュームは、Pacemaker クラスターと併用される場合のみサポートされるため、クラスターリソースとして設定する必要があります。

以下は、クラスター化された LVM ボリュームをクラスターリソースとして設定するために必要な手順の概要になります。

1. クラスターソフトウェアおよび LVM パッケージをインストールし、クラスターソフトウェアを起動してクラスターを作成します。クラスターにはフェンスを設定する必要があります。『High Availability Add-On の管理』ドキュメントには、クラスターを作成し、クラスター内にノードのフェンスを設定する手順例が記載されています。『High Availability Add-On Reference』ドキュメントには、クラスター設定のコンポーネントについて詳細情報が記載されています。
2. CLVM (Clustered Logical Volume Manager) には、クラスターロックを有効にするのに、各ノードの `/etc/lvm/lvm.conf` ファイルが必要になります。root で `lvmconf --enable-cluster` コマンドを使用すれば、クラスターロックを有効できます。このコマンドを実行するとロックタイプが変更するため、`lvmetad` デーモンは無効になります。`lvmetad` デーモンの詳細は、「[メタデータデーモン \(lvmetad\)](#)」を参照してください。

クラスター化されたロックをサポートするために `lvm.conf` ファイルを手動で設定する方法は、`lvm.conf` ファイルに記載されています。`lvm.conf` ファイルの詳細は、「[付録B LVM 設定ファイル](#)」を参照してください。

3. クラスターの `dlm` リソースをセットアップします。リソースをクローンリソースとして作成し、クラスター内のすべてのノードでそのリソースが実行されるようにします。

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

4. `clvmd` をクラスターリソースとして設定します。`dlm` リソースの場合と同様に、リソースをクローンリソースとして作成し、そのリソースがクラスター内のすべてのノードで実行されるようにします。

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

5. `clvmd` および `dlm` の依存関係をセットアップし、順番に起動します。`dlm` を起動してから、`dlm` と同じノードで `clvmd` を起動する必要があります。

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

6. クラスター化された論理ボリュームを作成します。クラスター環境に LVM 論理ボリュームを作成する場合は、単一ノード上に LVM 論理ボリュームを作成するのと同じ LVM コマンドを使用します。クラスター内に作成する LVM ボリュームを有効にするには、クラスターインフラストラクチャーが稼働中で、かつクラスターが定足数を満たしている必要があります。

デフォルトでは、CLVM で共有ストレージ上に作成した論理ボリュームは、その共有ストレージにアクセス可能なすべてのシステムに表示されます。ただし、すべてのストレージデバイスを、クラスター内の 1 つのノードからしか見れないようにボリュームグループを作成することも可能です。また、ボリュームグループのステータスを、ローカルボリュームグループからクラスター化されたボリュームグループへ変更することもできます。詳細は、「[クラスター内でのボリュームグループ作成](#)」および「[ボリュームグループのパラメーター変更](#)」を参照してください。



#### 警告

CLVM を使用して共有ストレージ上にボリュームグループを作成する際には、クラスター内のすべてのノードが、ボリュームグループを構成する物理ボリュームに確実にアクセスできるようにする必要があります。ストレージにアクセスできるノードとできないノードが混在する、非対称型のクラスター構成はサポートされていません。

クラスター内でミラー化された論理ボリュームを作成する例は、「[クラスター内でのミラー化 LVM 論理ボリュームの作成](#)」を参照してください。

## 3.2. 論理ボリューム作成の概要

以下は、LVM 論理ボリュームを作成するのに必要な手順の概要です。

1. LVM ボリューム用に使用するパーティションを物理ボリュームとして初期化します (この操作によってラベル付けされます)。
2. ボリュームグループを作成します。
3. 論理ボリュームを作成します。

論理ボリュームを作成したら、ファイルシステムを作成してマウントします。本書の例では、GFS2 ファイルシステムを使用します。

1. **mkfs.gfs2** コマンドを使用して、論理ボリューム上に GFS2 ファイルシステムを作成します。
2. **mkdir** コマンドで新規のマウントポイントを作成します。クラスター化システムでは、そのクラスター内のすべてのノードにマウントポイントを作成します。
3. ファイルシステムをマウントします。各ノードの **fstab** に一行追加することもできます。





## 注記

GFS2 ファイルシステムはスタンドアロンシステム、またはクラスター構成で実装することが可能ですが、Red Hat Enterprise Linux 7 リリースでは、GFS2 を 1 ノードのファイルシステムとして使用することはサポートしていません。ただし、(たとえばバックアップを取得するために) クラスターファイルシステムのスナップショットをマウントする場合は引き続きサポートされます。

LVM セットアップ情報の保存エリアは、ボリュームが作成されたマシンではなく物理ボリュームにあるため、LVM ボリュームの作成はマシンから独立して行われます。ストレージを使用するサーバーにはローカルコピーがありますが、物理ボリュームにあるものから再作成することもできます。LVM のバージョンが互換性を持つ場合には、物理ボリュームを異なるサーバーに接続できます。

## 3.3. 論理ボリュームのファイルシステムの拡張

論理ボリュームのファイルシステムを拡張するには、以下の手順を実行します。

1. 既存のボリュームグループに、論理ボリュームを拡張するのに十分な空き領域があるかどうかを調べます。空き領域が足りない場合は、次の手順を実行します。
  1. **pvcreate** コマンドを使用して、新しい物理ボリュームを作成します。
  2. **vgextend** コマンドを使用して、その物理ボリュームを組み込む、拡張対象の論理ボリュームが含まれるボリュームグループを拡張します。
2. ボリュームグループを拡張できたら、**lvresize** コマンドで論理ボリュームを拡張して、ファイルシステムが拡張できるようにします。
3. 論理ボリュームでファイルシステムのサイズを変更します。

**lvresize** コマンドの **-r** オプションを使用すれば、1 つのコマンドで、論理ボリュームを拡張し、基礎となるファイルシステムのサイズを変更できることに注意してください。

## 3.4. 論理ボリュームのバックアップ

メタデータのバックアップとアーカイブは、**lvm.conf** ファイルで無効になっていない限り、ボリュームグループと論理ボリューム設定の変更時に自動的に作成されます。デフォルトでは、メタデータのバックアップは **/etc/lvm/backup** ファイルに保存され、メタデータのアーカイブは **/etc/lvm/archive** ファイルに保存されます。**/etc/lvm/archive** ファイルに保存されるメタデータのアーカイブの保持期間と、保存されるアーカイブファイルの数は、**lvm.conf** ファイルのパラメーターで指定されます。日次のシステムバックアップでは、バックアップに **/etc/lvm** ディレクトリーの内容が含まれます。

メタデータバックアップでは、論理ボリュームのユーザーとシステムのデータはバックアップされない点に注意してください。

**vgcfgbackup** コマンドを使用すると、手動で **/etc/lvm/backup** ファイルにメタデータをバックアップできます。また **vgcfgrestore** コマンドを使用すると、メタデータを復元できます。**vgcfgbackup** コマンドと **vgcfgrestore** コマンドについては「[ボリュームグループメタデータのバックアップ](#)」で説明されています。

## 3.5. ロギング

すべてのメッセージ出力は、以下についてそれぞれ選択されたロギングレベルで、ロギングモジュールから出力されます。

- 標準出力/エラー
- syslog
- ログファイル
- 外部ログ関数

ロギングのレベルは `/etc/lvm/lvm.conf` ファイルに設定されます。これについては、「[付録B LVM 設定ファイル](#)」に説明されています。

### 3.6. メタデータデーモン (LVMETAD)

LVM はオプションで中央メタデータキャッシュを使用できます。これはデーモン (`lvmetad`) と `udev` ルールにより実装されます。このメタデータデーモンの目的は主に 2 つあります。1 つ目は LVM コマンドのパフォーマンスを向上すること、2 つ目は論理ボリュームまたはボリュームグループ全体がシステムで利用可能になったら、`udev` で自動的にアクティブにできるようにすることです。

`lvm.conf` 設定ファイル内で `global/use_lvmetad` 変数を 1 に設定すると、LVM を設定してデーモンを使用できるようになります。これはデフォルト値です。`lvm.conf` 設定ファイルの詳細は、「[付録B LVM 設定ファイル](#)」を参照してください。



#### 注記

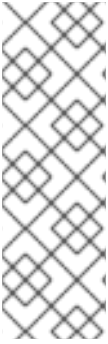
現在、`lvmetad` デーモンはノード間ではサポートされておらず、ロックタイプはローカルのファイルベースにする必要があります。`lvmetad --enable-cluster/--disable-cluster` コマンドを使用すると、`use_lvmetad` 設定が含まれる `lvm.conf` ファイルが適切に設定されます (`locking_type=3` の場合は `use_lvmetad` を 0 に設定する必要があります)。

`use_lvmetad` の値を 1 から 0 に設定するには、以下のコマンドを使用して `lvmetad` サービスを手動で再起動するか、または停止する必要があります。

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

通常、各 LVM コマンドを使用すると、ディスクスキャンが実行され、関連するすべての物理ボリュームが検索され、ボリュームグループのメタデータが読み取られます。ただし、メタデータデーモンが実行中で有効な場合、このスキャンは負荷がかかるため省略できます。代わりに、`udev` ルールを使って、各デバイスが利用可能になったら `lvmetad` デーモンが一度だけデバイスをスキャンするようにします。これにより I/O の量が大幅に削減されるため、特にディスクが多いシステムで LVM 操作を完了するのに必要な時間を減らすことができます。

実行時に新規のボリュームグループが利用可能な場合 (例: ホットプラグまたは iSCSI を使用)、その論理ボリュームを使用するにはアクティブにする必要があります。`lvmetad` デーモンが有効な場合は、`lvm.conf` 設定ファイルの `activation/auto_activation_volume_list` オプションを使用して、自動的にアクティブにするボリュームグループまたは論理ボリューム (あるいはその両方) の一覧を設定できます。`lvmetad` デーモンがアクティブでない場合は、手動でアクティブにする必要があります。



## 注記

**lvm**metad デーモンが実行していると、**pvscan --cache device** コマンドを実行しても **/etc/lvm/lvm.conf** ファイルの **filter =** 設定が適用されません。デバイスにフィルターを設定するには、**global\_filter =** 設定を使用する必要があります。グローバルフィルターに失敗したデバイスは LVM では開かれず、スキャンもされません。VM で LVM を使用しているときに、VM 内のデバイスのコンテンツを物理ホストでスキャンする必要がない場合などは、グローバルフィルターの使用が必要になる場合があります。

## 3.7. LVM コマンドによる LVM 情報の表示

**lvm** コマンドは、LVM サポートおよび設定に関する情報を表示するのに使用できる、いくつかのビルトインオプションを提供します。

- **lvm devtypes**

認識されているビルトインブロックデバイスのタイプを表示します (Red Hat Enterprise Linux リリース 6.6 以降)。

- **lvm formats**

認識されているメタデータ形式を表示します。

- **lvm help**

LVM ヘルプテキストを表示します。

- **lvm segtypes**

認識されている論理ボリュームセグメントタイプを表示します。

- **lvm tags**

このホストに定義されているタグを表示します。LVM オブジェクトタグの情報は、「[付録D LVM オブジェクトタグ](#)」を参照してください。

- **lvm version**

現在のバージョン情報を表示します。

## 第4章 CLI コマンドでの LVM 管理

この章では、論理ボリュームを作成し、保守するために LVM CLI (Command Line Interface) コマンドで実行できる個別の管理タスクについてまとめています。



### 注記

クラスター環境用に LVM ボリュームを作成または変更するには、**clvmd** デーモンが稼働している必要があります。詳細は「[クラスター内での LVM ボリューム作成](#)」を参照してください。

LVM コマンドラインインターフェース (CLI) の他にも、System Storage Manager (SSM) を使用して LVM 論理ボリュームを設定することができます。SSM と LVM の使用方法については、『ストレージ管理ガイド』を参照してください。

### 4.1. CLI コマンドの使用

すべての LVM CLI コマンドに共通する特性がいくつかあります。

コマンドライン引数でサイズ指定が必要な場合に、単位を指定することができます。単位を指定しない場合はデフォルトの単位 (通常 KB か MB) が使用されます。LVM CLI コマンドでは、小数は使用できません。

LVM では、コマンドライン引数で単位を指定する場合に、大文字と小文字は区別されず (たとえば、M と m はいずれも同じ単位)、2 の累乗 (1024 の倍数) が使用されます。ただし、**--units** 引数では、小文字の場合は単位が 1024 の倍数、大文字の場合は 1000 の倍数になります。

コマンドが、ボリュームグループ名または論理ボリューム名を引数として取る場合、完全パスにするかどうかはオプションとなります。たとえば、ボリュームグループ **vg0** 内の論理ボリューム **lv010** は、**vg0/lv010** と指定できます。ボリュームグループの一覧が必要なときに指定しないと、すべてのボリュームグループが対象になります。同様に、論理ボリュームの一覧が必要なときにボリュームグループを指定すると、そのボリュームグループ内の論理ボリュームの一覧が示されます。したがって、たとえば **lvdisplay vg0** コマンドは、ボリュームグループ **vg0** 内のすべての論理ボリュームを表示します。

すべての LVM コマンドで **-v** 引数を使用できるため、この引数を複数回指定して出力の詳細度を高くすることができます。たとえば、次の例は、**lvcreate** コマンドのデフォルト出力になります。

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

次の例は、**lvcreate** コマンドに **-v** 引数を使用した場合の出力になります。

```
# 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-lvol0 (253:2)
Clearing start of logical volume "lvol0"
Creating volume group backup "/etc/lvm/backup/new_vg" (seqno 5).
Logical volume "lvol0" created

```

引数を **-vv**、**-vvv**、**-vvvv** とすると、表示されるコマンドの出力は徐々に詳しくなります。**-vvvv** 引数は、現時点で最も詳細な情報を提供します。以下の例は、**lvcreate** コマンドに **-vvvv** 引数を指定した場合に出力される最初の数行になります。

```

# 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
...

```

LVM のすべての CLI コマンドで、**--help** 引数を付ければ、そのコマンドのヘルプが表示されます。

```
# commandname --help
```

コマンドの man ページを表示するには、**man** コマンドを実行します。

```
# man commandname
```

**man lvcreate** コマンドは、LVM に関する一般的なオンライン情報を提供します。

すべての LVM オブジェクトは、内部的には、オブジェクトを作成する際に割り当てられる UUID で参照されます。これは、たとえばボリュームグループの一部である物理ボリューム **/dev/sdf** を削除してから接続し直したときに、名前が **/dev/sdk** に変わった場合などに役立ちます。LVM は、物理ボリュームを、デバイス名ではなく UUID で識別するため、デバイス名が変わっても物理ボリュームを見つけることができます。物理ボリュームの作成時に物理ボリュームの UUID を指定する方法については、「[物理ボリュームメタデータの復元](#)」を参照してください。

## 4.2. 物理ボリュームの管理

このセクションでは、物理ボリューム管理の様々な場面で使用するコマンドについて説明します。

### 4.2.1. 物理ボリュームの作成

以下のセクションでは、物理ボリュームの作成に使用するコマンドを説明します。

#### 4.2.1.1. パーティションタイプの設定

ディスクデバイス全体を物理ボリュームに使用している場合は、そのディスクにパーティションテーブルがあってはなりません。ディスクパーティションが DOS の場合は、**fdisk**、**cfdisk** などのコマンドを使用して、パーティション ID を 0x8e に設定する必要があります。ディスクデバイス全体に物

理ボリュームがある場合はパーティションテーブルを消去する必要がありますが、このとき、そのディスクにあるデータはすべて効果的に破棄されます。以下のコマンドを使用すれば、最初のセクターをゼロで初期化し、既存のパーティションテーブルを削除できます。

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

#### 4.2.1.2. 物理ボリュームの初期化

**pvcreate** コマンドを使用して、物理ボリュームとして使用するブロックデバイスを初期化します。初期化は、ファイルシステムのフォーマットと同様です。

以下のコマンドは、**/dev/sdd**、**/dev/sde**、および **/dev/sdf** を LVM 物理ボリュームとして初期化します。作成したボリュームは、後で LVM 論理ボリュームに使用します。

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

ディスク全体でなく、パーティションを初期化するには、そのパーティションで **pvcreate** コマンドを実行します。以下の例では、パーティション **/dev/hdb1** を LVM 物理ボリュームとして初期化します。作成した物理ボリュームは、後で LVM 論理ボリュームに使用します。

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

#### 4.2.1.3. ブロックデバイスのスキャン

以下の例のように、**lvmdiskscan** コマンドを使用して、物理ボリュームに使用できるブロックデバイスをスキャンできます。

```
# 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. 物理ボリュームの表示

LVM 物理ボリュームのプロパティを表示するのに使用できるコマンドは 3 つあります。**pvs**、**pvdisplay**、および **pvscan** です。

**pvs** コマンドは、物理ボリュームの情報を設定可能な形式で出力します。1 つの物理ボリュームが 1 行で表示されます。**pvs** コマンドでは形式をかなり自由にカスタマイズできるため、スクリプト作成時に役に立ちます。**pvs** コマンドで出力をカスタマイズする詳細な方法は、「[LVM 用のカスタム報告](#)」を参照してください。

**pvdisplay** コマンドは、各物理ボリュームの詳細をそれぞれ複数行出力します。物理プロパティ (サイズ、エクステント、ボリュームグループなど) が、決められた形式で表示されます。

以下の例は、1 つの物理ボリュームについて、**pvdisplay** コマンドで出力した情報です。

```

# 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** コマンドは、システムにある、物理ボリュームでサポートされている LVM ブロックデバイスをすべてスキャンします。

以下のコマンドでは、検出された物理デバイスがすべて表示されます。

```

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

```

このコマンドで、特定の物理ボリュームがスキャンされないように、**lvm.conf** ファイルでフィルターを定義することができます。スキャンするデバイスを制御するフィルターの使用法は、「[フィルターを使用した LVM デバイススキャンの制御](#)」を参照してください。

#### 4.2.3. 物理ボリューム上での割り当て防止

**pvchange** コマンドを使用すると、単一または複数の物理ボリュームの空き領域で物理エクステントが割り当てられないようにすることができます。これは、ディスクエラーが発生した場合や、物理ボリュームを取り除く場合に必要となる可能性があります。

以下のコマンドは、**/dev/sdk1** での物理エクステントの割り当てを無効にします。

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

**pvchange** コマンドで **-xy** 引数を使用すると、無効にされていた割り当てを許可できます。

#### 4.2.4. 物理ボリュームのサイズ変更

なんらかの理由で基礎となるブロックデバイスのサイズを変更する必要がある場合は、**pvresize** コマンドを使用して LVM のサイズを更新します。このコマンドは、LVM が物理ボリュームを使用しているときに実行できます。

#### 4.2.5. 物理ボリュームの削除

デバイスを LVM で使用する必要がなくなった場合、**pvremove** コマンドを使用して LVM ラベルを削除できます。**pvremove** コマンドを実行すると、空の物理ボリュームにある LVM メタデータをゼロにします。

削除したい物理ボリュームがボリュームグループの一部になっている場合は、[「ボリュームグループからの物理ボリュームの削除」](#) で説明されているように、**vgreduce** コマンドで、ボリュームグループから物理ボリュームを取り除く必要があります。

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

### 4.3. ボリュームグループの管理

このセクションでは、ボリュームグループ管理の様々な場面で使用するコマンドについて説明します。

#### 4.3.1. ボリュームグループの作成

1 つまたは複数の物理ボリュームからボリュームグループを作成するには、**vgcreate** コマンドを使用します。**vgcreate** コマンドは、名前を指定してボリュームグループを作成し、そこに物理ボリュームを 1 つ以上追加します。

以下のコマンドは、物理ボリューム **/dev/sdd1** と **/dev/sde1** が含まれる **vg1** という名前のボリュームグループを作成します。

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

ボリュームグループの作成に物理ボリュームが使用されるとき、ディスク領域はデフォルトで 4MB のエクステントに分割されます。このエクステントは、論理ボリュームのサイズを拡張/縮小する最小単位です。エクステントの数が多くても、論理ボリュームの I/O パフォーマンスに影響を与えることはありません。

エクステントサイズのデフォルト設定が適切でない場合は、**vgcreate** コマンドに **-s** オプションを使用して、エクステントのサイズを指定することができます。**vgcreate** コマンドに **-p** と **-1** の引数を使用すると、ボリュームグループに追加可能な物理ボリュームまたは論理ボリュームの数に制限をかけ



ることができます。

デフォルトでは、ボリュームグループは、同じ物理ボリューム上に並行ストライプを配置しないなど、常識的な規則に従って物理エクステントを割り当てます。これが **normal** の割り当てポリシーです。**vgcreate** コマンドで **--alloc** 引数を使用して、**contiguous**、**anywhere**、または **cling** の割り当てポリシーを指定できます。一般的に、**normal** 以外の割り当てポリシーが必要となるのは、通常とは異なる、標準外のエクステント割り当てを必要とする特別なケースのみです。LVM で物理エクステントを割り当てる方法の詳細は、「[LVM の割り当て](#)」を参照してください。

LVM ボリュームグループと配下にある論理ボリュームは、**/dev** ディレクトリー内に、以下のような配置で、デバイス特有のファイルディレクトリーツリーに格納されます。

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

たとえば、**myvg1** と **myvg2** という名前のボリュームグループを作成し、それぞれに **lv01**、**lv02**、**lv03** という名前の論理ボリュームを作成した場合は、デバイス特有のファイルが 6 つ作成されます。

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

デバイス特有のファイルは、対応する論理ボリュームがアクティブになっていないと表示されません。

LVM におけるデバイスの最大サイズは、64 ビット CPU 上で 8 エクサバイトです。

### 4.3.2. LVM の割り当て

LVM の操作で物理エクステントを単一または複数の論理ボリュームに割り当てる必要がある場合、割り当ては以下のように行われます。

- ボリュームグループ上で割り当てられていない物理エクステントが、割り当ての候補になります。コマンドラインの末尾に物理エクステントの範囲を指定すると、指定した物理ボリュームの中で、その範囲内で割り当てられていない物理エクステントだけが、割り当て用エクステントとして考慮されます。
- 割り当てポリシーは順番に試行されます。最も厳格なポリシー (**contiguous**) から始まり、最後は **--alloc** オプションで指定した割り当てポリシーか、特定の論理ボリュームやボリュームグループにデフォルトとして設定されている割り当てポリシーが試行されます。割り当てポリシーでは、埋める必要がある空の論理ボリューム領域の最小番号の論理エクステントから始まり、割り当てポリシーによる制限に沿って、できるだけ多くの領域の割り当てを行います。領域が足りなくなると、LVM は次のポリシーに移動します。

割り当てポリシーの制限は以下のとおりです。

- **contiguous** の割り当てポリシーでは、論理ボリュームの最初の論理エクステントを除いたすべての論理エクステントは、その直前の論理エクステントに物理的に隣接している必要があります。

論理ボリュームがストライプ化またはミラー化されている場合は、**contiguous** の割り当て制限が、領域を必要とする各ストライプまたはミラーイメージ (レグ) に個別に適用されます。

- **cling** の割り当てポリシーでは、既存の論理ボリュームに追加する論理エクステントに使用される物理ボリュームが、その論理ボリューム内の別の (1 つ以上の) 論理エクステントですでに使用されている必要があります。**allocation/cling\_tag\_list** の設定パラメーターが定義されており、一覧表示されているいずれかのタグが 2 つの物理ボリュームにある場合、この 2 つの物理ボリュームは一致すると見なされます。これにより、割り当てのために、同様のプロパティー (物理的な場所など) を持つ物理ボリュームのグループにタグを付け、その物理ボリュームを同等なものとして処理することができます。**cling** ポリシーを、LVM ボリュームの拡張時に使用する追加の物理ボリュームを指定する LVM タグと併用する方法の詳細は、「**cling** 割り当てポリシーを使用した論理ボリュームの拡張」を参照してください。

論理ボリュームがストライプ化またはミラー化されると、**cling** の割り当て制限が、領域を必要とする各ストライプまたはミラーイメージ (レッグ) に個別に適用されます。

- **normal** の割り当てポリシーは、並列の論理ボリューム (異なるストライプまたはミラーイメージ/レッグ) 内の同じオフセットで、その並列の論理ボリュームにすでに割り当てられている論理エクステントと同じ物理ボリュームを共有する物理エクステントは選択しません。

ミラーデータを保持するために、論理ボリュームと同時にミラーログを割り当てる場合、**normal** の割り当てポリシーでは、最初にログとデータに対して、それぞれ別の物理ボリュームを選択しようとします。異なる物理ボリュームを選択できず、かつ **allocation/mirror\_logs\_require\_separate\_pvs** 設定パラメーターが 0 に設定されている場合は、データの一部とログが物理ボリュームを共有できるようになります。

また、シンプルメタデータを割り当てる場合も、**normal** の割り当てポリシーはミラーログを割り当てる場合と同じようになりますが、設定パラメーターは **allocation/thin\_pool\_metadata\_require\_separate\_pvs** の値が適用されます。

- 割り当て要求を満たすのに十分な空きエクステントがあっても、**normal** の割り当てポリシーによって使用されない場合は、たとえ同じ物理ボリュームに 2 つのストライプを配置することによってパフォーマンスが低下しても、**anywhere** 割り当てポリシーがその空きエクステントを使用します。

割り当てポリシーは、**vgchange** コマンドで変更できます。



## 注記

定義された割り当てポリシーに沿って、このセクションで説明されている以上のレイアウトの操作が必要な場合は、今後のバージョンでコードが変更する可能性があることに注意してください。たとえば、割り当て可能な空き物理エクステントの数が同じ 2 つの空の物理ボリュームをコマンドラインで指定する場合、現行バージョンの LVM では、それが表示されている順番に使用が検討されます。ただし、今後のリリースで、そのプロパティーが変更されない保証はありません。特定の論理ボリューム用に特定のレイアウトを取得することが重要な場合は、各手順に適用される割り当てポリシーに基づいて LVM がレイアウトを決定することがないように、**lvcreate** と **lvconvert** を順に使用してレイアウトを構築してください。

特定のケースで、割り当てプロセスがどのように行われているかを確認するには、コマンドに **-vvvv** オプションを追加するなどして、デバッグロギングの出力を表示します。

### 4.3.3. クラスター内でのボリュームグループ作成

**vgcreate** コマンドで、クラスター環境内にボリュームグループを作成します。単一ノードにボリュームグループを作成する場合と同様です。

デフォルトでは、CLVM で共有ストレージに作成されたボリュームグループは、その共有ストレージにアクセス可能なすべてのコンピューターに表示されます。ただし、**vgcreate** コマンドの **-c n** オプションを使用すれば、クラスター内の 1 つのノードからしか見れないローカルのボリュームグループを作成することもできます。

クラスター環境で以下のコマンドを実行すると、コマンドを実行しているノードに対してローカルとなるボリュームグループが作成されます。このコマンドは、物理ボリュームである **/dev/sdd1** と **/dev/sde1** を含むローカルボリューム **vg1** を作成します。

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

**vgchange** コマンドで **-c** オプションを使用すると、既存のボリュームグループをローカルにするか、もしくはクラスター化するかを変更できます。詳細は「[ボリュームグループのパラメーター変更](#)」で説明しています。

既存のボリュームグループがクラスター化したボリュームグループであるかどうかは、**vgs** コマンドでチェックできます。ボリュームがクラスター化されている場合は、**c** 属性を表示します。以下のコマンドは、**VolGroup00** と **testvg1** のボリュームグループの属性を表示します。この例では、**VolGroup00** はクラスター化されていませんが、**testvg1** は、**Attr** 列にある **c** 属性が示すようにクラスター化されています。

```
# 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
```

**vgs** コマンドの詳細は、「[ボリュームグループの表示](#)」、「[LVM 用のカスタム報告](#)」、および **vgs** の man ページを参照してください。

#### 4.3.4. ボリュームグループへの物理ボリュームの追加

物理ボリュームを既存ボリュームグループに追加するには、**vgextend** コマンドを使用します。**vgextend** コマンドは、空き物理ボリュームを 1 つまたは複数追加して、ボリュームグループの容量を増やします。

以下のコマンドは、物理ボリューム **/dev/sdf1** をボリュームグループ **vg1** に追加します。

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

#### 4.3.5. ボリュームグループの表示

LVM ボリュームグループのプロパティを表示するのに使用できるコマンドは 2 つあります。**vgs** と **vgdisplay** です。

**vgscan** コマンドは、ボリュームグループのすべてのディスクをスキャンして LVM キャッシュファイルを再構築するほかに、ボリュームグループを表示することもできます。**vgscan** コマンドに関する情報は「[キャッシュファイル構築のためのボリュームグループのディスクスキャン](#)」を参照してください。

**vgs** コマンドは、ボリュームグループの情報を設定可能な形式で提供し、1 ボリュームグループにつき 1 行ずつ表示します。**vgs** コマンドでは形式をかなり自由にカスタマイズできるため、スクリプト作成に役立ちます。出力をカスタマイズする **vgs** コマンドを使用する方法は、「[LVM 用のカスタム報告](#)」を参照してください。

**vgdisplay** コマンドは、決められた形式でボリュームグループのプロパティ (サイズ、エクステン  
ト、物理ボリュームの数など) を表示します。以下の例は、ボリュームグループ **new\_vg** に対する  
**vgdisplay** コマンドの出力を示しています。ボリュームグループを指定しないと、既存のボリューム  
グループがすべて表示されます。

```
# 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
```

#### 4.3.6. キャッシュファイル構築のためのボリュームグループのディスクスキャン

**vgscan** コマンドは、システムでサポートされるすべてのディスクデバイスをスキャンし、LVM 物理ボ  
リュームとボリュームグループを検索します。これにより、**/etc/lvm/cache/.cache** ファイルに  
LVM キャッシュファイルが作成され、ここに現在の LVM デバイスの一覧を保ちます。

LVM は、システムの起動時、**vgcreate** コマンドの実行時、LVM による不整合の検出時などの他の  
LVM 操作時に、**vgscan** コマンドを自動的に実行します。



#### 注記

ハードウェア設定を変更して、ノードに対してデバイスの追加/削除を行う場合、システ  
ムの起動時に存在していなかったデバイスがシステムに認識されるように、**vgscan** コ  
マンドを手動で実行しなければならない場合があります。これは、たとえば、SAN 上の  
システムに新しいディスクを追加したり、物理ボリュームとしてラベルが付けられた新  
しいディスクをホットプラグする場合に必要な可能性があります。

**/etc/lvm/lvm.conf** ファイルでフィルターを定義して、特定デバイスを除外するようにスキャンを  
設定できます。スキャンするデバイスを制御するフィルターの使用方法は、[「フィルターを使用した  
LVM デバイススキャンの制御」](#) を参照してください。

次の例は、**vgscan** コマンドの出力になります。

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

```
Found volume group "officevg" using metadata type lvm2
```

### 4.3.7. ボリュームグループからの物理ボリュームの削除

ボリュームグループから未使用の物理ボリュームを削除するには、**vgreduce** コマンドを使用します。**vgreduce** コマンドは、空の物理ボリュームを 1 つまたは複数削除して、ボリュームグループの容量を縮小します。これによって、物理ボリュームが解放され、異なるボリュームグループで使用したり、システムから削除できるようになります。

ボリュームグループから物理ボリュームを削除する前に、**pvdisplay** コマンドを使用して、その物理ボリュームが論理ボリュームで使用されていないことを確認することができます。

```
# pvdisplay /dev/hda1

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

物理ボリュームが使用中の場合は、**pvmove** コマンドを使用して、データを別の物理ボリュームに移行する必要があります。その後、**vgreduce** コマンドを使用してその物理ボリュームを削除します。

以下のコマンドは、物理ボリューム **/dev/hda1** を、ボリュームグループ **my\_volume\_group** から取り除きます。

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

論理ボリュームに、障害のある物理ボリュームが含まれる場合は、その論理ボリュームを使用することはできません。見つからない物理ボリュームをボリュームグループから削除します。その物理ボリュームに論理ボリュームが割り当てられていない場合は、**vgreduce** コマンドの **--removemissing** パラメーターを使用することができます。

障害が発生した物理ボリュームに、セグメントタイプが **mirror** の論理ボリュームのミラーイメージが含まれる場合、**vgreduce --removemissing --mirroronly --force** コマンドを使ってミラーからイメージを削除することができます。これにより、物理ボリュームのミラーイメージである論理ボリュームのみが削除されます。

LVM ミラーの障害から回復する方法は、[「LVM ミラー障害からの回復」](#) を参照してください。ボリュームグループから紛失した物理ボリュームを削除する方法は、[「紛失した物理ボリュームのボリュームグループからの削除」](#) を参照してください。

### 4.3.8. ボリュームグループのアクティブ化と非アクティブ化

ボリュームグループを作成すると、デフォルトでアクティブになります。これは、そのグループ内の論理ボリュームがアクセス可能で、かつ変更される可能性があることを意味します。



ボリュームグループを非アクティブ化し、カーネルに認識されないようにする必要のある様々な状況があります。ボリュームグループを非アクティブ化またはアクティブ化するには、**vgchange** コマンドで **-a (--available)** 引数を使用します。

以下の例では、ボリュームグループ **my\_volume\_group** を非アクティブ化します。

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

クラスターロックが有効な場合には、「e」を追加すると、1つのノードでボリュームグループが排他的にアクティブ化または非アクティブ化されます。「l」を追加すると、ローカルノードでのみボリュームグループがアクティブ化または非アクティブ化されます。単一ホストのスナップショットを使用する論理ボリュームは、一度に1つのノードでしか利用できないため、常に排他的にアクティブ化されます。

「[論理ボリュームグループのパラメーター変更](#)」で説明されているように、**lvchange** コマンドを使用して、個別の論理ボリュームを非アクティブ化できます。クラスター内の個別ノード上で論理ボリュームをアクティブ化する方法については、「[クラスター内の個別ノードでの論理ボリュームのアクティブ化](#)」を参照してください。

### 4.3.9. ボリュームグループのパラメーター変更

「[ボリュームグループのアクティブ化と非アクティブ化](#)」で説明されているように、**vgchange** コマンドは、ボリュームグループを非アクティブ化およびアクティブ化するのに使用されます。また、このコマンドを使用して、既存のボリュームグループに関するボリュームグループパラメーターを変更することもできます。

以下のコマンドは、ボリュームグループ **vg00** の論理ボリュームの最大数を 128 に変更します。

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

**vgchange** コマンドで変更できるボリュームグループパラメーターの説明については、**vgchange(8)** の man ページを参照してください。

### 4.3.10. ボリュームグループの削除

論理ボリュームがないボリュームグループを削除するには、**vgremove** コマンドを使用します。

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

### 4.3.11. ボリュームグループの分割

ボリュームグループの物理ボリュームを分割して、新しいボリュームグループを作成するには、**vgsplit** コマンドを使用します。

論理ボリュームはボリュームグループ間で分割することはできません。それぞれの既存の論理ボリュームは完全に物理ボリューム上に存在し、既存または新規のボリュームグループを形成している必要があります。ただし必要な場合は、**pvmove** コマンドを使用して、その分割を強制することができます。

以下の例は、元のボリュームグループ **bigvg** から新規のボリュームグループ **smallvg** を分割しています。

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

#### 4.3.12. ボリュームグループの統合

2つのボリュームグループを統合して1つのボリュームグループにするには、**vgmerge** コマンドを使用します。ボリュームの物理エクステンツサイズが同じで、かつ両ボリュームグループの物理および論理ボリュームのサマリーが「マージ先」ボリュームグループの制限内に収まる場合は、非アクティブな「マージ元」のボリュームを、アクティブまたは非アクティブの「マージ先」ボリュームにマージすることができます。

以下のコマンドは、非アクティブなボリュームグループ **my\_vg** をアクティブまたは非アクティブなボリュームグループ **databases** にマージして、詳細なランタイム情報を提供します。

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

#### 4.3.13. ボリュームグループメタデータのバックアップ

メタデータのバックアップとアーカイブは、**lvm.conf** ファイルで無効になっていない限り、ボリュームグループまたは論理ボリューム設定の変更時に自動的に作成されます。デフォルトでは、メタデータのバックアップは **/etc/lvm/backup** ファイルに保存され、メタデータのアーカイブは **/etc/lvm/archives** ファイルに保存されます。**vgcfgbackup** コマンドを使用すると、手動でメタデータを **/etc/lvm/backup** ファイルにバックアップできます。

**vgcfgrestore** コマンドは、ボリュームグループのメタデータを、アーカイブからボリュームグループのすべての物理ボリュームに復元します。

物理ボリュームのメタデータを復元する **vgcfgrestore** コマンドの使用例は、[「物理ボリュームメタデータの復元」](#) を参照してください。

#### 4.3.14. ボリュームグループの名前変更

既存ボリュームグループの名前を変更するには、**vgrename** コマンドを使用します。

以下のいずれかのコマンドで、既存ボリュームグループ **vg02** の名前を **my\_volume\_group** に変更できます。

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

```
# vgrename vg02 my_volume_group
```

#### 4.3.15. ボリュームグループの別のシステムへの移動

LVM ボリュームグループ全体を、別のシステムに移動することができます。これを実行するには、**vgexport** と **vgimport** のコマンドの使用が推奨されます。



#### 注記

**vgimport** コマンドの **--force** 引数を使用できます。これで物理ボリュームがないボリュームグループをインポートし、**vgreduce --removemissing** コマンドを実行します。



**vgexport** コマンドは、非アクティブのボリュームグループにシステムがアクセスできないようにするため、物理ボリュームの割り当て解除が可能になります。**vgimport** コマンドは、**vgexport** コマンドで非アクティブにしていたボリュームグループに、マシンが再度アクセスできるようにします。

ボリュームグループを 2 つのシステム間で移行するには、以下の手順に従います。

1. ボリュームグループ内のアクティブなボリュームのファイルにアクセスしているユーザーがいないことを確認してから、論理ボリュームをアンマウントします。
2. **vgchange** コマンドで **-a n** 引数を使用して、そのボリュームグループを非アクティブとしてマークします。これによりボリュームグループでこれ以上の動作が発生しないようにします。
3. **vgexport** コマンドを使用してボリュームグループをエクスポートします。これにより、削除中のシステムからボリュームグループへアクセスできなくなります。

ボリュームグループをエクスポートして **pvs** コマンドを実行すると、以下の例のように、エクスポート先のボリュームグループに物理ボリュームが表示されます。

```
# pvs
PV /dev/sda1      is in exported VG myvg [17.15 GB / 7.15 GB free]
PV /dev/sdc1      is in exported VG myvg [17.15 GB / 15.15 GB free]
PV /dev/sdd1      is in exported VG myvg [17.15 GB / 15.15 GB free]
...
```

次にシステムがシャットダウンする時に、ボリュームグループを構成していたディスクを外して、新しいシステムに接続できます。

4. ディスクが新しいシステムに接続したら、**vgimport** コマンドを使用してボリュームグループをインポートし、新しいシステムからアクセスできるようにします。
5. **vgchange** コマンドで **-a y** 引数を使用して、ボリュームグループをアクティブにします。
6. ファイルシステムをマウントして使用可能にします。

#### 4.3.16. ボリュームグループディレクトリーの再作成

ボリュームグループディレクトリーと、論理ボリューム特有ファイルを再作成するには、**vgmknodes** コマンドを使用します。このコマンドは、**/dev** ディレクトリー内の LVM2 特有ファイルをチェックします。このファイルはアクティブな論理ボリュームに必要です。このコマンドは不足しているファイルを作成し、未使用のファイルを削除します。

**vgscan** コマンドに **mknodes** 引数を指定して、**vgmknodes** コマンドを **vgscan** コマンドに統合することができます。

### 4.4. 論理ボリュームの管理

このセクションでは、論理ボリューム管理の様々な要素を実行するコマンドを説明します。

#### 4.4.1. リニア論理ボリュームの作成

論理ボリュームを作成するには、**lvcreate** コマンドを使用します。論理ボリュームに名前を指定しないと、デフォルトの名前 **lvol#** が使用されます (**#**の部分には論理ボリュームの内部番号が入ります)。

論理ボリュームを作成すると、論理ボリュームがボリュームグループから作成され、ボリュームグループを構成する物理ボリュームの空きエクステントが使用されます。通常、論理ボリュームは、その下層の物理ボリュームで次に使用可能な空き領域を使用します。論理ボリュームを変更すると、物理ボリューム領域の確保と再割り当てが可能になります。

以下のコマンドは、ボリュームグループ **vg1** に、10 ギガバイトの論理ボリュームを作成します。

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

論理ボリュームサイズのデフォルト単位はメガバイトです。次のコマンドは、ボリュームグループ **testvg** 内に、名前が **testlv** でサイズが 1500 メガバイトのリニア論理ボリュームを作成し、ブロックデバイス **/dev/testvg/testlv** を作成します。

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

次のコマンドは、ボリュームグループ **vg0** の空きエクステントから、名前が **gfslv** でサイズが 50 ギガバイトの論理ボリュームを作成します。

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

**lvcreate** コマンドで **-l** 引数を使用すると、論理ボリュームのサイズをエクステントで指定できます。この引数を使用すると、関連するボリュームグループ、論理ボリューム、または物理ボリュームセットのサイズの割合も指定できます。接尾辞 **%VG** はボリュームグループの合計サイズ、接尾辞 **%FREE** はボリュームグループの残りの空き容量、そして接尾辞 **%PVS** は物理ボリュームの空き容量を示します。スナップショットの場合は、サイズに接尾辞 **%ORIGIN** を使用して、元の論理ボリュームの合計サイズをパーセンテージで指定することができます (100%**ORIGIN** にすると、元の論理ボリューム全体が使用されます)。サイズをパーセンテージで指定した場合は、新規の論理ボリュームにおける論理エクステントに上限が設定されます。したがって、作成する LV の論理エクステントの正確な数は、コマンドが完了するまで決定されません。

以下のコマンドは、ボリュームグループ **testvg** の 60% の容量を使用する、論理ボリューム **mylv** を作成します。

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

以下のコマンドは、ボリュームグループ **testvg** の空き領域をすべて使用する、論理ボリューム **yourlv** を作成します。

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

**lvcreate** コマンドで **-l** 引数を使用すれば、ボリュームグループ全域を使用する論理ボリュームを作成できます。もしくは、**vgdisplay** コマンドを使用して「合計 PE」サイズを確認し、その結果を **lvcreate** コマンドへの入力として使用しても、ボリュームグループ全域を使用する論理ボリュームを作成することができます。

以下のコマンドは、ボリュームグループ **testvg** の全域を使用する論理ボリューム **mylv** を作成します。

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

論理ボリュームの作成に使用した下層の物理ボリュームは、物理ボリュームを削除する必要がある場合

に重要になる可能性があります。そのため、論理ボリュームを作成する際にはこの可能性を考慮する必要があります。ボリュームグループから物理ボリュームを削除する方法は、「[ボリュームグループからの物理ボリュームの削除](#)」を参照してください。

ボリュームグループから論理ボリュームを作成する際に、特定の物理ボリュームを割り当てる場合は、**lvcreate** コマンドラインの末尾に物理ボリュームを指定します。以下のコマンドは、ボリュームグループ **testvg** から、物理ボリューム **/dev/sdg1** を割り当てた論理ボリューム **testlv** を作成します。

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

論理ボリュームに使用する物理ボリュームのエクステントを指定することができます。以下の例では、ボリュームグループ **testvg** の、エクステントが 0 から 24 の物理ボリューム **/dev/sda1** と、エクステントが 50 から 124 の物理ボリューム **/dev/sdb1** から、リニア論理ボリュームを作成します。

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

以下の例では、エクステントが 0 から 25 の物理ボリューム **/dev/sda1** からリニア論理ボリュームを作成し、そのエクステントを 100 にします。

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

論理ボリュームにエクステントを割り当てる方法に関するデフォルトポリシーは **inherit** で、ボリュームグループにも同じポリシーが適用されます。このポリシーは、**lvchange** コマンドで変更できます。割り当てポリシーの詳細は、「[ボリュームグループの作成](#)」を参照してください。

#### 4.4.2. ストライプ化ボリュームの作成

連続的な読み取りと書き込みが大量に行われる場合は、ストライプ化論理ボリュームを作成すると、データ I/O が効率が上がります。ストライプ化ボリュームに関する一般情報は、「[ストライプ化論理ボリューム](#)」を参照してください。

ストライプ化論理ボリュームを作成する場合は、**lvcreate** コマンドで **-i** 引数を使用してストライプの数を指定します。これにより、論理ボリュームがストライプ化される物理ボリュームの数が決定します。ストライプ数は、ボリュームグループ内の物理ボリュームの数よりも多くすることはできません (**-alloc anywhere** 引数を使用される場合は除く)。

ストライプ化論理ボリュームを構成する下層の物理デバイスのサイズが異なる場合、ストライプ化ボリュームの最大サイズは、一番小さいデバイスにより決まります。2 レッグのストライプの最大サイズは、小さい方のデバイスの 2 倍になります。3 レッグのストライプの最大サイズは、一番小さいデバイスの 3 倍になります。

以下のコマンドは、64 キロバイトのストライプを持つ 2 つの物理ボリュームにまたがってストライプ化論理ボリュームを作成します。論理ボリュームは、ボリュームグループ **vg0** から作成され、サイズが 50 ギガバイトで、名前が **gfslv** になります。

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

リニアボリュームと同じく、ストライプに使用する物理ボリュームのエクステントを指定できます。以下のコマンドは、2 つの物理ボリュームにまたがってストライプ化する、名前が **stripelv** でサイズが 100 エクステントのストライプ化ボリュームを、ボリュームグループ **testvg** に作成します。ストライプは、**/dev/sda1** のセクター 0-49 と、**/dev/sdb1** のセクター 50-99 を使用します。

