Red Hat Enterprise Linux 6

論理ボリュームマネージャーの管理

LVM 管理者ガイド

エディッション 1

Last Updated: 2017-10-12

Red Hat Enterprise Linux 6 論理ボリュームマネージャーの管理

LVM 管理者ガイド
エディッション 1

法律上の通知

概要

本ガイドは、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ています。

目次

はじめに

1. 本ガイドについて

本ガイドは、クラスター環境における LVM の実行に関する情報など、論理ボリュームマネージャー (LVM) について説明しています。

2. 対象読者

本ガイドは、Linux オペレーティングシステムを実行するシステムを管理するシステム管理者を対象としています。Red Hat Enterprise Linux 6 と GFS2 ファイルシステム管理に精通している必要があります。

3. ソフトウェアのバージョン

表1 ソフトウェアのバージョン

ソフトウェア	説明
Red Hat Enterprise Linux 6	Red Hat Enterprise Linux 6 以降を指します。
GFS2	Red Hat Enterprise Linux 6 以降の GFS2 を指しま。

4. 関連ドキュメント

Red Hat Enterprise Linux の使用についての詳細情報は、以下の資料を参照してください。

- 『インストールガイド』 — Red Hat Enterprise Linux 6 のインストールについての関連情報を記載しています。
- 『導入ガイド』 — Red Hat Enterprise Linux 6 の導入、設定、および管理についての関連情報を記載しています。
- 『ストレージ管理ガイド』 — Red Hat Enterprise Linux 6 上でのストレージデバイスおよびファイルシステムの効果的な管理方法について説明しています。

Red Hat Enterprise Linux 6 用の High-Availabilty アドオンおよび Resilient Storage アドオンに関する詳細情報は、以下の資料を参照してください。

- 『High Availability Add-On の概要』 — Red Hat High Availability アドオンの全体的な概要を説明しています。
- 『クラスターの管理』 — Red Hat High Availability アドオンのインストール、設定、および管理に関する情報を提供しています。

- 『Global File System 2: 設定と管理』 — Resilient Storage アドオンに含まれている Red Hat GFS2 (Red Hat Global File System 2) のインストール、設定、および維持に関する情報を提供しています。
- 『DM マルチパス』 — Red Hat Enterprise Linux 6 のデバイスマッパーマルチパスの機能の使用法に関する情報を提供しています。
- 『ロードバランサーの管理』 — 実サーバー群全体にわたって IP 負荷を分散するための Linux Virtual Servers (LVS) を提供する一式の統合ソフトウェアコンポーネントであるロードバランサーアドオンを利用したハイパフォーマンスのシステムとサービスの設定に関する情報を提供しています。
- 『リリースノート』 — Red Hat 製品の現在のリリースに関する情報を提供しています。

Red Hat Cluster Suite ドキュメントとその他の Red Hat ドキュメントは、HTML と PDF 形式で、https://access.redhat.com/site/documentation/ よりオンラインで入手できます。また、Red Hat Enterprise Linux ドキュメント CD は RPM 形式で取得可能です。

5. フィードバック

誤字/脱字を発見された場合、または本ガイドを改善するためのご意見/ご提案がございましたら、弊社にご連絡ください。その場合は、製品 **Red Hat Enterprise Linux 6** およびコンポーネント **doc-Logical_Volume_Manager** に対して、Bugzilla (http://bugzilla.redhat.com/) 内でご報告ください。バグ報告を提出される時には、以下のマニュアルの識別子を必ずご記入