```
# lvcreate -l 100 -i 2 -n stripe1v testvg /dev/sda1:0-49 /dev/sdb1:50-99
Using default stripesize 64.00 KB
Logical volume "stripe1v" created
```

#### 4.4.3. RAID 論理ボリューム

LVM は RAID0/1/4/5/6/10 をサポートします。



##### 注記

RAID 論理ボリュームはクラスターに対応していません。RAID 論理ボリュームは 1 台のマシンに作成でき、かつ排他的にアクティブにすることができますが、複数のマシンで同時にアクティブにすることはできません。排他的ではないミラー化されたボリュームが必要な場合は、「[ミラー化ボリュームの作成](#)」に説明されているように、セグメントタイプの **mirror** を指定してボリュームを作成する必要があります。

RAID 論理ボリュームを作成するには、**lvcreate** コマンドの **--type** 引数に raid タイプを指定します。表4.1「[RAID のセグメントタイプ](#)」で、使用できる RAID セグメントタイプについて説明しています。

表4.1 RAID のセグメントタイプ

セグメントタイプ	説明
<b>raid1</b>	RAID1 ミラーリング。これは <b>lvcreate</b> コマンドの <b>--type</b> 引数のデフォルト値で、 <b>-m</b> を指定してもストライピングを指定しない場合に使用されます。
<b>raid4</b>	RAID4 専用パリティディスク
<b>raid5</b>	<b>raid5_ls</b> と同様
<b>raid5_la</b>	<div>RAID5 left asymmetric</div> <div>ローテートパリティ 0 + データ継続</div>
<b>raid5_ra</b>	<div>RAID5 right asymmetric</div> <div>ローテートパリティ N + データ継続</div>
<b>raid5_ls</b>	<div>RAID5 left symmetric</div> <div>ローテートパリティ 0 + データ再起動</div>

セグメントタイプ	説明
<b>raid5_rs</b>	<div>RAID5 right symmetric</div> <div>ローテートパリティー N + データ再起動</div>
<b>raid6</b>	<b>raid6_zr</b> と同様
<b>raid6_zr</b>	<div>RAID6 zero restart</div> <div>ローテートパリティーゼロ (左から右) + データ再起動</div>
<b>raid6_nr</b>	<div>RAID6 N restart</div> <div>ローテートパリティー N (左から右) + データ再起動</div>
<b>raid6_nc</b>	<div>RAID6 N continue</div> <div>ローテートパリティー N (左から右) + データを継続</div>
<b>raid10</b>	<div>ストライプ化ミラー。これは <b>lvcreate</b> コマンドの <b>--type</b> 引数のデフォルト値で、<b>-m</b> を指定し、ストライプを 2 つ以上指定する場合に使用されます。</div> <div>ミラーセットのストライピング</div>
<b>raid0/raid0_meta</b> (Red Hat Enterprise Linux 7.3 以降)	ストライピング。RAID0 では、ストライプ単位で、複数のデータサブボリュームに論理ボリュームデータが分散されます。これは、パフォーマンスを向上させるために使用します。論理ボリュームのデータは、いずれかのデータサブボリュームで障害が発生すると失われます。RAID0 ボリュームの作成については、「 <a href="#">RAID0 ボリュームの作成 (Red Hat Enterprise Linux 7.3 以降)</a> 」を参照してください。

通常、5 つのプライマリータイプ (**raid1**、**raid4**、**raid5**、**raid6**、**raid10**) の中から 1 つを指定するだけで十分です。RAID 5/6 が使用する各種アルゴリズムの詳細は、『Common RAID Disk Data Format Specification』 ([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)) の第 4 章を参照してください。

RAID 論理ボリュームを作成するとき、LVM は、データまたはアレイ内のパリティサブボリュームごとに、サイズが 1 エクステントのメタデータサブボリュームを作成します。たとえば、2 方向の RAID1 アレイを作成すると、メタデータサブボリュームが 2 つ (**lv\_rmeta\_0** および **lv\_rmeta\_1**)、データサブボリュームが 2 つ (**lv\_rimage\_0** および **lv\_rimage\_1**) 作成されます。同様に、3 方向のストラ

イプ (および暗黙的なパリティデバイスが 1 つ) の RAID4 を作成すると、メタデータサブボリュームが 4 つ (`lv_rmeta_0`、`lv_rmeta_1`、`lv_rmeta_2`、および `lv_rmeta_3`)、データサブボリュームが 4 つ (`lv_rimage_0`、`lv_rimage_1`、`lv_rimage_2`、および `lv_rimage_3`) が作成されます。

以下のコマンドは、ボリュームグループ `my_vg` 内に、1 ギガバイトの 2 方向 RAID1 アレイ `my_lv` を作成します。

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

`-m` 引数でコピー数を設定し、RAID1 アレイを作成することができます。同様に、`-i` オプションで、RAID 4/5/6 論理ボリュームのストライプ数を指定し、`-I` 引数で、ストライプのサイズを指定できます。

以下のコマンドは、ボリュームグループ `my_vg` に、サイズが 1 ギガバイトで、名前が `my_lv` の RAID5 アレイ (ストライプ 3 つ + 暗黙的なパリティドライブ 1 つ) を作成します。ストライプ数の指定は、LVM ストライプ化ボリュームの場合と同じように行います。パリティドライブは、正確な数だけ自動的に追加されます。

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

以下のコマンドは、ボリュームグループ `my_vg` に、サイズが 1 ギガバイトで、名前が `my_lv` の RAID6 アレイ (ストライプ 3 つ + 暗黙的なパリティドライブ 2 つ) を作成します。

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

LVM で RAID 論理ボリュームを作成したら、他の LVM 論理ボリュームと同じように、ボリュームのアクティブ化、変更、削除、表示、使用を行うことができます。

RAID10 論理ボリュームを作成する際に、**sync** 操作で論理ボリュームを初期化するのに必要なバックグラウンド I/O は、ボリュームグループメタデータへの更新などの他の I/O 操作を、LVM デバイスに押し出す可能性があります。これはとくに RAID 論理ボリュームを多数作成している場合に生じる可能性があります。これにより、他の LVM 操作の速度が遅くなる場合があります。

RAID 論理ボリュームが初期化される速度は、復旧スロットルを実装することで制御することができます。**sync** 操作が実施される速度は、`lvcreate` コマンドの `--minrecoveryrate` および `--maxrecoveryrate` オプションで、I/O 速度の最小および最大を設定することで制御できます。

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

RAID 論理ボリュームの最大復旧速度を設定し、通常の I/O 操作が押し出されないようにします。速度は、アレイ内の各デバイスに対して、1 秒あたりの量を指定します。サフィックスを指定しない場合は、kiB/sec/device と見なされます。復旧速度を 0 に設定すると無制限になります。

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

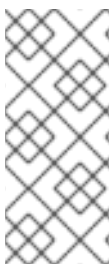
RAID 論理ボリュームの最小復旧速度を設定し、負荷の高い通常の I/O がある場合でも、**sync** 操作の I/O が最小スループットを達成できるようにします。速度アレイ内の各デバイスに対して、1 秒あたりの量を指定します。サフィックスを指定しない場合は、kiB/sec/device と見なされます。

以下のコマンドは、最大速度が 128 kiB/sec/device で、サイズが 10 ギガバイトのストライプが 3 つある、2 方向の RAID10 アレイを作成します。このアレイは名前は `my_lv` で、ボリュームグループは `my_vg` になります。

-

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

さらに、RAID スクラビング操作の最小および最大復旧速度を指定することもできます。RAID スクラビングの情報は、「[RAID 論理ボリュームのスクラビング](#)」を参照してください。



### 注記

LVM RAID Calculator を使用すると、RAID ストレージで論理ボリュームを作成するコマンドを生成できます。このアプリケーションは、現在のストレージまたは作成されるストレージについて入力した情報を使用してコマンドを生成します。LVM RAID Calculator アプリケーションは、<https://access.redhat.com/labs/lvmraidcalculator/> で使用できます。

以下のセクションでは、LVM RAID デバイスで実行できる管理タスクについて説明します。

- 「[RAID0 ボリュームの作成 \(Red Hat Enterprise Linux 7.3 以降\)](#)」.
- 「[リニアデバイスの RAID デバイスへの変換](#)」
- 「[LVM RAID1 論理ボリュームの LVM リニア論理ボリュームへの変換](#)」
- 「[ミラー化 LVM デバイスの RAID1 デバイスへの変換](#)」
- 「[RAID 論理ボリュームのサイズ変更](#)」
- 「[既存の RAID1 デバイス内のイメージ数の変更](#)」
- 「[RAID イメージを複数の論理ボリュームに分割](#)」
- 「[RAID イメージの分割とマージ](#)」
- 「[RAID 障害ポリシーの設定](#)」
- 「[RAID デバイスの置き換え](#)」
- 「[RAID 論理ボリュームのスクラビング](#)」
- 「[RAID テイクオーバー \(Red Hat Enterprise Linux 7.4 以降\)](#)」
- 「[RAID 論理ボリュームの再成形 \(Red Hat Enterprise Linux 7.4 以降\)](#)」
- 「[RAID1 論理ボリュームでの I/O 操作の制御](#)」
- 「[RAID 論理ボリュームのリージョンサイズの変更 \(Red Hat Enterprise Linux 7.4 以降\)](#)」

#### 4.4.3.1. RAID0 ボリュームの作成 (Red Hat Enterprise Linux 7.3 以降)

RAID0 ボリュームを作成するコマンドの書式は以下のとおりです。

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

表4.2 RAID0 コマンドの作成に関するパラメーター



パラメーター	説明
<b>--type</b> <b>raid0[_meta]</b>	<b>raid0</b> を指定すると、メタデータボリュームなしで RAID0 ボリュームが作成されます。 <b>raid0_meta</b> を指定すると、メタデータボリュームとともに RAID0 ボリュームが作成されます。RAID0 には耐障害性がないため、RAID1/10 の場合のようにミラーリングされたすべてのデータブロックを格納したり、RAID4/5/6 の場合のようにすべてのパリティブロックを格納したりする必要はありません。したがって、ミラーリングされたブロックまたはパリティブロックの再同期の進行状態を把握するメタデータボリュームは必要ありません。ただし、RAID0 から RAID4/5/6/10 に変換するには、メタデータボリュームが必要です。 <b>raid0_meta</b> を指定すると、割り当ての失敗を防ぐためにこれらのメタデータが事前に割り当てられます。
<b>--stripes</b> <i>Stripes</i>	論理ボリュームを分散するデバイスの数を指定します。
<b>--stripesize</b> <b>StripeSize</b>	各ストライプのサイズをキロバイト単位で指定します。これは、次のデバイスに移動する前にデバイスに書き込まれるデータの量です。
<b>VolumeGroup</b>	使用するボリュームグループを指定します。
<b>PhysicalVolumePath</b> ...	使用するデバイスを指定します。指定しない場合は、LVM によって、 <i>Stripes</i> オプションに指定されているデバイスの数が、各ストライプに 1 つずつ選択されます。

#### 4.4.3.2. リニアデバイスの RAID デバイスへの変換

既存のリニア論理ボリュームを RAID デバイスに変換するには、**lvconvert** コマンドの **--type** 引数を使用します。

以下のコマンドは、ボリュームグループ **my\_vg** のリニア論理ボリューム **my\_lv** を、2 方向の RAID1 アレイに変換します。

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

RAID 論理ボリュームは、メタデータとデータサブボリュームのペアで構成されているため、リニアデバイスを RAID1 アレイに変換すると、メタデータサブボリュームが作成され、リニアボリュームが存在する物理ボリューム (のいずれか) にある、複製元の論理ボリュームに関連付けられます。イメージは、メタデータ/データサブボリュームのペアに追加されます。たとえば、複製元のデバイスは以下のとおりです。

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

2 方向の RAID1 アレイに変換すると、デバイスには以下のデータとメタデータサブボリュームのペアが含まれます。

```
# 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)
```

複製元の論理ボリュームとペアのメタデータイメージを同じ物理ボリュームに配置できないと、**lvconvert** は失敗します。

#### 4.4.3.3. LVM RAID1 論理ボリュームの LVM リニア論理ボリュームへの変換

**lvconvert** コマンドを使用して、既存の RAID1 LVM 論理ボリュームを LVM リニア論理ボリュームに変換するには **-m0** 引数を指定します。これにより、すべての RAID データサブボリュームおよび RAID アレイを構成するすべての RAID メタデータサブボリュームが削除され、最高レベルの RAID1 イメージがリニア論理ボリュームとして残されます。

以下の例は、既存の LVM RAID1 論理ボリュームを表示しています。

```
# 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)
```

以下のコマンドは、LVM RAID1 論理ボリューム **my\_vg/my\_lv** を LVM リニアデバイスに変換します。

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

LVM RAID1 論理ボリューム を LVM リニアボリュームに変換する場合、削除する物理ボリュームを指定できます。以下の例は、**/dev/sda1** と **/dev/sdb1** の 2 つのイメージで構成される LVM RAID1 論理ボリュームのレイアウトを表示しています。この例で、**lvconvert** コマンドは **/dev/sda1** を削除して、**/dev/sdb1** をリニアデバイスを構成する物理ボリュームとして残すように指定します。

```
# 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)
```

#### 4.4.3.4. ミラー化 LVM デバイスの RAID1 デバイスへの変換

**lvconvert** コマンドを使用して、セグメントタイプの **mirror** を指定した既存のミラー化 LVM デバイスを、RAID1 LVM デバイスに変換するには、**--type raid1** 引数を指定します。これにより、ミ

ラーサブボリューム (\***\_mimage\_\***) の名前が、RAID サブボリューム (\***\_rimage\_\***) に変更になります。また、ミラーログは削除され、対応するデータサブボリュームと同じ物理ボリュームのデータサブボリューム用に、メタデータサブボリューム (\***\_rmeta\_\***) が作成されます。

以下の例は、ミラー化論理ボリューム **my\_vg/my\_lv** のレイアウトを示しています。

```
# 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)
```

以下のコマンドは、ミラー化論理ボリューム **my\_vg/my\_lv** を RAID1 論理ボリュームに変換します。

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

#### 4.4.3.5. RAID 論理ボリュームのサイズ変更

RAID 論理ボリュームのサイズ変更は、以下の方法でできます。

- いずれのタイプの RAID 論理ボリュームのサイズも、**lvresize** コマンドまたは **lvextend** コマンドで増やすことができます。これは、RAID イメージの数を変更するものではありません。ストライプ化 RAID 論理ボリュームの場合は、ストライプ化 RAID 論理ボリュームの作成時に、同様の丸め動作の制約が適用されます。RAID ボリュームの拡張の詳細は、「[RAID ボリュームの拡張](#)」を参照してください。
- いずれのタイプの RAID 論理ボリュームのサイズも、**lvresize** コマンドまたは **lvreduce** コマンドで減らすことができます。これは、RAID イメージの数を変更するものではありません。**lvextend** コマンドを使用して、ストライプ化 RAID 論理ボリュームの作成時に、同様のストライプの丸め動作の制約が適用されます。論理ボリュームのサイズを変更するコマンド例は、「[論理ボリュームの縮小](#)」を参照してください。
- Red Hat Enterprise Linux 7.4 以降、**lvconvert** コマンドに **--stripes N** パラメーターを付けて、ストライプ化 RAID 論理ボリューム (**raid4/5/6/10**) でストライプの数を変更できます。このように、ストライプを追加または削除することで、RAID 論理ボリュームのサイズを増やしたり削除します。**raid10** ボリュームには、ストライプを追加する機能しかありません。この機能は、同じ RAID レベルを維持しながら、RAID 論理ボリュームの属性を変更することができる、RAID の **再成形** 機能になります。RAID 再成形と、**lvconvert** コマンドを使用して RAID 論理ボリュームを再成形するコマンドの使用例は、**lvraid(7)** の man ページを参照してください。

#### 4.4.3.6. 既存の RAID1 デバイス内のイメージ数の変更

既存の RAID1 アレイ内のイメージ数は、LVM ミラーリングの初期実装でイメージ数を変更する場合と同様にできます。**lvconvert** コマンドを使用して、追加または削除するメタデータ/データサブボリュームのペアの数を指定します。LVM ミラーリングの初期実装におけるボリューム設定の変更方法

については、「[ミラー化ボリューム設定の変更](#)」を参照してください。

**lvconvert** コマンドを使用して RAID1 デバイスにイメージを追加する際、追加後のイメージ数を指定できます。または、デバイスに追加するイメージ数を指定することも可能です。また、メタデータ/データイメージのペアを置く物理ボリュームを指定することもできます。

メタデータサブボリューム (**\*\_rmeta\_\*** と呼ばれる) は、対応するデータサブボリューム (**\*\_rimage\_\***) と同じ物理デバイスに常に存在します。メタデータ/データのサブボリュームのペアは、(**--alloc anywhere** を指定しない限り) RAID アレイにある、別のメタデータ/データのサブボリュームのペアと同じ物理ボリュームには作成されません。

RAID1 ボリュームにイメージを追加するコマンドの形式は、以下のとおりです。

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

たとえば、以下のコマンドは、2 方向 RAID1 アレイである LVM デバイス **my\_vg/my\_lv** を表示します。

```
# 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)
```

以下のコマンドは、2 方向の RAID1 デバイス **my\_vg/my\_lv** を、3 方向の RAID1 デバイスに変換します。

```
# 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)
```

イメージを RAID1 アレイに追加する場合は、イメージに使用する物理ボリュームを指定できます。以下のコマンドは、2 方向の RAID1 デバイス **my\_vg/my\_lv** を、3 方向の RAID1 デバイスに変換し、物理ボリューム **/dev/sdd1** がアレイに使用されるようにします。

```
# 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)
```

RAID1 アレイからイメージを削除するには、以下のコマンドを使用します。**lvconvert** コマンドを使用して RAID1 デバイスからイメージを削除する場合は、削除後のイメージの合計数を指定できます。または、デバイスから削除するイメージ数を指定することも可能です。また、デバイスを削除する物理ボリュームを指定することもできます。

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

また、イメージとその関連付けられたメタデータサブボリュームを削除すると、それよりも大きな番号のイメージがそのスロットを引き継ぎます。たとえば、**lv\_rimage\_0**、**lv\_rimage\_1**、および **lv\_rimage\_2** で構成される 3 方向の RAID1 アレイから **lv\_rimage\_1** を削除すると、RAID1 アレイを構成するイメージの名前は **lv\_rimage\_0** と **lv\_rimage\_1** になります。サブボリューム **lv\_rimage\_2** の名前が、空のスロットを引き継いで **lv\_rimage\_1** になります。

以下の例は、3 方向の RAID1 論理ボリューム **my\_vg/my\_lv** のレイアウトを示しています。

```
# 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)
```

以下のコマンドは、3 方向の RAID1 論理ボリュームを、2 方向の RAID1 論理ボリュームに変換します。

```
# 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)
```

以下のコマンドは、3 方向の RAID1 論理ボリュームを、2 方向の RAID1 論理ボリュームに変換し、物理ボリューム **/dev/sde1** からイメージを削除することを指定します。

```
# lvconvert -m1 my_vg/my_lv /dev/sde1
```

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

#### 4.4.3.7. RAID イメージを複数の論理ボリュームに分割

RAID 論理ボリュームのイメージを分割して、新しい論理ボリュームを形成します。RAID イメージを分割する手順は、「[ミラー化論理ボリュームの冗長イメージの分割](#)」で説明されているように、ミラー化論理ボリュームの冗長イメージを分割する手順と同じです。

RAID イメージを分割するコマンドの形式は、以下のとおりです。

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

既存の RAID1 論理ボリュームから RAID イメージを削除する場合と同様に（「[既存の RAID1 デバイス内のイメージ数の変更](#)」で説明）、RAID データのサブボリューム（およびその関連付けられたメタデータのサブボリューム）をデバイスから削除する場合、それより大きい番号のイメージは、そのスロットを埋めるために番号が変更になります。そのため、RAID アレイを構成する論理ボリューム上のインデックス番号は連続する整数となります。



#### 注記

RAID1 アレイがまだ同期していない場合は、RAID イメージを分割できません。

以下の例は、2 方向の RAID1 論理ボリューム **my\_lv** を、**my\_lv** と **new** の 2 つのリニア論理ボリュームに分割します。

```
# 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)
```

以下の例は、3 方向の RAID1 論理ボリューム **my\_lv** を、2 方向の RAID1 論理ボリューム **my\_lv** と、リニア論理ボリューム **new** に分割します。

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

```

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

```

#### 4.4.3.8. RAID イメージの分割とマージ

**lvconvert** コマンドで **--splitmirrors** 引数とともに **--trackchanges** 引数を使用すると、すべての変更を追跡しながら、RAID1 アレイのイメージを一時的に読み取り専用に分割することができます。これにより、イメージの分割後に変更になったアレイの部分のみを再同期する一方で、後でそのイメージをアレイにマージし直すことができます。

RAID イメージを分割する **lvconvert** コマンドの形式は、以下のとおりです。

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

**--trackchanges** 引数を使用して RAID イメージを分割する場合、分割するイメージを指定することはできますが、分割されるボリューム名を変更することはできません。また、分割して作成されたボリュームには以下の制限があります。

- 作成された新規ボリュームは読み取り専用です。
- 新規ボリュームのサイズは変更できません。
- 残りのアレイの名前は変更できません。
- 残りのアレイのサイズは変更できません。
- 新規のボリュームと、残りのアレイを個別にアクティブ化することはできません。

**--trackchanges** 引数を使用して分割したイメージをマージするには、**lvconvert** コマンドで **--merge** 引数を指定して実行します。イメージをマージすると、イメージが分割されてから変更したアレイの部分のみが再同期されます。

RAID イメージをマージする **lvconvert** コマンドの形式は、以下のとおりです。

```
lvconvert --merge raid_image
```

以下の例は、残りのアレイへの変更を追跡する一方で、RAID1 論理ボリュームを作成し、そのボリュームからイメージを分割します。

```

# lvcreate --type raid1 -m 2 -L 1G -n my_lv .vg
Logical volume "my_lv" created
# lvs -a -o name,copy_percent,devices my_vg
LV              Copy%   Devices
my_lv           100.00

```



```

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

```

以下の例は、残りのアレイベンチンへの変更を追跡する一方で、RAID1 ボリュームからイメージを分割します。その後、ボリュームをアレイベンチンにマージし直しています。

```

# 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)

```

RAID1 ボリュームからイメージを分割後、その分割を永続的にするには **lvconvert --splitmirrors** コマンドを実行します。ここでは **--trackchanges** 引数は指定せず、イメージを分割する最初の **lvconvert** コマンドを繰り返します。これにより、**--trackchanges** 引数が作成したリンクが機能しなくなります。

追跡されるイメージを永久に分割する場合を除き、**--trackchanges** 引数を使用してイメージを分割してから、アレイベンチン上で **lvconvert --splitmirrors** コマンドを発行することはできません。

以下の一連のコマンドは、イメージを分割してこれを追跡してから、追跡されるイメージを永久に分割します。

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

ただし、以下の一連のコマンドは失敗する点に注意してください。

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

同様に、以下の一連のコマンドも失敗します。分割されたイメージが追跡されていないためです。

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

#### 4.4.3.9. RAID 障害ポリシーの設定

LVM RAID は、**lvm.conf** ファイルの **raid\_fault\_policy** フィールドで定義されている詳細設定に基づいて、デバイス障害を自動で処理します。

- **raid\_fault\_policy** フィールドが **allocate** に設定されている場合、システムは障害が発生したデバイスをボリュームグループの予備のデバイスに置き換えようとします。予備のデバイスがないと、システムログにレポートが送信されます。
- **raid\_fault\_policy** フィールドが **warn** に設定されている場合、システムは警告を生成して、ログにはデバイスが失敗したことが示されます。これにより、ユーザーは取るべき一連の動作を決めることができます。

該当するポリシーを使用するデバイスが残っている限り、RAID 論理ボリュームは操作を続行します。

##### 4.4.3.9.1. 「allocate」 RAID 障害ポリシー

以下の例では、**raid\_fault\_policy** フィールドは **lvm.conf** ファイルで **allocate** に設定されています。RAID 論理ボリュームは、以下のように配置されます。

```
# 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)

```

**/dev/sde** デバイスが失敗すると、システムログはエラーメッセージを表示します。

```

# 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.

```

**raid\_fault\_policy** フィールドが **allocate** に設定されているため、障害が発生したデバイスは、ボリュームグループの新しいデバイスに置き換わります。

```

# 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)

```

障害が発生したデバイスが置き換わっても、LVM は障害が発生したデバイスを見つけられないことが示される点に注意してください。これは、障害が発生したデバイスが、RAID 論理ボリュームからは削除されても、ボリュームグループからは削除されていないためです。障害が発生したデバイスをボリュームグループから削除するには、**vgreduce --removemissing VG** を実行できます。

**raid\_fault\_policy** が **allocate** に設定され、予備のデバイスがない場合、割り当ては失敗し、論理ボリュームはそのまま残ります。割り当てが失敗した場合は、「[「warn」 RAID 障害ポリシー](#)」で説明されているように、ドライブを修正し、その後で論理ボリュームを非アクティブ化およびアクティブ化できます。また、「[RAID デバイスの置き換え](#)」で説明されているように、障害が発生したデバイスを置き換えることも可能です。

#### 4.4.3.9.2. 「warn」 RAID 障害ポリシー

以下の例では、**raid\_fault\_policy** フィールドは **lvm.conf** ファイルで **warn** に設定されています。RAID 論理ボリュームは以下のように配置されます。

```
# 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)
```

**/dev/sdh** デバイスに障害が発生すると、システムログはエラーメッセージを表示します。ただし、この場合、LVM はイメージの 1 つを置き換えて RAID デバイスを自動的に修復しようとはしません。したがって、デバイスに障害が発生したら、以下のように **lvconvert** コマンドの **--repair** 引数を使用してデバイスを置き換えることができます。

```
# 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)
```

障害が発生したデバイスが置き換わっても、LVM は障害が発生したデバイスを見つけられないことが示される点に注意してください。これは、障害が発生したデバイスが、RAID 論理ボリュームからは削除されても、ボリュームグループからは削除されていないためです。障害が発生したデバイスをボリュームグループから削除するには、**vgreduce --removemissing VG** を実行できます。

デバイス障害が一時的か、または障害が発生したデバイスの修復が可能な場合は、**lvchange** コマンドの **--refresh** オプションを使って、障害が発生したデバイスの復旧を開始できます。これまでは、論理ボリュームを非アクティブ化してからアクティブ化することが必要でした。

以下のコマンドは論理ボリュームを更新します。

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

#### 4.4.3.10. RAID デバイスの置き換え

RAID は従来の LVM ミラーリングとは異なります。LVM ミラーリングでは、障害が発生したデバイスを削除する必要がありました。削除しないと、ミラー化論理ボリュームがハングするためです。RAID アレイは、障害があるデバイスがあっても稼働し続けることができます。RAID1 以外の RAID タイプでデバイスを削除すると、レベルが下の RAID に変わります (たとえば、RAID6 から RAID5、もしくは RAID4 または RAID5 から RAID0)。そのため、無条件に障害のあるデバイスを削除してから置き換えを行う代わりに、**lvconvert** コマンドに **--replace** 引数を使用すれば、LVM から、RAID ボリュームのデバイスを 1 回で置き換えることができます。

**lvconvert --replace** の形式は、以下のとおりです。

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

以下の例は、RAID1 論理ボリュームを作成した後に、そのボリューム内のデバイスを置き換えています。

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

以下の例は、RAID1 論理ボリュームを作成した後に、そのボリューム内のデバイスを置き換え、置き換えに使用する物理ボリュームを指定しています。

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

```

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

```

1 度に 2 つ以上の RAID デバイスを置き換えるには、以下の例のように複数の **replace** 引数を指定します。

```

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

```

## 注記

**lvconvert --replace** コマンドを使用して置き換えるドライブを指定する場合は、置き換えるドライブを、アレイ内ですでに使用されている予備のドライブ領域から割り当てないようにしてください。たとえば、**lv\_rimage\_0** と **lv\_rimage\_1** は、同じ物理ボリュームに置くことができません。

### 4.4.3.11. RAID 論理ボリュームのスクラビング

LVM は RAID 論理ボリュームのスクラビングサポートを提供します。RAID スクラビングは、すべてのデータおよびアレイ内のパリティブロックを読み込み、それらが一貫しているかどうかを確認するプロセスです。

**lvchange** コマンドの **--syncaction** オプションを使って、RAID スクラビングの操作を開始しま

す。**check** または **repair** のいずれかの操作を指定します。**check** 操作はアレイ全体を対象に、アレイ内の不一致の数を記録しますが、不一致を修復することはありません。**repair** 操作が、不一致が見つかる際に修復します。

RAID 論理ボリュームのスクラビングを実行するコマンドの形式は以下のとおりです。

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



#### 注記

**lvchange --syncaction repair vg/raid\_lv** の操作内容は、**lvconvert --repair vg/raid\_lv** と同じではありません。**lvchange --syncaction repair** は、アレイでバックグラウンドの同期操作を開始しますが、**lvconvert --repair** は、ミラーまたは RAID 論理ボリューム内の障害が発生したデバイスの修復/置換を行うように設計されています。

新しい RAID スクラビング操作をサポートするため、**lvs** コマンドは、**raid\_sync\_action** と **raid\_mismatch\_count** の 2 つの新しい出力可能なフィールドに対応しています。これらのフィールドはデフォルトでは出力されません。これらのフィールドを表示するには、以下のように **lvs** の **-o** パラメーターを使ってこれらを指定します。

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

**raid\_sync\_action** フィールドは、raid ボリュームが現在実行している同期操作を表示します。これには、以下のいずれかの値を使用することができます。

- **idle**: すべての同期操作が完了しています (何も実行していません)
- **resync**: アレイを初期化、またはマシン障害後の復旧を実行します
- **recover**: アレイ内のデバイスを置き換えます
- **check**: アレイの不一致を検索します
- **repair**: 不一致を検索し、修復します

**raid\_mismatch\_count** フィールドは、**check** 操作時に検出された不一致の数を表示します。

**lvs** コマンドの **Cpy%Sync** フィールドは、**check** および **repair** を含む **raid\_sync\_action** 操作のいずれかの進捗を出力するようになりました。

**lvs** コマンドの **lv\_attr** フィールドは、RAID スクラビング操作をサポートする追加のインジケータを提供するようになりました。このフィールドのビット 9 は、論理ボリュームの正常性を示し、以下のインジケータがサポートされるようになりました。

- 「(m)ismatches (不一致)」は、RAID 論理ボリュームに不一致があることを示します。この文字は、スクラビング操作で RAID に一貫性がない部分があることを検出した後に表示されません。
- 「(r)efresh (更新)」は、LVM がデバイスラベルを読み取り、デバイスを稼働できると認識した場合でも、RAID アレイのデバイスに障害が発生し、カーネルがこれを障害と認識していることを示します。この論理ボリュームを、デバイスが利用可能になったことをカーネルに通知するために更新する (refresh) か、もしくはデバイスに障害が発生したと思われる場合はデバイスを置き換える (replace) 必要があります。



**lv**s コマンドについての情報は、「[オブジェクト表示フィールド](#)」を参照してください。

RAID スクラビング操作を実行する際、**sync** 操作で必要になるバックグラウンド I/O が、ボリュームグループメタデータへの更新などの他の I/O 操作を LVM デバイスに押し出す可能性があります。これにより、他の LVM 操作が遅くなる可能性があります。復旧スロットルを実装して RAID 論理ボリュームのスクラビングを実行する速度を制御することができます。

**sync** 操作の実行される速度は、**lvchange** コマンドの **--minrecoveryrate** および **--maxrecoveryrate** オプションを使用して、それらの操作の最小および最大 I/O 速度を設定することにより制御できます。これらのオプションは以下のように指定します。

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

RAID 論理ボリュームの最大復旧速度を設定し、通常の I/O 操作が押し出されないようにします。**速度** は、アレイ内の各デバイスに対して、1 秒あたりの量を指定します。サフィックスを指定しない場合は、kiB/sec/device と見なされます。復旧速度を 0 に設定すると無制限になります。

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

RAID 論理ボリュームの最小復旧速度を設定し、負荷の高い通常の I/O がある場合でも、**sync** 操作の I/O が最小スループットを達成できるようにします。**速度** アレイ内の各デバイスに対して、1 秒あたりの量を指定します。サフィックスを指定しない場合は、kiB/sec/device と見なされます。

#### 4.4.3.12. RAID テイクオーバー (Red Hat Enterprise Linux 7.4 以降)

LVM は、Raid テイクオーバーをサポートします。これは、RAID 論理ボリュームの RAID レベルを別のレベル (たとえば RAID 5 から RAID 6) へ変えることを意味します。RAID レベルの変更は、通常、デバイスの耐障害性を増減したり、論理ボリュームのストライプ化をやり直すことで行います。RAID テイクオーバーには **lvconvert** を使用します。RAID テイクオーバーの詳細と、**lvconvert** を使用して RAID 論理ボリュームを変換する例は、**lvraid(7)** の man ページを参照してください。

#### 4.4.3.13. RAID 論理ボリュームの再成形 (Red Hat Enterprise Linux 7.4 以降)

RAID 再成形は、同じ RAID レベルを維持しつつ、RAID 論理ボリュームの属性を変更することを示しています。変更できる属性例としては、RAID レイアウトと、ストライプのサイズおよび数が挙げられます。RAID 再成形と、**lvconvert** コマンドを使用して RAID 論理ボリュームを再生成する例は、**lvraid(7)** の man ページを参照してください。

#### 4.4.3.14. RAID1 論理ボリュームでの I/O 操作の制御

**lvchange** コマンドの **--writemostly** および **--writebehind** パラメーターを使用して、RAID1 論理ボリュームのデバイスに対する I/O 操作を制御できます。これらのパラメーターを使用する形式は以下のとおりです。

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

RAID1 論理ボリューム内のデバイスを **write-mostly** とマークします。これらのドライブのすべての読み取りは、必要でない限り回避されます。このパラメーターを設定することにより、ドライブに対する I/O 操作の回数を最小限に抑えることができます。デフォルトでは、論理ボリュームに指定された物理ボリュームに対して、**write-mostly** 属性を **yes** に設定します。**:n** を物理ボリュームに追加して **write-mostly** フラグを削除したり、**:t** を指定して値

を切り替えたりすることができます。--**writemostly** 引数は、1 つのコマンドで複数回指定することができるため、1 回で論理ボリュームのすべての物理ボリュームで、write-mostly 属性を切り替えることができます。

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

**write-mostly** というマークが付いている RAID1 論理ボリュームのデバイスに許可される、未処理の書き込みの最大数を指定します。この値を上回ると書き込みは同期され、構成要素になっているデバイスへの書き込みがすべて、アレイが書き込みの完了を知らせる前に完了してしまいます。この値をゼロに設定すると、設定はクリアになり、システムが値を任意に選択できるようになります。

#### 4.4.3.15. RAID 論理ボリュームのリージョンサイズの変更 (Red Hat Enterprise Linux 7.4 以降)

RAID 論理ボリュームを作成すると、論理ボリュームのリージョンサイズは、`/etc/lvm/lvm.conf` ファイルの **raid\_region\_size** パラメーターの値になります。このデフォルト値は、**lvcreate** コマンドの **-R** オプションで上書きできます。

RAID 論理ボリュームを作成したら、**lvconvert** コマンドの **-R** オプションで、ボリュームのリージョンサイズを変更できます。以下の例では、論理ボリューム **vg/raidlv** のリージョンサイズを 4096K に変更します。リージョンサイズを変更するには、RAID ボリュームを同期する必要があります。

```
# lvconvert -R 4096K vg/raid1
Do you really want to change the region_size 512.00 KiB of LV vg/raid1 to
4.00 MiB? [y/n]: y
Changed region size on RAID LV vg/raid1 to 4.00 MiB.
```

#### 4.4.4. ミラー化ボリュームの作成

Red Hat Enterprise Linux 7.0 リリースの場合、「[RAID 論理ボリューム](#)」で説明されているように、LVM は RAID 1/4/5/6/10 に対応します。RAID 論理ボリュームはクラスターには対応していません。RAID 論理ボリュームは 1 台のマシン上で作成でき、かつ排他的にアクティブ化することができますが、複数のマシンで同時にアクティブにすることはできません。排他的ではないミラー化されたボリュームが必要な場合は、「[ミラー化ボリュームの作成](#)」に説明されているようにセグメントタイプの **mirror** を指定して、ボリュームを作成する必要があります。



#### 注記

**mirror** セグメントタイプを指定した既存の LVM デバイスを RAID1 LVM デバイスに変換する方法についての情報は、「[ミラー化 LVM デバイスの RAID1 デバイスへの変換](#)」を参照してください。

## 注記

ミラー化された LVM 論理ボリュームをクラスター内に作成するには、単一ノード上で **mirror** のセグメントタイプを指定したミラー化論理ボリュームを作成するのと同じコマンドと手順が必要です。しかし、クラスター内にミラー化 LVM ボリュームを作成するには、クラスターとクラスターミラーインフラストラクチャーが稼動中であり、クラスターが定足数を満たしており、かつクラスターロッキングを有効化するように、**lvm.conf** ファイルでロッキングタイプが正しく設定されている必要があります。クラスター内におけるミラー化ボリュームの作成例は、「[クラスター内でのミラー化 LVM 論理ボリュームの作成](#)」をご覧ください。

単一クラスター内の複数のノードから短時間に連続して複数の LVM ミラーを作成または変換するコマンドを実行しようとする、これらのコマンドのバックログが生じる場合があります。これによって、要求した操作がタイムアウトになって失敗する可能性があります。この問題を回避するために、そのクラスターのいずれかのノードから、クラスターミラー作成コマンドを実行することを推奨します。

ミラー化ボリュームを作成する場合、**lvcreate** コマンドの **-m** 引数を使用して、データのコピー数を指定します。**-m1** と指定すると、ミラーが 1 つ作成され、ファイルシステムのコピーが合計 2 つとなります (リニア論理ボリューム 1 つと、コピーが 1 つ)。同じように **-m2** と指定すると、ミラーが 2 つ作成され、ファイルシステムのコピーが合計 3 つとなります。

以下のコマンドは、ミラーが 1 つあるミラー化論理ボリュームを作成します。ボリュームのサイズは 50 ギガバイト、名前は **mirrorlv** で、ボリュームグループ **vg0** から作り出されます。

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

デフォルトでは、LVM ミラーデバイスは、複製されるデバイスを、サイズが 512KB のリージョンに分割します。**lvcreate** コマンドに **-R** 引数を使用すると、リージョンサイズをメガバイト単位で指定できます。また、**lvm.conf** ファイル内で **mirror\_region\_size** 設定で、デフォルトのリージョンサイズを変更することも可能です。

## 注記

クラスターインフラストラクチャーの制限により、デフォルトのリージョンサイズ (512KB) では、1.5TB を超えるクラスターミラーは作成できません。1.5TB よりも大きなミラーを必要とするユーザーは、リージョンサイズをデフォルトよりも大きくする必要があります。リージョンサイズが小さいままだと、LVM の作成がハングしてしまいます。また、その他の LVM コマンドでもハングの可能性があります。

1.5TB を超えるミラー用のリージョンサイズを指定するには、ミラーサイズをテラバイト単位で考えて、2 の次の累乗に切り上げ、その数を **lvcreate** コマンドの **-R** 引数として使用することが一般的な方法です。たとえば、ミラーサイズが 1.5TB の場合は、**-R 2** と指定することができます。また、ミラーサイズが 3TB の場合は **-R 4**、5TB の場合は **-R 8** と指定できます。

以下のコマンドは、リージョンサイズが 2MB のミラー化論理ボリュームを作成します。

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

ミラーが作成されると、ミラーのリージョンは同期されます。ミラーコンポーネントが大きい場合は、同期プロセスに時間がかかる可能性があります。作成しているミラーを回復させる必要がない場合は、**--nosync** 引数を指定して、最初のデバイスからの初期の同期は不要であることを指定することが

できます。

LVM は、単一または複数のミラーと同期するリージョンを追跡するのに使用する小さなログを維持します。デフォルトでは、このログはディスクに保持され、再起動後も永続化するため、マシンが再起動/クラッシュするたびにミラーを再同期する必要はありません。代わりに、**--corelog** 引数を使用すると、このログがメモリーで保持されるように指定できるため、余分なログデバイスが不要になります。ただし、この場合は、再起動のたびにミラー全体を再同期する必要がでてきます。

以下のコマンドは、ボリュームグループ **bigvg** からミラー化論理ボリュームを作成します。論理ボリュームの名前は **ondiskmirvol** で、ミラー が 1 つあります。ボリュームのサイズは 12MB で、ミラーログをメモリーに保持します。

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

このミラーログは、ミラーレグが作成されるデバイスとは異なるデバイスで作成されます。しかし、**vgcreate** コマンドに **--alloc anywhere** 引数を使用すれば、ミラーレグの 1 つと同じデバイス上ミラーログを作成することができます。ただし、これによりパフォーマンスが低下する場合がありますが、配下のデバイスが 2 つしかなくてもミラーを作成できます。

以下のコマンドは、単一のミラーを持つミラー化論理ボリュームを作成します。このミラーログはミラーレグの 1 つと同じデバイス上にあります。この例では、ボリュームグループ **vg0** は 2 つのデバイスで構成されています。このコマンドによって、ボリュームグループ **vg0** 内に、名前が **mirrorlv** で、サイズが 500 MB のボリュームが作成されます。

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

## 注記

クラスター化されたミラーでは、ミラーログ管理は、その時点でクラスター ID の最も低いクラスターノードによって行われます。そのため、クラスターミラーログを保持するデバイスがクラスターのサブセット上で利用できなくなる場合、最も低い ID を持つクラスターノードがミラーログへのアクセスを保持する限り、クラスター化されたミラーは影響を受けることなく、機能を継続することができます。ミラーは影響を受けないため、自動修正アクション (修復) も実行されません。ただし、最も低い ID のクラスターノードがミラーログにアクセスできなくなると、(他のノードからログへのアクセスが可能かどうかにかかわらず) 自動アクションが作動します。

自動的にミラー化されるミラーログを作成するために、**--mirrorlog mirrored** 引数を指定することができます。以下のコマンドはボリュームグループ **bigvg** からミラー化論理ボリュームを作成します。論理ボリュームは **twologvol** という名前で、単一のミラーを持ちます。このボリュームのサイズは 12MB で、ミラーログがミラー化され、各ログは別個のデバイス上に保管されます。

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

標準ミラーログと同様に、**vgcreate** コマンドの **--alloc anywhere** 引数を使用してミラーレグと同じデバイス上に冗長ミラーログを作成することが可能です。これによってパフォーマンスが低下する可能性があります。各ログを別個のデバイス上に保管するための配下のデバイス数がミラーレグに対して十分でない場合でも、冗長ミラーログの作成が可能となります。

ミラーが作成されると、ミラーのリージョンは同期されます。ミラーコンポーネントが大きい場合は、同期プロセスに時間がかかる可能性があります。作成しているミラーを回復させる必要がない場合は、**--nosync** 引数を指定して、最初のデバイスからの初期の同期は不要であることを指定することができます。

ミラーレグとログ用に使用するデバイス、およびそのデバイスで使用するエクステントを指定することができます。ログを特定のディスクに強制するには、それが配置されるディスク上のエクステントを正確に 1 つ指定します。LVM は、コマンドラインでデバイスが一覧表示される順序を必ずしも優先しません。物理ボリュームが一覧にあれば、それが割り当てが実行される唯一の領域になります。割り当て済みの物理エクステントが一覧にある場合、そのエクステントは無視されます。

以下のコマンドは、単一のミラーとミラー化されない単一ログを持つミラー化論理ボリュームを作成します。このボリュームは、サイズ 500 MB、名前は **mirrorlv** で、ボリュームグループ **vg0** から構築されます。第 1 のミラーレグはデバイス **/dev/sda1** 上にあり、第 2 のミラーレグはデバイス **/dev/sdb1** 上にあり、そのミラーログは **/dev/sdc1** 上にあります。

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

以下のコマンドは、単一のミラーを持つミラー化論理ボリュームを作成します。このボリュームは、サイズは 500 MB、名前は **mirrorlv** であり、ボリュームグループ **vg0** から構築されます。第 1 のミラーレグは、エクステントが 0 から 499 のデバイス **/dev/sda1** にあり、第 2 のミラーレグはエクステントが 0 から 499 のデバイス **/dev/sdb1** にあります。ミラーログは、エクステントが 0 のデバイス **/dev/sdc1** から始まります。エクステントサイズは 1MB です。指定されたエクステントのいずれかが割り当て済みである場合は、それらは無視されます。

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



#### 注記

単一の論理ボリューム内でストライピングとミラーリングを併用することが可能です。論理ボリュームの作成と同時にミラーの数 (**--mirrors X**) とストライプの数 (**--stripes Y**) を指定すると、ミラーデバイスの構成デバイスがストライプ化されます。

#### 4.4.4.1. ミラー化論理ボリュームの障害ポリシー

**lvm.conf** ファイルの **activation** セクション内の **mirror\_image\_fault\_policy** と **mirror\_log\_fault\_policy** のパラメーターを使用すると、デバイスの障害が発生した場合にミラー化論理ボリュームがどのような動作をするかを定義することができます。これらのパラメーターが **remove** に設定されると、システムは障害のあるデバイスを削除して、そのデバイスなしで実行しようとしします。これらのパラメーターが **allocate** に設定されていると、システムは障害のあるデバイスを削除して、そのデバイスの代わりとなる新たなデバイス上での領域の割り当てを試みます。代わりに割り当てることができる適切なデバイスと領域がない場合、このポリシーは **remove** ポリシーのように機能します。

デフォルトでは、**mirror\_log\_fault\_policy** パラメーターは **allocate** に設定されます。ログにこのポリシーを使用すると、処理が高速になり、クラッシュやシステムの再起動時にも同期状態を記憶する機能が維持されます。このポリシーを **remove** に設定すると、ログデバイスに障害が発生した際に、ミラーがメモリー内ログを使用するように切り替わります。この場合、ミラーはクラッシュ時とシステムの再起動時に同期状態を記憶せず、ミラー全体が再同期されます。

デフォルトでは、**mirror\_image\_fault\_policy** パラメーターは **remove** に設定されます。このポ



リシーでは、ミラーイメージに障害が発生すると、良好なコピーが1つしか残っていない場合は、ミラーが非ミラー化デバイスに変換されます。ミラーデバイスに対してこのポリシーを**allocate**に設定すると、ミラーはデバイスを再同期する必要があるため、処理に時間がかかりますが、これによってデバイスのミラー特性を保持することができます。

### 注記

LVM ミラーにデバイス障害が発生すると、2段階の回復プロセスが実行されます。第1段階では、障害が発生したデバイスの削除が行われます。これによってミラーは、単一のリニアデバイスに縮小されます。第2段階では、**mirror\_log\_fault\_policy** パラメーターが **allocate** に設定されている場合、障害の発生したデバイスの置き換えを試みます。ただし、第2段階では、他のデバイスが利用可能である場合、ミラーが以前使用していたデバイスの中から、障害とは関係のないデバイスが選択されるという保証はない点に注意してください。

LVM ミラー障害が発生した際の手動で回復する方法についての情報は、[「LVM ミラー障害からの回復」](#) を参照してください。

#### 4.4.4.2. ミラー化論理ボリュームの冗長イメージの分割

ミラー化論理ボリュームの冗長イメージを分割して、新たな論理ボリュームを形成することができます。イメージを分割するには、**lvconvert** コマンドの **--splitmirrors** 引数を使用して、分割する冗長イメージの数を指定します。新たに分割する論理ボリュームの名前を指定するには、このコマンドの **--name** 引数を使用する必要があります。

以下のコマンドは、ミラー化論理ボリューム **vg/lv** から、**copy** という名前の新たな論理ボリュームを分割します。新しい論理ボリュームには2つのミラーレグが含まれます。この例では、LVM は分割するデバイスを選択しています。

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

分割するデバイスを指定することが可能です。以下のコマンドは、ミラー化論理ボリューム **vg/lv** から **copy** という名前の新たな論理ボリュームを分割します。新しい論理ボリュームには、**/dev/sdc1** と **/dev/sde1** のデバイスで構成される、2つのミラーレグが含まれます。

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

#### 4.4.4.3. ミラー化論理ボリュームの修復

**lvconvert --repair** コマンドを使用すると、ディスクの障害後にミラーを修復することができます。これによって、ミラーは整合性のある状態に戻ります。**lvconvert --repair** コマンドは、インタラクティブなコマンドで、障害のあるデバイスの置き換えをシステムに試行させるかどうかを指定するようにプロンプトを出します。

- プロンプトを省略して障害の発生したデバイスをすべて置き換えるには、コマンドライン上で **-y** オプションを指定します。
- プロンプトを省略して、障害の発生したデバイスを一切置き換えないようにするには、コマンドライン上で **-f** オプションを指定します。
- プロンプトを省略し、かつミラーイメージとミラーログを対象とする異なる置き換えポリシーを示すには、**--use-policies** 引数を指定して、**lvm.conf** ファイル内の **mirror\_log\_fault\_policy** および **mirror\_device\_fault\_policy** パラメーターによ

て指定されているデバイス置き換えポリシーを使用することができます。

#### 4.4.4.4. ミラー化ボリューム設定の変更

**lvconvert** コマンドを使用して、論理ボリュームに含まれるミラーの数を増加/減少させることができます。これにより、論理ボリュームをミラー化ボリュームからリニアボリュームに、またはリニアボリュームからミラー化ボリュームに変換できます。また、このコマンドを使用して、**corelog** などの既存の論理ボリュームの他のミラーパラメーターも再設定できます。

リニアボリュームをミラー化ボリュームに変換する際には、既存ボリューム用にミラーレグを作成することになります。つまり、ボリュームグループにはミラーレグ用とミラーログ用のデバイスと領域がなければならないことを意味します。

ミラーレグを 1 つ失うと、LVM はそのボリュームをリニアボリュームに変換するため、ミラーの冗長性なしにボリュームに引き続きアクセスできます。そのレグを置き換えた後は、**lvconvert** コマンドを使用して、ミラーを復元できます。この手順は「[LVM ミラー障害からの回復](#)」で説明されています。

以下のコマンドは、リニア論理ボリューム **vg00/lvol1** をミラー化論理ボリュームに変換します。

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

以下のコマンドは、ミラー化論理ボリューム **vg00/lvol1** をリニア論理ボリュームに変換して、ミラーレグを削除します。

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

以下のコマンドは、既存の論理ボリューム **vg00/lvol1** にミラーレグを追加します。この例は、**lvconvert** コマンドがそのボリュームを 2 つのミラーレグがあるボリュームに変更する前後のボリュームの設定を示しています。

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

#### 4.4.5. シンプロビジョニングされた論理ボリュームの作成

論理ボリュームのシンプロビジョニングが可能になりました。これにより、利用可能なエクステントよりも大きい論理ボリュームを作成できます。シンプロビジョニングを使用すると、空き領域のストレージ



ジプール (シンプールと呼ばれる) を管理して、アプリケーションで必要になったときに任意の数のデバイスに割り当てることができます。その後、アプリケーションを実際に論理ボリュームに書き込むときに、後で割り当てる用に、シンプールにバインド可能なデバイスを作成できます。シンプールは、コスト効率が高いストレージ領域を割り当てる必要がある場合に、動的に拡張できます。



## 注記

このセクションでは、シンプロビジョニングされた論理ボリュームを作成し、拡張するために使用する基本的なコマンドの概要を説明します。LVM シンプロビジョニングの詳細情報と、シンプロビジョニングされた論理ボリュームと共に LVM コマンドおよびユーティリティを使用する方法についての情報は、**lvmtthin(7) man** ページを参照してください。



## 注記

シンボリュームはクラスターのノード間ではサポートされません。シンプールとそのすべてのシンボリュームは、1 つのクラスターノードで排他的にアクティブにする必要があります。

シンボリュームを作成するには、以下のタスクを実行します。

1. **vgcreate** コマンドを使用して、ボリュームグループを作成します。
2. **lvcreate** コマンドを使用して、シンプールを作成します。
3. **lvcreate** コマンドを使用して、シンプール内にシンプロビジョニングされたボリュームを作成します。

**lvcreate** コマンドに **-T** (または **--thin**) オプションを使用して、シンプールまたはシンプロビジョニングされたボリュームを作成します。また、**lvcreate** コマンドの **-T** オプションを使用して、1 つのコマンドで同時にプール内にシンプールとシンプロビジョニングされたボリュームの両方を作成することも可能です。

以下のコマンドは、**lvcreate** コマンドの **-T** オプションを使用して、**mythinpool** という名前のシンプールを作成します。これは、ボリュームグループ **vg001** 内にあり、サイズは 100M です。物理領域のプールを作成しているため、プールのサイズを指定する必要があります。**lvcreate** コマンドの **-T** オプションは引数を取りません。コマンドで指定する他のオプションから、作成されるデバイスのタイプが推定されます。

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

以下のコマンドは、**lvcreate** コマンドに **-T** オプションを使用して、シンプール **vg001/mythinpool** に **thinvolume** という名前のシンボリュームを作成します。ここでは、仮想サイズを指定して、ボリュームを含むプールよりも大きなボリュームの仮想サイズを指定している点に注意してください。

```
# lvcreate -V 1G -T vg001/mythinpool -n thinvolume
Logical volume "thinvolume" created
```

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

以下のコマンドは、**lvcreate** コマンドに **-T** オプションを使用して、プール内にシンプールとシンプロビジョニングされたボリュームを作成します。その際、**lvcreate** コマンドでサイズと仮想サイズの引数を指定します。また、このコマンドは、ボリュームグループ **vg001** にシンプール **mythinpool** を作成し、そのプールにシンプロビジョニングされたボリューム **thinvolume** も作成します。

```
# lvcreate -L 100M -T vg001/mythinpool -V 1G -n thinvolume
Rounding up size to full physical extent 4.00 MiB
Logical volume "thinvolume" created
# lvs
  LV          VG      Attr      LSize   Pool                Origin Data%  Move Log
Copy%  Convert
mythinpool  vg001    twi-a-tz 100.00m                0.00
thinvolume  vg001    Vwi-a-tz  1.00g mythinpool          0.00
```

また、**lvcreate** コマンドの **--thinpool** パラメーターを指定して、シンプールを作成することもできます。**-T** オプションとは異なり、**--thinpool** パラメーターには作成しているシンプール論理ボリューム名の引数が必要です。以下の例は、**lvcreate** コマンドで **--thinpool** パラメーターを指定して、**mythinpool** という名前のシンプールを作成します。これは、ボリュームグループ **vg001** 内にあり、サイズは 100M です。

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

ストライピングはプールを作成するためにサポートされています。以下のコマンドは、2つの 64 kB のストライプがあり、チャンクサイズが 256 kB のボリュームグループ **vg001** 内に **pool** という名前の 100M のシンプールを作成します。また、1T のシンボリューム **vg00/thin\_lv** も作成します。

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

**lvextend** コマンドを使用して、シンボリュームのサイズを拡張できます。ただし、シンプールのサイズを縮小することはできません。

以下のコマンドは、既存のシンプールのサイズ (100M) を変更し、100M 拡張します。

```
# lvextend -L+100M vg001/mythinpool
Extending logical volume mythinpool to 200.00 MiB
Logical volume mythinpool successfully resized
# lvs
  LV          VG      Attr      LSize   Pool                Origin Data%  Move Log
Copy%  Convert
mythinpool  vg001    twi-a-tz 200.00m                0.00
thinvolume  vg001    Vwi-a-tz  1.00g mythinpool          0.00
```

他の論理ボリュームのタイプと同様に、**lvrename** を使用してボリューム名の変更、**lvremove** を使用してボリュームの削除、**lvs** と **lvdisplay** のコマンドを使用してボリュームの情報の表示を行うことができます。

デフォルトでは、**lvcreate** コマンドは、計算式 ( $\text{Pool\_LV\_size} / \text{Pool\_LV\_chunk\_size} * 64$ ) から、シンプールのメタデータ論理ボリュームのサイズを設定します。スナップショットが大量にある場合、または、シンプールのサイズが小さい場合は、後でシンプールのサイズが急激に大きくなることが予測される場合は、**lvcreate** コマンドの **--poolmetadatasize** パラメーターで、シンプールのメタデータボリュームでデフォルト値の増加が必要となる場合があります。シンプールのメタデータ論理ボリュームでサポートされる値は 2MiB - 16GiB です。

**lvconvert** コマンドの **--thinpool** パラメーターを使用して、既存の論理ボリュームをシンプールボリュームに変換できます。既存の論理ボリュームをシンプールボリュームに変換する場合、**lvconvert** コマンドの **--thinpool** パラメーターとともに **--poolmetadata** パラメーターを使用して、既存の論理ボリュームをシンプールボリュームのメタデータボリュームに変換する必要があります。



### 注記

論理ボリュームをシンプールボリュームまたはシンプールメタデータボリュームに変換すると、論理ボリュームのコンテンツが破棄されます。この場合、**lvconvert** はデバイスのコンテンツを保存するのではなく、コンテンツを上書きするためです。

以下の例は、ボリュームグループ **vg001** の既存の論理ボリューム **lv1** をシンプールボリュームに変換します。また、ボリュームグループ **vg001** の既存の論理ボリューム **lv2** を、そのシンプールボリュームのメタデータボリュームに変換します。

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

## 4.4.6. スナップショットボリュームの作成



### 注記

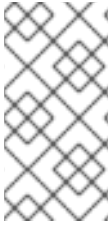
LVM はシンプロビジョニングされたスナップショットをサポートします。シンプロビジョニングされたスナップショットボリュームの作成については、[「シンプロビジョニングされたスナップショットボリュームの作成」](#) を参照してください。

スナップショットボリュームを作成するには、**lvcreate** コマンドで **-s** 引数を使用します。スナップショットボリュームは書き込み可能です。



### 注記

LVM スナップショットは、クラスター内のノード間ではサポートされていません。クラスター化されたボリュームグループ内にスナップショットボリュームは作成できません。ただし、クラスター論理ボリューム上でデータの一貫したバックアップ作成が必要な場合、ボリュームを排他的にアクティブ化した上で、スナップショットを作成することができます。1つのノード上で論理ボリュームを排他的にアクティブ化する方法は、[「クラスター内の個別ノードでの論理ボリュームのアクティブ化」](#) を参照してください。



## 注記

ミラー化論理ボリュームを対象とした LVM スナップショットがサポートされています。

RAID 論理ボリュームを対象としたスナップショットがサポートされています。RAID 論理ボリュームの作成方法は、「[RAID 論理ボリューム](#)」を参照してください。

LVM では、複製元のボリュームのサイズよりも大きく、そのボリュームのメタデータを必要とするスナップショットを作成できません。これよりも大きなスナップショットボリュームを指定しても、システムには、複製元のサイズに必要な大きさのスナップショットボリュームのみが作成されます。

デフォルトで、スナップショットボリュームは、通常のアクティブ化コマンドの実行時に省略されます。スナップショットボリュームのアクティブ化を制御する方法は、「[論理ボリュームのアクティブ化の制御](#)」を参照してください。

以下のコマンドは、名前が `/dev/vg00/snap` でサイズが 100 MB のスナップショット論理ボリュームを作成します。これは、`/dev/vg00/lvol1` という名前の元の論理ボリュームのスナップショットを作成します。元の論理ボリュームにファイルシステムが含まれている場合は、任意のディレクトリー上でスナップショット論理ボリュームをマウントしてから、そのファイルシステムのコンテンツにアクセスし、元のファイルシステムが更新を継続している間にバックアップを実行することができます。

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

スナップショット論理ボリュームを作成した後に、**lvdisplay** コマンドで、複製元のボリュームを指定すると、すべてのスナップショット論理ボリュームとそのステータス (アクティブまたは非アクティブ) の一覧が出力されます。

以下の例は、論理ボリューム `/dev/new_vg/lvol0` のステータスを示しています。これに対して、スナップショットボリューム `/dev/new_vg/newvgsnap` が作成されています。

```
# 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
```

デフォルトでは **lvs** コマンドは、複製元のボリュームと、使用されているスナップショットボリュームの現在の割合を表示します。以下の例は、論理ボリューム `/dev/new_vg/lvol0` を含むシステム用の **lvs** コマンドのデフォルト出力を示しています。スナップショットボリューム `/dev/new_vg/newvgsnap` はこの論理ボリューム用に作成されています。

```
# lvs
LV          VG      Attr      LSize   Origin Snap%   Move Log Copy%

```

```
lvvol0      new_vg owi-a- 52.00M
newvgsnap1  new_vg swi-a-  8.00M lvvol0      0.20
```



### 警告

複製元ボリュームが変更されると、スナップショットのサイズが拡大されるため、**lvs** コマンドを使用して、スナップショットボリュームのパーセンテージを定期的に監視して、満杯にならないように確認することが重要です。100% になったスナップショットは、完全に消失します。これは、複製元ボリュームの変更されていない部分への書き込みにより、スナップショットが必ず破損するためです。

スナップショットが満杯になると、スナップショット自体が無効になるだけでなく、そのスナップショットデバイスにマウントされているすべてのファイルシステムのマウントが強制的に解除されます。これにより、マウントポイントへのアクセス時に必ず発生するファイルシステムエラーを回避できます。さらに、**lvm.conf** ファイルで **snapshot\_autoextend\_threshold** オプションを指定することができます。このオプションによって、スナップショットの残りの領域が設定されたしきい値を下回ると、常にスナップショットを自動的に拡張できるようになりました。この機能の利用に際しては、ボリュームグループ内に未割り当ての領域があることが条件になります

LVM では、複製元ボリュームのサイズよりも大きく、そのボリュームのメタデータを必要とするスナップショットボリュームを作成できません。同様に、スナップショットの自動拡張を実行しても、スナップショットに必要なサイズとして計算される最大サイズを超えるまでにスナップショットボリュームが拡張されることはありません。スナップショットのサイズが複製元のボリュームを包含できるまで拡大されると、スナップショットの自動拡張はモニターされなくなります。

**snapshot\_autoextend\_threshold** と **snapshot\_autoextend\_percent** の設定についての詳細は、**lvm.conf** ファイルに記載されています。**lvm.conf** ファイルの詳細については、「[付録B LVM 設定ファイル](#)」を参照してください。

#### 4.4.7. シンプロビジョニングされたスナップショットボリュームの作成

Red Hat Enterprise Linux は、シンプロビジョニングされたスナップショットボリュームのサポートを提供します。シンプロビジョニングされたスナップショットボリュームのメリットとデメリットの詳細は、「[シンプロビジョニングされたスナップショットボリューム](#)」を参照してください。



### 注記

このセクションでは、シンプロビジョニングされたスナップショットボリュームを作成し、拡張するために使用する基本的なコマンドの概要を説明します。LVM シンプロビジョニングの詳細情報と、シンプロビジョニングされた論理ボリュームと共に LVM コマンドおよびユーティリティーを使用する方法は、**lvmtthin(7)** man ページを参照してください。

## 重要

シンプロビジョニングされたスナップショットボリュームを作成する場合、ボリュームのサイズは指定しません。サイズパラメーターを指定すると、作成されるスナップショットはシンプロビジョニングされたスナップショットボリュームにはならず、データを保管するためにシンプールを使用することもあります。たとえば、**lvcreate -s vg/thinvolume -L10M** コマンドは、作成元ボリュームがシンボリュームであっても、シンプロビジョニングされたスナップショット (シンスナップショット) を作成しません。

シンスナップショットは、シンプロビジョニングされた作成元ボリューム用に作成するか、またはシンプロビジョニングされない作成元ボリューム用にも作成できます。

**lvcreate** コマンドで **--name** オプションを使用してスナップショットボリューム名を指定することができます。以下のコマンドは、**mynsnapshot1** と呼ばれるシンプロビジョニングされた論理ボリューム **vg001/thinvolume** のシンプロビジョニングされたスナップショットボリュームを作成します。

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

シンスナップショットボリュームには、他のシンボリュームと同じ特性があります。ボリュームのアクティブ化、拡張、名前変更、削除、さらにはスナップショット作成も個別に行うことができます。

デフォルトで、スナップショットボリュームは、通常のアクティブ化コマンドの実行時に省略されます。スナップショットボリュームのアクティブ化を制御する方法は、[「論理ボリュームのアクティブ化の制御」](#) を参照してください。

シンプロビジョニングされていない論理ボリュームの、シンプロビジョニングされたスナップショットを作成することもできます。シンプロビジョニングされていない論理ボリュームはシンプール内に含まれていないため、**外部の複製元**と呼ばれます。外部の複製元ボリュームは、複数の異なるシンプールの、多くのシンプロビジョニングされたスナップショットボリュームによって使用され、共有されることも可能です。外部の複製元は、シンプロビジョニングされたスナップショットが作成される際に非アクティブであり、かつ読み取り専用である必要があります。

外部の複製元のシンプロビジョニングされたスナップショットを作成するには、**--thinpool** オプションを指定する必要があります。以下のコマンドは、読み取り専用の非アクティブなボリューム **origin\_volume** のシンスナップショットボリュームを作成します。このシンスナップショットボリュームの名前は **mythinsnap** です。論理ボリューム **origin\_volume** は、既存のシンプール **vg001/pool** を使用する、ボリュームグループ **vg001** 内のシンスナップショットボリューム **mythinsnap** に対する外部の複製元になります。複製元ボリュームは、スナップショットボリュームと同じボリュームグループ内に存在する必要があるため、複製元の論理ボリュームを指定する場合にボリュームグループを指定する必要はありません。

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

以下のコマンドにあるように、最初のスナップショットボリュームの 2 番目のシンプロビジョニングされたスナップショットボリュームを作成することができます。



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

Red Hat Enterprise Linux 7.2 以降、**lv** コマンドのレポートフィールド **lv\_ancestors** および **lv\_descendants** 指定すると、シンスナップショット論理ボリュームのすべての先祖 (ancestor) と子孫 (descendant) をそれぞれ表示できます。

下記の例は以下を意味します。

- **stack1** は、ボリュームグループ **vg001** で元となるボリュームです。
- **stack2** は、**stack1** のスナップショットです。
- **stack3** は、**stack2** のスナップショットです。
- **stack4** は、**stack3** のスナップショットです。

さらに

- **stack5** も、**stack2** のスナップショットです。
- **stack6** は、**stack5** のスナップショットです。

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

## 注記

**lv\_ancestors** フィールドおよび **lv\_descendants** フィールドは、既存の依存関係を表示しますが削除されたエントリは追跡しません。このチェーンの最中にエントリが削除されると、依存関係チェーンが壊れるからです。たとえば、この設定例から論理ボリューム **stack3** を削除すると、以下のように表示されます。

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

ただし、Red Hat Enterprise Linux 7.3 以降は、削除した論理ボリュームを追跡して表示するようにシステムを設定できます。**lv\_ancestors\_full** フィールドおよび **lv\_descendants\_full** フィールドを指定することで、このボリュームを含む完全依存チェーンを表示できます。過去の論理ボリュームを追跡、表示、および削除する方法は、[「過去の論理ボリュームの追跡および表示 \(Red Hat Enterprise Linux 7.3 以降\)」](#) を参照してください。



#### 4.4.8. LVM 論理ボリュームの作成

Red Hat Enterprise Linux 7.1 リリースでは、LVM により LVM キャッシュ論理ボリュームが完全にサポートされます。キャッシュ論理ボリュームでは高速なブロックデバイス (SSD ドライブなど) から構成される小さい論理ボリュームが使用されるため、頻繁に使用されるブロックを小さい高速な論理ボリュームに格納することにより、大きい低速な論理ボリュームのパフォーマンスが向上します。

LVM キャッシュは LVM 論理ボリュームタイプを使用します。関連するこれらすべての論理ボリュームは、同じボリュームグループ内に存在する必要があります。

- 複製元論理ボリューム — 大きい低速な論理ボリューム
- キャッシュプール論理ボリューム — 小さい高速な論理ボリューム。キャッシュデータ論理ボリュームとキャッシュメタデータ論理ボリュームの 2 つのデバイスから構成されます。
- キャッシュデータ論理ボリューム — キャッシュプール論理ボリューム用のデータブロックを含む論理ボリューム
- キャッシュメタデータ論理ボリューム — キャッシュプール論理ボリューム用のメタデータを含む論理ボリューム。データブロックが保存された場所を指定するアカウンティング情報を保持します (たとえば、複製元の論理ボリューム上またはキャッシュデータ論理ボリューム上)。
- キャッシュ論理ボリューム — 作成元の論理ボリュームとキャッシュプール論理ボリュームを含む論理ボリューム。これは、さまざまなキャッシュボリュームコンポーネントをカプセル化する使用可能なデバイスです。

以下の手順に従うと、LVM キャッシュ論理ボリュームを作成できます。

1. 低速な物理ボリュームと高速な物理ボリュームを含むボリュームグループを作成します。この例では、**/dev/sde1** は低速なデバイスであり、**/dev/sdf1** は高速なデバイスです。両方のデバイスはボリュームグループ **VG** に含まれます。

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

2. 作成元のボリュームを作成します。この例では、サイズが 10 ギガバイトであり、低速な物理ボリュームである **/dev/sde1** から構成される **lv** という名前の作成元ボリュームが作成されます。

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

3. キャッシュプール論理ボリュームを作成します。この例では、ボリュームグループ **VG** に含まれる高速なデバイス **/dev/sdf1** 上に **cpool** という名前のキャッシュプール論理ボリュームが作成されます。このコマンドにより作成されるキャッシュプール論理ボリュームは、隠しキャッシュデータ論理ボリューム **cpool\_cdata** と隠しキャッシュメタデータ論理ボリューム **cpool\_cmeta** から構成されます。

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

```
[cpool]          5.00g Cwi---C--- cpool_cdata(0)
[cpool_cdata]    5.00g Cwi-ao---- /dev/sdf1(4)
[cpool_cmeta]    8.00m ewi-ao---- /dev/sdf1(2)
```

さらに複雑な設定の場合は、キャッシュデータとキャッシュメタデータ論理ボリュームを個別に作成し、それらのボリュームをキャッシュプール論理ボリュームに結合する必要があります。この手順については、**lvmdcache(7)** の man ページを参照してください。

4. キャッシュプール論理ボリュームを作成元論理ボリュームにリンクして、キャッシュ論理ボリュームを作成します。作成されたユーザー使用可能なキャッシュ論理ボリュームには、作成元の論理ボリュームの名前が付けられます。作成元の論理ボリュームは、**\_corig** が元の名前に追加された状態で隠し論理ボリュームになります。

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

5. オプションとして、Red Hat Enterprise Linux リリース 7.2 では、キャッシュされた論理ボリュームをシンプル論理ボリュームに変換できます。プールから作成されたすべてのシン論理ボリュームがキャッシュを共有することに注意してください。

以下のコマンドでは、シンプルメタデータ (**lv\_tmeta**) を割り当てるために高速なデバイス **/dev/sdf1** を使用します。これは、キャッシュプールボリュームで使用されるのと同じデバイスです。つまり、シンプルメタデータボリュームはキャッシュデータ論理ボリューム **cpool\_cdata** とキャッシュメタデータ論理ボリューム **cpool\_cmeta** の両方とそのデバイスを共有します。

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

他の管理例を含む LVM キャッシュボリュームの詳細については、**lvmdcache(7)** の man ページを参照してください。

シンプロビジョニングされた論理ボリュームの作成については、「[シンプロビジョニングされた論理ボリュームの作成](#)」を参照してください。

#### 4.4.9. スナップショットボリュームのマージ

**lvconvert** コマンドの **--merge** オプションを使用して、スナップショットを複製元のボリュームにマージすることができます。複製元とスナップショットボリュームの両方が閉じている状態だと、マージはただちに開始します。そうでない場合は、複製元またはスナップショットのいずれかがアクティブになり、かつ両方が閉じられている状態に最初になったときにマージが開始します。root ファイルシステムのように、閉じることができない複製元へのスナップショットのマージは、次に複製元ボリュームがアクティブになるまで行われません。マージが開始すると、マージ後の論理ボリュームには、複製元の名前、マイナー番号、UUID が入ります。マージの進行中、複製元に対する読み取りまたは書き込みは、マージ中のスナップショットに対して実行されているかのように見えます。マージが完了すると、マージされたスナップショットは削除されます。

以下のコマンドは、スナップショットボリューム **vg00/lvol1\_snap** をその複製元にマージします。

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

コマンドライン上で複数のスナップショットを指定したり、LVM オブジェクトタグを使用して複数のスナップショットをそれぞれの複製元にマージしたりすることが可能です。以下の例では、論理ボリューム **vg00/lvol1**、**vg00/lvol2**、および **vg00/lvol3** にはすべて **@some\_tag** タグが付きます。以下のコマンドは、この 3 つのボリュームのスナップショット論理ボリュームを連続的にマージします。マージは **vg00/lvol1**、**vg00/lvol2**、**vg00/lvol3** の順で行われます。**--background** オプションを使用している場合は、すべてのスナップショット論理ボリュームのマージが並行して開始されます。

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

LVM オブジェクトのタグ付けに関する情報は、「[付録D LVM オブジェクトタグ](#)」を参照してください。**lvconvert --merge** コマンドについては、**lvconvert(8)** の man ページをご覧ください。

#### 4.4.10. 永続的なデバイス番号

メジャーデバイス番号とマイナーデバイス番号はモジュールのロード時に動的に割り当てられます。一部のアプリケーションは、ブロックデバイスが常に同じデバイス (メジャーとマイナー) 番号でアクティブ化されている場合に、最も効果的に機能します。これらは **lvcreate** と **lvchange** コマンドで、以下の引数を使用することによって指定できます。

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

別のデバイスにすでに動的に割り当てられている番号を使用しないように、マイナー番号は大きくします。

NFS を使用してファイルシステムをエクスポートする場合は、そのエクスポートファイルで **fsid** パラメーターを指定すると、LVM 内で永続的なデバイス番号を設定する必要がなくなります。

#### 4.4.11. 論理ボリュームの縮小

論理ボリュームのサイズを縮小するには、**lvreduce** コマンドを使用します。論理ボリュームにファイルシステムが含まれる場合は、最初にファイルシステムを縮小して、論理ボリュームのサイズが常に、少なくともファイルシステムが予期するサイズと同じになるようにします。

以下のコマンドは、ボリュームグループ **vg00** の論理ボリューム **lv01** のサイズを、3 つの論理エクステンツに縮小します。

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

#### 4.4.12. 論理ボリュームグループのパラメーター変更

論理ボリュームのパラメーターを変更するには、**lvchange** コマンドを使用します。変更可能なパラメーターの一覧は、**lvchange(8)** の man ページを参照してください。

**lvchange** コマンドを使用して論理ボリュームのアクティブ化と非アクティブ化を実行できます。ボリュームグループ内のすべての論理ボリュームのアクティブ化と非アクティブ化を同時に行うには、「[ボリュームグループのパラメーター変更](#)」で説明されているように **vgchange** コマンドを使用します。

以下のコマンドは、ボリュームグループ **vg00** 内のボリューム **lv01** のパーミッションを読み取り専用に変更します。

```
# lvchange -pr vg00/lv01
```

#### 4.4.13. 論理ボリュームの名前変更

既存の論理ボリューム名を変更するには、**lvrename** コマンドを使用します。

以下のいずれかのコマンドも、ボリュームグループ **vg02** 内の論理ボリューム **lvold** の名前を **lvnew** に変更します。

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

```
# lvrename vg02 lvold lvnew
```

クラスター内の個別ノード上で論理ボリュームをアクティブ化する方法は「[クラスター内の個別ノードでの論理ボリュームのアクティブ化](#)」を参照してください。

#### 4.4.14. 論理ボリュームの削除

非アクティブな論理ボリュームを削除するには、**lvremove** コマンドを使用します。論理ボリュームが現在マウントされている場合は、削除する前にボリュームをアンマウントしてください。また、クラスター環境では削除前に論理ボリュームを非アクティブ化しておく必要があります。

以下のコマンドは、論理ボリューム **/dev/testvg/testlv** をボリュームグループ **testvg** から削除します。このケースでは、論理ボリュームは非アクティブ化されていないことに注意してください。

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

**lvchange -an** コマンドを使用して論理ボリュームを削除する前に、これを明示的に非アクティブ化することができます。この場合、アクティブな論理ボリュームを削除したいかどうかを確認するプロンプトは表示されません。

#### 4.4.15. 論理ボリュームの表示

LVM 論理ボリュームのプロパティを表示するのに使用できるコマンドは、**lvs**、**lvdisplay**、および **lvscan** の 3 つです。

**lvs** コマンドは、論理ボリューム情報を設定可能な形式で提供して、1 つの論理ボリュームにつき 1 行ずつ表示します。**lvs** コマンドでは形式をかなり自由にカスタマイズできるため、スクリプト作成時に役立ちます。**lvs** コマンドで出力をカスタマイズする詳細な方法は「[LVM 用のカスタム報告](#)」を参照してください。

**lvdisplay** コマンドは、決められた形式で、論理ボリュームのプロパティ (サイズ、レイアウト、マッピングなど) を表示します。

以下のコマンドは、**vg00** 内にある **lvol2** の属性を示しています。スナップショット論理ボリュームがこの元の論理ボリューム用に作成されている場合、このコマンドはすべてのスナップショット論理ボリュームとそのステータス (アクティブまたは非アクティブ) の一覧を表示します。

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

**lvscan** コマンドは、システム内のすべての論理ボリュームをスキャンし、以下の例のようにそれらを一覧表示します。

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

#### 4.4.16. 論理ボリュームの拡張

論理ボリュームのサイズを拡張するには、**lvextend** コマンドを使用します。

論理ボリュームを拡張する場合、追加するボリュームの容量、または拡張後のボリュームのサイズを指定することができます。

以下のコマンドは、論理ボリューム **/dev/myvg/homevol** を 12 ギガバイトに拡張します。

```
# 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
```

以下のコマンドは、論理ボリューム **/dev/myvg/homevol** に 1 ギガバイト追加します。

```
# 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
```

**lvcreate** コマンドと同様に、**lvextend** コマンドで **-l** 引数を使用すると、論理ボリュームの拡張サイズをエクステント数で指定することができます。また、この引数を使用してボリュームグループのパーセンテージ、またはボリュームグループに残ってる空き領域をパーセンテージで指定することもできます。以下のコマンドは、**testlv** という論理ボリュームを拡張して、ボリュームグループ **myvg** の空き領域をすべて使用するようにします。

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

論理ボリュームを拡張したら、それに合わせてファイルシステムのサイズも拡張する必要があります。

多くのファイルシステムサイズ変更ツールは、デフォルトで、ファイルシステムのサイズを、その下の論理ボリュームのサイズまで拡大するため、2つのコマンドに同じサイズを指定する必要はありません。

#### 4.4.16.1. ストライプ化ボリュームの拡張

ストライプ化論理ボリュームのサイズを拡大するには、ボリュームグループを構成している物理ボリュームに、ストライプをサポートする十分な空き領域がなければなりません。たとえば、ボリュームグループ全域を使用する2方向ストライプがある場合は、ボリュームグループに物理ボリュームを1つ追加しただけでは、ストライプを拡張することはできません。拡張するには少なくとも2つの物理ボリュームを、ボリュームグループに追加する必要があります。

たとえば、以下の **vgs** コマンドで表示された、2つの物理ボリュームで構成されるボリュームグループ **vg** について考えてみましょう。

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

ボリュームグループの全領域を使用してストライプを作成することができます。

```
# 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)
```

ボリュームグループの空き領域がなくなっていることに注意してください。

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

以下のコマンドで、ボリュームグループに物理ボリュームをもう1つ追加します。これで、135 ギガバイトの領域が追加されます。

```
# 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
```

この時点では、ストライプ化論理ボリュームを、ボリュームグループの最大サイズまで拡大することはできません。データをストライプ化するには、2つの物理デバイスが必要です。

```
# 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
```

ストライプ化論理ボリュームを拡張するには、もう 1 つの物理ボリュームを追加してから、論理ボリュームを拡張します。この例では、ボリュームグループに物理ボリュームを 2 つ追加することにより、ボリュームグループの最大サイズまで、論理ボリュームを拡張できるようになっています。

```
# 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
```

ストライプ化論理ボリュームを拡張するのに十分な物理デバイスがない場合でも、その拡張部分がストライプ化されなくても問題がないならば、ボリュームの拡張は可能です。ただし、これによってパフォーマンスが一定でなくなる可能性があります。論理ボリュームに領域を追加する場合、デフォルトの動作では、既存論理ボリュームの最後のセグメントと同じストライピングパラメーターを使用するようになっていますが、これらのパラメーターはオーバーライドすることができます。以下の例では、初回の **lvextend** コマンドが失敗した後に、既存のストライプ化論理ボリュームを拡張して残りの空き領域を使用するようにしています。

```
# 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
```

#### 4.4.16.2. RAID ボリュームの拡張

RAID 論理ボリュームは、新規の RAID リージョンの同期を実行しなくても、**lvextend** コマンドを使って拡張することができます。

**lvcreate** コマンドで RAID 論理ボリュームを作成する場合は、**--nosync** オプションを指定すると、論理ボリュームを作成する際に RAID リージョンが同期されません。**--nosync** オプションを使って作成した RAID 論理ボリュームを後に拡張する場合も、その時に RAID 拡張が同期されることはありません。

**--nosync** オプションを使用して既存の論理ボリュームが作成されたかどうかを判別するには、**lvs** コマンドを使用してボリュームの属性を表示します。論理ボリュームが、初期同期を行わずに作成された RAID ボリュームの場合は、その論理ボリュームの属性フィールドの先頭に「R」と表示されます。初期同期が行われて論理ボリュームが作成された場合は、「r」と表示されます。

以下のコマンドは、初期同期を行わずに作成した RAID 論理ボリューム **lv** の属性を表示します。属性フィールドの先頭に「R」文字が表示されます。属性フィールドの 7 番目が「r」文字になっています



が、これは RAID 対象のタイプであることを示しています。属性フィールドの意味は、表4.5「lvs 表示フィールド」を参照してください。

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

この論理ボリュームを **lvextend** コマンドで拡張すると、RAID の拡張部分は再同期されません。

**lvcreate** コマンドで、**--nosync** オプションを指定せずに RAID 論理ボリュームを作成した場合、**lvextend** コマンドで **--nosync** オプションを指定すれば、ミラーを再同期することなく論理ボリュームを拡張することができます。

以下の例では、**--nosync** オプションを使わずに作成した RAID 論理ボリュームを拡張し、その論理ボリュームの作成時にそれが同期されたことを示しています。ただし、この例では、ボリュームの拡張時にボリュームが同期されていないことを示しています。ボリュームには「r」の属性が設定されていますが、**lvextend** コマンドに **--nosync** オプションを付けて実行すると、ボリュームには「R」の属性が設定されることに注意してください。

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

RAID ボリュームが非アクティブの場合、ボリュームの拡張時に同期が自動的に省略されることはありません。これは、**--nosync** オプションを指定してボリュームを作成する場合でも当てはまります。その代わりに、論理ボリュームの拡張部分を完全に再同期するかどうかのプロンプトが出されます。



## 注記

RAID ボリュームがリカバリーを実行しているときに、**--nosync** オプションを指定してボリュームの作成/拡張をしていた場合に、論理ボリュームを拡張することはできません。ただし、**--nosync** オプションを指定しないと、リカバリー中に RAID ボリュームを拡張できます。

### 4.4.16.3. cling 割り当てポリシーを使用した論理ボリュームの拡張

LVM ボリュームを拡張する際には、**lvextend** コマンドの **--alloc cling** オプションを使用して、**cling** 割り当てポリシーを指定することができます。このポリシーによって、同一の物理ボリューム上のスペースが、既存の論理ボリュームの最終セグメントとして選択されます。物理ボリューム上に十分な領域がなく、タグの一覧が **lvm.conf** ファイル内で定義されている場合には、LVM は、その物理ボリュームにいずれかのタグが付けられているかを確認し、既存エクステントと新規エクステント間で、物理ボリュームのタグを適合させようとします。

たとえば、ご使用の論理ボリュームが単一のボリュームグループ内の 2 サイト間でミラー化されている場合は、**@site1** タグと **@site2** タグを使用し、サイトの場所に応じて物理ボリュームにタグを付けることができます。この場合は、**lvm.conf** ファイル内に以下の行を指定します。

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

物理ボリュームのタグ付けに関する情報は、「[付録D LVM オブジェクトタグ](#)」を参照してください。

以下の例では、**lvm.conf** ファイルが変更されて、次のような行が追加されています。

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

また、この例で

は、**/dev/sdb1**、**/dev/sdc1**、**/dev/sdd1**、**/dev/sde1**、**/dev/sdf1**、**/dev/sdg1**、および **/dev/sdh1** の物理ボリュームで構成されるボリュームグループ **taft** が作成されています。これらの物理ボリュームには、**A**、**B** および **C** のタグが付けられています。この例では、**C** のタグは使用されていませんが、LVM がタグを使用して、ミラーレグに使用する物理ボリュームを選択することを示しています。

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

以下のコマンドは、ボリュームグループ **taft** から 10 ギガバイトのミラー化ボリュームを作成します。

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

以下のコマンドは、ミラーレグおよび RAID メタデータのサブボリュームに使用されるデバイスを表示します。

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

以下のコマンドは、ミラー化ボリュームのサイズを拡張します。**cling** 割り当てポリシーを使用して、同じタグが付いた物理ボリュームを使用してミラーレグが拡張される必要があることを示します。

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

以下に表示したコマンドは、ログとして同一のタグが付いた物理ボリュームを使用してミラーログが拡張されているのを示しています。**c** のタグが付いた物理ボリュームは無視される点に注意してください。

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

#### 4.4.17. 論理ボリュームの縮小

論理ボリュームのサイズを縮小するには、まずファイルシステムをアンマウントします。次に **lvreduce** コマンドを使用してボリュームを縮小します。ボリュームを縮小したら、ファイルシステムを再マウントします。



#### 警告

ボリューム自体を縮小する前に、ファイルシステムなど、ボリューム内に常駐するものはいずれもサイズを縮小しておくことが重要です。これを行っておかないとデータが喪失する恐れがあります。

論理ボリュームを縮小すると、ボリュームグループの一部が確保されて、そのボリュームグループ内の他の論理ボリュームに割り当てられます。

以下の例では、ボリュームグループ **vg00** 内の論理ボリューム **lv01** のサイズを、3 論理エクステント分縮小しています。

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

#### 4.4.18. 論理ボリュームのアクティブ化の制御

**lvcreate** または **lvchange** コマンドの **-k** または **--setactivationskip {y|n}** オプションを使って、通常のアクティブ化コマンドの実行時にスキップされるよう論理ボリュームにフラグを設定することができます。このフラグは非アクティブ化の実行中には適用されません。

**lvs** コマンドを使って、このフラグが論理ボリュームに設定されているかどうかを判別できます。以下の例にあるように **k** 属性が表示されます。

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

デフォルトでは、シンスナップショットボリュームには、アクティブ化のスキップのためにフラグが設定されます。標準的な **-ay** または **--activate y** オプションに加えて、**-K** または **--ignoreactivationskip** オプションを使用することにより、**k** 属性セットで論理ボリュームをアクティブ化することができます。

以下のコマンドはシンスナップショットの論理ボリュームをアクティブ化します。

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

永続的な「アクティブ化スキップ」フラグは、論理ボリュームの作成時に、**lvcreate** コマンドの **-kn** または **--setactivationskip n** オプションを指定してオフにすることができます。**lvchange** コマンドの **-kn** または **--setactivationskip n** オプションを指定して、既存の論理ボリュームに対するフラグをオフにすることができます。また、**-ky** または **--setactivationskip y** オプションを使って、フラグを再度オンにすることができます。

以下のコマンドは、アクティブ化スキップフラグがない、スナップショット論理ボリュームを作成します。

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

以下のコマンドは、スナップショット論理ボリュームから、アクティブ化スキップフラグを削除します。

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

**/etc/lvm/lvm.conf** ファイルの **auto\_set\_activation\_skip** 設定を使って、デフォルトのアクティブ化スキップ設定を制御することができます。

#### 4.4.19. 過去の論理ボリュームの追跡および表示 (Red Hat Enterprise Linux 7.3 以降)

Red Hat Enterprise Linux 7.3 以降、**lvm.conf** 設定ファイルで **record\_lvs\_history** メタデータオプションを有効にして、削除したシンスナップショットとシン論理ボリュームを追跡するように設定します。これにより、元の依存関係チェーンから削除し、過去の論理ボリュームになった論理ボリュームを含む、完全シンスナップショット依存関係を表示できます。

**lvm.conf** 設定ファイルで **lvs\_history\_retention\_time** メタデータオプションを使用し、保持時間(秒)を指定して、決められた期間、システムに過去のボリュームを保持するように設定できます。

過去の論理ボリュームは、削除された論理ボリュームを単純化したものを保持し、以下の、ボリュームのレポートフィールドを含みます。

- **lv\_time\_removed**: 論理ボリュームの削除時間
- **lv\_time**: 論理ボリュームの作成時間
- **lv\_name**: 論理ボリュームの名前
- **lv\_uuid**: 論理ボリュームの UUID
- **vg\_name**: 論理ボリュームを含むボリュームグループ

ボリュームを削除すると、過去の論理ボリューム名には頭にハイフンが付きます。たとえば、論理ボリューム **lvvol1** を削除すると、過去のボリューム名は **-lvvol1** となります。過去の論理ボリュームは再アクティベートすることができません。

**record\_lvs\_history** メタデータオプションを有効にしても、**lvremove** コマンドの **--nohistory** オプションを指定して論理ボリュームを削除すれば、過去の論理ボリュームを個別に保持しないようにすることができます。

ボリューム表示に過去の論理ボリュームを含むには、LVM 表示コマンドに **-H|--history** オプションを指定します。**-H** オプションとともに、レポートフィールド **lv\_full\_ancestors** および **lv\_full\_descendants** を指定すると、過去のボリュームを含む完全なシンスナップショット依存関係チェーンを表示できます。

以下のコマンド群は、過去の論理ボリュームを表示および管理する例を示します。

1. **lvm.conf** ファイルに **record\_lvs\_history=1** を設定して、過去の論理ボリュームを保持します。このメタデータオプションは、デフォルトでは有効ではありません。
2. 以下のコマンドを実行して、シンプロビジョニングスナップショットチェーンを表示します。

この例では、

- **lvvol1** は元となるボリュームで、チェーンの中で最初のボリュームになります。
- **lvvol2** は、**lvvol1** のスナップショットです。
- **lvvol3** は、**lvvol2** のスナップショットです。
- **lvvol4** は、**lvvol3** のスナップショットです。
- **lvvol5** は、**lvvol3** のスナップショットです。

例の **lvs** 表示コマンドに **-H** オプションを追加しても、シンスナップショットボリュームは削除されていないため、過去の論理ボリュームは表示されません。

```
# lvs -H -o name,full_ancestors,full_descendants
LV      FAncestors          FDescendants
lvvol1                                lvvol2,lvvol3,lvvol4,lvvol5
lvvol2 lvvol1                lvvol3,lvvol4,lvvol5
lvvol3 lvvol2,lvvol1          lvvol4,lvvol5
lvvol4 lvvol3,lvvol2,lvvol1
lvvol5 lvvol3,lvvol2,lvvol1
pool
```

3. スナップショットチェーンから論理ボリューム **lvvol3** を削除してから再度 **lvs** コマンドを実行し、過去の論理ボリュームが、先祖 (ancestor) と子孫 (descendant) とともに、どのように表示されるかを確認します。

```
# lvremove -f vg/lvol3
Logical volume "lvvol3" successfully removed
# lvs -H -o name,full_ancestors,full_descendants
LV      FAncestors          FDescendants
lvvol1                                lvvol2,-lvvol3,lvvol4,lvvol5
lvvol2 lvvol1                -lvvol3,lvvol4,lvvol5
-lvvol3 lvvol2,lvvol1          lvvol4,lvvol5
lvvol4 -lvvol3,lvvol2,lvvol1
lvvol5 -lvvol3,lvvol2,lvvol1
pool
```

4. 過去のボリュームが削除された時間を表示するには、**lv\_time\_removed** レポートフィールドを使用できます。

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

5. **vgname/lvname** フォーマットを以下の例のように指定すると、表示コマンドで過去の論理ボリュームを個別に参照できます。**lv\_attr** フィールドの 5 番目の例を **h** に設定して、ボリュームが過去のボリュームであることを示すように設定されています。

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

6. ボリュームにライブの子孫がないと、LVM は過去の論理ボリュームを保持しません。これは、スナップショットチェーンの最後に論理ボリュームを削除すると、論理ボリュームが過去の論理ボリュームとして保持されないことを示しています。

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

7. 以下のコマンドを実行して、ボリューム **lvol1** および **lvol2** を削除します。次に **lvs** コマンドを実行して、ボリュームが削除されるとどのように表示されるかを確認します。

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

8. 過去の論理ボリュームを完全に削除したら、**lvremove** コマンドを再度実行します。以下の例のように、ハイフンが追加された、過去のボリューム名を指定します。

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

9. 子孫にライブボリュームが含まれるチェーンがある場合に限り、過去の論理ボリュームは保持されます。これは、以下の例のように、そのボリュームにリンクされている子孫がない場合に、過去の論理ボリュームを削除すると、チェーンの論理ボリュームがすべて削除されることを意味します。

```
# lvremove -f vg/lvol4
Automatically removing historical logical volume vg/-lvol1.
Automatically removing historical logical volume vg/-lvol2.
Automatically removing historical logical volume vg/-lvol4.
Logical volume "lvol4" successfully removed
```

## 4.5. フィルターを使用した LVM デバイススキンの制御

起動時に、**vgscan** コマンドが実行して、システム上のブロックデバイスのスキンによる LVM ラベルの確認、物理ボリュームの判別、メタデータの読み取り、およびボリュームグループの一覧の構築が行われます。物理ボリュームの名前は、システム内の各ノードの LVM キャッシュファイル **/etc/lvm/cache/.cache** に保存されます。それ以後のコマンドはそのファイルを読み込み、再スキンを防止することになります。

**lvm.conf** 設定ファイル内にフィルターを設定することにより、LVM がスキンするデバイスを制御することができます。**lvm.conf** ファイル内のフィルターは、一連の簡単な正規表現で構成されており、**/dev** ディレクトリー内のデバイス名に適用され、検出されるそれぞれのブロックデバイスを受理するか、または拒否するかの決定が行われます。

以下の例は、LVM がスキンするデバイスを制御するフィルターの使用を示しています。正規表現は完全なパス名に対して自由に照合されるため、これらの例の一部は必ずしも推奨される方法ではないことに注意してください。たとえば、**a/loop/** は **a/\*.loop.\*** と同等であり、**/dev/solooperation/lvol1** に一致します。

以下のフィルターは、検出されたすべてのデバイスを追加します。これは、設定ファイル内で設定されているフィルターはないため、デフォルトの動作になります。

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

以下のフィルターは、ドライブにメディアが入っていない場合の遅延を回避するために **cdrom** デバイスを削除します。

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

以下のフィルターはすべてのループを追加して、その他のすべてのブロックデバイスを削除します。

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



以下のフィルターはすべてのループと IDE を追加して、その他のすべてのブロックデバイスを削除します。

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

以下のフィルターは 1 番目の IDE ドライブ上にパーティション 8 のみを追加して、他のすべてのブロックデバイスを削除します。

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



#### 注記

**lvmetad** デーモンの実行中は、**/etc/lvm/lvm.conf** ファイルの **filter =** 設定は、**pvscan --cache device** コマンドを実行する場合には適用されません。デバイスをフィルターするには、**global\_filter =** 設定を使用する必要があります。グローバルフィルターに失敗するデバイスは LVM では開かれず、スキャンは一切実行されません。VM で LVM デバイスを使用する場合や、VM 内のデバイスのコンテンツを物理ホストでスキャンする必要がある場合などには、グローバルフィルターを使用する必要があります。

**lvm.conf** ファイルの詳細情報は、「[付録B LVM 設定ファイル](#)」および **lvm.conf(5)** の man ページを参照してください。

## 4.6. オンラインデータ移動

**pvmove** コマンドを使用すると、システムの使用中にデータを移動することができます。

**pvmove** コマンドは、データを分割してセクションに移動して、各セクションを移動する一時的なミラーを作成します。**pvmove** コマンドの操作に関する詳細は、**pvmove(8)** の man ページを参照してください。



#### 注記

クラスター内で **pvmove** 操作を実行するためには、**cmirror** パッケージがインストールされており、**cmirrord** サービスが実行中である必要があります。

以下のコマンドは、物理ボリューム **/dev/sdc1** から、ボリュームグループ内の他の空き物理ボリュームに、すべての割り当て領域を移動します。

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

以下のコマンドは、論理ボリューム **MyLV** のエクステントのみを移動します。

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

**pvmove** コマンドの実行には時間がかかるため、バックグラウンドでコマンドを実行して、フォアグラウンドでの進捗状況の表示を回避した方が良いでしょう。以下のコマンドは、物理ボリューム **/dev/sdc1** に割り当てられているすべてのエクステントを、バックグラウンドで **/dev/sdf1** に移動します。

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

■

以下のコマンドは、5 秒間ごとに **pvmove** コマンドの進行状況をパーセンテージで報告します。

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

## 4.7. クラスター内の個別ノードでの論理ボリュームのアクティブ化

クラスター環境に LVM をインストールしている場合は、単一のノード上で論理ボリュームを排他的にアクティブ化しなければならない場合があります。

論理ボリュームを単一のノード上で排他的にアクティブ化するには、**lvchange -aey** コマンドを使用します。または、**lvchange -aly** コマンドを使用して、論理ボリュームを排他的ではなくローカルノード上のみでアクティブ化することもできます。これらの論理ボリュームは、後で追加のノード上で同時にアクティブ化することができます。

また、「[付録D LVM オブジェクトタグ](#)」に説明されているように、LVM タグを使用した個別ノード上での論理ボリュームのアクティブ化も可能です。さらに、設定ファイル内でノードのアクティブ化を指定することもできます。これについては、「[付録B LVM 設定ファイル](#)」で説明されています。

## 4.8. LVM 用のカスタム報告

LVM では、カスタマイズされたレポートを生成したり、レポートの出力をフィルタリングしたりするためのさまざまな設定およびコマンドラインオプションが提供されます。LVM レポート機能の完全な説明については、**lvreport(7)** の man ページを参照してください。

**pvs**、**lvs**、および **vgs** コマンドを使用して、LVM オブジェクトについての簡潔でカスタマイズ可能なレポートを作成することができます。これらのコマンドが生成するレポートには、オブジェクトごとに 1 行の出力が含まれます。各行には、オブジェクトに関連するプロパティのフィールドの順序付けられた一覧が含まれます。レポートするオブジェクトを選択する方法には、物理ボリューム、ボリュームグループ、論理ボリューム、物理ボリュームセグメント、および論理ボリュームセグメント別の 5 つの方法があります。

Red Hat Enterprise Linux 7.3 リリースでは、**lv fullreport** コマンドを使用して、物理ボリューム、ボリュームグループ、論理ボリューム、物理ボリュームセグメント、および論理ボリュームセグメントに関する情報を一度に報告できます。このコマンドとその機能については、**lv-fullreport(8)** の man ページを参照してください。

Red Hat Enterprise Linux 7.3 リリースでは、LVM は、LVM コマンドの実行中に収集された操作、メッセージ、およびオブジェクトごとのステータス (完全なオブジェクト ID 付き) が含まれるメッセージログレポートをサポートします。LVM ログレポートの例は、「[コマンドログレポート \(Red Hat Enterprise Linux 7.3 以降\)](#)」を参照してください。LVM ログレポートの詳細は、**lvreport(7)** の man ページを参照してください。

以下のセクションでは、**pvs**、**lvs**、および **vgs** コマンドを使用したレポートのカスタマイズに関する概要情報を提供しています。

- 「[形式の制御](#)」。レポートの書式を制御するために使用できるコマンド引数の概要を提供します。
- 「[オブジェクト表示フィールド](#)」。各 LVM オブジェクトに対して表示できるフィールドの一覧を提供します。
- 「[LVM 報告のソート](#)」。生成されたレポートをソートするために使用できるコマンド引数の概要を提供します。

- 「[ユニットの指定](#)」。レポート出力の単位を指定する手順を提供します。
- 「[JSON 形式の出力 \(Red Hat Enterprise Linux 7.3 以降\)](#)」。JSON 形式の出力を指定する例を提供します (Red Hat Enterprise Linux 7.3 以降)。
- 「[コマンドログレポート \(Red Hat Enterprise Linux 7.3 以降\)](#)」。コマンドログの例を提供します。

#### 4.8.1. 形式の制御

**pvs**、**lvs**、または **vgs** コマンドのどれを使用するかによって、表示されるデフォルトのフィールドセットとソート順序が決定されます。これらのコマンドの出力は、以下の引数を使用することによって制御できます。

- **-o** 引数を使用すると、表示するフィールドをデフォルト以外に変更することができます。たとえば、以下の出力は **pvs** コマンドのデフォルト表示です (物理ボリュームに関する情報を表示)。

```
# 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
```

以下のコマンドは、物理ボリュームの名前とサイズだけを表示します。

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

- **-o** 引数との組み合わせで使用するプラス記号 (+) を使って、出力にフィールドを追加することができます。

以下の例は、デフォルトフィールドに加えて、物理ボリュームの UUID を表示しています。

```
# 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
```

- コマンドに **-v** 引数を追加すると、追加のフィールドが含まれます。たとえば、**pvs -v** コマンドは、デフォルトフィールドに加えて、**DevSize** と **PV UUID** のフィールドも表示します。

```
# 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** 引数は、見出し行を表示しません。これはスクリプトを作成する際に便利です。

以下の例は、**pv\_name** 引数と共に **--noheadings** 引数を使用して、すべての物理ボリュームの一覧を生成しています。

```

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

```

- **--separator separator** 引数は、*separator* を使用して各フィールドを分離します。

次の例は、**pvs** コマンドのデフォルト出力フィールドを等号 (=) で分割しています。

```

# 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

```

**separator** 引数の使用時にフィールドを配置するには、**--aligned** 引数と共に **separator** 引数を使用します。

```

# 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

```

**lvs** または **vgs** コマンドに **-P** 引数を追加して、通常の出力では表示されない、障害が発生したボリュームに関する情報を表示することができます。この引数を使用した場合に出力される情報については、「[障害の発生したデバイスの情報を表示](#)」をご覧ください。

表示に関する引数の詳細は、**pvs(8)**、**vgs(8)**、および **lvs(8)** の man ページを参照してください。

ボリュームグループフィールドは、物理ボリューム (および物理ボリュームセグメント) フィールド、または論理ボリューム (および論理ボリュームセグメント) フィールドと混在させることができますが、物理ボリュームフィールドと論理ボリュームフィールドは混在させることはできません。たとえば、以下のコマンドは、1 つの物理ボリュームにつき 1 行の出力を表示します。

```

# 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. オブジェクト表示フィールド

このセクションでは、**pvs**、**vgs**、および **lvs** コマンドを使って LVM オブジェクトについて表示できる情報を一覧表示する一連の表を提供します。

便宜上、フィールド名の接頭辞は、コマンドのデフォルトと一致する場合は省略できます。たとえば、**pvs** コマンドでは、**name** は **pv\_name**、**vgs** コマンドでは、**name** は **vg\_name** と解釈されます。

以下のコマンドの実行は、**pvs -o pv\_free** の実行に相当します。

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

### 注記

**pvs**、**vgs**、および **lvs** 出力の属性フィールドにある文字数は、以降のリリースで増える可能性があります。既存の文字フィールドの位置は変更しませんが、新しいフィールドが末尾に追加される可能性があります。相対的な位置を使って特定の属性文字を検索するスクリプトを作成する場合は、このことを考慮して、フィールドの終点ではなく、フィールドの始点を基点として文字検索を行います。たとえば、**lv\_attr** フィールドの 9 番目のビットの文字 **p** を検索する場合、文字列 **"^/.....p/"** を指定することはできませんが、文字列 **"/\*p\$/"** は使用しないでください。

## pvs コマンド

表4.3「**pvs コマンド表示フィールド**」は、**pvs** コマンドの表示引数、ヘッダーに表示されるフィールド名、フィールドの説明を一覧にまとめています。

表4.3 **pvs** コマンド表示フィールド

引数	ヘッダー	説明
<b>dev_size</b>	DevSize	物理ボリュームを作成する基となる配下のデバイスのサイズ
<b>pe_start</b>	1st PE	配下のデバイス内の最初の物理エクステンツの開始点までのオフセット
<b>pv_attr</b>	Attr	物理ボリュームのステータス: (a)llocatable または e(x)ported
<b>pv_fmt</b>	Fmt	物理ボリュームのメタデータ形式 ( <b>lvm2</b> または <b>lvm1</b> )
<b>pv_free</b>	PFree	物理ボリュームにある残りの空き領域
<b>pv_name</b>	PV	物理ボリュームの名前
<b>pv_pe_alloc_count</b>	Alloc	使用される物理エクステンツの数
<b>pv_pe_count</b>	PE	物理エクステンツの数

引数	ヘッダー	説明
<b>pvseg_size</b>	SSize	物理ボリュームのセグメントサイズ
<b>pvseg_start</b>	Start	物理ボリュームセグメントの最初の物理エクステンツ
<b>pv_size</b>	PSize	物理ボリュームのサイズ
<b>pv_tags</b>	PV Tags	物理ボリュームに割り当てられた LVM タグ
<b>pv_used</b>	Used	物理ボリューム上で現在使用中の領域の量
<b>pv_uuid</b>	PV UUID	物理ボリュームの UUID

デフォルトで **pvs** コマンドが表示するフィールド

は、**pv\_name**、**vg\_name**、**pv\_fmt**、**pv\_attr**、**pv\_size**、**pv\_free** です。表示は **pv\_name** でソートされます。

```
# 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
```

**pvs** コマンドに **-v** 引数を使用すると、デフォルトの表示に **dev\_size** および **pv\_uuid** のフィールドが追加されます。

```
# 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
```

**pvs** コマンドに **--segments** 引数を使用すると、各物理ボリュームセグメントの情報を表示します。セグメントはエクステンツの集合です。セグメントの表示は、論理ボリュームがフラグメント化 (断片化) しているかどうかを確認するのに役立ちます。

デフォルトで **pvs --segments** コマンドが表示するフィールド

は、**pv\_name**、**vg\_name**、**pv\_fmt**、**pv\_attr**、**pv\_size**、**pv\_free**、**pvseg\_start**、**pvseg\_size** です。この表示は、物理ボリューム内で **pv\_name** と **pvseg\_size** でソートされます。

```
# 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
```

**pvs -a** コマンドを使用して、LVM が検出した、LVM 物理ボリュームとして初期化していないデバイスを確認できます。

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

vgs コマンド

表4.4 「vgs 表示フィールド」 は、**vgs** コマンドの表示引数、ヘッダーに表示されるフィールド名、およびフィールドの説明を一覧にまとめています。

表4.4 vgs 表示フィールド

引数	ヘッダー	説明
lv_count	#LV	ボリュームグループに含まれる論理ボリュームの数
max_lv	MaxLV	ボリュームグループ内で許容される論理ボリュームの最大数 (無制限の場合は 0)
max_pv	MaxPV	ボリュームグループ内で許容される物理ボリュームの最大数 (無制限の場合は 0)



引数	ヘッダー	説明
<b>pv_count</b>	#PV	ボリュームグループを定義する物理ボリューム数
<b>snap_count</b>	#SN	ボリュームグループに含まれるスナップショット数
<b>vg_attr</b>	Attr	ボリュームグループのステータス: (w)riteable、(r)eadonly、resi(z)eable、e(x)ported、(p)artial、および (c)lustered
<b>vg_extent_count</b>	#Ext	ボリュームグループ内の物理エクステントの数
<b>vg_extent_size</b>	Ext	ボリュームグループ内の物理エクステントのサイズ
<b>vg_fmt</b>	Fmt	ボリュームグループのメタデータ形式 ( <b>lvm2</b> または <b>lvm1</b> )
<b>vg_free</b>	VFree	ボリュームグループ内の残りの空き領域のサイズ
<b>vg_free_count</b>	Free	ボリュームグループ内の空き物理エクステントの数
<b>vg_name</b>	VG	ボリュームグループ名
<b>vg_seqno</b>	Seq	ボリュームグループの改訂を示す番号
<b>vg_size</b>	VSize	ボリュームグループのサイズ
<b>vg_sysid</b>	SYS ID	LVM1 システム ID
<b>vg_tags</b>	VG Tags	ボリュームグループに割り当てられた LVM タグ
<b>vg_uuid</b>	VG UUID	ボリュームグループの UUID

デフォルトで **vgs** コマンドが表示するフィールド

は、**vg\_name**、**pv\_count**、**lv\_count**、**snap\_count**、**vg\_attr**、**vg\_size**、**vg\_free** です。表示は **vg\_name** でソートされます。

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

**vgs** コマンドに **-v** 引数を使用すると、デフォルトの表示に **vg\_extent\_size** および **vg\_uuid** のフィールドが追加されます。

```
# 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
```

## lvs コマンド

表4.5「lvs 表示フィールド」は、**lvs** コマンドの表示引数、ヘッダーに表示されるフィールド名、およびフィールドの説明を一覧にまとめています。

表4.5 lvs 表示フィールド

引数	ヘッダー	説明
<div>chunksize</div> <div>chunk_size</div>	Chunk	スナップショットボリュームのユニットサイズ
copy_percent	Copy%	ミラー化論理ボリュームの同期のパーセンテージ。さらに <b>pv_move</b> コマンドで物理エクステントを移動する時にも使用されます。
devices	Devices	論理ボリュームを構成する配下のデバイス: 物理ボリューム、論理ボリューム、および物理エクステントと論理エクステントの開始点
lv_ancestors	Ancestors	(Red Hat Enterprise Linux 7.2 以降) シンプルスナップショットにおける、論理ボリュームの先祖 (ancestor)
lv_descendants	Descendants	(Red Hat Enterprise Linux 7.2 以降) シンプルスナップショットにおける、論理ボリュームの子孫 (descendant)
lv_attr	Attr	<p>論理ボリュームのステータス。論理ボリュームの属性ビットは以下ようになります。</p> <div> <p>ビット 1: ボリュームタイプ: (m)irrored (ミラー化)、(M)irrored without initial sync (初期同期なしのミラー化)、(o)rigin (複製元)、(O)rigin with merging snapshot (マージするスナップショットがある複製元)、(r)aid (RAID)、(R)aid without initial sync (初期同期なしの RAID)、(s)napshot (スナップショット)、merging (S)napshot (マージするスナップショット)、(p)vmove (物理ボリュームの移動)、(v)irtual (仮想)、mirror or raid (i)mage (ミラーまたは RAID イメージ)、mirror or raid (l)mage out-of-sync (ミラーまたは RAID イメージの非同期)、mirror (l)og device (ミラーログデバイス)、under (c)onversion (変換中)、thin (V)olume (シンボリューム)、(t)hin pool (シンプール)、(T)hin pool data (シンプールデータ)、raid or thin pool m(e)tadata or pool metadata spare (RAID またはシンプールメタデータまたはプールメタデータのスペア)</p> <p>ビット 2: パーミッション: (w)riteable (書き込み可能)、(r)ead-only (読み取り専用)、(R)ead-only activation of non-read-only volume (読み取り専用でないボリュームを読み取り専用にアクティブ化)</p> </div>

引数	ヘッダー	説明
		<p>ビット 3: 割り当てポリシー: (a)nywhere、(c)ontiguous、(i)nherited、(c)liling、(n)ormal。これは、たとえば <b>pvmove</b> コマンドの実行時など、割り当ての変更に対してボリュームが現在ロックされている場合に大文字になります。</p> <p>ビット 4: 固定されたマイナー番号</p> <p>ビット 5: 状態: (a)ctive (アクティブ)、(s)uspended (サスペンド)、(l)invalid snapshot (無効なスナップショット)、invalid (S)uspended snapshot (無効なサスペンドされたスナップショット)、snapshot (m)erge failed (スナップショットのマージが失敗)、suspended snapshot (M)erge failed (サスペンドされたスナップショットのマージが失敗)、mapped (d)evice present without tables (テーブルのないマッピングされたデバイス)、mapped device present with (i)nactive table (非アクティブのテーブルを持つマッピングされたデバイス)</p> <p>ビット 6: デバイス開放(o)</p> <p>ビット 7: ターゲットタイプ: (m)irror (ミラー)、(r)aid (RAID)、(s)napshot (スナップショット)、(t)hin (シン)、(u)nknown (不明)、(v)irtual (仮想)。これは、同じターゲットのカーネルに関連する論理ボリュームをグループにまとめます。たとえば、ミラーイメージ、ミラーログ、ミラー自体が元のデバイスマッパーのミラーカーネルドライバを使用する場合、それらは (m) と表示されます。md raid カーネルドライバを使用する同等の raid はすべて (r) と表示されます。元のデバイスマッパードライバを使用するスナップショットは (s) と表示され、シンプロビジョニングドライバを使用するシンボリュームのスナップショットは (t) と表示されます。</p> <p>ビット 8: 新しく割り当てられたデータブロックは使用前にゼロ(z) のブロックで上書きされます。</p> <p>ビット 9: ボリュームの正常性: (p)artial (部分的)、(r)efresh needed (更新が必要)、(m)ismatches exist (不一致が存在)、(w)ritemostly (書き込み多発)。部分的 (p) は、この論理ボリュームが使用する 1 つ以上の物理ボリュームがシステムから欠落していることを表します。更新 (r) は、この RAID 論理ボリュームが使用する 1 つ以上の物理ボリュームが書き込みエラーが生じたことを表します。書き込みエラーは、その物理ボリュームの一時的な障害によって引き起こされたか、または物理ボリュームに障害があることを示すかのいずれかの可能性があります。デバイスを更新するか、または置き換える必要があります。不一致 (m) は、RAID 論理ボリュームのアレイに一貫していない部分があることを表します。不整合は、RAID 論理ボリューム上で <b>check</b> 操作を開始することによって検出されます。(スクラビング操作 <b>check</b> および <b>repair</b> は、<b>lvchange</b> コマンドによって RAID 論理ボリューム上で実行できます。) 書き込み多発 (w) は write-mostly とマークが付けられた RAID 1 論理ボリュームのデバイスを表します。</p>

引数	ヘッダー	説明
		ット 10: s(k)ip activation (アクティブ化のスキップ): このボリュームには、アクティブ化の実行時にスキップされるようにフラグが設定されます。
<b>lv_kernel_major</b>	KMaj	論理ボリュームの実際のメジャーデバイス番号 (非アクティブの場合は -1)
<b>lv_kernel_minor</b>	KMIN	論理ボリュームの実際のマイナーデバイス番号 (非アクティブの場合は -1)
<b>lv_major</b>	Maj	論理ボリュームの永続的なメジャーデバイス番号 (未指定の場合は -1)
<b>lv_minor</b>	Min	論理ボリュームの永続的なマイナーデバイス番号 (未指定の場合は -1)
<b>lv_name</b>	LV	論理ボリュームの名前
<b>lv_size</b>	LSize	論理ボリュームのサイズ
<b>lv_tags</b>	LV Tags	論理ボリュームに割り当てられた LVM タグ
<b>lv_uuid</b>	LV UUID	論理ボリュームの UUID
<b>mirror_log</b>	Log	ミラーログが存在するデバイス
<b>modules</b>	Modules	この論理ボリュームを使用するのに必要な対応するカーネルデバイスマッパーターゲット
<b>move_pv</b>	Move	<b>pvmove</b> コマンドで作成された一時的な論理ボリュームの元となる物理ボリューム
<b>origin</b>	Origin	スナップショットボリュームの複製元のデバイス
<div>regionsize</div> <div>region_size</div>	Region	ミラー化論理ボリュームのユニットサイズ
<b>seg_count</b>	#Seg	論理ボリューム内のセグメント数
<b>seg_size</b>	SSize	論理ボリューム内のセグメントサイズ
<b>seg_start</b>	Start	論理ボリューム内のセグメントのオフセット
<b>seg_tags</b>	Seg Tags	論理ボリュームのセグメントに割り当てられた LVM タグ

引数	ヘッダー	説明
<b>segtype</b>	Type	論理ボリュームのセグメントタイプ (例: ミラー、ストライプ、リニア)
<b>snap_percent</b>	Snap%	使用中スナップショットボリュームの現在のパーセンテージ
<b>stripes</b>	#Str	論理ボリューム内のストライプ、またはミラーの数
<div> <b>stripesize</b> </div> <div> <b>stripe_size</b> </div>	Stripe	ストライプ化論理ボリューム内のストライプのユニットサイズ

デフォルトで **lvs** コマンドが表示するフィールド

は、**lv\_name**、**vg\_name**、**lv\_attr**、**lv\_size**、**origin**、**snap\_percent**、**move\_pv**、**mirror\_log**、**copy\_percent**、**convert\_lv** です。デフォルトの表示は、ボリュームグループ内で **vg\_name** と **lv\_name** でソートされます。

```
# lvs
LV          VG      Attr   LSize  Origin Snap%  Move Log Copy%  Convert
lv010       new_vg  owi-a- 52.00M
newvgssnap1 new_vg  swi-a- 8.00M lv010    0.20
```

**lvs** コマンドで **-v** 引数を使用すると、デフォルトの表示に

**seg\_count**、**lv\_major**、**lv\_minor**、**lv\_kernel\_major**、**lv\_kernel\_minor**、**lv\_uuid** のフィールドが追加されます。

```
# lvs -v
Finding all logical volumes
LV          VG      #Seg Attr   LSize  Maj Min KMaj KMin Origin Snap%
Move Copy%  Log Convert LV UUID
lv010       new_vg    1 owi-a- 52.00M  -1  -1 253  3
LBBy1Tz-sr23-0jsI-LT03-nHLC-y8XW-EhC178
newvgssnap1 new_vg    1 swi-a- 8.00M  -1  -1 253  5   lv010    0.20
1ye10U-1cIu-o79k-20h2-ZGF0-qCJm-CfbsIx
```

**lvs** コマンドで **--segments** 引数を使用すると、セグメント情報を強調したデフォルトのコラムで情報を表示します。**segments** 引数を使用すると、**seg** 接頭辞はオプションとなります。デフォルトで **lvs --segments** コマンドが表示するフィールド

は、**lv\_name**、**vg\_name**、**lv\_attr**、**stripes**、**segtype**、**seg\_size** です。デフォルトの表示は、ボリュームグループ内の **vg\_name** と **lv\_name** でソートされ、論理ボリューム内では **seg\_start** でソートされます。論理ボリュームがフラグメント化されている場合、このコマンドの出力は以下を表示します。

```
# lvs --segments
LV          VG          Attr   #Str Type   SSize
LogVol100  VolGroup00 -wi-ao    1 linear 36.62G
LogVol101  VolGroup00 -wi-ao    1 linear512.00M
lv          vg          -wi-a-    1 linear104.00M
lv          vg          -wi-a-    1 linear104.00M
```

```
lv      vg      -wi-a-    1 linear 104.00M
lv      vg      -wi-a-    1 linear  88.00M
```

**lvs --segments** コマンドで **-v** 引数を使用すると、デフォルトの表示に **seg\_start**、**stripesize**、**chunksize** のフィールドが追加されます。

```
# 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
```

以下の例は、1つの設定された論理ボリュームを持つシステム上での **lvs** コマンドのデフォルト出力を示しています。その後に、**segments** 引数を指定した **lvs** コマンドのデフォルト出力を表示しています。

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

### 4.8.3. LVM 報告のソート

通常、**lvs**、**vgs**、または **pvs** のコマンドの出力全体をソートして、コラムを正しく配置するには、まずそれを生成して内部に保管する必要があります。**--unbuffered** 引数を指定すると、生成直後にソートされていないままの出力で表示することができます。

別の順列のコラム一覧のソートを指定するには、報告コマンドのいずれかと一緒に **-o** 引数を使用します。出力自体の中にこれらのフィールドを含める必要はありません。

以下の例は、物理ボリュームの名前、サイズ、および空き領域を表示する **pvs** コマンドの出力を示しています。

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

以下の例では、空き領域のフィールドでソートされた同じ出力を示しています。

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

以下の例では、ソートするフィールドを表示する必要がないことを示しています。

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

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

逆順でソートするには、**-o** 引数の後で指定するフィールドの先頭に **-** 印を付けます。

```
# 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. ユニットの指定

LVM 報告表示用の単位を指定するには、報告コマンドに **--units** 引数を使用します。バイト(b)、キロバイト(k)、メガバイト(m)、ギガバイト(g)、テラバイト(t)、エクサバイト(e)、ペタバイト(p)、および人間が読める表示(h) を指定できます。デフォルトは人間が読める表示です。このデフォルト設定を上書きするには、**lvm.conf** ファイルの **global** セクション内の **units** パラメーターを設定します。

以下の例は、**pvs** コマンドの出力をデフォルトのギガバイトでなく、メガバイトで指定しています。

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

デフォルトでは、単位は 2 の累乗 (1024 の倍数) で表示されます。単位を 1000 の倍数で表示するには、大文字 (B、K、M、G、T、H) で単位を指定することができます。

以下のコマンドは、デフォルト動作である 1024 の倍数として出力を表示します。

```
# 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
```

以下のコマンドは 1000 の倍数として出力を表示します。

```
# 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
```

セクター (512 バイトとして定義) またはカスタム単位も指定できます。

以下の例は、**pvs** コマンドの出力をセクター数として表示します。

```
# 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
```

以下の例は、**pvs** コマンドの出力を 4 MB 単位で表示しています。

```
# 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
```

#### 4.8.5. JSON 形式の出力 (Red Hat Enterprise Linux 7.3 以降)

Red Hat Enterprise Linux 7.3 では、LVM 表示コマンドで、**--reportformat** オプションを使用して JSON 形式で出力を表示できます。

以下の例は、標準的なデフォルト形式の **lvs** の出力を示しています。

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

以下のコマンドは、JSON 形式を指定する場合と同じ LVM 設定の出力を表示します。

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

また、**/etc/lvm/lvm.conf** ファイルで **output\_format** 設定を使用して、レポート形式を設定オプションとして設定することもできます。ただし、コマンドラインの **--reportformat** 設定はこの設定よりも優先されます。

#### 4.8.6. コマンドログレポート (Red Hat Enterprise Linux 7.3 以降)

Red Hat Enterprise Linux 7.3 では、レポート指向および処理指向の LVM コマンドを使用して、コマンドログを報告できます (これが **log/report\_command\_log** 設定で有効になっている場合)。このレポートで表示およびソートするフィールドセットを決定できます。

以下の例では、LVM コマンド向けの完全なログレポートを生成するよう LVM を設定します。この例では、論理ボリューム **lv010** と **lv011** の両方が、それらの論理ボリュームを含むボリュームグループ **VG** とともに正常に処理されたことを確認できます。

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

# lvs
Logical Volume
=====
LV      LSize Cpy%Sync
lv011  4.00m 100.00
lv010  4.00m

Command Log
=====
Seq LogType Context      ObjType ObjName ObjGrp  Msg      Errno RetCode
  1 status  processing lv      lv010  vg      success    0        1
  2 status  processing lv      lv011  vg      success    0        1
  3 status  processing vg      vg      success    0        1

# lvchange -an vg/lv011
Command Log
=====
Seq LogType Context      ObjType ObjName ObjGrp  Msg      Errno RetCode
  1 status  processing lv      lv011  vg      success    0        1
  2 status  processing vg      vg      success    0        1
```

LVM レポートおよびコマンドログの設定の詳細については、**lvmreport** の man ページを参照してください。

## 第5章 LVM 設定の例

この章では、基本的な LVM 設定の例をいくつか紹介します。

### 5.1. 3つのディスク上での LVM 論理ボリューム作成

この手順例では、`/dev/sda1`、`/dev/sdb1`、および `/dev/sdc1` のディスクで構成される `new_logical_volume` という名前の LVM 論理ボリュームを作成します。

1. ボリュームグループのディスクを使用するには、`pvcreate` コマンドで、そのディスクに LVM 物理ボリュームラベルを付けます。



#### 警告

このコマンドは、`/dev/sda1`、`/dev/sdb1`、および `/dev/sdc1` にあるデータを破棄します。

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

2. 作成した LVM 物理ボリュームで構成されるボリュームグループを作成します。以下のコマンドを使用すると、ボリュームグループ `new_vol_group` が作成されます。

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

`vgs` コマンドを使用すると、作成したボリュームグループの属性を表示できます。

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

3. 作成したボリュームグループから論理ボリュームを作成します。以下のコマンドを使用して、ボリュームグループ `new_vol_group` から、論理ボリューム `new_logical_volume` を作成します。この例では、ボリュームグループの 2 ギガバイトを使用する論理ボリュームが作成されます。

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

4. 論理ボリュームにファイルシステムを作成します。以下のコマンドを使用すると、論理ボリュームに GFS2 ファイルシステムが作成されます。

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

```

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

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

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

Syncing...
All Done

```

以下のコマンドは、論理ボリュームをマウントして、ファイルシステムディスクの領域使用率を報告します。

```

# mount /dev/new_vol_group/new_logical_volume /mnt
# df
Filesystem          1K-blocks      Used Available Use% Mounted on
/dev/new_vol_group/new_logical_volume
                    1965840         20   1965820   1% /mnt

```

## 5.2. ストライプ化論理ボリュームの作成

以下の手順例では、**/dev/sda1**、**/dev/sdb1**、および **/dev/sdc1** のディスクにデータをストライプ化する、**striped\_logical\_volume** という名前の LVM ストライプ化論理ボリュームを作成します。

1. **pvcreate** コマンドを使用し、LVM 物理ボリュームとして使用するディスクにラベルを付けます。



### 警告

このコマンドは、**/dev/sda1**、**/dev/sdb1**、および **/dev/sdc1** にあるデータを破棄します。

```

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

```

2. ボリュームグループ **volgroup01** を作成します。以下のコマンドを使用すると、ボリュームグループ **volgroup01** が作成されます。

```

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

```

**vgs** コマンドを使用すると、作成したボリュームグループの属性を表示できます。

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

- 作成したボリュームグループから、ストライプ化論理ボリュームを作成します。以下のコマンドを使用すると、ボリュームグループ **volgroup01** から、ストライプ化論理ボリューム **striped\_logical\_volume** が作成されます。この例では、2 ギガバイトサイズで、ストライプサイズが 4 キロバイトのストライプが 3 つある、論理ボリュームが作成されます。

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

- ストライプ化論理ボリュームでファイルシステムを作成します。以下のコマンドを使用すると、論理ボリュームに GFS2 ファイルシステムが作成されます。

```
# mkfs.gfs2 -p lock_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
```

以下のコマンドは、論理ボリュームをマウントして、ファイルシステムディスクの領域使用率を報告します。

```
# mount /dev/volgroup01/striped_logical_volume /mnt
# 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. ボリュームグループの分割

この手順例では、3つの物理ボリュームから構成されるボリュームグループを使用します。この物理ボリュームに未使用領域が十分にあれば、新たにディスクを追加しなくてもボリュームグループを作成できます。

はじめに、ボリュームグループ **myvol** から論理ボリューム **mylv** を作成します。そのボリュームグループは、**/dev/sda1**、**/dev/sdb1**、および **/dev/sdc1** の3つの物理ボリュームで構成されます。

その後、**/dev/sda1** と **/dev/sdb1** を使用したボリュームグループ **myvg** と、**/dev/sdc1** を使用したボリュームグループ **yourvg** を使用します。

1. **pvscan** コマンドを使用すると、現在ボリュームグループで利用可能な空き領域の容量を確認できます。

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

2. **pvmove** コマンドを使用して、**/dev/sdc1** で使用中の物理エクステンツをすべて **/dev/sdb1** に移動することができます。**pvmove** コマンドの実行には時間がかかる場合があります。

```
# 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%
```

データを移動したら、**/dev/sdc1** 上のすべての領域が空き領域になっていることが確認できます。

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

3. 新規ボリュームグループ **yourvg** を作成するために、**vgsplit** コマンドを使用して、ボリュームグループ **myvg** を分割します。

ボリュームグループを分割するには、論理ボリュームが非アクティブな状態である必要があります。ファイルシステムがマウントされている場合は、論理ボリュームを非アクティブ化する前に、そのファイルシステムをアンマウントする必要があります。

論理ボリュームを非アクティブ化するには、**lvchange** コマンドまたは **vgchange** コマンドを使用します。以下のコマンドを実行すると、論理ボリューム **mylv** が非アクティブ化され、ボリュームグループ **myvg** からボリュームグループ **yourvg** が分割され、物理ボリューム **/dev/sdc1** が新規のボリュームグループ **yourvg** に移動します。

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

**vgs** を使用すると、2つのボリュームグループの属性を確認できます。

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

4. ボリュームグループを新しく作成したら、新規の論理ボリューム **yourlv** を作成します。

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

5. 新規の論理ボリュームにファイルシステムを作成し、その論理ボリュームをマウントします。

```
# mkfs.gfs2 -p lock_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

# mount /dev/yourvg/yourlv /mnt
```

6. 前の手順で、論理ボリューム **mylv** を非アクティブ化しました。したがって、この論理ボリュームをマウントする前に、再度アクティブ化する必要があります。

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

# mount /dev/myvg/mylv /mnt
# 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. 論理ボリュームからディスクを削除する

この手順例では、ディスクを交換するか、または別のボリュームで使用するために、既存の論理ボリュームからディスクを削除する方法を示しています。ディスクを削除する前に、LVM 物理ボリューム上のエクステントを、別のディスクまたはディスクセットに移動する必要があります。



### 5.4.1. 既存の物理ボリュームへのエクステントの移動

この例では、論理ボリュームが、ボリュームグループ **myvg** の 4 つの物理ボリュームに分配されています。

```
# 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
```

この例では、物理ボリューム **/dev/sdb1** からエクステントを移動して、この物理ボリュームをボリュームグループから削除できるようにします。

1. 同じボリュームグループの物理ボリュームに空きエクステントが十分にある場合は、削除したいデバイスに対して (他のオプションを指定せずに) **pvmove** コマンドを実行すると、そのエクステントは他のデバイスに分配されます。

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

**pvmove** コマンドの実行が終了すると、エクステントの分配は次のようになります。

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

2. **vgreduce** コマンドを使用して、ボリュームグループから物理ボリューム **/dev/sdb1** を削除することができます。

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

これでディスクは物理的に削除できるようになり、他のユーザーに割り当てることも可能になります。

### 5.4.2. 新規ディスクへのエクステントの移動

この例では、以下のように、ボリュームグループ **myvg** の 3 つの物理ボリュームに、論理ボリュームが分配されています。

-

```
# 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
```

以下の手順では、**/dev/sdb1** のエクステンツを、新しいデバイス **/dev/sdd1** に移動します。

1. **/dev/sdd1** から、物理ボリュームを作成します。

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

2. 新たに作成した物理ボリューム **/dev/sdd1** を、既存のボリュームグループ **myvg** に追加します。

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

3. **pvmove** を使用して、データを **/dev/sdb1** から **/dev/sdd1** へ移動します。

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

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

4. データを **/dev/sdb1** から移動したら、この物理ボリュームをボリュームグループから削除できます。

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

これで、このディスクを別のボリュームグループに再割り当てしたり、システムから削除できるようになりました。

## 5.5. クラスター内でのミラー化 LVM 論理ボリュームの作成

ミラー化 LVM 論理ボリュームをクラスターに作成するには、(セグメントタイプ **mirror** を設定して)

1 つのノードにミラー化 LVM 論理ボリュームを作成する場合と同じコマンドと手順を使用する必要があります。ただし、クラスターに、ミラー化 LVM ボリュームを作成するには、以下の条件を満たす必要があります。

- クラスターおよびクラスターミラーインフラストラクチャーが稼働している
- クラスターが定足数を満たす
- クラスターロックを有効にするよう **lvm.conf** ファイルのロックタイプを正しく設定する (直接設定するか、[「クラスター内での LVM ボリューム作成」](#) で説明されたように **lvmconf** コマンドを使用する)

Red Hat Enterprise Linux 7 では、クラスターは Pacemaker で管理されます。クラスター化された LVM 論理ボリュームは、Pacemaker クラスターと併用される場合のみサポートされるため、クラスターリソースとして設定する必要があります。

以下の手順は、クラスターに、ミラー化された LVM ボリュームを作成します。

1. クラスターソフトウェアおよび LVM パッケージをインストールし、クラスターソフトウェアを起動してクラスターを作成します。クラスターにはフェンスを設定する必要があります。『High Availability Add-On の管理』ドキュメントには、クラスターを作成し、クラスターにノードのフェンスを設定する手順例が記載されています。『High Availability Add-On Reference』ドキュメントには、クラスター設定のコンポーネントについての詳細情報が記載されています。
2. クラスター内の全ノードで共有するミラー化論理ボリュームを作成するには、クラスターの各ノードの **lvm.conf** ファイルにロックタイプが正しく設定されている必要があります。デフォルトでは、ロックタイプはローカルに設定されます。これを変更するには、クラスターの各ノードで以下のコマンドを実行し、クラスターロックを有効にします。

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

3. クラスターの **dlm** リソースをセットアップします。リソースをクローンリソースとして作成し、そのリソースがクラスター内のすべてのノードで実行されるようにします。

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

4. クラスターリソースとして **clvmd** を設定します。**dlm** リソースの場合と同様に、リソースを、クローン作成したリソースとして作成し、それがクラスター内のすべてのノードで実行されるようにします。**with\_cmirrord=true** パラメーターで、**clvmd** が実行されるすべてのノードで **cmirrord** デモンを有効にするように設定する必要があります。

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

**clvmd** リソースを設定していても、**with\_cmirrord=true** パラメーターを設定していない場合は、以下のコマンドでそのパラメーターを組み込むようにリソースを更新することができます。

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

5. **clvmd** および **dlm** の依存関係をセットアップし、順番に起動します。**clvmd** は **dlm** の後に起動し、**dlm** と同じノードで実行する必要があります。

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

6. ミラーを作成します。最初のステップは、物理ボリュームの作成です。次のコマンドは、3つの物理ボリュームを作成します。これらの内の2つの物理ボリュームはミラーレグとして使用され、3つ目の物理ボリュームにミラーログが格納されます。

```
# pvcreate /dev/sdb1
Physical volume "/dev/sdb1" successfully created
# pvcreate /dev/sdc1
Physical volume "/dev/sdc1" successfully created
# pvcreate /dev/sdd1
Physical volume "/dev/sdd1" successfully created
```

7. ボリュームグループを作成します。この例では、直前のステップで作成した3つの物理ボリュームで構成されるボリュームグループ **vg001** を作成します。

```
# vgcreate vg001 /dev/sdb1 /dev/sdc1 /dev/sdd1
Clustered volume group "vg001" successfully created
```

**vgcreate** コマンドの出力で、ボリュームグループがクラスター化されていることが示されている点に注意してください。ボリュームグループの属性を表示する **vgs** コマンドを使用すると、ボリュームグループがクラスター化されていることを検証することができます。ボリュームグループがクラスター化されている場合は、**c** 属性が表示されます。

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

8. ミラー化論理ボリュームを作成します。この例では、ボリュームグループ **vg001** から論理ボリューム **mirrorlv** を作成します。このボリュームのミラーレグは1つです。この例では、物理ボリュームのどのエクステンツが論理ボリュームに使用されるかを指定します。

```
# lvcreate --type mirror -l 1000 -m 1 vg001 -n mirrorlv /dev/sdb1:1-1000 /dev/sdc1:1-1000 /dev/sdd1:0
Logical volume "mirrorlv" created
```

**lvs** コマンドを使用すると、ミラー作成の進捗状況を表示できます。以下の例では、ミラーの同期が、47%、91% と進み、ミラー完了時には 100% 同期になったことを示しています。

```
# lvs vg001/mirrorlv
LV      VG      Attr      LSize Origin Snap%   Move Log
Copy%   Convert
mirrorlv vg001    mwi-a- 3.91G                                vg001_mlog
47.00
# lvs vg001/mirrorlv
LV      VG      Attr      LSize Origin Snap%   Move Log
Copy%   Convert
mirrorlv vg001    mwi-a- 3.91G                                vg001_mlog
91.00
```

```
# lvs vg001/mirrorlv
  LV          VG      Attr   LSize Origin Snap%   Move Log
Copy% Convert
mirrorlv vg001      mwi-a- 3.91G                vg001_mlog
100.00
```

ミラーの完了は、システムログに記録されます。

```
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
```

9. **lvs** コマンドで **-o +devices** オプションを使用すると、ミラーの設定を表示できます。これには、ミラーレグを構成するデバイスの情報が含まれます。この例では、論理ボリュームが2つのリニアイメージと1つのログで構成されていることがわかります。

```
# 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/sdb1(1)
[mirrorlv_mimage_1] vg001      iwi-ao 3.91G
/dev/sdc1(1)
[mirrorlv_mlog]    vg001      lwi-ao 4.00M
/dev/sdd1(0)
```

**lvs** の **seg\_pe\_ranges** オプションを使用すると、データレイアウトを表示することができます。このオプションを使用すれば、レイアウトに適切な冗長性があることを検証することができます。このコマンドの出力は、**lvcreate** と **lvresize** コマンドが入力として受け取る形式と同じ形式で PE 範囲を表示します。

```
# lvs -a -o +seg_pe_ranges --segments
PE Ranges
mirrorlv_mimage_0:0-999 mirrorlv_mimage_1:0-999
/dev/sdb1:1-1000
/dev/sdc1:1-1000
/dev/sdd1:0-0
```



## 注記

LVM ミラー化ボリュームのいずれかのレグに障害が発生した際の回復方法は、[「LVM ミラー障害からの回復」](#) を参照してください。

## 第6章 LVM トラブルシューティング

本章では、さまざまな LVM の問題のトラブルシューティング手順について説明します。

### 6.1. トラブルシューティング診断

コマンドが期待通りに機能しない場合は、以下の方法で診断情報を収集できます。

- 任意のコマンドに **-v**、**-vv**、**-vvv**、**-vvvv** のいずれかの引数を使用して、出力の詳細レベルを徐々に高くしていくことができます
- 問題が論理ボリュームのアクティブ化に関連している場合は、設定ファイルの **log** セクションに **activation = 1** と設定して、**-vvvv** 引数を付けてコマンドを実行します。この出力を検証したら、このパラメーターを 0 に戻し、低メモリー状況で起こり得るマシンのロッキング問題を回避します。
- **lvm dump** コマンドを実行すると、診断目的の情報ダンプが提供されます。詳細は **lvm dump(8)** man ページで参照してください。
- 追加のシステム情報を得るには、**lvs -v**、**pvs -a**、または **dmsetup info -c** コマンドを実行します。
- **/etc/lvm/backup** ファイル内のメタデータの最近のバックアップと **/etc/lvm/archive** ファイル内のアーカイブバージョンを検証します。
- **lvmconfig** コマンドを実行して、現在の設定情報をチェックします。
- 物理ボリュームを持つデバイスについての記録を調べるために、**/etc/lvm** ディレクトリーの **.cache** ファイルをチェックします。

### 6.2. 障害の発生したデバイスの情報を表示

**lvs** または **vgs** コマンドに **-P** 引数を使用すると、他の方法では出力に表示されないような障害ボリュームに関する情報を表示することができます。この引数は、メタデータが内部で完全に一貫していない場合でも、一部の操作を許可します。たとえば、ボリュームグループ **vg** を構成するデバイスのいずれかに障害が発生した場合、**vgs** コマンドで以下のような出力が表示される場合があります。

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

**vgs** コマンドで **-P** 引数を指定すると、ボリュームグループは使用不可のままですが、障害のあるデバイスに関するより多く情報を確認することができます。

```
# 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)
```

この例では、障害デバイスが、ボリュームグループ内のリニア論理ボリュームと、ストライプ化論理ボリュームの両方の障害の原因になっています。**-P** 引数を付けない **lvs** コマンドでは、以下のような出力が表示されます。

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

-P 引数を使用すると、障害の発生した論理ボリュームが表示されます。

```
# 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)
```

以下の例は、ミラー化論理ボリュームの 1 つのレッグに障害が発生した場合の、-P 引数を指定した **pvs** と **lvs** コマンドの出力を示しています。

```
# 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. LVM ミラー障害からの回復

このセクションでは、物理ボリュームの基礎となるデバイスが停止したため、LVM ミラー化ボリュームのレッグの 1 つに障害が発生し、**mirror\_log\_fault\_policy** パラメーターが **remove** に設定されている状態から復旧する例を説明します。この場合は、ミラーを手動で再構築する必要があります。**mirror\_log\_fault\_policy** パラメーターの設定については、「[ミラー化論理ボリュームの障害ポリシー](#)」を参照してください。

ミラーレッグに障害が発生すると、LVM はミラー化ボリュームをリニアボリュームに変換します。このボリュームは以前と同様に動作を継続しますが、ミラー化による冗長性はありません。この時点で、代替物理デバイスとして使用し、ミラーを再構築するために新たなディスクデバイスをシステムに追加することができます。

以下のコマンドは、ミラー用に使用される物理ボリューム群を作成します。



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

以下のコマンドは、ボリュームグループ **vg** とミラー化ボリューム **groupfs** を作成します。

```
# vgcreate vg /dev/sd[abcdefgh][12]
Volume group "vg" successfully created
# 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
```

**lvs** コマンドを使用すると、ミラー化ボリュームのレイアウトとミラーレッグの配下のデバイスとミラーログを確認できます。最初の例では、ミラーは完全には同期化されていないことに注意してください。**Copy%** フィールドが 100.00 になるのを待ってから続行する必要があります。

```
# 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)

# 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)
```

この例では、ミラーのプライマリーレッグ **/dev/sda1** に障害が発生しています。ミラー化ボリューム

への書き込み作業はいずれも LVM がミラーの障害を検知する結果となります。これが発生すると、LVM はミラーを単一のリニアボリュームに変換します。この場合は、この変換をトリガーするために **dd** コマンドを実行します。

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

**lvs** コマンドを使用して、デバイスがリニアデバイスになっていることを確認することができます。障害が発生したディスクが原因で I/O エラーが発生します。

```
# lvs -a -o +devices
/dev/sda1: 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)
```

この時点では、論理ボリュームは使用することができるはずですが、ミラーの冗長性がなくなります。

ミラー化ボリュームを再構築するには、破損したボリュームを交換して、物理ボリュームを再作成します。新規ディスクに交換しないで同じディスクを使用すると、**pvcreate** コマンドを実行した時に、"inconsistent" の警告が表示されます。この警告が表示されないようにするには、**vgreduce --removemissing** のコマンドを実行します。

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

# 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]
```

次に、新規物理ボリュームで元のボリュームグループを拡張します。

```
# 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  ]

```

リアボリュームを変換して、元のミラー化された状態に戻します。

```

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

```

**lvs** コマンドを使用すると、ミラーが復元したことを確認できます。

```

# 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. 物理ボリュームメタデータの復元

物理ボリュームのボリュームグループのメタデータ領域が誤って上書きされたり、破棄されたりする場合は、メタデータ領域が正しくないことを示すエラーメッセージか、システムで特定 UUID の物理ボリュームが見つからなかったことを示すエラーメッセージが表示されます。物理ボリュームからのデータの復元は、失われたメタデータと同じ UUID を指定して、物理ボリューム上に新規のメタデータ領域を書き込むことによって実行できる場合があります。



### 警告

作業用の LVM 論理ボリュームを使ってこの手順を試行しないでください。間違った UUID を指定するとデータ損失の原因となります。

以下の例は、メタデータ領域が見つからなかったり、破損している場合に表示される出力の種類を示しています。

```
# 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.
...
```

上書きされている物理ボリュームの UUID は、`/etc/lvm/archive` ディレクトリーで見つけることができます。**VolumeGroupName\_xxxx.vg** ファイルで、該当するボリュームで最後にアーカイブ化された、有効な LVM メタデータを確認します。

または、ボリュームを非アクティブ化し、**partial (-P)** 引数を設定すると、見つからない破損した物理ボリュームの UUID を見つけれられる可能性があります。

```
# 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'.
...
```

**pvcreate** コマンドで、**--uuid** と **--restorefile** 引数を使用して、物理ボリュームを復元します。以下の例では、`/dev/sdh1` デバイスを上記の UUID (**FmGRh3-zhok-iVI8-7qTD-S5BI-MAEN-NYM5Sk**) を持つ物理ボリュームとしてラベル付けします。このコマンドは、ボリュームグループ用の最近の正しいアーカイブのメタデータ **VG\_00050.vg** に含まれているメタデータ情報で、物理ボリュームラベルを復元します。**restorefile** 引数は **pvcreate** コマンドに対して、ボリュームグループ上の古いものと互換性のある物理ボリュームを作成するように指示し、古い物理ボリュームのデータが含まれていた場所に新しいメタデータを配置しないようにします。(これは、たとえば元の **pvcreate** コマンドが、メタデータの配置を制御をするコマンドライン引数を使用していた場合や、物理ボリュームが複数の異なるデフォルトを使用するソフトウェアの別バージョンを使用して作成されていた場合などに発生する可能性があります)。**pvcreate** コマンドは LVM メタデータ領域のみを上書きし、既存のデータ領域には影響を与えません。

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

次に、**vgcfgrestore** コマンドを使用して、ボリュームグループのメタデータを復元することができます。

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

これで論理ボリュームが表示できるようになります。

```
# 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)
```

以下のコマンドはボリュームをアクティブにし、アクティブになったボリュームを表示します。

```
# lvchange -ay /dev/VG/stripe
# 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)
```

オンディスク LVM メタデータが、それを書き換えるデータと同じ容量である場合は、このコマンドで物理ボリュームを復元できます。メタデータの書き換えがメタデータ領域を超える場合、ボリューム上のデータは影響を受ける可能性があります。そのデータを復元するには、**fsck** コマンドを使用することができます。

## 6.5. 紛失した物理ボリュームの入れ替え

物理ボリュームに障害が発生した場合、または交換の必要がある場合、「[物理ボリュームメタデータの復元](#)」で説明されている物理ボリュームのメタデータを復旧するのと同じ手順に従って、既存ボリュームグループ内の紛失した物理ボリュームに置き換わる新しい物理ボリュームにラベル付けすることができます。**vgdisplay** コマンドで **--partial** と **--verbose** 引数を使用すると、すでに存在しない物理ボリュームの UUID およびサイズを表示することができます。別の同じサイズの物理ボリュームを置き換える場合は、**pvccreate** コマンドで **--restorefile** と **--uuid** 引数を使用して、紛失した物理ボリュームと同じ UUID を持つ新規デバイスを初期化することができます。その後、**vgcfgrestore** コマンドを使用してボリュームグループのメタデータを復元します。

## 6.6. 紛失した物理ボリュームのボリュームグループからの削除

物理ボリュームがなくなった場合、ボリュームグループ内の残りの物理ボリュームをアクティブ化するには、**vgchange** コマンドで **--partial** 引数を使用します。その物理ボリュームを使用していたすべての論理ボリュームをボリュームグループから削除するには **vgreduce** コマンドで **--removemissing** 引数を使用します。

**--test** 引数を指定して **vgreduce** コマンドを実行して、破棄する対象を確認する必要があります。

ほとんどの LVM 操作と同様に、**vgcfgrestore** コマンドを **vgreduce** コマンドの実行直後に使用して、ボリュームグループのメタデータをその以前の状態に戻すと、**vgreduce** コマンドを元に戻すことが可能です。たとえば、**--test** 引数なしで **vgreduce** コマンドで **--removemissing** 引数を使用し、保持するつもりだった論理ボリュームを削除してしまった場合でも、その物理ボリュームの入れ替えは可能であり、別の **vgcfgrestore** コマンドを使用してボリュームグループを以前の状態に戻すことができます。

## 6.7. 論理ボリュームでの不十分な空きエクステンツ

論理ボリュームの作成時に「Insufficient free extents」というエラーメッセージが表示されることがあります。これは **vgdisplay** または **vgs** のコマンドの出力からは十分なエクステンツがあると思われる場合でも発生することがあります。その理由は、これらのコマンドが数字を第 2 小数点まで丸め、人間に認識可能な出力を提供するためです。実際のサイズを指定するには、物理ボリュームのサイズの判別にバイトの倍数を使用せずに、空き物理エクステンツ数を使用します。

**vgdisplay** コマンドの出力には、デフォルトで、空き物理エクステンツを示す以下のような行が含まれます。

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

別の方法として、**vg**s コマンドで **vg\_free\_count** と **vg\_extent\_count** 引数を使用して、空きエクステントと合計エクステント数を表示します。

```
# 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
```

8780 の空き物理エクステントがあるため、小文字 l の引数を使用してバイトの代わりにエクステントを使用し、次のコマンドを実行できます。

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

これで、ボリュームグループ内のすべてのエクステントが使用されます。

```
# 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
```

もしくは、**lvcreate** コマンドで **-l** 引数を使用して、ボリュームグループ内の残りの空き領域のパーセントを使用するために論理ボリュームを拡大することができます。詳細は、[「リニア論理ボリュームの作成」](#) を参照してください。

## 6.8. マルチパスデバイスに対する重複した PV 警告

マルチパスストレージで LVM を使用する場合は、一部の LVM コマンド (**vgs** や **lvchange** など) で、ボリュームグループまたは論理ボリュームを一覧表示するときに、以下のようなメッセージが表示されることがあります。

```
Found duplicate PV GDjTZf7Y03GJHjteq0wrye2dcSCjdaUi: using /dev/dm-5 not
/dev/sdd
Found duplicate PV GDjTZf7Y03GJHjteq0wrye2dcSCjdaUi: using /dev/emcpowerb
not /dev/sde
Found duplicate PV GDjTZf7Y03GJHjteq0wrye2dcSCjdaUi: using /dev/sddlmb
not /dev/sdf
```

このセクションでは、これらの警告の原因について説明したあとに、以下の 2 つのケースでこの問題を解決する方法について説明します。

- 出力に表示された 2 つのデバイスが、両方とも同じデバイスへの単一パスである
- 出力に表示された 2 つのデバイスが、両方ともマルチパスマップである

### 6.8.1. 重複した PV 警告の原因

デフォルト設定では、LVM コマンドは **/dev** のデバイスをスキャンし、検出された各デバイスで LVM メタデータをチェックします。これは、以下のような **/etc/lvm/lvm.conf** のデフォルトフィルターにより実行されます。

■

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

Device Mapper Multipath、または EMC PowerPath や Hitachi Dynamic Link Manager (HDLM) などの他のマルチパスソフトウェアを使用している場合、特定の論理ユニット番号 (LUN) に対する各パスは、`/dev/sdb` または `/dev/sdc` などの異なる SCSI デバイスとして登録されます。マルチパスソフトウェアは、この各パスに対してマッピングされる新しいデバイスを作成します (たとえば Device Mapper Multipath の場合は `/dev/mapper/mpath1` または `/dev/mapper/mpatha`、EMC PowerPath の場合は `/dev/emcpowera`、Hitachi HDLM の場合は `/dev/sddl1ab`)。各 LUN は、基礎となる同じデータを参照する、`/dev` 内の複数のデバイスノードを持つため、すべてのデバイスには同じ LVM メタデータが含まれます。したがって、LVM コマンドは同じメタデータを複数回検出し、重複したものとして報告します。

これらの重複メッセージはただの警告であり、LVM 操作が失敗したことは意味しません。これらのメッセージは、1 つのデバイスだけが物理ボリュームとして使用され、他のデバイスは無視されることをユーザーに通知します。メッセージが、正しくないデバイスが選択されていることを示す場合、または警告がユーザーにとって重大である場合は、物理ボリュームに必要なデバイスのみを検索し、マルチパスデバイスへの基礎となるパスを省略するために、フィルターを適用できます。

### 6.8.2. 単一パスに対する重複した警告

以下の例は、表示された重複デバイスが、両方とも同じデバイスへの単一パスであることを示す、重複した PV 警告を示しています。この場合、`/dev/sdd` と `/dev/sdf` は、`multipath -ll` コマンドの出力の同じマルチパスマップ下にあります。

```
Found duplicate PV GDjTZf7Y03GJHjteq0wrye2dcSCjdaUi: using **/dev/sdd**
not **/dev/sdf**
```

この警告が表示されないようにするには、LVM がメタデータを検索するデバイスを制限するように、`/etc/lvm/lvm.conf` ファイルでフィルターを設定します。フィルターは、`/dev` (または `/etc/lvm/lvm.conf` ファイルで `dir` キーワードにより指定されたディレクトリー) のスキャンによって検出された、各デバイスに適用されるパターンのリストです。パターンは、任意の文字で区切られた正規表現と、`a` (許可) または `r` (拒否) が先頭に付けられた正規表現です。リストは順番に評価され、デバイスに一致する最初の正規表現によって、デバイスが許可または拒否 (無視) されるかどうかが決まります。どのパターンにも一致しないデバイスは許可されます。LVM フィルターの一般的な情報については、「[フィルターを使用した LVM デバイススキャンの制御](#)」を参照してください。

設定するフィルターには、LVM メタデータをチェックする必要があるすべてのデバイス (root ボリュームグループがあるローカルハードドライブやマルチパスデバイスなど) を含める必要があります。マルチパスデバイスへの基礎となるパス (`/dev/sdb` や `/dev/sdd` など) を拒否することにより、これらの重複する PV 警告を回避できます (一意の各メタデータ領域はマルチパスデバイス自体で 1 度しか検出されないため)。

以下の例は、複数のストレージパスが利用可能であることが原因で発生する重複 PV 警告を回避するフィルターを示しています。

- このフィルターは、最初のハードドライブ (`/dev/sda`) の 2 番目のパーティションと、すべての device-mapper-multipath デバイスを許可し、他のすべてのパーティションとデバイスを拒否します。

```
filter = [ "a|/dev/sda2$|", "a|/dev/mapper/mpath.*|", "r|.*)" ]
```

- このフィルターは、すべての HP SmartArray コントローラと、EMC PowerPath デバイスを許可します。



```
filter = [ "a|/dev/cciss/.*|", "a|/dev/emcpower.*|", "r|.*)" ]
```

- このフィルターは、最初の IDE ドライブのすべてのパーティションと、すべてのマルチパスデバイスを許可します。

```
filter = [ "a|/dev/hda.*|", "a|/dev/mapper/mpath.*|", "r|.*)" ]
```



#### 注記

新しいフィルターを **/etc/lvm/lvm.conf** ファイルに追加する場合は、元のフィルターを # でコメントアウトするか、削除してください。

フィルターが設定され、**/etc/lvm/lvm.conf** ファイルが保存されたら、これらのコマンドの出力をチェックして、物理ボリュームまたはボリュームグループが不明でないことを確認します。

```
# pvscan
# vgscan
```

また、以下の例で示されているように、**/etc/lvm/lvm.conf** ファイルを変更しなくても、LVM コマンドに **--config** 引数を追加すれば、すぐにフィルターをテストすることができます。

```
# lvs --config 'devices{ filter = [ "a|/dev/emcpower.*|", "r|.*)" ] }'
```



#### 注記

**--config** 引数を使用してフィルターをテストした場合、サーバーの設定には永続的な変更が加えられません。テスト後に、有効なフィルターを **/etc/lvm/lvm.conf** ファイルに設定してください。

LVM フィルターの設定後に、再起動時に必要なデバイスのみがスキャンされるように、**dracut** コマンドで **initrd** デバイスを再ビルドすることが推奨されます。

### 6.8.3. マルチパスマップの重複する警告

以下の例は、両方ともマルチパスマップである 2 つのデバイスに対する、重複する PV 警告を示しています。これらの例では、2 つの異なるパス (ただし、デバイスは異なる) について説明します。

```
Found duplicate PV GDjTZf7Y03GJHjtec0wrye2dcSCjdaUi: using
**/dev/mapper/mpatha** not **/dev/mapper/mpathc**
```

```
Found duplicate PV GDjTZf7Y03GJHjtec0wrye2dcSCjdaUi: using
**/dev/emcpowera** not **/dev/emcpowerh**
```

この状況は、同じデバイスへの 1 つのパスであるデバイスに対する重複の警告よりも深刻です。多くの場合、これらの警告は、マシンが確認すべきでないデバイス (たとえば、LUN クローンまたはミラー) がマシンに提供されたことを意味します。この場合は、マシンから取り外すデバイスが明確にわかっていない限り、状況は改善しません。この問題を解決するには、Red Hat テクニカルサポートにお問い合わせいただくことが推奨されます。



## 付録A デバイスマッパー

デバイスマッパーとは、ボリューム管理用のフレームワークを提供するカーネルドライバーです。これは論理ボリュームとして使用可能な、マップされたデバイスを作成する一般的な方法を提供します。デバイスマッパーは、ボリュームグループやメタデータ形式をとくに認識する訳ではありません。

デバイスマッパーは多くの高度な技術のための土台を提供します。LVM のほかにも、デバイスマッパーマルチパスと **dmraid** コマンドがデバイスマッパーを使用します。デバイスマッパーに対するアプリケーションインターフェースは **ioctl** システムコールになり、ユーザーインターフェースは **dmsetup** コマンドになります。

LVM 論理ボリュームは、デバイスマッパーを使用してアクティブにされます。それぞれの論理ボリュームは、マッピングされたデバイスに変換され、それぞれのセグメントが、そのデバイスを記述するマッピングテーブルの行に変換されます。デバイスマッパーは、リニアマッピング、ストライプ化マッピング、エラーマッピングを含む各種のマッピングターゲットをサポートします。たとえば、2つのディスクを1つの論理ボリュームとして連結することができます。これは、各ディスクに対してそれぞれ一対のリニアマッピングを1つ維持することで行います。LVM がボリュームを作成する場合は、**dmsetup** コマンドでクエリー可能な、配下のデバイスマッパーデバイスを作成します。マッピングテーブルのデバイスの形式に関する情報は、「[デバイステーブルのマッピング](#)」を参照してください。デバイスをクエリーする **dmsetup** コマンドの使用方法については、「[dmsetup コマンド](#)」を参照してください。

### A.1. デバイステーブルのマッピング

マップされたデバイスは、サポートされているデバイステーブルのマッピングを使用してデバイスの論理セクターの各範囲をマップする方法を指定するテーブルによって定義されます。マップされたデバイスのテーブルは以下の形式の行の一覧から作成されます。

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

デバイスマッパーテーブルの最初の行では、**start** パラメーターはゼロ (0) でなければなりません。1行にある **start** + **length** のパラメーター群は、次の行の **start** と同じでなければなりません。マッピングテーブルの行に指定されるマッピングパラメーターの種類は、その行に指定される **mapping** のタイプによって決まります。

デバイスマッパー内のサイズは常にセクターで指定されます (512 バイト)。

1つのデバイスがデバイスマッパー内でマッピングパラメーターとして指定される場合、そのデバイスはファイルシステム内のデバイス名で参照されるか (例: **/dev/hda**)、または **major:minor** の形式でメジャー番号とマイナー番号で参照されます。major:minor の形式は、パス名ルックアップを回避するので推奨されます。

デバイスのマッピングテーブルの例を以下に示します。このテーブルには4つのリニアターゲットがあります。

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

各行の最初の2つのパラメーターはセグメントの始点ブロックとセグメントの長さです。次のキーワードはマッピングターゲットであり、この例ではすべて **linear** になっています。行のその他の部分は **linear** ターゲットのパラメーターで構成されています。

以下のセクションでは以下のマッピングの形式について説明しています。

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

### A.1.1. リニアマッピングターゲット

リニアマッピングターゲットは連続範囲のブロックを別のブロックデバイスにマップします。リニアターゲットの形式は以下のようになります。

```
start length linear device offset
```

#### **start**

仮想デバイス内の始点ブロック

#### **length**

このセグメントの長さ

#### **device**

ブロックデバイス。ファイルシステム内のデバイス名で参照、または *major:minor* の形式のメジャー番号とマイナー番号で参照されます。

#### **offset**

デバイス上のマッピングの始点オフセット

以下の例は、仮想デバイスの始点ブロックが 0、セグメントの長さが 1638400、メジャー/マイナー番号ペアが 8:2、デバイスの始点オフセットが 41146992 であるリニアターゲットを示しています。

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

以下の例は、デバイスパラメーターがデバイス **/dev/hda** として指定されたリニアターゲットを示しています。

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

### A.1.2. ストライプ化マッピングターゲット

ストライプ化マッピングターゲットは物理デバイス全体でのストライピングをサポートします。これは、ストライプの数、ストライプのチャンクサイズ、およびデバイス名とセクターのペアの一覧を引数として受け取ります。ストライプ化ターゲットの形式は以下のようになります。

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

それぞれのストライプについて **device** と **offset** のパラメーターの 1 つのセットが指定されます。

### **start**

仮想デバイス内の始点ブロック

### **length**

このセグメントの長さ

### **#stripes**

仮想デバイスのストライプの数

### **chunk\_size**

次にスイッチするまでに各ストライプに書き込まれるセクターの数。2 の累乗であり、最低でもカーネルページサイズの大きさでなければなりません。

### **device**

ブロックデバイス。ファイルシステム内のデバイス名で参照されるか、または **major:minor** の形式でメジャー番号とマイナーの番号で参照されます。

### **offset**

デバイス上のマッピングの始点オフセット

以下の例は、3 つのストライプと、チャンクサイズ 128 を持つストライプ化ターゲットを示しています。

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

### **0**

仮想デバイス内の始点ブロック

### **73728**

このセグメントの長さ

### **striped 3 128**

チャンクサイズが 128 ブロックの 3 つのデバイスにわたるストライプ

### **8:9**

最初のデバイスのメジャー番号:マイナー番号

### **384**

最初のデバイス上のマッピングの始点オフセット

**8:8**

2 つ目のデバイスのメジャー番号:マイナー番号

**384**

2 つ目のデバイスのマッピングの始点オフセット

**8:7**

3 つ目のデバイスのメジャー番号:マイナー番号

**9789824**

3 つ目のデバイス上のマッピングの始点オフセット

以下の例は、2 つのストライプ、256 KiB のチャンク、およびメジャー番号とマイナー番号の代わりにファイルシステム内のデバイス名で指定されたデバイスパラメーターを持つストライプ化ターゲットを示しています。

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

### A.1.3. ミラーマッピングターゲット

ミラーマッピングターゲットはミラー化した論理デバイスのマッピングをサポートします。ミラー化ターゲットの形式は次のようになります。

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

**start**

仮想デバイス内の始点ブロック

**length**

このセグメントの長さ

**log\_type**

使用可能なログタイプとそれらの引数は以下のようになります。

**core**

ミラーはローカルであり、ミラーログはコアメモリーに保持されます。このログタイプは 1 - 3 の引数を取ります。

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

**disk**

ミラーはローカルであり、ミラーログはディスクに保持されます。ログタイプは 2 - 4 の引数を取ります。

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

**clustered\_core**

ミラーはクラスター化されており、ミラーログはコアメモリーに保持されます。このログタイプは 2 - 4 の引数を取ります。

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

**clustered\_disk**

ミラーはクラスター化されており、ミラーログはディスクに保持されます。このログタイプは 3 - 5 の引数を取ります。

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

LVM は、どのリージョンがミラー (ミラー群) と同期しているかを追跡するのに使用する小さなログを維持します。*regionsize* 引数は、それらのリージョンのサイズを指定します。

クラスター環境では、*UUID* 引数はミラーログデバイスに関連付けられた一意の識別子であるため、ログの状態はクラスター全体で維持することができます。

オプションの **[no]sync** 引数を使用して、ミラーを "in-sync" か "out-of-sync" として指定することができます。**block\_on\_error** 引数はミラーに対して、エラーを無視するのではなくエラーに対応するように指示するために使用されます。

**#log\_args**

マッピング内で指定されるログ引数の数

**logargs**

ミラー用のログ引数。提供されるログ引数の数は **#log-args** パラメーターで指定され、有効なログ引数は **log\_type** パラメーターで決定されます。

**#devs**

ミラー内のレグ数。デバイスとオフセットが各レグに指定されます。

**device**

それぞれのレグ用のブロックデバイス。ファイルシステム内のデバイス名で参照されるか、または **major:minor** の形式でメジャーとマイナーの番号で参照されます。**#devs** パラメーターに示されるように、ブロックデバイスとオフセットは各ミラーレグに指定されます。

**offset**

デバイス上のマッピングの始点オフセット。ブロックデバイスとオフセットは **#devs** で示されるようにそれぞれのミラーレグ用に指定されます。

以下の例は、ミラーログがディスク上に保持されたクラスター化されたミラー用のミラーマッピングターゲットを示しています。

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

0

仮想デバイス内の始点ブロック

**52428800**

このセグメントの長さ

**mirror clustered\_disk**

ミラーがクラスター化されており、ディスク上でミラーログを維持することを指定するログタイプが指定されたミラーターゲット

**4**

4 つのミラーログ引数が続きます。

**253:2**

ログデバイスのメジャー番号:マイナー番号

**1024**

何が同期しているかを追跡記録するためにミラーログが使用するリージョンのサイズ

**UUID**

クラスター全域でログ情報を維持するためのミラーログデバイスの UUID

**block\_on\_error**

ミラーはエラーに対応する必要があります。

**3**

ミラー内のレッグ数

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

ミラーの各レッグを構成しているデバイス用のメジャー番号:マイナー番号およびオフセット

### **A.1.4. スナップショットとスナップショット複製元のマッピングターゲット**

ボリュームの最初の LVM スナップショットを作成する場合に、4 つのデバイスマッパーデバイスが使用されます。

1. **linear** マッピングを持つデバイス。ソースボリュームの元のマッピングテーブルが含まれます。
2. ソースボリューム用の COW (copy-on-write) デバイスとして使用される **linear** マッピングを持つデバイス。書き込みを行うたびに、元のデータは各スナップショットの COW デバイス内に保存され、その可視コンテンツはそのまま維持されます (COW デバイスが満杯になるまで)。
3. 上記の 1 と 2 を組み合わせた **snapshot** マッピングを持つデバイス。これは可視スナップショットボリュームです。
4. 「元」のボリューム (これは、元のソースボリュームで使用されるデバイス番号を使用します)。このテーブルはデバイス #1 からの「snapshot-origin」マッピングに置き換えられます。

これらのデバイスを作成するには固定された命名スキームが使用されます。たとえば、以下のようなコマンドを使用して **base** という名前の LVM ボリュームを作成し、**snap** という名前のスナップショットボリュームをそのボリューム上に作成することができます。

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

これによって 4 つのデバイスが作成され、以下のコマンドで表示できます。

```
# 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** ターゲットの形式は以下ようになります。

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

#### **start**

仮想デバイス内の始点ブロック

#### **length**

このセグメントの長さ

#### **origin**

スナップショットのベースボリューム

**snapshot-origin** には通常、それをベースにした 1 つまたは複数のスナップショットがあります。読み取りは直接バックアップデバイスにマップされます。それぞれの書き込みには、元のデータが各スナップショットの COW デバイス内に保存されて、COW デバイスが満杯になるまでその可視コンテンツをそのまま維持します。

**snapshot** ターゲットの形式は以下ようになります。

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

#### **start**

仮想デバイス内の始点ブロック

#### **length**

このセグメントの長さ

## **origin**

スナップショットのベースボリューム

## **COW-device**

変更したデータチャンクが保存されるデバイス

## **P|N**

P (Persistent) または N (Not persistent) は、スナップショットが再起動後に維持されるかどうかを示します。一時的なスナップショット (N) では、ディスクに保存できるメタデータが少なくなり、代わりに、カーネルがメモリーに保持します。

## **chunksize**

COW デバイスに保存されるデータにおける、変更したチャンクのセクターサイズ

次の例では、複製元デバイスが 254:11 の **snapshot-origin** ターゲットを示しています。

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

以下の例では、複製元デバイスが 254:11 で、COW デバイスが 254:12 の **snapshot** ターゲットを示しています。このスナップショットデバイスは再起動後も永続し、COW デバイス上に保存されるデータのチャンクサイズは 16 セクターです。

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

### **A.1.5. エラーマッピングターゲット**

エラーマッピングターゲットを使用すると、マッピングされたセクターへの I/O オペレーションはいずれも失敗します。

エラーマッピングターゲットはテスト用に使用できます。障害時にデバイスがどのように動作するかをテストするには、1 つのデバイスの中に不良セクターがあるデバイスマッピングを 1 つ作成するか、またはミラーレグをスワップアウトして、そのレグをエラーターゲットに置き換えます。

エラーターゲットは、実際のデバイス上でのタイムアウトおよび再試行を回避する方法として、障害のあるデバイスの代わりに使用することができます。エラーターゲットは、障害時に LVM メタデータを再配置している間に中間ターゲットとして機能します。

**error** マッピングターゲットは、*start* と *length* のパラメーター以外には追加のパラメーターを取りません。

以下の例は、**error** ターゲットを示しています。

```
0 65536 error
```

### **A.1.6. ゼロマッピングターゲット**

**zero** マッピングターゲットは、**/dev/zero** と同等のブロックデバイスです。このマッピングの読み取り操作はゼロのブロックを返します。このマッピングに書き込まれたデータは破棄されますが、書き込みは正常に実行されます。**zero** マッピングターゲットは *start* と *length* パラメーター以外には追加のパラメーターは取りません。



以下の例は、16Tb デバイス用の **zero** ターゲットを示しています。

```
0 65536 zero
```

### A.1.7. マルチパスマッピングターゲット

マルチパスマッピングターゲットはマルチパス化したデバイスのマッピングをサポートします。**multipath** ターゲットの形式は以下のようになります。

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

それぞれのパスグループ用に 1 つのセットの **pathgroupargs** パラメーター群があります。

#### **start**

仮想デバイス内の始点ブロック

#### **length**

このセグメントの長さ

#### **#features**

マルチパス機能の数。その後にそれらの機能が続きます。このパラメーターがゼロであれば、**feature** パラメーターは存在せず、次のデバイスマッピングパラメーターは **#handlerargs** となります。現在、**multipath.conf** ファイルの **features** 属性で設定できる **queue\_if\_no\_path** というサポートされている機能が 1 つあります。これは、利用可能なパスがない場合には、マルチパス化したデバイスが I/O 操作をキューに登録するよう現在設定されていることを示します。

以下の例では、**multipath.conf** ファイルの **no\_path\_retry** 属性が設定されています。これは、パスを使用する試行が所定回数行われた後にすべてのパスが失敗 (failed) とマークされるまで I/O 操作をキューに登録するために設定されます。この場合、すべてのパスチェッカーによるチェックが所定回数失敗するまでマッピングは以下のように表示されます。

```
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
```

すべてのパスチェッカーがチェックに所定回数失敗した後は、マッピングは以下のように表示されます。

```
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**

ハードウェアハンドラー引数の数です。それらの引数がその後に続きます。ハードウェアハンドラーは、パスグループを切り替える場合か、または I/O エラーを処理する場合に、ハードウェア固有のアクションを実行するために使用されるモジュールを指定します。これがゼロに設定されている場合は、次のパラメーターは **#pathgroups** となります。

#### **#pathgroups**

パスグループの数です。パスグループとは、マルチパス化したデバイスがロードバランシングを行うパスのセットのことです。それぞれのパスグループに 1 つのセットの **pathgroupargs** パラメータがあります。

### **pathgroup**

試行する次のパスグループ

### **pathgroupsargs**

各パスグループは以下の引数で構成されます。

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

パスグループ内の各パス用にパス引数の 1 つのセットがあります。

### **pathselector**

次の I/O 操作で使用するための、このパスグループ内のパスを決定するために使用するアルゴリズムを指定します。

### **#selectorargs**

マルチパスマッピングでこの引数に続くパスセレクター引数の数。現在、この引数の値は常に 0 (ゼロ) です。

### **#paths**

このパスグループ内のパスの数。

### **#pathargs**

このグループ内の各パスに指定されたパス引数の数。現在、この数は常に 1 で **ioreqs** 引数になります。

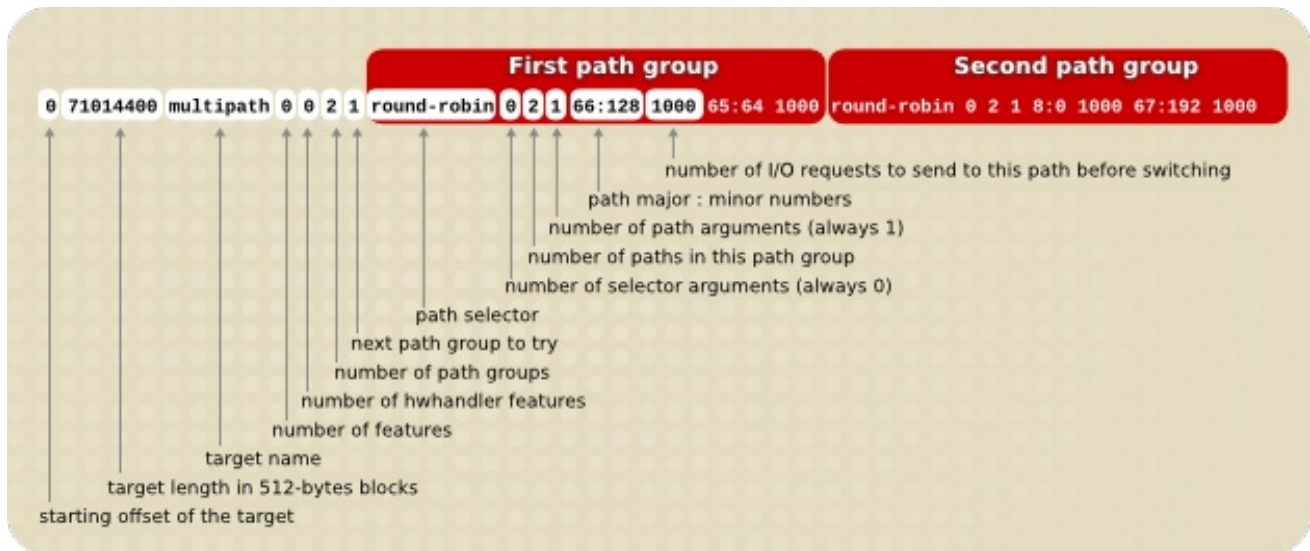
### **device**

パスのブロックデバイス数。**major:minor** の形式で、メジャー番号とマイナー番号によって参照されます。

### **ioreqs**

現在のグループ内の次のパスへ切り替えるまでこのパスにルーティングする I/O 要求数。

図A.1「マルチパスマッピングターゲット」は、2つのパスグループを持つマルチパスターゲットの形式を示しています。



図A.1 マルチパスマッピングターゲット

以下の例は、同じマルチパスデバイスのための純粋なフェイルオーバーターゲットの定義を示しています。このターゲットには、4つのパスグループがあります。マルチパス化したデバイスが1度に1つのパスのみを使用するように、パスグループごとに1つのパスのみが開いています。

```
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
```

次の例は、同じマルチパス化したデバイスを対象とする、完全に分散した (multibus) ターゲットの定義を示しています。このターゲットでは、パスグループが1つだけ存在し、そこにすべてのパスが含まれます。このセットアップでは、マルチパスが、負荷をすべてのパスに均等に分散します。

```
0 71014400 multipath 0 0 1 1 round-robin 0 4 1 66:112 1000 \
67:176 1000 68:240 1000 65:48 1000
```

マルチパス化の詳細は、『DM マルチパス』を参照してください。

### A.1.8. 暗号マッピングターゲット

**crypt** ターゲットは、指定したデバイスを経由してデータパッシングを暗号化します。これは、kernel Crypto API を使用します。

**crypt** ターゲットの形式は以下のようになります。

```
start length crypt cipher key IV-offset device offset
```

#### **start**

仮想デバイス内の始点ブロック

#### **length**

このセグメントの長さ

#### **cipher**

Cipher は、***cipher[-chainmode]-ivmode[:iv options]*** で構成されます。

**cipher**

利用できる Cipher は **/proc/crypto** (例: **aes**) 内に一覧表示されています。

**chainmode**

常に **cbc** を使用します。**ebc** は 使用しません。これは初期ベクトル (IV) を使いません。

**ivmode[:iv options]**

IV は暗号化を変更するために使用する初期ベクトルです。IV モードは **plain** または **essiv:hash** です。**-plain** の **ivmode** は、セクター番号 (および IV オフセット) を IV として使用します。**-essiv** の **ivmode** はウォーターマークの弱点を回避するための機能強化です。

**key**

暗号化キー (16 進数で指定)

**IV-offset**

初期ベクトル (IV) オフセット

**device**

ブロックデバイス。ファイルシステム内のデバイス名で参照、または **major:minor** の形式のメジャー番号とマイナー番号で参照されます。

**offset**

デバイス上のマッピングの始点オフセット

以下に **crypt** ターゲットの例を示します。

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

## A.2. DMSETUP コマンド

**dmsetup** コマンドはデバイスマッパーと通信するためのコマンドラインラッパーです。LVM デバイスに関する全般的なシステム情報については、以下のセクションで説明されているように **dmsetup** コマンドの **info**、**ls**、**status**、および **deps** オプションの使用が役に立ちます。

**dmsetup** コマンドのその他のオプションとその機能に関する情報は、**dmsetup(8)** の man ページを参照してください。

### A.2.1. dmsetup info コマンド

**dmsetup info device** コマンドは デバイスマッパーデバイスに関する要約情報を提供します。デバイス名を指定しないと、その出力は現在設定されているすべてのデバイスマッパーデバイスに関する情報となります。デバイスを 1 つ指定すると、このコマンドはそのデバイスのみについて情報を出力します。

**dmsetup info** コマンドは以下のカテゴリーで情報を提供します。

**Name**

デバイスの名前です。LVM デバイスは、ハイフンで区切られたボリュームグループ名と論理ボリューム名で表現されます。元の名前のハイフンは2つのハイフンに変換されます。標準的な LVM 操作時は、LVM デバイスを直接指定するために、この形式で LVM デバイスの名前を使用することはできず、代わりに代替名の `vg/lv` を使用する必要があります。

## State

使用可能なデバイスの状態 (state) は、**SUSPENDED**、**ACTIVE**、および **READ-ONLY** です。**dmsetup suspend** コマンドはデバイスを **SUSPENDED** の状態にします。デバイスが一時停止 (SUSPENDED) になっている場合、そのデバイスへのすべての I/O 操作は停止します。**dmsetup resume** コマンドはそのデバイスの状態を **ACTIVE** に復元します。

## Read Ahead

読み取り操作が実行されている間に開かれているファイルを対象にシステムが先読みをするデータブロックの数です。デフォルトでカーネルは適切な値を自動選択します。この値を変更するには、**dmsetup** コマンドの **--readahead** オプションを使用します。

## Tables present

このカテゴリで利用できる状態は **LIVE** と **INACTIVE** です。**INACTIVE** 状態は、テーブルの状態が **LIVE** となるように **dmsetup resume** コマンドがデバイス状態を **ACTIVE** に復元する際に入れ替えられるテーブルがロードされていることを示します。詳細は **dmsetup man** ページを参照してください。

## Open count

open reference count はデバイスが開かれた回数を示します。**mount** コマンドはデバイスを開きます。

## Event number

現在のイベントの受信数。**dmsetup wait n** コマンドを発行すると、n 番目のイベントが受理されるまでコールをブロックして待機します。

## Major, minor

メジャーとマイナーのデバイス番号

## Number of targets

デバイスを構成するフラグメントの数です。たとえば、リニアデバイスが3つのディスクにまたがる場合は、ターゲットは3つになります。リニアデバイスがディスクの先頭と終点で構成され、中間を持たない場合は、ターゲットは2つになります。

## UUID

デバイスの UUID

以下の例では、**dmsetup info** コマンド用の一部の出力を示しています。

```
# 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-t0cS1kqFV9drb0X1Vr8sxeYP0tqcrpdegyqj5lZxe45JMGlmvvtqLmbLpBcenh2L3

```

### A.2.2. dmsetup ls コマンド

マップされたデバイスのデバイス名は、**dmsetup ls** コマンドで一覧表示できます。指定タイプのターゲットを1つ以上持つデバイスは、**dmsetup ls --target *target\_type*** コマンドで一覧表示できます。その他の **dmsetup ls** コマンドオプションについては、**dmsetup man** ページを参照してください。

以下の例は、現在設定されているマップ済みデバイスのデバイス名を一覧表示するコマンドを示します。

```

# dmsetup ls
testgfsvg-testgfslv3    (253:4)
testgfsvg-testgfslv2    (253:3)
testgfsvg-testgfslv1    (253:2)
VolGroup00-LogVol01     (253:1)
VolGroup00-LogVol00     (253:0)

```

以下の例は、現在設定されているミラーマッピングのデバイス名を一覧表示するコマンドを示します。

```

# 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)

```

マルチパスまたはその他のデバイスマッパーデバイス上にスタックされる LVM 設定は複雑でわかりにくい場合があります。**dmsetup ls** のコマンドには、以下に示す例のように、デバイス間の依存関係をツリーで表示する **--tree** オプションがあります。

```

# dmsetup ls --tree
vgtest-lvmir (253:13)
├─vgtest-lvmir_mimage_1 (253:12)
│   └─mpathep1 (253:8)
│       └─mpathe (253:5)

```

```

├── (8:112)
│   └── (8:64)
├── vgtest-lvmir_mimage_0 (253:11)
│   ├── mpathcp1 (253:3)
│   │   └── mpathc (253:2)
│   │       ├── (8:32)
│   │       └── (8:16)
├── vgtest-lvmir_mlog (253:4)
│   ├── mpathfp1 (253:10)
│   │   └── mpathf (253:6)
│   │       ├── (8:128)
│   │       └── (8:80)

```

### A.2.3. dmsetup status コマンド

**dmsetup status *device*** コマンドは、指定したデバイスにある各ターゲットの状態情報を提供します。デバイス名を指定しないと、現在設定されているすべてのデバイスマッパーデバイスに関する情報が出力されます。**dmsetup status --target *target\_type*** コマンドを使用すると、指定した 1 つのタイプのターゲットを 1 つ以上持つデバイスの状態だけを表示できます。

以下の例は、現在設定されているすべてのマップ済みデバイス内のターゲットの状態を一覧表示するコマンドを示しています。

```

# dmsetup status
testgfsvg-testgfslv3: 0 312352768 linear
testgfsvg-testgfslv2: 0 312352768 linear
testgfsvg-testgfslv1: 0 312352768 linear
testgfsvg-testgfslv1: 312352768 50331648 linear
VolGroup00-LogVol01: 0 4063232 linear
VolGroup00-LogVol00: 0 151912448 linear

```

### A.2.4. dmsetup deps コマンド

**dmsetup deps *device*** コマンドは、指定デバイスのマッピングテーブルが参照する、デバイス用の (メジャー/マイナー) ペアの一覧を提供します。デバイス名を指定しないと、その出力は現在設定されているすべてのデバイスマッパーデバイスに関する情報になります。

以下の例は、現在設定されているマップ済みデバイスのすべての依存関係を一覧表示するコマンドを示しています。

```

# dmsetup deps
testgfsvg-testgfslv3: 1 dependencies      : (8, 16)
testgfsvg-testgfslv2: 1 dependencies      : (8, 16)
testgfsvg-testgfslv1: 1 dependencies      : (8, 16)
VolGroup00-LogVol01: 1 dependencies      : (8, 2)
VolGroup00-LogVol00: 1 dependencies      : (8, 2)

```

以下の例では、デバイス **lock\_stress-grant--02.1722** の依存関係のみを一覧表示するコマンドを示しています。

```

# dmsetup deps lock_stress-grant--02.1722
3 dependencies : (253, 33) (253, 32) (253, 31)

```

## A.3. デバイスマッパーの UDEV デバイスマネージャサポート

**udev** デバイスマネージャーの主な役割は、**/dev** ディレクトリー内でノードを設定する動的な手段を提供することです。これらのノードの作成は、ユーザースペースで **udev** ルールを適用することによって実行されます。これらのルールは、特定のデバイスを追加、削除、または変更した結果、カーネルから直接送信される **udev** イベント上で処理されます。これはホットプラグサポートの便利な中央メカニズムを提供します。

**udev** デバイスマネージャーは、実際のノードを作成するだけでなく、ユーザーによる名前付けが可能なシンボリックリンクも作成できます。これによりユーザーは、独自のカスタマイズされた名前付け、および **/dev** ディレクトリーのディレクトリー構造を必要に応じて自由に選択できるようになります。

それぞれの **udev** イベントには、処理されるデバイスに関する基本情報が含まれます。これには、デバイスの名前、デバイスが属するサブシステム、デバイスのタイプ、使用されているメジャーとマイナーの番号、イベントのタイプなどが含まれます。**udev** ルール内でもアクセス可能な **/sys** ディレクトリー内のすべての情報にアクセスする可能性があることを考慮すると、ユーザーはこの情報に基づく単純なフィルターを利用し、この情報に基づいて条件付きでルールを実行することができます。

**udev** デバイスマネージャーは、ノードのパーミッション設定の一元化された方法も提供します。ユーザーはカスタマイズされたルールセットを簡単に追加し、イベント処理中に入手可能な情報のいずれかのビットで指定する、任意のデバイスのパーミッションを定義することができます。

**udev** ルール内にプログラムのフックを直接追加することも可能です。**udev** デバイスマネージャーは、これらのプログラムを呼び出して、イベントを処理するために必要とされる追加処理を行うことができます。さらにプログラムは、この処理の結果として、環境変数をエクスポートすることもできます。追加の情報源として、任意の結果をルール内で追加で 사용할 ことが可能です。

**udev** ライブラリーを使用するソフトウェアは、入手可能なすべての情報とともに、**udev** イベントを受信し、処理することができます。このため、処理は、**udev** デーモンのみにバインドされません。

### A.3.1. udev のデバイスマッパーとの統合

デバイスマッパーは **udev** 統合に対して直接のサポートを提供します。これによって、デバイスマッパーは、LVM デバイスを 含むデバイスマッパーデバイスに関連したすべての **udev** 処理と同期します。**udev** デーモンにルールを適用する方式は、デバイスの変更元であるプログラム (**dmsetup** や LVM など) を使用した並列処理であるため、同期が必要です。このサポートがなかったため、前回の変更イベントの結果、引き続きオープンな **udev** ルールで処理されたデバイスをユーザーが削除しようとする問題が頻繁に発生していました。この問題は、デバイスに対する変更の間隔が短い場合にとくに多く発生していました。

Red Hat Enterprise Linux は、一般的なデバイスマッパーデバイスおよび LVM 向けの **udev** ルールを正式にサポートしています。表A.1「[デバイスマッパーデバイス向けの udev ルール](#)」は、**/lib/udev/rules.d** にインストールされているこれらのルールについてまとめています。

表A.1 デバイスマッパーデバイス向けの udev ルール

ファイル名	説明
-------	----



ファイル名	説明
<b>10-dm.rules</b>	<p>一般的なデバイスマッパールールを格納し、<b>/dev/mapper</b> 内に <b>/dev/dm-N</b> をターゲットとするシンボリックリンクを作成します。ここで N は、カーネルによってデバイスに動的に割り当てられる数です (<b>/dev/dm-N</b> はノードです)。</p> <p>注意: <b>/dev/dm-N</b> ノードは、デバイスにアクセスするスクリプトには <b>決して</b> 使用されるべきではありません。N の数は動的に割り当てられ、デバイスがどのように有効化される順序とともに変化します。したがって、<b>/dev/mapper</b> ディレクトリー内の真の名前を使用すべきです。このレイアウトは、ノード/シンボリックリンクが作成されるべき方法の要件をサポートします。</p>
<b>11-dm-lvm.rules</b>	<p>LVM デバイス用に適用されるルールを格納し、ボリュームグループの論理ボリュームのシンボリックリンクを作成します。このシンボリックリンクは、<b>/dev/vgname</b> ディレクトリーに、<b>/dev/dm-N</b> をターゲットとして作成されます。</p> <p>注意: デバイスマッパーサブシステムの今後のすべての命名基準に一致させるには、<b>udev</b> ルールは、<b>11-dm-subsystem_name.rules</b> の形式に従う必要があります。<b>udev</b> を提供する <b>libdevmapper</b> ユーザーはいずれも、この基準に従う必要があります。</p>
<b>13-dm-disk.rules</b>	<p>全デバイスマッパーデバイスに適用されるルールを格納し、<b>/dev/disk/by-id</b> ディレクトリーおよび <b>/dev/disk/by-uuid</b> ディレクトリーにシンボリックリンクを作成します。</p>
<b>95-dm-notify.rules</b>	<p><b>libdevmapper</b> を使用する待機中のプロセスを通知するルールを格納します。(LVM や <b>dmsetup</b> と同様)。以前の全ルールが適用された後で通知が行われ、<b>udev</b> 処理が確実に完了するようにします。通知されたプロセスは、その後で再開します。</p>
<b>69-dm-lvm-metad.rules</b>	<p>システムに新たに表示されるブロックデバイスで LVM スキャンをトリガーするためのフックが含まれ、可能な場合は LVM の自動アクティブ化を実行します。これは、<b>lvmetad</b> デーモンをサポートし、<b>lvm.conf</b> ファイルの <b>use_lvmetad=1</b> で設定されます。クラスター環境では、<b>lvmetad</b> デーモンと自動アクティブ化はサポートされません。</p>

**12-dm-permissions.rules** ファイルを用いて、カスタマイズされたパーミッションルールをさらに追加することができます。このファイルは **/lib/udev/rules** ディレクトリーにはインストールされず、**/usr/share/doc/device-mapper-version** ディレクトリーにあります。**12-dm-permissions.rules** ファイルは、パーミッションの設定方法のヒントが記載されたテンプレート

で、一例として取り上げられている一部のマッチングルールをベースとしています。このファイルには、一般的な状況についての例が記載されています。このファイルを編集して、**/etc/udev/rules.d** ディレクトリーに手動で配置すると、アップデート後もそのまま残り、設定がそのまま維持されます。

これらのルールは、イベントの処理中に、他のルールによっても使用可能なすべての基本的な変数を設定します。

以下の変数は、**10-dm.rules** で設定されています。

- **DM\_NAME**: デバイスマッパーデバイスの名前
- **DM\_UUID**: デバイスマッパーデバイスのUUID
- **DM\_SUSPENDED**: デバイスマッパーデバイスの停止状態
- **DM\_UDEV\_RULES\_VSN**: **udev** ルールバージョン (これは主に、前述の変数が正式なデバイスマッパールールによって直接設定されていることを、他すべてのルールが確認するためのものです)

以下の変数は、**11-dm-lvm.rules** で設定されています。

- **DM\_LV\_NAME**: 論理ボリューム名
- **DM\_VG\_NAME**: ボリュームグループ名
- **DM\_LV\_LAYER**: LVM レイヤー名

**12-dm-permissions.rules** ファイルに文書化されているように、これらの変数すべてを**12-dm-permissions.rules** ファイル内で使用して、特定のデバイスマッパーデバイスのパーミッションを定義することができます。

### A.3.2. udev をサポートするコマンドとインターフェース

表A.2「udev をサポートする dmsetup コマンド」には、udev の統合をサポートする **dmsetup** コマンドについてまとめています。

表A.2 udev をサポートする dmsetup コマンド

コマンド	説明
<b>dmsetup udevcomplete</b>	<b>udev</b> がルールの処理を完了し、待機中のプロセスのロックを解除したことを通知するために使用されます ( <b>95-dm-notify.rules</b> 内の <b>udev</b> ルールの中から呼び出されます)。
<b>dmsetup udevcomplete_all</b>	デバッグの目的で使用され、待機中の全プロセスのロックを手動で解除します。
<b>dmsetup udevcookies</b>	デバッグの目的で使用され、既存のすべての Cookie (システム全体のセマフォ) を表示します。
<b>dmsetup udevcreatecookie</b>	Cookie (セマフォ) を手動で作成するのに使用されます。これは、単一の同期リソース下で、より多くのプロセスを実行するのに役立ちます。

コマンド	説明
<b>dmsetup udevreleasecookie</b>	単一の同期 Cookie の下に置かれるすべてのプロセスに関連した、すべての <b>udev</b> 処理を待機するのに使用されます。

**udev** 統合をサポートする **dmsetup** オプションは以下のとおりです。

#### --udevcookie

**udev** トランザクションに追加したいすべての **dmsetup** プロセスを対象に定義する必要があります。これは、**udevcreatecookie** および **udevreleasecookie** と併用されます。

```
COOKIE=$(dmsetup udevcreatecookie)
dmsetup command --udevcookie $COOKIE ....
dmsetup command --udevcookie $COOKIE ....
....
dmsetup command --udevcookie $COOKIE ....
dmsetup udevreleasecookie --udevcookie $COOKIE
```

**--udevcookie** オプションを使用する以外には、プロセスの環境に変数を単にエクスポートできません。

```
export DM_UDEV_COOKIE=$(dmsetup udevcreatecookie)
dmsetup command ...
dmsetup command ...
...
dmsetup command ...
```

#### --noudevrules

**udev** ルールを無効にします。ノード/シンボリックリンクは **libdevmapper** によって作成されます (以前の方法)。このオプションは、**udev** が適切に機能しない場合のデバッグを目的としています。

#### --noudevsync

**udev** の同期を無効にします。これもデバッグを目的としています。

**dmsetup** コマンドとそのオプションに関する情報は、**dmsetup(8)** の man ページを参照してください。

LVM コマンドは、**udev** の統合に対応した以下のオプションをサポートします。

- **--noudevrules**: **dmsetup** コマンドについて、**udev** ルールを無効にします。
- **--noudevsync**: **dmsetup** コマンドについて、**udev** 同期を無効にします。

**lvm.conf** ファイルには、**udev** の統合をサポートする以下のオプションが含まれます。

- **udev\_rules**: すべての LVM2 コマンドを対象に **udev\_rules** をグローバルに有効/無効にします。
- **udev\_sync**: すべての LVM コマンドを対象に **udev** 同期をグローバルに有効/無効にします。

**lvm.conf** ファイルオプションの詳細は、**lvm.conf** ファイルのインラインコメントを参照してください。

## 付録B LVM 設定ファイル

LVM は複数の設定ファイルに対応しています。システム起動時に **lvm.conf** 設定ファイルが、環境変数 **LVM\_SYSTEM\_DIR** によって指定されたディレクトリーからロードされます。このディレクトリーはデフォルトでは **/etc/lvm** に設定されます。

**lvm.conf** ファイルはロードする追加の設定ファイルを指定できます。最新のファイルの設定は、以前のファイルの設定を上書きします。すべての設定ファイルをロードした後に、使用中の設定を表示するには、**lvmconfig** コマンドを実行します。

追加の設定ファイルのロードに関する情報は、「[ホストタグ](#)」を参照してください。

### B.1. LVM 設定ファイル

LVM 設定に使用されるファイルは以下のとおりです。

#### **/etc/lvm/lvm.conf**

ツールで読み込まれる中央設定ファイル

#### **etc/lvm/lvm\_hosttag.conf**

各ホストタグについて、**lvm\_hosttag.conf** という追加の設定ファイルが読み込まれます (存在する場合)。そのファイルが新規のタグを定義する場合、追加の設定ファイルが読み取られるようにファイルの一覧に追加されます。ホストタグに関する情報は、「[ホストタグ](#)」を参照してください。

LVM 設定ファイルのほかにも、LVM を実行しているシステムには LVM システムセットアップに影響する以下のようなファイルが含まれます。

#### **/etc/lvm/cache/.cache**

デバイス名フィルターキャッシュファイル (設定可能)

#### **/etc/lvm/backup/**

ボリュームグループメタデータの自動バックアップ用ディレクトリー (設定可能)

#### **/etc/lvm/archive/**

ボリュームグループメタデータの自動アーカイブ用ディレクトリー (ディレクトリーパスとアーカイブ履歴の範囲に関する設定が可能)

#### **/var/lock/lvm/**

単一ホストの設定では、並行ツールの実行によってメタデータの破損を防止するロックファイルが使用され、クラスターでは、クラスター全域の DLM が使用されます。

### B.2. LVMCONFIG コマンド

**lvmconfig** コマンドを使用して、現在の LVM 設定を表示したり、設定をファイルに保存したりすることができます。**lvmconfig** コマンドは、以下の機能を含むさまざまな機能を提供します。

- 任意のタグ設定ファイルとマージした現在の lvm 設定をダンプできます。

- 値がデフォルトと異なる現在の設定すべてをダンプできます。
- 現在の LVM バージョンに導入されたすべての新規設定を特定の LVM バージョンにダンプできます。
- コマンドおよびメタデータプロファイルに対して、すべての設定を全体としてまたは個別にダンプできます。LVM プロファイルについては、[「LVM プロファイル」](#) を参照してください。
- 特定バージョンの LVM の設定のみをダンプできます。
- 現在の設定を検証することができます。

サポートされている機能の完全な一覧と `lvmconfig` オプションの指定方法については、`lvmconfig man` ページを参照してください。

### B.3. LVM プロファイル

LVM プロファイルは、各種の環境または使用において一部の特性を実現するために使用できる選択されたカスタマイズ可能な設定のセットです。通常、プロファイルには該当する環境または使用を反映する名前が付けられます。LVM プロファイルは既存の設定を上書きします。

LVM が認識する LVM プロファイルには、コマンドプロファイルとメタデータプロファイルの 2 つのグループがあります。

- コマンドプロファイルは、グローバルな LVM コマンドレベルで選択された設定を上書きするために使用されます。このプロファイルは LVM コマンドの実行開始時に適用され、LVM コマンドの実行中に使用されます。コマンドプロファイルは、LVM コマンドの実行時に `--commandprofile ProfileName` オプションを指定することによって適用します。
- メタデータプロファイルは、ボリュームグループ/論理ボリュームレベルで選択された設定を上書きするために使用されます。このプロファイルは、処理される各ボリュームグループ/論理グループについて個別に適用されます。そのため、各ボリュームグループ/論理ボリュームは、そのメタデータで使用するプロファイル名を保存でき、ボリュームグループ/論理ボリュームが次に処理される際には、該当プロファイルが自動的に適用されます。ボリュームグループとその論理ボリュームのいずれかに異なるプロファイルが定義されている場合は、論理ボリュームに定義されるプロファイルが優先されます。
  - `vgcreate` または `lvcreate` コマンドを使用してボリュームグループまたは論理ボリュームを作成する場合、`--metadataprofile ProfileName` オプションを指定して、メタデータプロファイルをボリュームグループまたは論理グループに割り当てることができます。
  - 既存のボリュームグループまたは論理グループへのメタデータプロファイルの割り当てまたは割り当て解除は、`lvchange` または `vgchange` コマンドの `--metadataprofile ProfileName` または `--detachprofile` オプションを指定して実行できます。
  - `vgs` および `lvs` コマンドの `-o vg_profile` および `-o lv_profile` 出力オプションを指定することにより、ボリュームグループまたは論理ボリュームに現在割り当てられているメタデータプロファイルを表示できます。

コマンドプロファイルに許可される一連のオプションとメタデータプロファイルに許可される一連のオプションは相互に排他的です。これらの 2 つのセットのいずれかに属する設定を他方の設定に混在させることができず、LVM ツールはこのように混在したプロファイルを拒否します。

LVM はいくつかの事前に定義された設定プロファイルを提供します。LVM プロファイルはデフォルトで `/etc/lvm/profile` ディレクトリーに保存されます。このロケーション

は、`/etc/lvm/lvm.conf` ファイルの `profile_dir` 設定を使用して変更できます。それぞれのプロファイル設定は、`profile` ディレクトリーの `ProfileName.profile` file に保存されます。LVM コマンドでプロファイルを参照する場合、`.profile` という接尾辞は省略されます。

複数の異なる値で追加のプロファイルを作成することができます。このために、LVM は `command_profile_template.profile` ファイル (コマンドプロファイル用) および `metadata_profile_template.profile` ファイル (メタデータプロファイル用) を提供します。これらのファイルには、それぞれのタイプのプロファイルでカスタマイズできるすべての設定が含まれます。これらのテンプレートプロファイルは、随時コピーし、編集することができます。

または、`lvmconfig` コマンドを使用してそれぞれのプロファイルタイプのプロファイルファイルの指定セクションについて新規プロファイルを生成できます。以下のコマンドは、`section` の設定で構成される `ProfileName.profile` という名前の新規コマンドプロファイルを作成します。

```
lvmconfig --file ProfileName.profile --type profilable-command section
```

以下のコマンドは、`section` の設定で構成される `ProfileName.profile` という名前の新規メタデータプロファイルを作成します。

```
lvmconfig --file ProfileName.profile --type profilable-metadata section
```

セクションが指定されない場合は、プロファイルでカスタマイズできるすべての設定がレポートされます。

## B.4. サンプル LVM.CONF ファイル

`lvm.conf` 設定ファイルのサンプルを以下に示します。ご使用のデフォルト設定とは若干異なる可能性があります。



### 注記

以下のコマンドを実行すると、すべての値にデフォルトが設定され、コメントが含まれる `lvm.conf` ファイルを生成できます。

```
lvmconfig --type default --withcomments
```

```
# 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.
#
# Refer to 'man lvm.conf' for information about how settings configured in
# this file are combined with built-in values and command line options to
# arrive at the final values used by LVM.
#
# Refer to 'man lvmconfig' for information about displaying the built-in
# and configured values used by LVM.
#
# If a default value is set in this file (not commented out), then a
```

```
# new version of LVM using this file will continue using that value,
# even if the new version of LVM changes the built-in default value.
#
# 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.

# Configuration section config.
# How LVM configuration settings are handled.
config {

    # Configuration option config/checks.
    # If enabled, any LVM configuration mismatch is reported.
    # This implies checking that the configuration key is understood by
    # LVM and that the value of the key is the proper type. If disabled,
    # any configuration mismatch is ignored and the default value is used
    # without any warning (a message about the configuration key not being
    # found is issued in verbose mode only).
    checks = 1

    # Configuration option config/abort_on_errors.
    # Abort the LVM process if a configuration mismatch is found.
    abort_on_errors = 0

    # Configuration option config/profile_dir.
    # Directory where LVM looks for configuration profiles.
    profile_dir = "/etc/lvm/profile"
}

# Configuration section devices.
# How LVM uses block devices.
devices {

    # Configuration option devices/dir.
    # Directory in which to create volume group device nodes.
    # Commands also accept this as a prefix on volume group names.
    # This configuration option is advanced.
    dir = "/dev"

    # Configuration option devices/scan.
    # Directories containing device nodes to use with LVM.
    # This configuration option is advanced.
    scan = [ "/dev" ]

    # Configuration option devices/obtain_device_list_from_udev.
    # Obtain the list of available devices from udev.
    # This avoids opening or using any inapplicable non-block devices or
    # subdirectories found in the udev directory. Any device node or
    # symlink not managed by udev in the udev directory is ignored. This
    # setting applies only to the udev-managed device directory; other
    # directories will be scanned fully. LVM needs to be compiled with
    # udev support for this setting to apply.
    obtain_device_list_from_udev = 1
}
```



```

# Configuration option devices/external_device_info_source.
# Select an external device information source.
# Some information may already be available in the system and LVM can
# use this information to determine the exact type or use of devices it
# processes. Using an existing external device information source can
# speed up device processing as LVM does not need to run its own native
# routines to acquire this information. For example, this information
# is used to drive LVM filtering like MD component detection, multipath
# component detection, partition detection and others.
#
# Accepted values:
#   none
#       No external device information source is used.
#   udev
#       Reuse existing udev database records. Applicable only if LVM is
#       compiled with udev support.
#
external_device_info_source = "none"

# Configuration option devices/preferred_names.
# Select which path name to display for a block device.
# If multiple path names exist for a block device, and LVM needs to
# display a name for the device, the path names are matched against
# each item in this list of regular expressions. The first match is
# used. Try to avoid using undescriptive /dev/dm-N names, if present.
# If no preferred name matches, or if preferred_names are not defined,
# the following built-in preferences are applied in order until one
# produces a preferred name:
# Prefer names with path prefixes in the order of:
# /dev/mapper, /dev/disk, /dev/dm-*, /dev/block.
# Prefer the name with the least number of slashes.
# Prefer a name that is a symlink.
# Prefer the path with least value in lexicographical order.
#
# Example
# preferred_names = [ "^/dev/mpath/", "^/dev/mapper/mpath", "^/dev/[hs]d"
]
#
preferred_names = [ "^/dev/mpath/", "^/dev/mapper/mpath", "^/dev/[hs]d" ]

# Configuration option devices/filter.
# Limit the block devices that are used by LVM commands.
# This is a list of regular expressions used to accept or reject block
# device path names. Each regex is delimited by a vertical bar '|'
# (or any character) and is preceded by 'a' to accept the path, or
# by 'r' to reject the path. The first regex in the list to match the
# path is used, producing the 'a' or 'r' result for the device.
# When multiple path names exist for a block device, if any path name
# matches an 'a' pattern before an 'r' pattern, then the device is
# accepted. If all the path names match an 'r' pattern first, then the
# device is rejected. Unmatching path names do not affect the accept
# or reject decision. If no path names for a device match a pattern,
# then the device is accepted. Be careful mixing 'a' and 'r' patterns,
# as the combination might produce unexpected results (test changes.)
# Run vgscan after changing the filter to regenerate the cache.

```

```
# See the use_lvmetad comment for a special case regarding filters.
#
# Example
# Accept every block device:
# filter = [ "a|.*|/" ]
# Reject the cdrom drive:
# filter = [ "r|/dev/cdrom|" ]
# Work with just loopback devices, e.g. for testing:
# filter = [ "a|loop|", "r|.*|" ]
# Accept all loop devices and ide drives except hdc:
# filter = [ "a|loop|", "r|/dev/hdc|", "a|/dev/ide|", "r|.*|" ]
# Use anchors to be very specific:
# filter = [ "a|^/dev/hda8$|", "r|.*|/" ]
#
# This configuration option has an automatic default value.
# filter = [ "a|.*|/" ]

# Configuration option devices/global_filter.
# Limit the block devices that are used by LVM system components.
# Because devices/filter may be overridden from the command line, it is
# not suitable for system-wide device filtering, e.g. udev and lvmetad.
# Use global_filter to hide devices from these LVM system components.
# The syntax is the same as devices/filter. Devices rejected by
# global_filter are not opened by LVM.
# This configuration option has an automatic default value.
# global_filter = [ "a|.*|/" ]

# Configuration option devices/cache_dir.
# Directory in which to store the device cache file.
# The results of filtering are cached on disk to avoid rescanning dud
# devices (which can take a very long time). By default this cache is
# stored in a file named .cache. It is safe to delete this file; the
# tools regenerate it. If obtain_device_list_from_udev is enabled, the
# list of devices is obtained from udev and any existing .cache file
# is removed.
cache_dir = "/etc/lvm/cache"

# Configuration option devices/cache_file_prefix.
# A prefix used before the .cache file name. See devices/cache_dir.
cache_file_prefix = ""

# Configuration option devices/write_cache_state.
# Enable/disable writing the cache file. See devices/cache_dir.
write_cache_state = 1

# Configuration option devices/types.
# List of additional acceptable block device types.
# These are of device type names from /proc/devices, followed by the
# maximum number of partitions.
#
# Example
# types = [ "fd", 16 ]
#
# This configuration option is advanced.
# This configuration option does not have a default value defined.
```

```
# Configuration option devices/sysfs_scan.
# Restrict device scanning to block devices appearing in sysfs.
# This is a quick way of filtering out block devices that are not
# present on the system. sysfs must be part of the kernel and mounted.)
sysfs_scan = 1

# Configuration option devices/multipath_component_detection.
# Ignore devices that are components of DM multipath devices.
multipath_component_detection = 1

# Configuration option devices/md_component_detection.
# Ignore devices that are components of software RAID (md) devices.
md_component_detection = 1

# Configuration option devices/fw_raid_component_detection.
# Ignore devices that are components of firmware RAID devices.
# LVM must use an external_device_info_source other than none for this
# detection to execute.
fw_raid_component_detection = 0

# Configuration option devices/md_chunk_alignment.
# Align PV data blocks with md device's stripe-width.
# This applies if a PV is placed directly on an md device.
md_chunk_alignment = 1

# Configuration option devices/default_data_alignment.
# Default alignment of the start of a PV data area in MB.
# If set to 0, a value of 64KiB will be used.
# Set to 1 for 1MiB, 2 for 2MiB, etc.
# This configuration option has an automatic default value.
default_data_alignment = 1

# Configuration option devices/data_alignment_detection.
# Detect PV data alignment based on sysfs device information.
# The start of a PV data area will be a multiple of minimum_io_size or
# optimal_io_size exposed in sysfs. minimum_io_size is the smallest
# request the device can perform without incurring a read-modify-write
# penalty, e.g. MD chunk size. optimal_io_size is the device's
# preferred unit of receiving I/O, e.g. MD 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.
data_alignment_detection = 1

# Configuration option devices/data_alignment.
# Alignment of the start of a PV data area in KiB.
# If a PV is placed directly on an md device and md_chunk_alignment or
# data_alignment_detection are enabled, then this setting is ignored.
# Otherwise, md_chunk_alignment and data_alignment_detection are
# disabled if this is set. Set to 0 to use the default alignment or the
# page size, if larger.
data_alignment = 0

# Configuration option devices/data_alignment_offset_detection.
# Detect PV data alignment offset based on sysfs device information.
# The start of a PV aligned data area will be shifted by the
```

```
# alignment_offset exposed in sysfs. This offset is often 0, but may
# be non-zero. Certain 4KiB sector drives that compensate for windows
# partitioning will have an alignment_offset of 3584 bytes (sector 7
# is the lowest aligned logical block, the 4KiB sectors start at
# LBA -1, and consequently sector 63 is aligned on a 4KiB boundary).
# pvcreate --dataalignmentoffset will skip this detection.
data_alignment_offset_detection = 1

# Configuration option devices/ignore_suspended_devices.
# Ignore DM devices that have I/O suspended while scanning devices.
# Otherwise, LVM waits for a suspended device to become accessible.
# This should only be needed in recovery situations.
ignore_suspended_devices = 0

# Configuration option devices/ignore_lvm_mirrors.
# Do not scan 'mirror' LVs to avoid possible deadlocks.
# This avoids possible deadlocks when using the 'mirror' segment type.
# This setting determines whether LVs using the 'mirror' segment type
# are scanned for LVM labels. This affects the ability of mirrors to
# be used as physical volumes. If this setting is enabled, it is
# impossible to create VGs on top of mirror LVs, i.e. to stack VGs on
# mirror LVs. If this setting is disabled, allowing mirror LVs to be
# scanned, it may cause LVM processes and I/O to the mirror to become
# blocked. This is due to the way that the mirror segment type handles
# failures. In order for the hang to occur, an LVM command must be run
# just after a failure and before the automatic LVM repair process
# takes place, or there must be failures in multiple mirrors in the
# same VG at the same time with write failures occurring moments before
# a scan of the mirror's labels. The 'mirror' scanning problems do not
# apply to LVM RAID types like 'raid1' which handle failures in a
# different way, making them a better choice for VG stacking.
ignore_lvm_mirrors = 1

# Configuration option devices/disable_after_error_count.
# Number of I/O errors after which a device is skipped.
# During each LVM operation, errors received from each device are
# counted. If the counter of a device exceeds the limit set here,
# no further I/O is sent to that device for the remainder of the
# operation. Setting this to 0 disables the counters altogether.
disable_after_error_count = 0

# Configuration option devices/require_restorefile_with_uuid.
# Allow use of pvcreate --uuid without requiring --restorefile.
require_restorefile_with_uuid = 1

# Configuration option devices/pv_min_size.
# Minimum size in KiB of block devices which can be used as PVs.
# In a clustered environment all nodes must use the same value.
# Any value smaller than 512KiB is ignored. The previous built-in
# value was 512.
pv_min_size = 2048

# Configuration option devices/issue_discards.
# Issue discards to PVs that are no longer used by an LV.
# Discards are sent to an LV's underlying physical volumes when the LV
# is no longer using the physical volumes' space, e.g. lvremove,
```

```

# lvreduce. Discards inform the storage that a region is no longer
# used. 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 enabled, discards will only be issued if both the
# storage and kernel provide support.
issue_discards = 0

# Configuration option devices/allow_changes_with_duplicate_pvs.
# Allow VG modification while a PV appears on multiple devices.
# When a PV appears on multiple devices, LVM attempts to choose the
# best device to use for the PV. If the devices represent the same
# underlying storage, the choice has minimal consequence. If the
# devices represent different underlying storage, the wrong choice
# can result in data loss if the VG is modified. Disabling this
# setting is the safest option because it prevents modifying a VG
# or activating LVs in it while a PV appears on multiple devices.
# Enabling this setting allows the VG to be used as usual even with
# uncertain devices.
allow_changes_with_duplicate_pvs = 0
}

# Configuration section allocation.
# How LVM selects space and applies properties to LVs.
allocation {

# Configuration option allocation/cling_tag_list.
# Advise LVM which PVs to use when searching for new space.
# 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.
#
# Example
# Use the special tag "@" as a wildcard to match any PV tag:
# cling_tag_list = [ "@" ]
# LVs are mirrored between two sites within a single VG, and
# PVs are tagged with either @site1 or @site2 to indicate where
# they are situated:
# cling_tag_list = [ "@site1", "@site2" ]
#
# This configuration option does not have a default value defined.

# Configuration option allocation/maximise_cling.
# Use a previous allocation algorithm.
# 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. This setting can be used to disable the changes
# and revert to the previous algorithm.
maximise_cling = 1

# Configuration option allocation/use_blkid_wiping.
# Use blkid to detect existing signatures on new PVs and LVs.

```

```
# The blkid library can detect more signatures than the native LVM
# detection code, but may take longer. LVM needs to be compiled with
# blkid wiping support for this setting to apply. LVM native detection
# code is currently able to recognize: MD device signatures,
# swap signature, and LUKS signatures. To see the list of signatures
# recognized by blkid, check the output of the 'blkid -k' command.
use_blkid_wiping = 1

# Configuration option allocation/wipe_signatures_when_zeroing_new_lvs.
# Look for and erase any signatures while zeroing a new LV.
# The --wipesignatures option overrides this setting.
# Zeroing is controlled by the -Z/--zero option, and if not specified,
# zeroing is used by default if possible. Zeroing simply overwrites the
# first 4KiB of a new LV with zeroes and does no signature detection or
# wiping. Signature wiping goes beyond zeroing and detects exact types
# and positions of signatures within the whole LV. It provides a
# cleaner LV after creation as all known signatures are wiped. The LV
# is not claimed incorrectly by other tools because of old signatures
# from previous use. The number of signatures that LVM can detect
# depends on the detection code that is selected (see
# use_blkid_wiping.) Wiping each detected signature must be confirmed.
# When this setting is disabled, signatures on new LVs are not detected
# or erased unless the --wipesignatures option is used directly.
wipe_signatures_when_zeroing_new_lvs = 1

# Configuration option allocation/mirror_logs_require_separate_pvs.
# Mirror logs and images will always use different PVs.
# The default setting changed in version 2.02.85.
mirror_logs_require_separate_pvs = 0

# Configuration option allocation/raid_stripe_all_devices.
# Stripe across all PVs when RAID stripes are not specified.
# If enabled, all PVs in the VG or on the command line are used for
raid0/4/5/6/10
# when the command does not specify the number of stripes to use.
# This was the default behaviour until release 2.02.162.
# This configuration option has an automatic default value.
# raid_stripe_all_devices = 0

# Configuration option
allocation/cache_pool_metadata_require_separate_pvs.
# Cache pool metadata and data will always use different PVs.
cache_pool_metadata_require_separate_pvs = 0

# Configuration option allocation/cache_mode.
# The default cache mode used for new cache.
#
# Accepted values:
#   writethrough
#     Data blocks are immediately written from the cache to disk.
#   writeback
#     Data blocks are written from the cache back to disk after some
#     delay to improve performance.
#
# This setting replaces allocation/cache_pool_cachemode.
# This configuration option has an automatic default value.
```

```

# cache_mode = "writethrough"

# Configuration option allocation/cache_policy.
# The default cache policy used for new cache volume.
# Since kernel 4.2 the default policy is smq (Stochastic multique),
# otherwise the older mq (Multiqueue) policy is selected.
# This configuration option does not have a default value defined.

# Configuration section allocation/cache_settings.
# Settings for the cache policy.
# See documentation for individual cache policies for more info.
# This configuration section has an automatic default value.
# cache_settings {
# }

# Configuration option allocation/cache_pool_chunk_size.
# The minimal chunk size in KiB for cache pool volumes.
# Using a chunk_size that is too large can result in wasteful use of
# the cache, where small reads and writes can cause large sections of
# an LV to be mapped into the cache. However, choosing a chunk_size
# that is too small can result in more overhead trying to manage the
# numerous chunks that become mapped into the cache. The former is
# more of a problem than the latter in most cases, so the default is
# on the smaller end of the spectrum. Supported values range from
# 32KiB to 1GiB in multiples of 32.
# This configuration option does not have a default value defined.

# Configuration option
allocation/thin_pool_metadata_require_separate_pvs.
# Thin pool metadata and data will always use different PVs.
thin_pool_metadata_require_separate_pvs = 0

# Configuration option allocation/thin_pool_zero.
# Thin pool data chunks are zeroed before they are first used.
# Zeroing with a larger thin pool chunk size reduces performance.
# This configuration option has an automatic default value.
# thin_pool_zero = 1

# Configuration option allocation/thin_pool_discards.
# The discards behaviour of thin pool volumes.
#
# Accepted values:
#   ignore
#   nopassdown
#   passdown
#
# This configuration option has an automatic default value.
# thin_pool_discards = "passdown"

# Configuration option allocation/thin_pool_chunk_size_policy.
# The chunk size calculation policy for thin pool volumes.
#
# Accepted values:
#   generic
#       If thin_pool_chunk_size is defined, use it. Otherwise, calculate
#       the chunk size based on estimation and device hints exposed in

```

```
# sysfs - the minimum_io_size. The chunk size is always at least
# 64KiB.
# performance
# If thin_pool_chunk_size is defined, use it. Otherwise, calculate
# the chunk size for performance based on device hints exposed in
# sysfs - the optimal_io_size. The chunk size is always at least
# 512KiB.
#
# This configuration option has an automatic default value.
# thin_pool_chunk_size_policy = "generic"

# Configuration option allocation/thin_pool_chunk_size.
# The minimal chunk size in KiB for thin pool volumes.
# Larger chunk sizes may improve performance for plain thin volumes,
# however using them for snapshot volumes is less efficient, as it
# consumes more space and takes extra time for copying. When unset,
# lvm tries to estimate chunk size starting from 64KiB. Supported
# values are in the range 64KiB to 1GiB.
# This configuration option does not have a default value defined.

# Configuration option allocation/physical_extent_size.
# Default physical extent size in KiB to use for new VGs.
# This configuration option has an automatic default value.
# physical_extent_size = 4096
}

# Configuration section log.
# How LVM log information is reported.
log {

# Configuration option log/report_command_log.
# Enable or disable LVM log reporting.
# If enabled, LVM will collect a log of operations, messages,
# per-object return codes with object identification and associated
# error numbers (errno) during LVM command processing. Then the
# log is either reported solely or in addition to any existing
# reports, depending on LVM command used. If it is a reporting command
# (e.g. pvs, vgs, lvs, lvm fullreport), then the log is reported in
# addition to any existing reports. Otherwise, there's only log report
# on output. For all applicable LVM commands, you can request that
# the output has only log report by using --logonly command line
# option. Use log/command_log_cols and log/command_log_sort settings
# to define fields to display and sort fields for the log report.
# You can also use log/command_log_selection to define selection
# criteria used each time the log is reported.
# This configuration option has an automatic default value.
# report_command_log = 0

# Configuration option log/command_log_sort.
# List of columns to sort by when reporting command log.
# See <lvm command> --logonly --configreport log -o help
# for the list of possible fields.
# This configuration option has an automatic default value.
# command_log_sort = "log_seq_num"

# Configuration option log/command_log_cols.
```



```

# List of columns to report when reporting command log.
# See <lvm command> --logonly --configreport log -o help
# for the list of possible fields.
# This configuration option has an automatic default value.
# command_log_cols =
"log_seq_num,log_type,log_context,log_object_type,log_object_name,log_object_id,log_object_group,log_object_group_id,log_message,log_errno,log_ret_code"

# Configuration option log/command_log_selection.
# Selection criteria used when reporting command log.
# You can define selection criteria that are applied each
# time log is reported. This way, it is possible to control the
# amount of log that is displayed on output and you can select
# only parts of the log that are important for you. To define
# selection criteria, use fields from log report. See also
# <lvm command> --logonly --configreport log -S help for the
# list of possible fields and selection operators. You can also
# define selection criteria for log report on command line directly
# using <lvm command> --configreport log -S <selection criteria>
# which has precedence over log/command_log_selection setting.
# For more information about selection criteria in general, see
# lvm(8) man page.
# This configuration option has an automatic default value.
# command_log_selection = "!(log_type=status && message=success)"

# Configuration option log/verbose.
# Controls the messages sent to stdout or stderr.
verbose = 0

# Configuration option log/silent.
# Suppress all non-essential messages from stdout.
# This has the same effect as -qq. When enabled, the following commands
# still produce output: dumpconfig, lvdisplay, lvmdiskscan, lvs, pvck,
# pvdisplay, pvs, version, vgcfgrestore -l, vgdisplay, vgs.
# Non-essential messages are shifted from log level 4 to log level 5
# for syslog and lvm2_log_fn purposes.
# Any 'yes' or 'no' questions not overridden by other arguments are
# suppressed and default to 'no'.
silent = 0

# Configuration option log/syslog.
# Send log messages through syslog.
syslog = 1

# Configuration option log/file.
# Write error and debug log messages to a file specified here.
# This configuration option does not have a default value defined.

# Configuration option log/overwrite.
# Overwrite the log file each time the program is run.
overwrite = 0

# Configuration option log/level.
# The level of log messages that are sent to the log file 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

# Configuration option log/indent.
# Indent messages according to their severity.
indent = 1

# Configuration option log/command_names.
# Display the command name on each line of output.
command_names = 0

# Configuration option log/prefix.
# A prefix to use before the log message text.
# (After the command name, if selected).
# Two spaces allows you to see/grep the severity of each message.
# To make the messages look similar to the original LVM tools use:
# indent = 0, command_names = 1, prefix = " -- "
prefix = "  "

# Configuration option log/activation.
# Log messages during activation.
# Don't use this in low memory situations (can deadlock).
activation = 0

# Configuration option log/debug_classes.
# Select log messages by class.
# Some debugging messages are assigned to a class and only appear in
# debug output if the class is listed here. Classes currently
# available: memory, devices, activation, allocation, lvmetad,
# metadata, cache, locking, lvmpolld. Use "all" to see everything.
debug_classes = [ "memory", "devices", "activation", "allocation",
"lvmetad", "metadata", "cache", "locking", "lvmpolld", "dbus" ]
}

# Configuration section backup.
# How LVM metadata is backed up and archived.
# In LVM, a 'backup' is a copy of the metadata for the current system,
# and an 'archive' contains old metadata configurations. They are
# stored in a human readable text format.
backup {

# Configuration option backup/backup.
# Maintain a backup of the current metadata configuration.
# Think very hard before turning this off!
backup = 1

# Configuration option backup/backup_dir.
# Location of the metadata backup files.
# Remember to back up this directory regularly!
backup_dir = "/etc/lvm/backup"

# Configuration option backup/archive.
# Maintain an archive of old metadata configurations.
# Think very hard before turning this off.
archive = 1
```

```
# Configuration option backup/archive_dir.
# Location of the metadata archive files.
# Remember to back up this directory regularly!
archive_dir = "/etc/lvm/archive"

# Configuration option backup/retain_min.
# Minimum number of archives to keep.
retain_min = 10

# Configuration option backup/retain_days.
# Minimum number of days to keep archive files.
retain_days = 30
}

# Configuration section shell.
# Settings for running LVM in shell (readline) mode.
shell {

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

# Configuration section global.
# Miscellaneous global LVM settings.
global {

    # Configuration option global/umask.
    # The file creation mask for any files and directories created.
    # Interpreted as octal if the first digit is zero.
    umask = 077

    # Configuration option global/test.
    # No on-disk metadata changes will be made in test mode.
    # Equivalent to having the -t option on every command.
    test = 0

    # Configuration option global/units.
    # Default value for --units argument.
    units = "h"

    # Configuration option global/si_unit_consistency.
    # Distinguish between powers of 1024 and 1000 bytes.
    # The LVM commands distinguish between powers of 1024 bytes,
    # e.g. KiB, MiB, GiB, and powers of 1000 bytes, e.g. KB, MB, GB.
    # If scripts depend on the old behaviour, disable this setting
    # temporarily until they are updated.
    si_unit_consistency = 1

    # Configuration option global/suffix.
    # Display unit suffix for sizes.
    # This setting has no effect if the units are in human-readable form
    # (global/units = "h") in which case the suffix is always displayed.
    suffix = 1

    # Configuration option global/activation.
```

```
# Enable/disable communication with the kernel device-mapper.
# Disable to use the tools to manipulate LVM metadata without
# activating any logical volumes. If the device-mapper driver
# is not present in the kernel, disabling this should suppress
# the error messages.
activation = 1

# Configuration option global/fallback_to_lvm1.
# Try running LVM1 tools if LVM cannot communicate with DM.
# This option only applies to 2.4 kernels and is provided to help
# switch between device-mapper kernels and LVM1 kernels. The LVM1
# tools need to be installed with .lvm1 suffices, e.g. vgscan.lvm1.
# They will stop working once the lvm2 on-disk metadata format is used.
# This configuration option has an automatic default value.
# fallback_to_lvm1 = 1

# Configuration option global/format.
# The default metadata format that commands should use.
# The -M 1|2 option overrides this setting.
#
# Accepted values:
#   lvm1
#   lvm2
#
# This configuration option has an automatic default value.
# format = "lvm2"

# Configuration option global/format_libraries.
# Shared libraries that process different metadata formats.
# If support for LVM1 metadata was compiled as a shared library use
# format_libraries = "liblvm2format1.so"
# This configuration option does not have a default value defined.

# Configuration option global/segment_libraries.
# This configuration option does not have a default value defined.

# Configuration option global/proc.
# Location of proc filesystem.
# This configuration option is advanced.
proc = "/proc"

# Configuration option global/etc.
# Location of /etc system configuration directory.
etc = "/etc"

# Configuration option global/locking_type.
# Type of locking to use.
#
# Accepted values:
#   0
#   Turns off locking. Warning: this risks metadata corruption if
#   commands run concurrently.
#   1
#   LVM uses local file-based locking, the standard mode.
#   2
#   LVM uses the external shared library locking_library.
```

```

# 3
# LVM uses built-in clustered locking with clvmd.
# This is incompatible with lvmetad. If use_lvmetad is enabled,
# LVM prints a warning and disables lvmetad use.
# 4
# LVM uses read-only locking which forbids any operations that
# might change metadata.
# 5
# Offers dummy locking for tools that do not need any locks.
# You should not need to set this directly; the tools will select
# when to use it instead of the configured locking_type.
# Do not use lvmetad or the kernel device-mapper driver with this
# locking type. It is used by the --readonly option that offers
# read-only access to Volume Group metadata that cannot be locked
# safely because it belongs to an inaccessible domain and might be
# in use, for example a virtual machine image or a disk that is
# shared by a clustered machine.
#
locking_type = 3

# Configuration option global/wait_for_locks.
# When disabled, fail if a lock request would block.
wait_for_locks = 1

# Configuration option global/fallback_to_clustered_locking.
# Attempt to use built-in cluster locking if locking_type 2 fails.
# If using external locking (type 2) and initialisation fails, with
# this enabled, an attempt will be made to use the built-in clustered
# locking. Disable this if using a customised locking_library.
fallback_to_clustered_locking = 1

# Configuration option global/fallback_to_local_locking.
# Use locking_type 1 (local) if locking_type 2 or 3 fail.
# If an attempt to initialise type 2 or type 3 locking failed, perhaps
# because cluster components such as clvmd are not running, with this
# enabled, an attempt will be made to use local file-based locking
# (type 1). If this succeeds, only commands against local VGs will
# proceed. VGs marked as clustered will be ignored.
fallback_to_local_locking = 1

# Configuration option global/locking_dir.
# Directory to use for LVM command file locks.
# Local non-LV directory that holds file-based locks while commands are
# in progress. A directory like /tmp that may get wiped on reboot is OK.
locking_dir = "/run/lock/lvm"

# Configuration option global/prioritise_write_locks.
# Allow quicker VG write access during high volume read access.
# When 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. This option only affects
# locking_type 1 viz. local file-based locking.
prioritise_write_locks = 1

```

```
# Configuration option global/library_dir.
# Search this directory first for shared libraries.
# This configuration option does not have a default value defined.

# Configuration option global/locking_library.
# The external locking library to use for locking_type 2.
# This configuration option has an automatic default value.
# locking_library = "liblvm2clusterlock.so"

# Configuration option global/abort_on_internal_errors.
# Abort a command that encounters an internal error.
# 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

# Configuration option global/detect_internal_vg_cache_corruption.
# Internal verification of VG structures.
# Check if CRC matches when a parsed VG is used multiple times. This
# is useful to catch unexpected changes to cached VG structures.
# Please only enable for debugging.
detect_internal_vg_cache_corruption = 0

# Configuration option global/metadata_read_only.
# No operations that change on-disk metadata are 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

# Configuration option global/mirror_segtype_default.
# The segment type used by the short mirroring option -m.
# The --type mirror|raid1 option overrides this setting.
#
# Accepted values:
#   mirror
#       The original RAID1 implementation from LVM/DM. It is
#       characterized by a flexible log solution (core, disk, mirrored),
#       and by the necessity to block I/O while handling a failure.
#       There is an inherent race in the dmeventd failure handling logic
#       with snapshots of devices using this type of RAID1 that in the
#       worst case could cause a deadlock. (Also see
#       devices/ignore_lvm_mirrors.)
#   raid1
#       This is a newer RAID1 implementation using the MD 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,
#       so no separate devices are required.) This mirror
#       implementation does not require I/O to be blocked while
#       handling a failure. This mirror implementation is not
#       cluster-aware and cannot be used in a shared (active/active)
#       fashion in a cluster.
#
mirror_segtype_default = "raid1"
```

```

# Configuration option global/raid10_segtype_default.
# The segment type used by the -i -m combination.
# The --type raid10|mirror option overrides this setting.
# The --stripes/-i and --mirrors/-m options can both be specified
# during the creation of a logical volume to use both striping and
# mirroring for the LV. There are two different implementations.
#
# Accepted values:
#   raid10
#       LVM uses MD's RAID10 personality through DM. This is the
#       preferred option.
#   mirror
#       LVM layers the 'mirror' and 'stripe' segment types. The layering
#       is done by creating a mirror LV on top of striped sub-LVs,
#       effectively creating a RAID 0+1 array. The layering is suboptimal
#       in terms of providing redundancy and performance.
#
raid10_segtype_default = "raid10"

# Configuration option global/sparse_segtype_default.
# The segment type used by the -V -L combination.
# The --type snapshot|thin option overrides this setting.
# The combination of -V and -L options creates a sparse LV. There are
# two different implementations.
#
# Accepted values:
#   snapshot
#       The original snapshot implementation from LVM/DM. It uses an old
#       snapshot that mixes data and metadata within a single COW
#       storage volume and performs poorly when the size of stored data
#       passes hundreds of MB.
#   thin
#       A newer implementation that uses thin provisioning. It has a
#       bigger minimal chunk size (64KiB) and uses a separate volume for
#       metadata. It has better performance, especially when more data
#       is used. It also supports full snapshots.
#
sparse_segtype_default = "thin"

# Configuration option global/lvdisplay_shows_full_device_path.
# Enable this to reinstate the previous lvdisplay name format.
# The default format for displaying LV names in lvdisplay was changed
# in version 2.02.89 to show the LV name and path separately.
# Previously this was always shown as /dev/vgname/lvname even when that
# was never a valid path in the /dev filesystem.
# This configuration option has an automatic default value.
# lvdisplay_shows_full_device_path = 0

# Configuration option global/use_lvmetad.
# Use lvmetad to cache metadata and reduce disk scanning.
# When enabled (and running), lvmetad provides LVM commands with VG
# metadata and PV state. LVM commands then avoid reading this
# information from disks which can be slow. When disabled (or not
# running), LVM commands fall back to scanning disks to obtain VG
# metadata. lvmetad is kept updated via udev rules which must be set
# up for LVM to work correctly. (The udev rules should be installed

```

```
# by default.) Without a proper udev setup, changes in the system's
# block device configuration will be unknown to LVM, and ignored
# until a manual 'pvscan --cache' is run. If lvmetad was running
# while use_lvmetad was disabled, it must be stopped, use_lvmetad
# enabled, and then started. When using lvmetad, LV activation is
# switched to an automatic, event-based mode. In this mode, LVs are
# activated based on incoming udev events that inform lvmetad when
# PVs appear on the system. When a VG is complete (all PVs present),
# it is auto-activated. The auto_activation_volume_list setting
# controls which LVs are auto-activated (all by default.)
# When lvmetad is updated (automatically by udev events, or directly
# by pvscan --cache), devices/filter is ignored and all devices are
# scanned by default. lvmetad always keeps unfiltered information
# which is provided to LVM commands. Each LVM command then filters
# based on devices/filter. This does not apply to other, non-regexp,
# filtering settings: component filters such as multipath and MD
# are checked during pvscan --cache. To filter a device and prevent
# scanning from the LVM system entirely, including lvmetad, use
# devices/global_filter.
    use_lvmetad = 0

# Configuration option global/lvmetad_update_wait_time.
# The number of seconds a command will wait for lvmetad update to finish.
# After waiting for this period, a command will not use lvmetad, and
# will revert to disk scanning.
# This configuration option has an automatic default value.
# lvmetad_update_wait_time = 10

# Configuration option global/use_lvmlockd.
# Use lvmlockd for locking among hosts using LVM on shared storage.
# Applicable only if LVM is compiled with lockd support in which
# case there is also lvmlockd(8) man page available for more
# information.
use_lvmlockd = 0

# Configuration option global/lvmlockd_lock_retries.
# Retry lvmlockd lock requests this many times.
# Applicable only if LVM is compiled with lockd support
# This configuration option has an automatic default value.
# lvmlockd_lock_retries = 3

# Configuration option global/sanlock_lv_extend.
# Size in MiB to extend the internal LV holding sanlock locks.
# The internal LV holds locks for each LV in the VG, and after enough
# LVs have been created, the internal LV needs to be extended. lvcreate
# will automatically extend the internal LV when needed by the amount
# specified here. Setting this to 0 disables the automatic extension
# and can cause lvcreate to fail. Applicable only if LVM is compiled
# with lockd support
# This configuration option has an automatic default value.
# sanlock_lv_extend = 256

# Configuration option global/thin_check_executable.
# The full path to the thin_check command.
# LVM uses this command to check that a thin metadata device is in a
# usable state. When a thin pool is activated and after it is
```



```

# deactivated, this command is run. Activation will only proceed if
# the command has an exit status of 0. Set to "" to skip this check.
# (Not recommended.) Also see thin_check_options.
# (See package device-mapper-persistent-data or thin-provisioning-tools)
# This configuration option has an automatic default value.
# thin_check_executable = "/usr/sbin/thin_check"

# Configuration option global/thin_dump_executable.
# The full path to the thin_dump command.
# LVM uses this command to dump thin pool metadata.
# (See package device-mapper-persistent-data or thin-provisioning-tools)
# This configuration option has an automatic default value.
# thin_dump_executable = "/usr/sbin/thin_dump"

# Configuration option global/thin_repair_executable.
# The full path to the thin_repair command.
# LVM uses this command to repair a thin metadata device if it is in
# an unusable state. Also see thin_repair_options.
# (See package device-mapper-persistent-data or thin-provisioning-tools)
# This configuration option has an automatic default value.
# thin_repair_executable = "/usr/sbin/thin_repair"

# Configuration option global/thin_check_options.
# List of options passed to the thin_check command.
# With thin_check version 2.1 or newer you can add the option
# --ignore-non-fatal-errors to let it pass through ignorable errors
# and fix them later. With thin_check version 3.2 or newer you should
# include the option --clear-needs-check-flag.
# This configuration option has an automatic default value.
# thin_check_options = [ "-q", "--clear-needs-check-flag" ]

# Configuration option global/thin_repair_options.
# List of options passed to the thin_repair command.
# This configuration option has an automatic default value.
# thin_repair_options = [ "" ]

# Configuration option global/thin_disabled_features.
# Features to not use in the thin driver.
# This can be helpful for testing, or to avoid using a feature that is
# causing problems. Features include: block_size, discards,
# discards_non_power_2, external_origin, metadata_resize,
# external_origin_extend, error_if_no_space.
#
# Example
# thin_disabled_features = [ "discards", "block_size" ]
#
# This configuration option does not have a default value defined.

# Configuration option global/cache_disabled_features.
# Features to not use in the cache driver.
# This can be helpful for testing, or to avoid using a feature that is
# causing problems. Features include: policy_mq, policy_smq.
#
# Example
# cache_disabled_features = [ "policy_smq" ]
#

```

```
# This configuration option does not have a default value defined.

# Configuration option global/cache_check_executable.
# The full path to the cache_check command.
# LVM uses this command to check that a cache metadata device is in a
# usable state. When a cached LV is activated and after it is
# deactivated, this command is run. Activation will only proceed if the
# command has an exit status of 0. Set to "" to skip this check.
# (Not recommended.) Also see cache_check_options.
# (See package device-mapper-persistent-data or thin-provisioning-tools)
# This configuration option has an automatic default value.
# cache_check_executable = "/usr/sbin/cache_check"

# Configuration option global/cache_dump_executable.
# The full path to the cache_dump command.
# LVM uses this command to dump cache pool metadata.
# (See package device-mapper-persistent-data or thin-provisioning-tools)
# This configuration option has an automatic default value.
# cache_dump_executable = "/usr/sbin/cache_dump"

# Configuration option global/cache_repair_executable.
# The full path to the cache_repair command.
# LVM uses this command to repair a cache metadata device if it is in
# an unusable state. Also see cache_repair_options.
# (See package device-mapper-persistent-data or thin-provisioning-tools)
# This configuration option has an automatic default value.
# cache_repair_executable = "/usr/sbin/cache_repair"

# Configuration option global/cache_check_options.
# List of options passed to the cache_check command.
# With cache_check version 5.0 or newer you should include the option
# --clear-needs-check-flag.
# This configuration option has an automatic default value.
# cache_check_options = [ "-q", "--clear-needs-check-flag" ]

# Configuration option global/cache_repair_options.
# List of options passed to the cache_repair command.
# This configuration option has an automatic default value.
# cache_repair_options = [ "" ]

# Configuration option global/system_id_source.
# The method LVM uses to set the local system ID.
# Volume Groups can also be given a system ID (by vgcreate, vgchange,
# or vgimport.) A VG on shared storage devices is accessible only to
# the host with a matching system ID. See 'man lvmsystemid' for
# information on limitations and correct usage.
#
# Accepted values:
#   none
#       The host has no system ID.
#   lvmlocal
#       Obtain the system ID from the system_id setting in the 'local'
#       section of an lvm configuration file, e.g. lvmlocal.conf.
#   uname
#       Set the system ID from the hostname (uname) of the system.
#       System IDs beginning localhost are not permitted.
```

```

# machineid
#   Use the contents of the machine-id file to set the system ID.
#   Some systems create this file at installation time.
#   See 'man machine-id' and global/etc.
# file
#   Use the contents of another file (system_id_file) to set the
#   system ID.
#
system_id_source = "none"

# Configuration option global/system_id_file.
# The full path to the file containing a system ID.
# This is used when system_id_source is set to 'file'.
# Comments starting with the character # are ignored.
# This configuration option does not have a default value defined.

# Configuration option global/use_lvmpolld.
# Use lvmpolld to supervise long running LVM commands.
# When enabled, control of long running LVM commands is transferred
# from the original LVM command to the lvmpolld daemon. This allows
# the operation to continue independent of the original LVM command.
# After lvmpolld takes over, the LVM command displays the progress
# of the ongoing operation. lvmpolld itself runs LVM commands to
# manage the progress of ongoing operations. lvmpolld can be used as
# a native systemd service, which allows it to be started on demand,
# and to use its own control group. When this option is disabled, LVM
# commands will supervise long running operations by forking themselves.
# Applicable only if LVM is compiled with lvmpolld support.
use_lvmpolld = 1

# Configuration option global/notify_dbus.
# Enable D-Bus notification from LVM commands.
# When enabled, an LVM command that changes PVs, changes VG metadata,
# or changes the activation state of an LV will send a notification.
notify_dbus = 1
}

# Configuration section activation.
activation {

# Configuration option activation/checks.
# Perform internal checks of libdevmapper operations.
# 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

# Configuration option activation/udev_sync.
# Use udev notifications to synchronize udev and LVM.
# The --nodevsysync option overrides this setting.
# When disabled, LVM commands will not wait for notifications from
# udev, but continue irrespective of any possible udev processing in
# the background. Only use this if udev is not running or has rules
# that ignore the devices LVM creates. If enabled when udev is not
# running, and LVM processes are waiting for udev, run the command
# 'dmsetup udevcomplete_all' to wake them up.

```

```
udev_sync = 1

# Configuration option activation/udev_rules.
# Use udev rules to manage LV device nodes and symlinks.
# When disabled, LVM will manage the device nodes and symlinks for
# active LVs itself. Manual intervention may be required if this
# setting is changed while LVs are active.
udev_rules = 1

# Configuration option activation/verify_udev_operations.
# Use extra checks in LVM to verify udev operations.
# This enables additional checks (and if necessary, repairs) on entries
# in the device directory after udev has completed processing its
# events. Useful for diagnosing problems with LVM/udev interactions.
verify_udev_operations = 0

# Configuration option activation/retry_deactivation.
# Retry failed LV deactivation.
# If LV deactivation fails, LVM will retry for a few seconds before
# failing. This may happen because a process run from a quick udev rule
# temporarily opened the device.
retry_deactivation = 1

# Configuration option activation/missing_stripe_filler.
# Method to fill missing stripes when activating an incomplete LV.
# 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 in place of missing stripes. Using anything
# other than 'error' with mirrored or snapshotted volumes is likely to
# result in data corruption.
# This configuration option is advanced.
missing_stripe_filler = "error"

# Configuration option activation/use_linear_target.
# Use the linear target to optimize single stripe LVs.
# When disabled, the striped target is used. The linear target is an
# optimised version of the striped target that only handles a single
# stripe.
use_linear_target = 1

# Configuration option activation/reserved_stack.
# Stack size in KiB to reserve for use while devices are suspended.
# Insufficient reserve risks I/O deadlock during device suspension.
reserved_stack = 64

# Configuration option activation/reserved_memory.
# Memory size in KiB to reserve for use while devices are suspended.
# Insufficient reserve risks I/O deadlock during device suspension.
reserved_memory = 8192

# Configuration option activation/process_priority.
# Nice value used while devices are suspended.
# Use a high priority so that LVs are suspended
# for the shortest possible time.
process_priority = -18
```

```

# Configuration option activation/volume_list.
# Only LVs selected by this list are activated.
# If this list is defined, an LV is only activated if it matches an
# entry in this list. If this list is undefined, it imposes no limits
# on LV activation (all are allowed).
#
# Accepted values:
#   vgname
#       The VG name is matched exactly and selects all LVs in the VG.
#   vgname/lvname
#       The VG name and LV name are matched exactly and selects the LV.
#   @tag
#       Selects an LV if the specified tag matches a tag set on the LV
#       or VG.
#   @*
#       Selects an LV if a tag defined on the host is also set on the LV
#       or VG. See tags/hosttags. If any host tags exist but volume_list
#       is not defined, a default single-entry list containing '@*'
#       is assumed.
#
# Example
# volume_list = [ "vg1", "vg2/lvol1", "@tag1", "@*" ]
#
# This configuration option does not have a default value defined.

# Configuration option activation/auto_activation_volume_list.
# Only LVs selected by this list are auto-activated.
# This list works like volume_list, but it is used only by
# auto-activation commands. It does not apply to direct activation
# commands. If this list is defined, an LV is only auto-activated
# if it matches an entry in this list. If this list is undefined, it
# imposes no limits on LV auto-activation (all are allowed.) If this
# list is defined and empty, i.e. "[]", then no LVs are selected for
# auto-activation. An LV that is selected by this list for
# auto-activation, must also be selected by volume_list (if defined)
# before it is activated. Auto-activation is an activation command that
# includes the 'a' argument: --activate ay or -a ay. The 'a' (auto)
# argument for auto-activation is meant to be used by activation
# commands that are run automatically by the system, as opposed to LVM
# commands run directly by a user. A user may also use the 'a' flag
# directly to perform auto-activation. Also see pvscan(8) for more
# information about auto-activation.
#
# Accepted values:
#   vgname
#       The VG name is matched exactly and selects all LVs in the VG.
#   vgname/lvname
#       The VG name and LV name are matched exactly and selects the LV.
#   @tag
#       Selects an LV if the specified tag matches a tag set on the LV
#       or VG.
#   @*
#       Selects an LV if a tag defined on the host is also set on the LV
#       or VG. See tags/hosttags. If any host tags exist but volume_list
#       is not defined, a default single-entry list containing '@*'
#       is assumed.

```

```
#
# Example
# auto_activation_volume_list = [ "vg1", "vg2/lvol1", "@tag1", "@*" ]
#
# This configuration option does not have a default value defined.

# Configuration option activation/read_only_volume_list.
# LVs in this list are activated in read-only mode.
# If this list is defined, each LV that is to be activated is checked
# against this list, and if it matches, it is activated in read-only
# mode. This overrides the permission setting stored in the metadata,
# e.g. from --permission rw.
#
# Accepted values:
#   vname
#     The VG name is matched exactly and selects all LVs in the VG.
#   vname/lvname
#     The VG name and LV name are matched exactly and selects the LV.
#   @tag
#     Selects an LV if the specified tag matches a tag set on the LV
#     or VG.
#   @*
#     Selects an LV if a tag defined on the host is also set on the LV
#     or VG. See tags/hosttags. If any host tags exist but volume_list
#     is not defined, a default single-entry list containing '@*'
#     is assumed.
#
# Example
# read_only_volume_list = [ "vg1", "vg2/lvol1", "@tag1", "@*" ]
#
# This configuration option does not have a default value defined.

# Configuration option activation/raid_region_size.
# Size in KiB of each raid or mirror synchronization region.
# For raid or mirror segment types, this is the amount of data that is
# copied at once when initializing, or moved at once by pvmove.
raid_region_size = 512

# Configuration option activation/error_when_full.
# Return errors if a thin pool runs out of space.
# The --errorwhenfull option overrides this setting.
# When enabled, writes to thin LVs immediately return an error if the
# thin pool is out of data space. When disabled, writes to thin LVs
# are queued if the thin pool is out of space, and processed when the
# thin pool data space is extended. New thin pools are assigned the
# behavior defined here.
# This configuration option has an automatic default value.
# error_when_full = 0

# Configuration option activation/readahead.
# Setting to use when there is no readahead setting in metadata.
#
# Accepted values:
#   none
#     Disable readahead.
#   auto
```

```

#       Use default value chosen by kernel.
#
readahead = "auto"

# Configuration option activation/raid_fault_policy.
# Defines how a device failure in a RAID LV is handled.
# This includes LVs that have the following segment types:
# raid1, raid4, raid5*, and raid6*.
# If a device in the LV fails, the policy determines the steps
# performed by dmeventd automatically, and the steps performed by the
# manual command lvconvert --repair --use-policies.
# Automatic handling requires dmeventd to be monitoring the LV.
#
# Accepted values:
#   warn
#       Use the system log to warn the user that a device in the RAID LV
#       has failed. It is left to the user to run lvconvert --repair
#       manually to remove or replace the failed device. As long as the
#       number of failed devices does not exceed the redundancy of the LV
#       (1 device for raid4/5, 2 for raid6), the LV will remain usable.
#   allocate
#       Attempt to use any extra physical volumes in the VG as spares and
#       replace faulty devices.
#
raid_fault_policy = "warn"

# Configuration option activation/mirror_image_fault_policy.
# Defines how a device failure in a 'mirror' LV is handled.
# An LV with the 'mirror' segment type is composed of mirror images
# (copies) and a mirror log. A disk log ensures that a mirror LV does
# not need to be re-synced (all copies made the same) every time a
# machine reboots or crashes. If a device in the LV fails, this policy
# determines the steps performed by dmeventd automatically, and the steps
# performed by the manual command lvconvert --repair --use-policies.
# Automatic handling requires dmeventd to be monitoring the LV.
#
# Accepted values:
#   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 the 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_image_fault_policy = "remove"

# Configuration option activation/mirror_log_fault_policy.
# Defines how a device failure in a 'mirror' log LV is handled.
# The mirror_image_fault_policy description for mirrored LVs also
# applies to mirrored log LVs.
mirror_log_fault_policy = "allocate"

# Configuration option activation/snapshot_autoextend_threshold.
# Auto-extend a snapshot when its usage exceeds this percent.
# Setting this to 100 disables automatic extension.
# The minimum value is 50 (a smaller value is treated as 50.)
# Also see snapshot_autoextend_percent.
# Automatic extension requires dmeventd to be monitoring the LV.
#
# Example
# Using 70% autoextend threshold and 20% autoextend size, when a 1G
# snapshot exceeds 700M, it is extended to 1.2G, and when it exceeds
# 840M, it is extended to 1.44G:
# snapshot_autoextend_threshold = 70
#
snapshot_autoextend_threshold = 100

# Configuration option activation/snapshot_autoextend_percent.
# Auto-extending a snapshot adds this percent extra space.
# The amount of additional space added to a snapshot is this
# percent of its current size.
#
# Example
# Using 70% autoextend threshold and 20% autoextend size, when a 1G
# snapshot exceeds 700M, it is extended to 1.2G, and when it exceeds
# 840M, it is extended to 1.44G:
# snapshot_autoextend_percent = 20
#
snapshot_autoextend_percent = 20

# Configuration option activation/thin_pool_autoextend_threshold.
# Auto-extend a thin pool when its usage exceeds this percent.
# Setting this to 100 disables automatic extension.
# The minimum value is 50 (a smaller value is treated as 50.)
# Also see thin_pool_autoextend_percent.
# Automatic extension requires dmeventd to be monitoring the LV.
#
# Example
# Using 70% autoextend threshold and 20% autoextend size, when a 1G
# thin pool exceeds 700M, it is extended to 1.2G, and when it exceeds
# 840M, it is extended to 1.44G:
# thin_pool_autoextend_threshold = 70
#
thin_pool_autoextend_threshold = 100
```



```

# Configuration option activation/thin_pool_autoextend_percent.
# Auto-extending a thin pool adds this percent extra space.
# The amount of additional space added to a thin pool is this
# percent of its current size.
#
# Example
# Using 70% autoextend threshold and 20% autoextend size, when a 1G
# thin pool exceeds 700M, it is extended to 1.2G, and when it exceeds
# 840M, it is extended to 1.44G:
# thin_pool_autoextend_percent = 20
#
thin_pool_autoextend_percent = 20

# Configuration option activation/mlock_filter.
# Do not mlock these memory areas.
# While activating devices, I/O to devices being (re)configured is
# suspended. As a precaution against deadlocks, LVM pins memory it is
# using so it is not paged out, and will not require I/O to reread.
# Groups of pages that are known not to be accessed during activation
# do not need to be pinned into memory. Each string listed in this
# setting is compared against each line in /proc/self/maps, and the
# pages corresponding to 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.
#
# Example
# mlock_filter = [ "locale/locale-archive", "gconv/gconv-modules.cache" ]
#
# This configuration option is advanced.
# This configuration option does not have a default value defined.

# Configuration option activation/use_mlockall.
# Use the old behavior of mlockall to pin all memory.
# Prior to version 2.02.62, LVM used mlockall() to pin the whole
# process's memory while activating devices.
use_mlockall = 0

# Configuration option activation/monitoring.
# Monitor LVs that are activated.
# The --ignoremonitoring option overrides this setting.
# When enabled, LVM will ask dmeventd to monitor activated LVs.
monitoring = 1

# Configuration option activation/polling_interval.
# Check pvmove or lvconvert progress at this interval (seconds).
# 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. 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 once the operation is complete.
polling_interval = 15

# Configuration option activation/auto_set_activation_skip.
# Set the activation skip flag on new thin snapshot LVs.
# The --setactivationskip option overrides this setting.
# An LV can have a persistent 'activation skip' flag. The flag causes

```

```
# the LV to be skipped during normal activation. The lvchange/vgchange
# -K option is required to activate LVs that have the activation skip
# flag set. When this setting is enabled, the activation skip flag is
# set on new thin snapshot LVs.
# This configuration option has an automatic default value.
# auto_set_activation_skip = 1

# Configuration option activation/activation_mode.
# How LVs with missing devices are activated.
# The --activationmode option overrides this setting.
#
# Accepted values:
#   complete
#     Only allow activation of an LV if all of the Physical Volumes it
#     uses are present. Other PVs in the Volume Group may be missing.
#   degraded
#     Like complete, but additionally RAID LVs of segment type raid1,
#     raid4, raid5, raid6 and raid10 will be activated if there is no
#     data loss, i.e. they have sufficient redundancy to present the
#     entire addressable range of the Logical Volume.
#   partial
#     Allows the activation of any LV even if a missing or failed PV
#     could cause data loss with a portion of the LV inaccessible.
#     This setting should not normally be used, but may sometimes
#     assist with data recovery.
#
activation_mode = "degraded"

# Configuration option activation/lock_start_list.
# Locking is started only for VGs selected by this list.
# The rules are the same as those for volume_list.
# This configuration option does not have a default value defined.

# Configuration option activation/auto_lock_start_list.
# Locking is auto-started only for VGs selected by this list.
# The rules are the same as those for auto_activation_volume_list.
# This configuration option does not have a default value defined.
}

# Configuration section metadata.
# This configuration section has an automatic default value.
# metadata {

# Configuration option metadata/check_pv_device_sizes.
# Check device sizes are not smaller than corresponding PV sizes.
# If device size is less than corresponding PV size found in metadata,
# there is always a risk of data loss. If this option is set, then LVM
# issues a warning message each time it finds that the device size is
# less than corresponding PV size. You should not disable this unless
# you are absolutely sure about what you are doing!
# This configuration option is advanced.
# This configuration option has an automatic default value.
# check_pv_device_sizes = 1

# Configuration option metadata/record_lvs_history.
# When enabled, LVM keeps history records about removed LVs in
```

```
# metadata. The information that is recorded in metadata for
# historical LVs is reduced when compared to original
# information kept in metadata for live LVs. Currently, this
# feature is supported for thin and thin snapshot LVs only.
# This configuration option has an automatic default value.
# record_lvs_history = 0

# Configuration option metadata/lvs_history_retention_time.
# Retention time in seconds after which a record about individual
# historical logical volume is automatically destroyed.
# A value of 0 disables this feature.
# This configuration option has an automatic default value.
# lvs_history_retention_time = 0

# Configuration option metadata/pvmetadatasize.
# Number of copies of metadata to store on each PV.
# The --pvmetadatasize option overrides this setting.
#
# Accepted values:
# 2
#   Two copies of the VG metadata are stored on the PV, one at the
#   front of the PV, and one at the end.
# 1
#   One copy of VG metadata is stored at the front of the PV.
# 0
#   No copies of VG metadata are stored on the PV. This may be
#   useful for VGs containing large numbers of PVs.
#
# This configuration option is advanced.
# This configuration option has an automatic default value.
# pvmetadatasize = 1

# Configuration option metadata/vgmetadatasize.
# Number of copies of metadata to maintain for each VG.
# The --vgmetadatasize option overrides this setting.
# 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 the
# total number of metadata areas available, then metadata is stored in
# them all. The value 0 (unmanaged) disables this automatic management
# and allows you to control which metadata areas are used at the
# individual PV level using pvchange --metadatasize y|n.
# This configuration option has an automatic default value.
# vgmetadatasize = 0

# Configuration option metadata/pvmetadatasize.
# Approximate number of sectors to use for each metadata copy.
# VGs with large numbers of PVs or LVs, or VGs containing complex LV
# structures, may need additional space for VG metadata. The metadata
# areas are treated as circular buffers, so unused space becomes filled
# with an archive of the most recent previous versions of the metadata.
# This configuration option has an automatic default value.
# pvmetadatasize = 255

# Configuration option metadata/pvmetadataignore.
# Ignore metadata areas on a new PV.
```

```
# The --metadataignore option overrides this setting.
# If metadata areas on a PV are ignored, LVM will not store metadata
# in them.
# This configuration option is advanced.
# This configuration option has an automatic default value.
# pvmetadataignore = 0

# Configuration option metadata/stripesize.
# This configuration option is advanced.
# This configuration option has an automatic default value.
# stripesize = 64

# Configuration option metadata/dirs.
# Directories holding live copies of text format metadata.
# These directories must not be on logical volumes!
# It's possible to use LVM with a couple of directories here,
# preferably on different (non-LV) filesystems, and with no other
# on-disk metadata (pvmetadaticopies = 0). Or this can be in addition
# to on-disk metadata areas. The feature was originally added to
# simplify testing and is not supported under low memory situations -
# the machine could lock up. Never edit any files in these directories
# by hand unless you are absolutely sure you know what you are doing!
# Use the supplied toolset to make changes (e.g. vgcfgrestore).
#
# Example
# dirs = [ "/etc/lvm/metadata", "/mnt/disk2/lvm/metadata2" ]
#
# This configuration option is advanced.
# This configuration option does not have a default value defined.
# }

# Configuration section report.
# LVM report command output formatting.
# This configuration section has an automatic default value.
# report {

# Configuration option report/output_format.
# Format of LVM command's report output.
# If there is more than one report per command, then the format
# is applied for all reports. You can also change output format
# directly on command line using --reportformat option which
# has precedence over log/output_format setting.
# Accepted values:
#   basic
#     Original format with columns and rows. If there is more than
#     one report per command, each report is prefixed with report's
#     name for identification.
#   json
#     JSON format.
# This configuration option has an automatic default value.
# output_format = "basic"

# Configuration option report/compact_output.
# Do not print empty values for all report fields.
# If enabled, all fields that don't have a value set for any of the
# rows reported are skipped and not printed. Compact output is
```

```
# applicable only if report/buffered is enabled. If you need to
# compact only specified fields, use compact_output=0 and define
# report/compact_output_cols configuration setting instead.
# This configuration option has an automatic default value.
# compact_output = 0

# Configuration option report/compact_output_cols.
# Do not print empty values for specified report fields.
# If defined, specified fields that don't have a value set for any
# of the rows reported are skipped and not printed. Compact output
# is applicable only if report/buffered is enabled. If you need to
# compact all fields, use compact_output=1 instead in which case
# the compact_output_cols setting is then ignored.
# This configuration option has an automatic default value.
# compact_output_cols = ""

# Configuration option report/aligned.
# Align columns in report output.
# This configuration option has an automatic default value.
# aligned = 1

# Configuration option report/buffered.
# Buffer report output.
# When buffered reporting is used, the report's content is appended
# incrementally to include each object being reported until the report
# is flushed to output which normally happens at the end of command
# execution. Otherwise, if buffering is not used, each object is
# reported as soon as its processing is finished.
# This configuration option has an automatic default value.
# buffered = 1

# Configuration option report/headings.
# Show headings for columns on report.
# This configuration option has an automatic default value.
# headings = 1

# Configuration option report/separator.
# A separator to use on report after each field.
# This configuration option has an automatic default value.
# separator = " "

# Configuration option report/list_item_separator.
# A separator to use for list items when reported.
# This configuration option has an automatic default value.
# list_item_separator = ","

# Configuration option report/prefixes.
# Use a field name prefix for each field reported.
# This configuration option has an automatic default value.
# prefixes = 0

# Configuration option report/quoted.
# Quote field values when using field name prefixes.
# This configuration option has an automatic default value.
# quoted = 1
```

```
# Configuration option report/columns_as_rows.
# Output each column as a row.
# If set, this also implies report/prefixes=1.
# This configuration option has an automatic default value.
# columns_as_rows = 0

# Configuration option report/binary_values_as_numeric.
# Use binary values 0 or 1 instead of descriptive literal values.
# For columns that have exactly two valid values to report
# (not counting the 'unknown' value which denotes that the
# value could not be determined).
# This configuration option has an automatic default value.
# binary_values_as_numeric = 0

# Configuration option report/time_format.
# Set time format for fields reporting time values.
# Format specification is a string which may contain special character
# sequences and ordinary character sequences. Ordinary character
# sequences are copied verbatim. Each special character sequence is
# introduced by the '%' character and such sequence is then
# substituted with a value as described below.
#
# Accepted values:
# %a
#     The abbreviated name of the day of the week according to the
#     current locale.
# %A
#     The full name of the day of the week according to the current
#     locale.
# %b
#     The abbreviated month name according to the current locale.
# %B
#     The full month name according to the current locale.
# %C
#     The preferred date and time representation for the current
#     locale (alt E)
# %C
#     The century number (year/100) as a 2-digit integer. (alt E)
# %d
#     The day of the month as a decimal number (range 01 to 31).
#     (alt 0)
# %D
#     Equivalent to %m/%d/%y. (For Americans only. Americans should
#     note that in other countries %d/%m/%y is rather common. This
#     means that in international context this format is ambiguous and
#     should not be used.
# %e
#     Like %d, the day of the month as a decimal number, but a leading
#     zero is replaced by a space. (alt 0)
# %E
#     Modifier: use alternative local-dependent representation if
#     available.
# %F
#     Equivalent to %Y-%m-%d (the ISO 8601 date format).
# %G
#     The ISO 8601 week-based year with century as a decimal number.
```

```

# The 4-digit year corresponding to the ISO week number (see %V).
# This has the same format and value as %Y, except that if the
# ISO week number belongs to the previous or next year, that year
# is used instead.
# %g
# Like %G, but without century, that is, with a 2-digit year
# (00-99).
# %h
# Equivalent to %b.
# %H
# The hour as a decimal number using a 24-hour clock
# (range 00 to 23). (alt 0)
# %I
# The hour as a decimal number using a 12-hour clock
# (range 01 to 12). (alt 0)
# %j
# The day of the year as a decimal number (range 001 to 366).
# %k
# The hour (24-hour clock) as a decimal number (range 0 to 23);
# single digits are preceded by a blank. (See also %H.)
# %l
# The hour (12-hour clock) as a decimal number (range 1 to 12);
# single digits are preceded by a blank. (See also %I.)
# %m
# The month as a decimal number (range 01 to 12). (alt 0)
# %M
# The minute as a decimal number (range 00 to 59). (alt 0)
# %O
# Modifier: use alternative numeric symbols.
# %p
# Either "AM" or "PM" according to the given time value,
# or the corresponding strings for the current locale. Noon is
# treated as "PM" and midnight as "AM".
# %P
# Like %p but in lowercase: "am" or "pm" or a corresponding
# string for the current locale.
# %r
# The time in a.m. or p.m. notation. In the POSIX locale this is
# equivalent to %I:%M:%S %p.
# %R
# The time in 24-hour notation (%H:%M). For a version including
# the seconds, see %T below.
# %s
# The number of seconds since the Epoch,
# 1970-01-01 00:00:00 +0000 (UTC)
# %S
# The second as a decimal number (range 00 to 60). (The range is
# up to 60 to allow for occasional leap seconds.) (alt 0)
# %t
# A tab character.
# %T
# The time in 24-hour notation (%H:%M:%S).
# %u
# The day of the week as a decimal, range 1 to 7, Monday being 1.
# See also %w. (alt 0)
# %U

```

```
# The week number of the current year as a decimal number,
# range 00 to 53, starting with the first Sunday as the first
# day of week 01. See also %V and %W. (alt 0)
# %V
# The ISO 8601 week number of the current year as a decimal number,
# range 01 to 53, where week 1 is the first week that has at least
# 4 days in the new year. See also %U and %W. (alt 0)
# %W
# The day of the week as a decimal, range 0 to 6, Sunday being 0.
# See also %u. (alt 0)
# %W
# The week number of the current year as a decimal number,
# range 00 to 53, starting with the first Monday as the first day
# of week 01. (alt 0)
# %x
# The preferred date representation for the current locale without
# the time. (alt E)
# %X
# The preferred time representation for the current locale without
# the date. (alt E)
# %y
# The year as a decimal number without a century (range 00 to 99).
# (alt E, alt 0)
# %Y
# The year as a decimal number including the century. (alt E)
# %Z
# The +hhmm or -hhmm numeric timezone (that is, the hour and minute
# offset from UTC).
# %Z
# The timezone name or abbreviation.
# %%
# A literal '%' character.
#
# This configuration option has an automatic default value.
# time_format = "%Y-%m-%d %T %Z"

# Configuration option report/devtypes_sort.
# List of columns to sort by when reporting 'lvm devtypes' command.
# See 'lvm devtypes -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# devtypes_sort = "devtype_name"

# Configuration option report/devtypes_cols.
# List of columns to report for 'lvm devtypes' command.
# See 'lvm devtypes -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# devtypes_cols =
"devtype_name,devtype_max_partitions,devtype_description"

# Configuration option report/devtypes_cols_verbose.
# List of columns to report for 'lvm devtypes' command in verbose mode.
# See 'lvm devtypes -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# devtypes_cols_verbose =
"devtype_name,devtype_max_partitions,devtype_description"
```



```

# Configuration option report/lvs_sort.
# List of columns to sort by when reporting 'lvs' command.
# See 'lvs -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# lvs_sort = "vg_name,lv_name"

# Configuration option report/lvs_cols.
# List of columns to report for 'lvs' command.
# See 'lvs -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# lvs_cols =
"lv_name,vg_name,lv_attr,lv_size,pool_lv,origin,data_percent,metadata_percent,move_pv,mirror_log,copy_percent,convert_lv"

# Configuration option report/lvs_cols_verbose.
# List of columns to report for 'lvs' command in verbose mode.
# See 'lvs -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# lvs_cols_verbose =
"lv_name,vg_name,seg_count,lv_attr,lv_size,lv_major,lv_minor,lv_kernel_major,lv_kernel_minor,pool_lv,origin,data_percent,metadata_percent,move_pv,copy_percent,mirror_log,convert_lv,lv_uuid,lv_profile"

# Configuration option report/vgs_sort.
# List of columns to sort by when reporting 'vgs' command.
# See 'vgs -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# vgs_sort = "vg_name"

# Configuration option report/vgs_cols.
# List of columns to report for 'vgs' command.
# See 'vgs -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# vgs_cols =
"vg_name,pv_count,lv_count,snap_count,vg_attr,vg_size,vg_free"

# Configuration option report/vgs_cols_verbose.
# List of columns to report for 'vgs' command in verbose mode.
# See 'vgs -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# vgs_cols_verbose =
"vg_name,vg_attr,vg_extent_size,pv_count,lv_count,snap_count,vg_size,vg_free,vg_uuid,vg_profile"

# Configuration option report/pvs_sort.
# List of columns to sort by when reporting 'pvs' command.
# See 'pvs -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# pvs_sort = "pv_name"

# Configuration option report/pvs_cols.
# List of columns to report for 'pvs' command.
# See 'pvs -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# pvs_cols = "pv_name,vg_name,pv_fmt,pv_attr,pv_size,pv_free"

```

```
# Configuration option report/pvs_cols_verbose.
# List of columns to report for 'pvs' command in verbose mode.
# See 'pvs -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# pvs_cols_verbose =
"pv_name,vg_name,pv_fmt,pv_attr,pv_size,pv_free,dev_size,pv_uuid"

# Configuration option report/secs_sort.
# List of columns to sort by when reporting 'lvs --segments' command.
# See 'lvs --segments -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# secs_sort = "vg_name,lv_name,seg_start"

# Configuration option report/secs_cols.
# List of columns to report for 'lvs --segments' command.
# See 'lvs --segments -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# secs_cols = "lv_name,vg_name,lv_attr,stripes,segtype,seg_size"

# Configuration option report/secs_cols_verbose.
# List of columns to report for 'lvs --segments' command in verbose mode.
# See 'lvs --segments -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# secs_cols_verbose =
"lv_name,vg_name,lv_attr,seg_start,seg_size,stripes,segtype,stripesize,chunksize"

# Configuration option report/pvsecs_sort.
# List of columns to sort by when reporting 'pvs --segments' command.
# See 'pvs --segments -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# pvsecs_sort = "pv_name,pvseg_start"

# Configuration option report/pvsecs_cols.
# List of columns to sort by when reporting 'pvs --segments' command.
# See 'pvs --segments -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# pvsecs_cols =
"pv_name,vg_name,pv_fmt,pv_attr,pv_size,pv_free,pvseg_start,pvseg_size"

# Configuration option report/pvsecs_cols_verbose.
# List of columns to sort by when reporting 'pvs --segments' command in verbose mode.
# See 'pvs --segments -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# pvsecs_cols_verbose =
"pv_name,vg_name,pv_fmt,pv_attr,pv_size,pv_free,pvseg_start,pvseg_size,lv_name,seg_start_pe,segtype,seg_pe_ranges"

# Configuration option report/vgs_cols_full.
# List of columns to report for lvm fullreport's 'vgs' subreport.
# See 'vgs -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# vgs_cols_full = "vg_all"

# Configuration option report/pvs_cols_full.
```

```
# List of columns to report for lvm fullreport's 'vgs' subreport.
# See 'pvs -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# pvs_cols_full = "pv_all"

# Configuration option report/lvs_cols_full.
# List of columns to report for lvm fullreport's 'lvs' subreport.
# See 'lvs -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# lvs_cols_full = "lv_all"

# Configuration option report/pvsegs_cols_full.
# List of columns to report for lvm fullreport's 'pvseg' subreport.
# See 'pvs --segments -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# pvsegs_cols_full = "pvseg_all,pv_uuid,lv_uuid"

# Configuration option report/segs_cols_full.
# List of columns to report for lvm fullreport's 'seg' subreport.
# See 'lvs --segments -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# segs_cols_full = "seg_all,lv_uuid"

# Configuration option report/vgs_sort_full.
# List of columns to sort by when reporting lvm fullreport's 'vgs'
subreport.
# See 'vgs -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# vgs_sort_full = "vg_name"

# Configuration option report/pvs_sort_full.
# List of columns to sort by when reporting lvm fullreport's 'vgs'
subreport.
# See 'pvs -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# pvs_sort_full = "pv_name"

# Configuration option report/lvs_sort_full.
# List of columns to sort by when reporting lvm fullreport's 'lvs'
subreport.
# See 'lvs -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# lvs_sort_full = "vg_name,lv_name"

# Configuration option report/pvsegs_sort_full.
# List of columns to sort by when reporting for lvm fullreport's 'pvseg'
subreport.
# See 'pvs --segments -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# pvsegs_sort_full = "pv_uuid,pvseg_start"

# Configuration option report/segs_sort_full.
# List of columns to sort by when reporting lvm fullreport's 'seg'
subreport.
# See 'lvs --segments -o help' for the list of possible fields.
# This configuration option has an automatic default value.
```

```
# segs_sort_full = "lv_uuid,seg_start"

# Configuration option report/mark_hidden_devices.
# Use brackets [] to mark hidden devices.
# This configuration option has an automatic default value.
# mark_hidden_devices = 1

# Configuration option report/two_word_unknown_device.
# Use the two words 'unknown device' in place of '[unknown]'.
# This is displayed when the device for a PV is not known.
# This configuration option has an automatic default value.
# two_word_unknown_device = 0
# }

# Configuration section dmeventd.
# Settings for the LVM event daemon.
dmeventd {

# Configuration option dmeventd/mirror_library.
# The library dmeventd uses 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"

# Configuration option dmeventd/raid_library.
# This configuration option has an automatic default value.
# raid_library = "libdevmapper-event-lvm2raid.so"

# Configuration option dmeventd/snapshot_library.
# The library dmeventd uses when monitoring a snapshot device.
# libdevmapper-event-lvm2snapshot.so monitors the filling of snapshots
# and emits a warning through syslog when the usage exceeds 80%. The
# warning is repeated when 85%, 90% and 95% of the snapshot is filled.
snapshot_library = "libdevmapper-event-lvm2snapshot.so"

# Configuration option dmeventd/thin_library.
# The library dmeventd uses when monitoring a thin device.
# libdevmapper-event-lvm2thin.so monitors the filling of a pool
# and emits a warning through syslog when the usage exceeds 80%. The
# warning is repeated when 85%, 90% and 95% of the pool is filled.
thin_library = "libdevmapper-event-lvm2thin.so"

# Configuration option dmeventd/executable.
# The full path to the dmeventd binary.
# This configuration option has an automatic default value.
# executable = "/usr/sbin/dmeventd"
}

# Configuration section tags.
# Host tag settings.
# This configuration section has an automatic default value.
# tags {

# Configuration option tags/hosttags.
```

```
# Create a host tag using the machine name.
# The machine name is nodename returned by uname(2).
# This configuration option has an automatic default value.
# hosttags = 0

# Configuration section tags/<tag>.
# Replace this subsection name with a custom tag name.
# Multiple subsections like this can be created. The '@' prefix for
# tags is optional. This subsection can contain host_list, which is a
# list of machine names. If the name of the local machine is found in
# host_list, then the name of this subsection is used as a tag and is
# applied to the local machine as a 'host tag'. If this subsection is
# empty (has no host_list), then the subsection name is always applied
# as a 'host tag'.
#
# Example
# The host tag foo is given to all hosts, and the host tag
# bar is given to the hosts named machine1 and machine2.
# tags { foo { } bar { host_list = [ "machine1", "machine2" ] } }
#
# This configuration section has variable name.
# This configuration section has an automatic default value.
# tag {

# Configuration option tags/<tag>/host_list.
# A list of machine names.
# These machine names are compared to the nodename returned
# by uname(2). If the local machine name matches an entry in
# this list, the name of the subsection is applied to the
# machine as a 'host tag'.
# This configuration option does not have a default value defined.
# }
# }
```

## 付録C LVM 選択基準

Red Hat Enterprise Linux リリース 7.1 では、多くの LVM レポートコマンドで **-S** または **--select** オプションを使用してこれらのコマンドの選択基準を定義できます。Red Hat Enterprise Linux リリース 7.2 では、多くの処理コマンドでも選択基準がサポートされます。選択基準を定義できるこれら 2 つのカテゴリのコマンドは以下のように定義されます。

- レポートコマンド — 選択基準を満たす行のみを表示します。選択基準を定義できるレポートコマンドの例には、**pvs**、**vgs**、**lvs**、**pvdiskdisplay**、**vgdisplay**、**lvdisplay**、**lvm devtypes**、**dmsetup info -c** などがあります。  
  
**-S** オプション以外に **-o selected** オプションを指定すると、すべての行が表示され、行が選択基準に一致する場合は 1、一致しない場合は 0 を示す "selected" 列が追加されます。
- 処理コマンド — 選択基準を満たすアイテムのみを処理します。選択基準を定義できる処理コマンドの例には、**pvchange**、**vgchange**、**lvchange**、**vgimport**、**vgexport**、**vgremove**、**lvremove** などがあります。

選択基準は、表示または処理する特定のフィールドの有効な値を定義するために比較演算子を使用する一連のステートメントです。選択されたフィールドは論理演算子とグループ演算子によって順番に結合されます。

選択基準を使用して表示するフィールドを指定する場合は、表示する選択基準に含まれるフィールドは必要ありません。出力にはさまざまなフィールドセットを含めることができますが、選択基準には 1 つのフィールドセットしか含めることができません。

- さまざまな LVM コンポーネントの利用可能なフィールドの一覧については、「[選択基準フィールド](#)」を参照してください。
- 許可された操作の一覧については、「[選択基準演算子](#)」を参照してください。演算子は `lvm(8)` man ページにも記載されています。
- また、レポートコマンドの **-S/--select** に対して **help (or ?)** キーワードを指定することにより、すべてのフィールドと使用可能な演算子を表示することもできます。たとえば、以下のコマンドは **lvs** コマンド向けのフィールドと使用可能な演算子を表示します。

```
# lvs -S help
```

Red Hat Enterprise Linux 7.2 リリースの場合は、**time** のフィールドタイプのフィールドの選択基準として時間値を指定できます。時間値の指定については、「[時間値の指定](#)」を参照してください。

### C.1. 選択基準フィールドタイプ

選択基準に指定するフィールドは特定のタイプです。各フィールドのヘルプ出力では、フィールドタイプがカッコで囲まれて表示されます。以下のヘルプ出力例は、フィールドタイプ **string**、**string\_list**、**number**、**percent**、**size**、および **time** を示す出力を示しています。

```
lv_name          - Name. LVs created for internal use are enclosed in
brackets.[string]
lv_role          - LV role. [string list]
raid_mismatch_count - For RAID, number of mismatches found or repaired.
[number]
copy_percent     - For RAID, mirrors and pvmove, current percentage in-
```

```

sync. [percent]
lv_size          - Size of LV in current units. [size]
lv_time          - Creation time of the LV, if known [time]

```

表C.1「選択基準フィールドタイプ」では、選択基準フィールドタイプについて説明しています。

表C.1 選択基準フィールドタイプ

フィールドタイプ	説明
数値	正の整数値。
サイズ	単位付きの浮動小数点値。指定しない場合はデフォルトで 'm' 単位が使用されます。
パーセント	% 接尾辞あり、または無しの正の整数。
文字列	引用符 ' または " で囲まれた文字または引用符で囲まれていない文字。
文字列リスト	[] または {} で囲まれた文字列と "all items must match" または "at least one item must match" 演算子で区切られた要素。

フィールドには以下の値を指定できます。

- フィールドタイプの具体的な値。
- **string** フィールドタイプのフィールドを含む正規表現 ("+" 演算子など)。
- 予約済みの値。たとえば、-1、unknown、undefined、undef はすべて未定義の数値を示すキーワードです。
- フィールド値の定義済み同意語。元の値の場合と同様に値の選択基準で使用できます。フィールド値の定義済み同意語の一覧については、表C.14「選択基準同意義」を参照してください。

## C.2. 選択基準演算子

表C.2「選択基準グループ演算子」では、選択基準グループ演算子について説明しています。

表C.2 選択基準グループ演算子

グループ演算子	説明
()	グループステートメントに使用
[]	文字列を文字列リストにグループ化するために使用 (完全一致)
{}	文字列を文字列リストにグループ化するために使用 (サブセット一致)

表C.3「選択基準比較演算子」では、選択基準比較演算子それらの演算子を使用できるフィールドタイプについて説明しています。

表C.3 選択基準比較演算子

比較演算子	説明	フィールドタイプ
=~	一致する正規表現	正規表現
!~	一致しない正規表現	正規表現
=	等しい	数字、サイズ、パーセント、文字列、文字列リスト、時間
!=	等しくない	数字、サイズ、パーセント、文字列、文字列リスト、時間
>=	より大か等しい	数字、サイズ、パーセント、時間
>	より大きい	数字、サイズ、パーセント、時間
<=	より小か等しい	数字、サイズ、パーセント、時間
<	より小さい	数字、サイズ、パーセント、時間
since	指定された時間以降 (>= と同じ)	時間
after	指定された時間よりも後 (> と同じ)	時間
until	指定された時間まで (<= と同じ)	時間
before	指定された時間よりも前 (< と同じ)	時間

表C.4「選択基準論理演算子およびグループ演算子」では、選択基準論理演算子およびグループ演算子について説明しています。

表C.4 選択基準論理演算子およびグループ演算子

論理およびグループ演算子	説明
&&	すべてのフィールドが一致する必要がある
,	すべてのフィールドが一致する必要がある (&& と同じ)
	少なくとも 1 つのフィールドが一致する必要がある
#	少なくとも 1 つのフィールドが一致する必要がある (   と同じ)
!	論理否定
(	左かっこ (グループ演算子)



論理およびグループ演算子	説明
)	右かっこ (グループ演算子)
[	リストの始まり (グループ演算子)
]	リストの終わり (グループ演算子)
{	リストサブセットの始まり (グループ演算子)
}	リストサブセットの終わり (グループ演算子)

### C.3. 選択基準フィールド

このセクションでは、指定できる論理および物理ボリューム選択基準について説明します。

表C.5「論理ボリュームフィールド」では、論理ボリュームフィールドとそのフィールドタイプについて説明しています。

表C.5 論理ボリュームフィールド

論理ボリュームフィールド	説明	フィールドタイプ
<b>lv_uuid</b>	一意の ID	文字列
<b>lv_name</b>	名前 (内部使用のために作成された論理ボリュームはかっこで囲まれます)	文字列
<b>lv_full_name</b>	ボリュームグループを含む論理ボリュームの完全な名前 (VG/LV など)	文字列
<b>lv_path</b>	論理ボリュームの完全なパス名 (内部論理ボリュームの場合は空白)	文字列
<b>lv_dm_path</b>	論理ボリューム用の内部デバイスマッパーパス名 (/dev/mapper ディレクトリー内)	文字列
<b>lv_parent</b>	別の論理ボリュームのコンポーネントである論理ボリュームの場合は、親論理ボリューム	文字列
<b>lv_layout</b>	論理ボリュームレイアウト	文字列リスト
<b>lv_role</b>	論理ボリュームロール	文字列リスト
<b>lv_initial_image_sync</b>	ミラー/RAID イメージで最初の再同期が行われた場合に設定される	数値

論理ボリュームフィールド	説明	フィールドタイプ
<b>lv_image_synced</b>	ミラー/RAID イメージが同期された場合に設定される	数値
<b>lv_merging</b>	スナップショット論理ボリュームが元の論理ボリュームにマージされる場合に設定される	数値
<b>lv_converting</b>	論理ボリュームが変換される場合に設定される	数値
<b>lv_allocation_policy</b>	論理ボリューム割り当てポリシー	文字列
<b>lv_allocation_locked</b>	論理ボリュームが割り当ての変更に対してロックされている場合に設定される	数値
<b>lv_fixed_minor</b>	論理ボリュームで固定マイナー番号が割り当てられた場合に設定される	数値
<b>lv_merge_failed</b>	スナップショットマージが失敗した場合に設定される	数値
<b>lv_snapshot_invalid</b>	スナップショット論理ボリュームが無効な場合に設定される	数値
<b>lv_skip_activation</b>	アクティブ化のときに論理ボリュームが省略された場合に設定される	数値
<b>lv_when_full</b>	シンプールの場合は、いっぱいになったときの動作	文字列
<b>lv_active</b>	論理ボリュームのアクティブな状態	文字列
<b>lv_active_locally</b>	論理ボリュームがローカルでアクティブな場合に設定される	数値
<b>lv_active_remotely</b>	論理ボリュームがリモートでアクティブな場合に設定される	数値
<b>lv_active_exclusively</b>	論理ボリュームが排他的にアクティブな場合に設定される	数値
<b>lv_major</b>	永続的なメジャー番号または -1 (永続的でない場合)	数値
<b>lv_minor</b>	永続的なマイナー番号または -1 (永続的でない場合)	数値
<b>lv_read_ahead</b>	現在の単位の先読み設定	サイズ

論理ボリュームフィールド	説明	フィールドタイプ
<b>lv_size</b>	現在の単位の論理ボリュームのサイズ	サイズ
<b>lv_metadata_size</b>	シンプルおよびキャッシュプールの場合は、メタデータを保持する論理ボリュームのサイズ	サイズ
<b>seg_count</b>	論理ボリューム内のセグメント数	数値
<b>origin</b>	スナップショットの場合は、この論理ボリュームの元のデバイス	文字列
<b>origin_size</b>	スナップショットの場合は、この論理ボリュームの元のデバイスのサイズ	サイズ
<b>data_percent</b>	スナップショットおよびシンプルとボリュームの場合は、使用領域の割合 (論理ボリュームがアクティブなとき)	パーセント
<b>snap_percent</b>	スナップショットの場合は、使用領域の割合 (論理ボリュームがアクティブなとき)	パーセント
<b>metadata_percent</b>	シンプルの場合は、メタデータの使用領域の割合 (論理ボリュームがアクティブなとき)	パーセント
<b>copy_percent</b>	RAID、ミラー、および pvmove の場合は、現在同期されている領域の割合	パーセント
<b>sync_percent</b>	RAID、ミラー、および pvmove の場合は、現在同期されている領域の割合	パーセント
<b>raid_mismatch_count</b>	RAID の場合は、検出または修復された不一致の数	数値
<b>raid_sync_action</b>	RAID の場合は、実行中の現在の同期アクション	文字列
<b>raid_write_behind</b>	RAID1 の場合は、writemostly デバイスに許可された未処理の書き込みの数	数値
<b>raid_min_recover_y_rate</b>	RAID1 の場合は、最小リカバリー I/O ロード (kiB/sec/disk 単位)	数値
<b>raid_max_recover_y_rate</b>	RAID1 の場合は、最大リカバリー I/O ロード (kiB/sec/disk 単位)	数値
<b>move_pv</b>	pvmove の場合は、pvmove で作成された一時的な論理ボリュームの元となる物理ボリューム	文字列

論理ボリュームフィールド	説明	フィールドタイプ
<b>convert_lv</b>	lvconvert の場合は、lvconvert で作成された一時的な論理ボリュームの名前	文字列
<b>mirror_log</b>	ミラーの場合は、同期ログを保持する論理ボリューム	文字列
<b>data_lv</b>	シンプルーおよびキャッシュプールの場合は、関連データを保持する論理ボリューム	文字列
<b>metadata_lv</b>	シンプルーおよびキャッシュプールの場合は、関連メタデータを保持する論理ボリューム	文字列
<b>pool_lv</b>	シンボリュームの場合は、このボリュームのシンプルー論理ボリューム	文字列
<b>lv_tags</b>	タグ (存在する場合)	文字列リスト
<b>lv_profile</b>	この論理ボリュームに割り当てられた設定プロファイル	文字列
<b>lv_time</b>	論理ボリュームの作成時刻 (既知の場合)	時間
<b>lv_host</b>	論理ボリュームの作成ホスト (既知の場合)	文字列
<b>lv_modules</b>	この論理ボリュームに必要なカーネル device-mapper モジュール	文字列リスト

表C.6「論理ボリュームデバイスの情報とステータスを組み合わせたフィールド」では、論理デバイス情報と論理デバイスステータスの両方を組み合わせる論理ボリュームデバイスフィールドについて説明しています。

表C.6 論理ボリュームデバイスの情報とステータスを組み合わせたフィールド

論理ボリュームフィールド	説明	フィールドタイプ
<b>lv_attr</b>	論理ボリュームデバイス情報と論理ボリュームステータスの両方に基づいて選択します。	文字列

表C.7「論理ボリュームデバイス情報フィールド」では、論理ボリュームデバイス情報フィールドとそのフィールドタイプについて説明しています。

表C.7 論理ボリュームデバイス情報フィールド

論理ボリュームフィールド	説明	フィールドタイプ
<b>lv_kernel_major</b>	現在割り当てられているメジャー番号または -1 (論理ボリュームがアクティブでない場合)	数値
<b>lv_kernel_minor</b>	現在割り当てられているマイナー番号または -1 (論理ボリュームがアクティブでない場合)	数値
<b>lv_kernel_read_ahead</b>	現在の単位の現在使用されている先読み設定	サイズ
<b>lv_permissions</b>	論理ボリュームパーミッション	文字列
<b>lv_suspended</b>	論理ボリュームが一時停止している場合に設定される	数値
<b>lv_live_table</b>	論理ボリュームにライブテーブルが存在する場合に設定される	数値
<b>lv_inactive_table</b>	論理ボリュームに非アクティブなテーブルが存在する場合に設定される	数値
<b>lv_device_open</b>	論理ボリュームデバイスがオープンな場合に設定される	数値

表C.8「論理ボリュームデバイスステータスフィールド」では、論理ボリュームデバイスステータスフィールドとそのフィールドタイプについて説明しています。

表C.8 論理ボリュームデバイスステータスフィールド

論理ボリュームフィールド	説明	フィールドタイプ
<b>cache_total_blocks</b>	キャッシュブロック合計数	数値
<b>cache_used_blocks</b>	使用済みキャッシュブロック数	数値
<b>cache_dirty_blocks</b>	ダーティーキャッシュブロック数	数値
<b>cache_read_hits</b>	キャッシュ読み取りヒット数	数値
<b>cache_read_misses</b>	キャッシュ読み取りミス数	数値
<b>cache_write_hits</b>	キャッシュ書き込みヒット数	数値

論理ボリュームフィールド	説明	フィールドタイプ
<b>cache_write_misses</b>	キャッシュ書き込みミス数	数値
<b>lv_health_status</b>	論理ボリューム正常性状態	文字列

表C.9「物理ボリュームラベルフィールド」では、物理ボリュームラベルフィールドとそのフィールドタイプについて説明しています。

表C.9 物理ボリュームラベルフィールド

物理ボリュームフィールド	説明	フィールドタイプ
<b>pv_fmt</b>	メタデータのタイプ	文字列
<b>pv_uuid</b>	一意の ID	文字列
<b>dev_size</b>	基礎となるデバイスのサイズ (現在の単位)	サイズ
<b>pv_name</b>	名前	文字列
<b>pv_mda_free</b>	このデバイスの空きメタデータ領域 (現在の単位)	サイズ
<b>pv_mda_size</b>	このデバイスの最小メタデータ領域のサイズ (現在の単位)	サイズ

表C.5「論理ボリュームフィールド」では、物理ボリュームフィールドとそのフィールドタイプについて説明しています。

表C.10 物理ボリュームフィールド

物理ボリュームフィールド	説明	フィールドタイプ
<b>pe_start</b>	基礎となるデバイスのデータの始まりに対するオフセット	数値
<b>pv_size</b>	物理ボリュームのサイズ (現在の単位)	サイズ
<b>pv_free</b>	未割り当て領域の合計サイズ (現在の単位)	サイズ
<b>pv_used</b>	割り当て済み領域の合計サイズ (現在の単位)	サイズ
<b>pv_attr</b>	さまざまな属性	文字列

物理ボリュームフィールド	説明	フィールドタイプ
<b>pv_allocatable</b>	このデバイスを割り当てに使用できる場合に設定される	数値
<b>pv_exported</b>	このデバイスがエクスポートされる場合に設定される	数値
<b>pv_missing</b>	このデバイスがシステムで不明な場合に設定される	数値
<b>pv_pe_count</b>	物理エクステンツの合計数	数値
<b>pv_pe_alloc_count</b>	割り当て済み物理エクステンツの合計数	数値
<b>pv_tags</b>	タグ (存在する場合)	文字列リスト
<b>pv_mda_count</b>	このデバイスのメタデータ領域の数	数値
<b>pv_mda_used_count</b>	このデバイスの使用中メタデータ領域の数	数値
<b>pv_ba_start</b>	基礎となるデバイスの PV ブートローダー領域の始まりに対するオフセット (現在の単位)	サイズ
<b>pv_ba_size</b>	PV ブートローダー領域のサイズ (現在の単位)	サイズ

表C.11「ボリュームグループフィールド」では、ボリュームグループフィールドとそのフィールドタイプについて説明しています。

表C.11 ボリュームグループフィールド

ボリュームグループフィールド	説明	フィールドタイプ
<b>vg_fmt</b>	メタデータのタイプ	文字列
<b>vg_uuid</b>	一意の ID	文字列
<b>vg_name</b>	名前	文字列
<b>vg_attr</b>	さまざまな属性	文字列
<b>vg_permissions</b>	ボリュームグループパーミッション	文字列
<b>vg_extendable</b>	ボリュームグループが拡張可能である場合に設定される	数値

ボリュームグループ フィールド	説明	フィールドタイプ
<b>vg_exported</b>	ボリュームグループがエクスポートされる場合に設定される	数値
<b>vg_partial</b>	ボリュームグループが部分的である場合に設定される	数値
<b>vg_allocation_policy</b>	ボリュームグループ割り当てポリシー	文字列
<b>vg_clustered</b>	ボリュームグループがクラスター化されている場合に設定される	数値
<b>vg_size</b>	ボリュームグループの合計サイズ (現在の単位)	サイズ
<b>vg_free</b>	空き領域の合計サイズ (現在の単位)	サイズ
<b>vg_sysid</b>	ボリュームグループのシステム ID (所有しているホストを示す)	文字列
<b>vg_systemid</b>	ボリュームグループのシステム ID (所有しているホストを示す)	文字列
<b>vg_extent_size</b>	物理エクステントのサイズ (現在の単位)	サイズ
<b>vg_extent_count</b>	物理エクステントの合計数	数値
<b>vg_free_count</b>	未割り当て物理エクステントの合計数	数値
<b>max_lv</b>	ボリュームグループ内で許可される論理ボリュームの最大数 (無制限の場合は 0)	数値
<b>max_pv</b>	ボリュームグループ内で許可される物理ボリュームの最大数 (無制限の場合は 0)	数値
<b>pv_count</b>	物理ボリュームの数	数値
<b>lv_count</b>	論理ボリュームの数	数値
<b>snap_count</b>	スナップショットの数	数値
<b>vg_seqno</b>	内部メタデータの改訂番号 — 変更されるたびに増分される	数値
<b>vg_tags</b>	タグ (存在する場合)	文字列リスト



ボリュームグループ フィールド	説明	フィールドタイプ
<b>vg_profile</b>	このボリュームグループに割り当てられた設定プロファイル	文字列
<b>vg_mda_count</b>	このボリュームグループのメタデータ領域の数	数値
<b>vg_mda_used_count</b>	このボリュームグループの使用メタデータ領域の数	数値
<b>vg_mda_free</b>	このボリュームグループの空きメタデータ領域 (現在の単位)	サイズ
<b>vg_mda_size</b>	このボリュームグループの最小メタデータ領域のサイズ (現在の単位)	サイズ
<b>vg_mda_copies</b>	ボリュームグループの使用メタデータ領域のターゲット数	数値

表C.12「論理ボリュームセグメントフィールド」では、論理ボリュームセグメントフィールドとそのフィールドタイプについて説明しています。

表C.12 論理ボリュームセグメントフィールド

論理ボリュームセグメント フィールド	説明	フィールドタイプ
<b>segtype</b>	論理ボリュームセグメントのタイプ	文字列
<b>stripes</b>	ストライプまたはミラーレグの数	数値
<b>stripesize</b>	ストライプの場合は、次のデバイスに切り替わる前の該当するデバイスに配置されたデータのサイズ	サイズ
<b>stripe_size</b>	ストライプの場合は、次のデバイスに切り替わる前の該当するデバイスに配置されたデータのサイズ	サイズ
<b>regionsize</b>	ミラーの場合は、デバイスを同期するときにコピーされたデータの単位	サイズ
<b>region_size</b>	ミラーの場合は、デバイスを同期するときにコピーされたデータの単位	サイズ
<b>chunksize</b>	スナップショットの場合は、変更を追跡するときに使用されたデータの単位	サイズ
<b>chunk_size</b>	スナップショットの場合は、変更を追跡するときに使用されたデータの単位	サイズ

論理ボリュームセグメントフィールド	説明	フィールドタイプ
<b>thin_count</b>	シンプールの場合は、このプール内のシンボリウムの数	数値
<b>discards</b>	シンプールの場合は、破棄が処理される方法	文字列
<b>cachemode</b>	キャッシュプールの場合は、書き込みがキャッシュされる方法	文字列
<b>zero</b>	シンプールの場合は、ゼロ化が有効なとき	数値
<b>transaction_id</b>	シンプールの場合は、トランザクション ID	数値
<b>thin_id</b>	シンボリウムの場合は、シンデバイス ID	数値
<b>seg_start</b>	セグメントの始まりに対する論理ボリューム内のオフセット (現在の単位)	サイズ
<b>seg_start_pe</b>	セグメントの始まりに対する論理ボリューム内のオフセット (物理エクステント数)	数値
<b>seg_size</b>	セグメントのサイズ (現在の単位)	サイズ
<b>seg_size_pe</b>	セグメントのサイズ (物理エクステント数)	サイズ
<b>seg_tags</b>	タグ (存在する場合)	文字列リスト
<b>seg_pe_ranges</b>	基礎となるデバイスの物理エクステントの範囲 (コマンドライン形式)	文字列
<b>devices</b>	開始エクステント番号で使用されている基礎となるデバイス	文字列
<b>seg_monitor</b>	セグメントの dmeventd 監視ステータス	文字列
<b>cache_policy</b>	キャッシュポリシー (キャッシュされたセグメントのみ)	文字列
<b>cache_settings</b>	キャッシュ設定/パラメーター (キャッシュされたセグメントのみ)	文字列リスト

表C.13「物理ボリュームセグメントフィールド」では、物理ボリュームセグメントフィールドとそのフィールドタイプについて説明しています。

表C.13 物理ボリュームセグメントフィールド

物理ボリュームセグメントフィールド	説明	フィールドタイプ
<b>pvseg_start</b>	セグメントの始まりの物理エクステント番号	数値
<b>pvseg_size</b>	セグメントのエクステントの数	数値

表C.14「[選択基準同意義](#)」には、フィールド値に使用できる同意語が一覧表示されています。これらの同意語は選択基準と元の値と同様な値で使用できます。以下の表では、"" のフィールド値は、空の文字列を示し、**-S 'field\_name=""** と指定することにより一致させることができます。

この表では、0 または 1 で示されたフィールドはバイナリー値を示します。レポートツールに **--binary** オプションを指定し、バイナリーフィールドにおいて、この表で "some text" または "" と示されているものの代わりに 0 または 1 を表示させることができます。

表C.14 選択基準同意義

フィールド	フィールド値	同意義
<b>pv_allocatable</b>	allocatable	1
<b>pv_allocatable</b>	""	0
<b>pv_exported</b>	exported	1
<b>pv_exported</b>	""	0
<b>pv_missing</b>	missing	1
<b>pv_missing</b>	""	0
<b>vg_extendable</b>	extendable	1
<b>vg_extendable</b>	""	0
<b>vg_exported</b>	exported	1
<b>vg_exported</b>	""	0
<b>vg_partial</b>	partial	1
<b>vg_partial</b>	""	0
<b>vg_clustered</b>	clustered	1
<b>vg_clustered</b>	""	0
<b>vg_permissions</b>	writable	rw、read-write

フィールド	フィールド値	同意義
<b>vg_permissions</b>	read-only	r、ro
<b>vg_mda_copies</b>	unmanaged	unknown、undefined、undef、-1
<b>lv_initial_image_sync</b>	initial image sync	sync、1
<b>lv_initial_image_sync</b>	""	0
<b>lv_image_synced</b>	image synced	synced、1
<b>lv_image_synce</b>	""	0
<b>lv_merging</b>	merging	1
<b>lv_merging</b>	""	0
<b>lv_converting</b>	converting	1
<b>lv_converting</b>	""	0
<b>lv_allocation_locked</b>	allocation locked	locked、1
<b>lv_allocation_locked</b>	""	0
<b>lv_fixed_minor</b>	fixed minor	fixed、1
<b>lv_fixed_minor</b>	""	0
<b>lv_active_locally</b>	active locally	active、locally、1
<b>lv_active_locally</b>	""	0
<b>lv_active_remotely</b>	active remotely	active、remotely、1
<b>lv_active_remotely</b>	""	0
<b>lv_active_exclusively</b>	active exclusively	active、exclusively、1
<b>lv_active_exclusively</b>	""	0
<b>lv_merge_failed</b>	merge failed	failed、1
<b>lv_merge_failed</b>	""	0
<b>lv_snapshot_invalid</b>	snapshot invalid	invalid、1

フィールド	フィールド値	同意義
<b>lv_snapshot_invalid</b>	""	0
<b>lv_suspended</b>	suspended	1
<b>lv_suspended</b>	""	0
<b>lv_live_table</b>	live table present	live table、live、1
<b>lv_live_table</b>	""	0
<b>lv_inactive_table</b>	inactive table present	inactive table、inactive、1
<b>lv_inactive_table</b>	""	0
<b>lv_device_open</b>	open	1
<b>lv_device_open</b>	""	0
<b>lv_skip_activation</b>	skip activation	skip、1
<b>lv_skip_activation</b>	""	0
<b>zero</b>	zero	1
<b>zero</b>	""	0
<b>lv_permissions</b>	writable	rw、read-write
<b>lv_permissions</b>	read-only	r、ro
<b>lv_permissions</b>	read-only-override	ro-override、r-override、R
<b>lv_when_full</b>	error	error when full、error if no space
<b>lv_when_full</b>	queue	queue when full、queue if no space
<b>lv_when_full</b>	""	undefined
<b>cache_policy</b>	""	undefined
<b>seg_monitor</b>	""	undefined
<b>lv_health_status</b>	""	undefined

## C.4. 時間値の指定

LVM 選択に時間値を指定する場合は、「標準的な時間選択形式」と「自由な形式の時間選択形式」で説明されたように、標準化された時間仕様形式またはより自由な形式のいずれかを使用できます。

レポート/時間形式設定オプションで時間値が表示される方法は、`/etc/lvm/lvm.conf` 設定ファイルで指定できます。このオプションの指定に関する情報は、`lvm.conf` ファイルで提供されています。

時間値を指定する場合は、表C.3「選択基準比較演算子」で説明されたように、比較演算子エイリアス **since**、**after**、**until**、および **before** を使用できます。

### C.4.1. 標準的な時間選択形式

LVM 選択の時間値は、以下の形式で指定できます。

```
date time timezone
```

表C.15「時間仕様形式」では、これらの時間値を指定するときに使用できる形式について簡単に説明しています。

表C.15 時間仕様形式

フィールド	フィールド値
date	<div>YYYY-MM-DD</div> <div>YYYY-MM、auto DD=1</div> <div>YYYY、auto MM=01、および DD=01</div>
時間	<div>hh:mm:ss</div> <div>hh:mm、auto ss=0</div> <div>hh、auto mm=0、auto ss=0</div>
timezone (常に + または - 記号が使用される)	<div>+hh:mm または -hh:mm</div> <div>+hh または -hh</div>

完全な日付/時間仕様は YYYY-MM-DD hh:mm:ss です。ユーザーは日付/時間部分を右から左の順序で省略できます。これらの部分が省略されると、秒単位で範囲が自動的に計算されます。以下に例を示します。

- "2015-07-07 9:51" は、"2015-07-07 9:51:00"～"2015-07-07 9:51:59" の範囲を意味します。
- "2015-07" は "2015-07-01 0:00:00"～"2015-07-31 23:59:59" の範囲を意味します。

- "2015" は "2015-01-01 0:00:00"～"2015-12-31 23:59:59" の範囲を意味します。

以下の例は、選択基準で使用する日付/時間仕様を示しています。

```
lvs -S 'time since "2015-07-07 9:51"'
lvs -S 'time = "2015-07"'
lvs -S 'time = "2015"'
```

### C.4.2. 自由な形式の時間選択形式

LVM 選択基準の日付/時間仕様は、以下の項目を使用して指定できます。

- 平日の名前 ("Sunday"～"Saturday" または省略した "Sun"～"Sat")
- 特定の時間のラベル ("noon"、"midnight")
- 現在の日に相対する日のラベル ("today"、"yesterday")
- 今日からの相対的なオフセット値を使用した特定の時間 (N は数字)
- ( "N" "seconds"/"minutes"/"hours"/"days"/"weeks"/"years" "ago")
- ( "N" "secs"/"mins"/"hrs" ... "ago")
- ( "N" "s"/"m"/"h" ... "ago")
- hh:mm:ss 形式または AM/PM 接尾辞を使用した時間形式
- 月の名前 ("January"～"December" または省略した "Jan"～"Dec")

以下の例は、選択基準で使用する **freeform** 日付/時間仕様を示しています。

```
lvs -S 'time since "yesterday 9AM"'
lvs -S 'time since "Feb 3 years 2 months ago"'
lvs -S 'time = "February 2015"'
lvs -S 'time since "Jan 15 2015" && time until yesterday'
lvs -S 'time since "today 6AM"'
```

## C.5. 選択基準表示の例

このセクションでは、LVM 表示コマンドの選択基準を使用する方法を示す一連の例を提供します。このセクションの例では、選択基準が使用されていない場合に以下の出力を提供する LVM ボリュームが設定されたシステムを使用します。

```
# lvs -a -o+layout,role
LV          VG Attr          LSize   Pool Origin Data%   Meta%   Layout
Role
  root      f1 -wi-ao-----   9.01g
public
  swap      f1 -wi-ao----- 512.00m
public
  [lvol0_pmspare] vg ewi-----   4.00m
private,    \

pool, spare
```

```

    lvol1          vg Vwi-a-tz--  1.00g pool          0.00
thin,sparse public
    lvol2          vg Vwi-a-tz--  1.00g pool          0.00
thin,sparse public, \

origin, \

thinorigin
    lvol3          vg Vwi---tz-k   1.00g pool lvol2
thin,sparse public, \

snapshot, \

thinsnapshot
    pool          vg twi-aotz-- 100.00m          0.00   1.07
thin,pool private
    [pool_tdata]   vg Twi-ao---- 100.00m          linear
private, \

thin,pool, \

data
    [pool_tmeta]   vg ewi-ao----  4.00m          linear
private, \

thin,pool, \

metadata

```

以下のコマンドは、名前が "lvol[13]" のすべての論理ボリュームを正規表現を使用して表示します。

```

# lvs -a -o+layout,role -S 'lv_name=~lvol[13]'
LV   VG   Attr      LSize Pool Origin Data% Layout      Role
lvol1 vg   Vwi-a-tz-- 1.00g pool          0.00 thin,sparse public
lvol3 vg   Vwi---tz-k 1.00g pool lvol2      thin,sparse
public,snapshot,thinsnapshot

```

以下のコマンドは、サイズが 500 メガバイトを超えるすべての論理ボリュームを表示します。

```

# lvs -a -o+layout,role -S 'lv_size>500m'
LV   VG   Attr      LSize Pool Origin Data% Layout      Role
root  f1   -wi-ao---- 9.01g                linear      public
swap  f1   -wi-ao---- 512.00m              linear      public
lvol1 vg   Vwi-a-tz-- 1.00g pool          0.00 thin,sparse public
lvol2 vg   Vwi-a-tz-- 1.00g pool          0.00 thin,sparse
public,origin,thinorigin
lvol3 vg   Vwi---tz-k 1.00g pool lvol2      thin,sparse
public,snapshot, \

thinsnapshot

```

以下のコマンドは、論理ボリュームロールとして **thin** を含むすべての論理ボリュームを表示します (シンプルな構築で論理ボリュームが使用されたことを示しています)。この例では、表示でのサブセットを示すためにカッコ ({} ) を使用します。



```
# lvs -a -o+layout,role -S 'lv_role={thin}'
LV          VG      Attr          LSize   Layout        Role
[pool_tdata] vg      Twi-ao---- 100.00m linear        private,thin,pool,data
[pool_tmeta] vg      ewi-ao----   4.00m linear        private,thin,pool,metadata
```

以下のコマンドは、使用可能なすべての最上位論理ボリューム (ロールが "public" の論理ボリューム) を表示します。サブセットを示すために文字列リストでかっこ ({} ) を指定しない場合は、デフォルトで指定されたものと見なされます。**lv\_role=public** を指定することは **lv\_role={public}** を指定することと同じです。

```
# lvs -a -o+layout,role -S 'lv_role=public'
LV      VG Attr          LSize   Pool Origin Data%   Layout        Role
root    f1 -wi-ao----   9.01g                linear        public
swap    f1 -wi-ao---- 512.00m                linear        public
lvol1   vg Vwi-a-tz--   1.00g pool          0.00   thin,sparse public
lvol2   vg Vwi-a-tz--   1.00g pool          0.00   thin,sparse
public,origin,thinorigin
lvol3   vg Vwi---tz-k   1.00g pool lvol2          thin,sparse
public,snapshot,thinsnapshot
```

以下のコマンドは、シンレイアウトとともにすべての論理ボリュームを表示します。

```
# lvs -a -o+layout,role -S 'lv_layout={thin}'
LV      VG Attr          LSize   Pool Origin Data% Meta% Layout        Role
lvol1   vg Vwi-a-tz--   1.00g pool          0.00          thin,sparse public
lvol2   vg Vwi-a-tz--   1.00g pool          0.00          thin,sparse
public,origin,
\
thinorigin
lvol3   vg Vwi---tz-k   1.00g pool lvol2          thin,sparse
public,snapshot,
\
thinsnapshot
pool    vg twi-aotz-- 100.00m                0.00  1.07   thin,pool   private
```

以下のコマンドは、"sparse,thin" に完全一致するレイアウトフィールドとともにすべての論理ボリュームを表示します。一致候補を検出するには文字列リストメンバーを指定する必要がないことに注意してください。

```
# lvs -a -o+layout,role -S 'lv_layout=[sparse,thin]'
LV      VG Attr          LSize   Pool Origin Data%   Layout        Role
lvol1   vg Vwi-a-tz--   1.00g pool          0.00   thin,sparse public
lvol2   vg Vwi-a-tz--   1.00g pool          0.00   thin,sparse
public,origin,thinorigin
lvol3   vg Vwi---tz-k 1.00g pool lvol2          thin,sparse
public,snapshot,thinsnapshot
```

以下のコマンドはシンおよびスパーズ論理ボリュームである論理ボリュームの論理ボリューム名を表示します。選択基準に使用されるフィールドのリストは表示するフィールドのリストと同じである必要がないことに注意してください。

```
# lvs -a -o lv_name -S 'lv_layout=[sparse,thin]'
LV
```

```
lvol1
lvol2
lvol3
```

## C.6. 選択基準処理の例

このセクションでは、LVM 論理ボリュームを処理するコマンドで選択基準を使用する方法を示す一連の例を提供します。

この例は、シンスナップショットを含む論理ボリュームのグループの初期設定を示しています。シンスナップショットには "skip activation" フラグがデフォルトで設定されています。また、この例には、"skip activation" フラグが設定された論理ボリューム **lvol4** も含まれます。

```
# lvs -o name,skip_activation,layout,role
LV      SkipAct      Layout      Role
root                    linear      public
swap                    linear      public
lvol1                    thin,sparse public
lvol2                    thin,sparse public,origin,thinorigin
lvol3 skip activation thin,sparse public,snapshot,thinsnapshot
lvol4 skip activation linear      public
pool                    thin,pool   private
```

以下のコマンドは、シンスナップショットであるすべての論理ボリュームから skip activation フラグを削除します。

```
# lvchange --setactivationskip n -S 'role=thinsnapshot'
Logical volume "lvol3" changed.
```

以下のコマンドは、**lvchange** コマンドの実行後に論理ボリュームの設定を表示します。"skip activation" フラグはシンスナップショットではない論理ボリュームから設定解除されていないことに注意してください。

```
# lvs -o name,active,skip_activation,layout,role
LV      Active SkipAct      Layout      Role
root    active                    linear      public
swap    active                    linear      public
lvol1    active                    thin,sparse public
lvol2    active                    thin,sparse public,origin,thinorigin
lvol3                    thin,sparse public,snapshot,thinsnapshot
lvol4    active skip activation linear      public
pool     active                    thin,pool   private
```

以下のコマンドは、追加のシン作成元/スナップショットボリュームが作成されたあとに論理ボリュームの設定を表示します。

```
# lvs -o name,active,skip_activation,origin,layout,role
LV      Active SkipAct      Origin Layout      Role
root    active                    linear      public
swap    active                    linear      public
lvol1    active                    thin,sparse public
lvol2    active                    thin,sparse
public,origin,thinorigin
lvol3                    lvol2 thin,sparse
```

```

public, snapshot, thinsnapshot
  lv14 active skip activation      linear      public
  lv15 active                      thin, sparse
public, origin, thinorigin
  lv16                               lv15  thin, sparse
public, snapshot, thinsnapshot
  pool  active                      thin, pool   private

```

以下のコマンドは、シンスナップショットボリュームであり、作成元ボリュームが **lv12** の論理ボリュームをアクティブ化します。

```

# lvchange -ay -S 'lv_role=thinsnapshot && origin=lv12'

# lvs -o name,active,skip_activation,origin,layout,role
LV      Active SkipAct      Origin Layout      Role
root    active              linear      public
swap    active              linear      public
lv11    active              thin, sparse public
lv12    active              thin, sparse
public, origin, thinorigin
  lv13    active              lv12  thin, sparse
public, snapshot, thinsnapshot
  lv14    active skip activation      linear      public
  lv15    active                      thin, sparse
public, origin, thinorigin
  lv16                               lv15  thin, sparse
public, snapshot, thinsnapshot
  pool    active                      thin, pool   private

```

アイテム全体から1つのアイテムに一致する選択基準を指定するときにアイテム全体に対してコマンドを実行する場合は、アイテム全体が処理されます。たとえば、ボリュームグループから1つまたは複数のアイテムを選択するときにボリュームグループを変更する場合、ボリュームグループ全体が選択されます。この例では、ボリュームグループ **vg** の一部である論理ボリューム **lv11** が選択されます。また、ボリュームグループ **vg** 内のすべての論理ボリュームが処理されます。

```

# lvs -o name,vg_name
LV      VG
root    fedora
swap    fedora
lv11    vg
lv12    vg
lv13    vg
lv14    vg
lv15    vg
lv16    vg
pool    vg

# vgchange -ay -S 'lv_name=lv11'
7 logical volume(s) in volume group "vg" now active

```

以下の例は、より複雑な選択基準ステートメントを示しています。この例では、すべての論理ボリュームのロールが作成元であり、ボリュームの名前が **lv1[456]** である場合、または論理ボリュームのサイズが5ギガバイトを超える場合、すべての論理ボリュームには **mytag** というタグが付けられます。

```

# lvchange --addtag mytag -S '(role=origin && lv_name=~lv1[456]) ||

```

```
lv_size > 5g'  
Logical volume "root" changed.  
Logical volume "lvol5" changed.
```

## 付録D LVM オブジェクトタグ

LVM タグは、同じタイプの LVM2 オブジェクトを 1 つのグループにまとめるのに使用される用語です。タグは、物理ボリューム、ボリュームグループ、論理ボリュームなどのオブジェクトに割り当てることができます。クラスター構成ではタグをホストに割り当てることができます。

タグは、コマンドラインで引数 PV、VG、または LV の代わりに表示することができます。混乱を防ぐために、タグの先頭には @ を付ける必要があります。各タグは、コマンドライン上の位置から想定されるタイプの、そのタグを処理するすべてのオブジェクトに置き換えることによって拡張されます。

LVM タグは、最長 1024 文字の文字列です。LVM タグの先頭にハイフンを使用することはできません。

有効なタグには、限定された範囲の文字のみで構成されます。使用可能な文字は [A-Za-z0-9\_+.-] です。Red Hat Enterprise Linux 6.1 リリースでは、使用可能な文字の一覧が拡大され、タグには "/"、"="、"!", ":", "#", および "&" の文字が使用できるようになりました。

タグは、ボリュームグループ内のオブジェクトにのみ付けられます。ボリュームグループから物理ボリュームを削除すると、そのタグはなくなります。タグは、ボリュームグループメタデータの一部として保存されているため、物理ボリュームを削除する際にともに削除されるためです。

以下のコマンドは、**database** タグを持つすべての論理ボリュームを一覧表示します。

```
# lvs @database
```

以下のコマンドは、現在アクティブなホストタグを一覧表示します。

```
# lvm tags
```

### D.1. オブジェクトタグの追加と削除

物理ボリュームにタグを追加したり、そこからタグを削除したりするには、**pvchange** コマンドで **--addtag** オプションや **--deltag** オプションを使用します。

ボリュームグループにタグを追加したり、そこからタグを削除するには、**vgchange** または **vgcreate** コマンドで **--addtag** や、**--deltag** オプションを使用します。

論理ボリュームにタグを追加したり、そこからタグを削除するには、**lvchange** または **lvcreate** コマンドで **--addtag** や **--deltag** オプションを使用します。

**pvchange**、**vgchange**、または **lvchange** の単一のコマンドで、**--addtag** および **--deltag** の引数を複数指定することができます。たとえば、以下のコマンドでは、タグ **T9** と **T10** が削除され、the タグ **T13** と **T14** がボリュームグループ **grant** に追加されます。

```
# vgchange --deltag T9 --deltag T10 --addtag T13 --addtag T14 grant
```

### D.2. ホストタグ

クラスター構成では、設定ファイルにホストタグを定義できます。**tags** セクションに **hosttags = 1** を設定すると、マシンのホスト名からホストタグが自動的に定義されます。これにより、すべてのマシンに複製できる共通の設定ファイルを使用できるようになり、各マシンにファイルの同一コピーが維持されますが、ホスト名に応じてマシン間で動作が異なる可能性があります。

設定ファイルに関する情報は、「[付録B LVM 設定ファイル](#)」を参照してください。

各ホストタグには、存在する場合は余分の設定ファイルが読み込まれます (`lvm_hosttag.conf`)。このファイルが新規タグを定義する場合、更なる設定ファイルが読み取りのためにファイルの一覧に追記されます。

たとえば、設定ファイル内の以下の以下のエントリは常に、**tag1** を定義し、ホスト名が **host1** の場合は、**tag2** を定義します。

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

### D.3. タグを使用したアクティブ化の制御

特定の論理ボリュームのみがホスト上でアクティブ化されるように設定ファイルで指定することができます。たとえば、以下のエントリはアクティベーション要求 (**vgchange -ay** など) のフィルタとして動作して、**vg1/lvol0** とホスト上のメタデータ内に **database** タグを持ついずれかの論理ボリューム、またはボリュームグループのみをアクティブ化します。

```
activation { volume_list = ["vg1/lvol0", "@database" ] }
```

いずれかのメタデータタグがマシンのホストタグのいずれかに一致する場合のみに一致する要因となる特別一致記号 `"@"` が存在します。

もう一つの例として、クラスター内の各マシンの設定ファイルに以下のエントリがあるような状況を考えてみます。

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

ホスト **db2** 上のみで **vg1/lvol2** をアクティブ化したい場合は、以下のようにします:

1. クラスター内のいずれかのホストから **lvchange --addtag @db2 vg1/lvol2** を実行します。
2. **lvchange -ay vg1/lvol2** を実行します。

この解決法では、ボリュームグループメタデータの中にホスト名を保存する必要があります。

## 付録E LVM ボリュームグループメタデータ

ボリュームグループの設定の詳細は、メタデータと呼ばれます。デフォルトでは、メタデータの同一のコピーは、ボリュームグループ内のすべての物理ボリュームのすべてのメタデータ領域で管理されます。LVM ボリュームグループメタデータは ASCII として保存されます。

ボリュームグループに物理ボリュームが多数含まれる場合は、メタデータのコピーが過剰になるため、効率的ではありません。**pvcreate** コマンドで **--metadatasize 0** オプションを使用すれば、作成する物理ボリュームにメタデータのコピーが含まれません。物理ボリュームに含まれるメタデータコピーの数は、一度選択すると後で変更することはできません。コピー数を 0 と選択すると、設定変更時の更新が迅速になります。しかし、すべてのボリュームグループには常に 1 つのメタデータ領域を持つ物理ボリュームが最低 1 つ含まれる必要があることに注意してください (高度な設定を使用して、ボリュームグループメタデータをファイルシステムに保存できる場合を除く)。後でボリュームグループを分割する予定がある場合は、それぞれのボリュームグループにメタデータのコピーが 1 つ以上必要になります。

核となるメタデータは ASCII 形式で保存されます。メタデータ領域は循環バッファです。新規のメタデータは古いメタデータに追記され、その開始点へのポインターが更新されます。

メタデータ領域のサイズは、**pvcreate** コマンドの **--metadatasize** オプションで指定できます。ボリュームグループに物理ボリュームや論理ボリュームが数百もある場合は、デフォルトのサイズでは小さすぎる場合があります。

### E.1. 物理ボリュームラベル

**pvcreate** コマンドは、デフォルトで、物理ボリュームラベルを 2 番目の 512 バイトセクターに配置しますが、別のセクター (最初の 4 つのセクターのいずれか) に配置できます。物理ボリュームラベルをスキャンする LVM ツールは、この 4 つのセクターをチェックするためです。物理ボリュームラベルは文字列 **LABELONE** で始まります。

物理ボリュームラベルに含まれる内容:

- 物理ボリューム UUID
- ブロックデバイスの容量 (バイト)
- NULL で終了するデータ領域ロケーションの一覧
- NULL で終了するメタデータ領域ロケーションの一覧

メタデータロケーションは、オフセットおよびサイズ (バイト単位) で保存されます。ラベルには 15 ロケーション用のスペースがありますが、現在、LVM ツールでは、そのうち 3 つしか使いません。データ領域が 1 つと、最大 2 つのメタデータ領域です。

### E.2. メタデータの内容

ボリュームグループメタデータに含まれる内容:

- それが作成された手段と日時の情報
- ボリュームグループに関する情報

ボリュームグループ情報に含まれる内容:

- 名前と一意識別子

- バージョン番号 (メタデータが更新される度に増加します)
- プロパティ (read、write、resizable など)
- そこに含まれる物理ボリュームと論理ボリュームの数に対する管理制限
- エクステントのサイズ (512 バイトで定義されるセクターの単位)
- ボリュームグループを構成する物理ボリュームの一覧 (順序が付けられていません)。一覧には以下が含まれます。
  - UUID (物理ボリュームを格納するブロックデバイスの確認に使用)
  - プロパティ (物理ボリュームの割り当て可能性など)
  - 物理ボリューム内の 1 番目のエクステントの開始点までのオフセット (セクターで表示)
  - エクステントの数
- 論理ボリュームの一覧 (順序は付けられていません)。以下の要素で構成されています。
  - 論理ボリュームセグメントの一覧 (順序は付けられています)。各セグメントについて、メタデータに、物理ボリュームセグメントまたは論理ボリュームセグメント一覧 (順序は付けられています) に適用されるマッピングが含まれます。

### E.3. サンプルのメタデータ

以下の例は、**myvg** ボリュームグループの LVM ボリュームグループメタデータになります。

```
# 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
]
}
}
}
```

## 付録F 改訂履歴

<b>改訂 1.0-11.1</b> 翻訳ファイルを XML ソースバージョン 1.0-11 と同期	<b>Sun Sep 24 2017</b>	<b>Terry Chuang</b>
<b>改訂 1.0-11</b> 7.4 GA 公開用バージョン。	<b>Wed Jul 19 2017</b>	<b>Steven Levine</b>
<b>改訂 1.0-9</b> 7.4 ベータ公開用ドキュメントの準備	<b>Mon May 15 2017</b>	<b>Steven Levine</b>
<b>改訂 1.0-7</b> 7.3 のバージョンを更新	<b>Mon Mar 27 2017</b>	<b>Steven Levine</b>
<b>改訂 1.0-5</b> 7.3 GA 公開用バージョン	<b>Mon Oct 17 2016</b>	<b>Steven Levine</b>
<b>改訂 1.0-4</b> 7.3 ベータ公開用ドキュメントの準備	<b>Wed Aug 17 2016</b>	<b>Steven Levine</b>
<b>改訂 0.3-4</b> 7.2 GA 公開用ドキュメントの準備	<b>Mon Nov 9 2015</b>	<b>Steven Levine</b>
<b>改訂 0.3-2</b> 7.2 ベータ公開用ドキュメントの準備	<b>Wed Aug 19 2015</b>	<b>Steven Levine</b>
<b>改訂 0.2-7</b> 7.1 GA リリース向けのバージョン	<b>Mon Feb 16 2015</b>	<b>Steven Levine</b>
<b>改訂 0.2-6</b> 7.1 Beta リリースバージョン	<b>Thu Dec 11 2014</b>	<b>Steven Levine</b>
<b>改訂 0.1-22</b> 7.0 GA リリース向けバージョン	<b>Mon Jun 2 2014</b>	<b>Steven Levine</b>
<b>改訂 0.1-1</b> ドキュメントの Red Hat Enterprise Linux 6 バージョンからブランチ作成	<b>Wed Jan 16 2013</b>	<b>Steven Levine</b>

## 索引

### シンボル

[/lib/udev/rules.d ディレクトリー](#), [udev のデバイスマッパーとの統合](#)

[アーカイブファイル](#), [論理ボリュームのバックアップ](#)

### エクステント

[割り当て](#), [ボリュームグループの作成](#), [LVM の割り当て](#)

[定義](#), [ボリュームグループ](#), [ボリュームグループの作成](#)

[オンラインデータ移動](#), [オンラインデータ移動](#)

### キャッシュファイル

[構築](#), [キャッシュファイル構築のためのボリュームグループのディスクスキャン](#)

[キャッシュボリューム](#), [キャッシュボリューム](#)

### キャッシュ論理ボリューム

[作成](#), [LVM 論理ボリュームの作成](#)

[クラスター環境](#), [クラスター化論理ボリュームマネージャー \(CLVM\)](#), [クラスター内での LVM ボリューム作成](#)

[コマンドラインユニット](#), [CLI コマンドの使用](#)

### サイズ変更

[物理ボリューム](#), [物理ボリュームのサイズ変更](#)

[シンプロビジョニングされたスナップショットボリューム](#), [シンプロビジョニングされたスナップショットボリューム](#)

[シンプロビジョニングされたスナップショット論理ボリューム](#)

[作成](#), [シンプロビジョニングされたスナップショットボリュームの作成](#)

[シンプロビジョニングされた論理ボリューム](#), [シンプロビジョニングされた論理ボリューム \(シンボリック\)](#)

[作成](#), [シンプロビジョニングされた論理ボリュームの作成](#)

### スキャン

[ブロックデバイス](#), [ブロックデバイスのスキャン](#)

### ストライプ化論理ボリューム

[作成](#), [ストライプ化ボリュームの作成](#)

[作成例](#), [ストライプ化論理ボリュームの作成](#)

[定義](#), [ストライプ化論理ボリューム](#)

[拡張](#), [ストライプ化ボリュームの拡張](#)

### スナップショットボリューム

[定義](#), [スナップショットボリューム](#)

### スナップショット論理ボリューム

[作成](#), [スナップショットボリュームの作成](#)

デバイスのスキャン、フィルター、[フィルターを使用した LVM デバイススキャンの制御](#)

デバイスサイズ、最大、[ボリュームグループの作成](#)

デバイススキャンフィルター、[フィルターを使用した LVM デバイススキャンの制御](#)

デバイスパス名、[CLI コマンドの使用](#)

デバイス特有のファイルがあるディレクトリー、[ボリュームグループの作成](#)

デバイス番号

マイナー、[永続的なデバイス番号](#)

メジャー、[永続的なデバイス番号](#)

永続的、[永続的なデバイス番号](#)

データ移動、オンライン、[オンラインデータ移動](#)

トラブルシューティング、[LVM トラブルシューティング](#)

バックアップ

ファイル、[論理ボリュームのバックアップ](#)

メタデータ、[論理ボリュームのバックアップ](#)、[ボリュームグループメタデータのバックアップ](#)

パス名、[CLI コマンドの使用](#)

パーティション

複数、[ディスク上の複数パーティション](#)

パーティションタイプ、設定、[パーティションタイプの設定](#)

ファイルシステム

[論理ボリューム上における拡張](#)、[論理ボリュームのファイルシステムの拡張](#)

ファイルシステムの拡張

[論理ボリューム](#)、[論理ボリュームのファイルシステムの拡張](#)

フィルター、[フィルターを使用した LVM デバイススキャンの制御](#)

ブロックデバイス

スキャン、[ブロックデバイスのスキャン](#)

ヘルプの表示、[CLI コマンドの使用](#)

ボリュームグループ

[vgs](#) 引数の表示、[vgs コマンド](#)

アクティブ化、[ボリュームグループのアクティブ化と非アクティブ化](#)

クラスター内での作成、[クラスター内でのボリュームグループ作成](#)

システム間での移動、[ボリュームグループの別のシステムへの移動](#)

パラメーターの変更、[ボリュームグループのパラメーター変更](#)

マージ、[ボリュームグループの統合](#)

作成、[ボリュームグループの作成](#)

分割、[ボリュームグループの分割](#)

手順の例、[ボリュームグループの分割](#)

削除、[ボリュームグループの削除](#)

名前変更, [ボリュームグループの名前変更](#)

定義, [ボリュームグループ](#)

拡張, [ボリュームグループへの物理ボリュームの追加](#)

管理、一般, [ボリュームグループの管理](#)

結合, [ボリュームグループの統合](#)

縮小, [ボリュームグループからの物理ボリュームの削除](#)

表示, [ボリュームグループの表示](#), [LVM 用のカスタム報告](#), [vgs コマンド](#)

非アクティブ化, [ボリュームグループのアクティブ化と非アクティブ化](#)

[ボリュームグループのアクティブ化](#), [ボリュームグループのアクティブ化と非アクティブ化](#)

[ボリュームグループの非アクティブ化](#), [ボリュームグループのアクティブ化と非アクティブ化](#)

[ミラー化論理ボリューム](#)

クラスター化された, [クラスター内でのミラー化 LVM 論理ボリュームの作成](#)

リニアへの変換, [ミラー化ボリューム設定の変更](#)

作成, [ミラー化ボリュームの作成](#)

再設定, [ミラー化ボリューム設定の変更](#)

障害ポリシー, [ミラー化論理ボリュームの障害ポリシー](#)

障害回復, [LVM ミラー障害からの回復](#)

[メタデータ](#)

バックアップ, [論理ボリュームのバックアップ](#), [ボリュームグループメタデータのバックアップ](#)

回復, [物理ボリュームメタデータの復元](#)

[メタデータデーモン](#), [メタデータデーモン \(lvmetad\)](#)

[ユニット](#), [コマンドライン](#), [CLI コマンドの使用](#)

[リニア論理ボリューム](#)

ミラー化への変換, [ミラー化ボリューム設定の変更](#)

作成, [リニア論理ボリュームの作成](#)

定義, [リニアボリューム](#)

[レポートのフォーマット](#), [LVM デバイス](#), [LVM 用のカスタム報告](#)

[ロギング](#), [ロギング](#)

[不十分な空きエクステンツメッセージ](#), [論理ボリュームでの不十分な空きエクステンツ](#)

[作成](#)

[クラスター内でのLVM ボリューム](#), [クラスター内での LVM ボリューム作成](#)

[ストライプ化論理ボリューム](#), 例, [ストライプ化論理ボリュームの作成](#)

[ボリュームグループ](#), [ボリュームグループの作成](#)

[ボリュームグループ](#), [クラスター化](#), [クラスター内でのボリュームグループ作成](#)

[物理ボリューム](#), [物理ボリュームの作成](#)

[論理ボリューム](#), [リニア論理ボリュームの作成](#)

[論理ボリューム](#), 例, [3つのディスク上での LVM 論理ボリューム作成](#)

[初期化](#)

[パーティション](#), [物理ボリュームの初期化](#)

物理ボリューム, [物理ボリュームの初期化](#)

## 削除

物理ボリューム, [物理ボリュームの削除](#)

論理ボリューム, [論理ボリュームの削除](#)

論理ボリュームからディスクを, [論理ボリュームからディスクを削除する](#)

## 割り当て, [LVM の割り当て](#)

ポリシー, [ボリュームグループの作成](#)

防止, [物理ボリューム上での割り当て防止](#)

## 名前変更

ボリュームグループ, [ボリュームグループの名前変更](#)

論理ボリューム, [論理ボリュームの名前変更](#)

新機能および変更された機能, [新機能および変更された機能](#)

## 概要

新機能および変更された機能, [新機能および変更された機能](#)

永続的なデバイス番号, [永続的なデバイス番号](#)

## 物理エクステンツ

割り当ての防止, [物理ボリューム上での割り当て防止](#)

## 物理ボリューム

[pvs 引数の表示](#), [pvs コマンド](#)

サイズ変更, [物理ボリュームのサイズ変更](#)

ボリュームグループから削除, [ボリュームグループからの物理ボリュームの削除](#)

ボリュームグループに追加, [ボリュームグループへの物理ボリュームの追加](#)

レイアウト, [LVM 物理ボリュームのレイアウト](#)

作成, [物理ボリュームの作成](#)

初期化, [物理ボリュームの初期化](#)

削除, [物理ボリュームの削除](#)

回復, [紛失した物理ボリュームの入れ替え](#)

図, [LVM 物理ボリュームのレイアウト](#)

定義, [物理ボリューム](#)

管理、一般, [物理ボリュームの管理](#)

紛失したボリュームの削除, [紛失した物理ボリュームのボリュームグループからの削除](#)

表示, [物理ボリュームの表示](#), [LVM 用のカスタム報告](#), [pvs コマンド](#)

管理の手順, [LVM 管理の概要](#)

## 縮小

論理ボリューム, [論理ボリュームの縮小](#)

## 表示

ボリュームグループ, [ボリュームグループの表示](#), [vgs コマンド](#)

出力のソート, [LVM 報告のソート](#)

物理ボリューム, [物理ボリュームの表示](#), [pvs コマンド](#)

論理ボリューム, [論理ボリュームの表示](#), [lvs コマンド](#)

設定の例, [LVM 設定の例](#)

詳細出力, [CLI コマンドの使用](#)

論理ボリューム

[lvs 引数の表示](#), [lvs コマンド](#)

キャッシュ, [LVM 論理ボリュームの作成](#)

[シンプロビジョニング](#), [シンプロビジョニングされた論理ボリュームの作成](#)

[シンプロビジョニングされたスナップショット](#), [シンプロビジョニングされたスナップショットボリュームの作成](#)

ストライプ化, [ストライプ化ボリュームの作成](#)

スナップショット, [スナップショットボリュームの作成](#)

パラメーターの変更, [論理ボリュームグループのパラメーター変更](#)

ミラー化, [ミラー化ボリュームの作成](#)

リニア, [リニア論理ボリュームの作成](#)

ローカルアクセス, [クラスター内の個別ノードでの論理ボリュームのアクティブ化](#)

作成, [リニア論理ボリュームの作成](#)

作成例, [3つのディスク上での LVM 論理ボリューム作成](#)

削除, [論理ボリュームの削除](#)

名前変更, [論理ボリュームの名前変更](#)

定義, [論理ボリューム](#), [LVM 論理ボリューム](#)

拡張, [論理ボリュームの拡張](#)

排他的アクセス, [クラスター内の個別ノードでの論理ボリュームのアクティブ化](#)

管理、一般, [論理ボリュームの管理](#)

縮小, [論理ボリュームの縮小](#), [論理ボリュームの縮小](#)

表示, [論理ボリュームの表示](#), [LVM 用のカスタム報告](#), [lvs コマンド](#)

過去, [過去の論理ボリュームの追跡および表示 \(Red Hat Enterprise Linux 7.3 以降\)](#)

論理ボリュームのアクティブ化

個別のノード, [クラスター内の個別ノードでの論理ボリュームのアクティブ化](#)

障害の発生したデバイス

表示, [障害の発生したデバイスの情報を表示](#)

## A

[archive ファイル](#), [ボリュームグループメタデータのバックアップ](#)

## B

[backup ファイル](#), [ボリュームグループメタデータのバックアップ](#)

## C



## CLVM

定義, [クラスター化論理ボリュームマネージャー \(CLVM\)](#)

clvmd デーモン, [クラスター化論理ボリュームマネージャー \(CLVM\)](#)

## L

### logical volume

activation, [論理ボリュームのアクティブ化の制御](#)

lvchange コマンド, [論理ボリュームグループのパラメーター変更](#)

lvconvert コマンド, [ミラー化ボリューム設定の変更](#)

lvcreate コマンド, [リニア論理ボリュームの作成](#)

lvdisplay コマンド, [論理ボリュームの表示](#)

lvextend コマンド, [論理ボリュームの拡張](#)

## LVM

アーキテクチャーの概要, [LVM アーキテクチャーの概要](#)

カスタムレポートのフォーマット, [LVM 用のカスタム報告](#)

クラスター化, [クラスター化論理ボリュームマネージャー \(CLVM\)](#)

コンポーネント, [LVM アーキテクチャーの概要](#), [LVM コンポーネント](#)

ディレクトリー構造, [ボリュームグループの作成](#)

ヘルプ, [CLI コマンドの使用](#)

ボリュームグループ、定義, [ボリュームグループ](#)

ラベル, [物理ボリューム](#)

ロギング, [ロギング](#)

物理ボリューム、定義, [物理ボリューム](#)

物理ボリュームの管理, [物理ボリュームの管理](#)

論理ボリュームの管理, [論理ボリュームの管理](#)

### LVM ボリュームの作成

概要, [論理ボリューム作成の概要](#)

lvmdiskscan コマンド, [ブロックデバイスのスキャン](#)

lvmetad デーモン, [メタデータデーモン \(lvmetad\)](#)

lvreduce コマンド, [論理ボリュームの縮小](#), [論理ボリュームの縮小](#)

lvremove コマンド, [論理ボリュームの削除](#)

lvrename コマンド, [論理ボリュームの名前変更](#)

lvs コマンド, [LVM 用のカスタム報告](#), [lvs コマンド](#)

引数の表示, [lvs コマンド](#)

lvscan コマンド, [論理ボリュームの表示](#)

## M

man ページの表示, [CLI コマンドの使用](#)

mirror\_image\_fault\_policy 設定パラメーター, [ミラー化論理ボリュームの障害ポリシー](#)

`mirror_log_fault_policy` 設定パラメーター, [ミラー化論理ボリュームの障害ポリシー](#)

## P

`pvdisplay` コマンド, [物理ボリュームの表示](#)

`pvmove` コマンド, [オンラインデータ移動](#)

`pvremove` コマンド, [物理ボリュームの削除](#)

`pvresize` コマンド, [物理ボリュームのサイズ変更](#)

`pvs` コマンド, [LVM 用のカスタム報告](#)

引数の表示, [pvs コマンド](#)

`pvscan` コマンド, [物理ボリュームの表示](#)

## R

[RAID 論理ボリューム](#), [RAID 論理ボリューム](#)

拡張, [RAID ボリュームの拡張](#)

`rules.d` ディレクトリー, [udev のデバイスマッパーとの統合](#)

## U

`udev` デバイスマネージャー, [デバイスマッパーの udev デバイスマネージャサポート](#)

`udev` ルール, [udev のデバイスマッパーとの統合](#)

## V

`vgcfgbackup` コマンド, [ボリュームグループメタデータのバックアップ](#)

`vgcfgrestore` コマンド, [ボリュームグループメタデータのバックアップ](#)

`vgchange` コマンド, [ボリュームグループのパラメーター変更](#)

`vgcreate` コマンド, [ボリュームグループの作成, クラスタ内でのボリュームグループ作成](#)

`vgdisplay` コマンド, [ボリュームグループの表示](#)

`vgexport` コマンド, [ボリュームグループの別のシステムへの移動](#)

`vgextend` コマンド, [ボリュームグループへの物理ボリュームの追加](#)

`vgimport` コマンド, [ボリュームグループの別のシステムへの移動](#)

`vgmerge` コマンド, [ボリュームグループの統合](#)

`vgmknodes` コマンド, [ボリュームグループディレクトリーの再作成](#)

`vgreduce` コマンド, [ボリュームグループからの物理ボリュームの削除](#)

`vgrename` コマンド, [ボリュームグループの名前変更](#)

`vgs` コマンド, [LVM 用のカスタム報告](#)

引数の表示, [vgs コマンド](#)

`vgscan` コマンド, [キャッシュファイル構築のためのボリュームグループのディスクスキャン](#)

`vgsplit` コマンド, [ボリュームグループの分割](#)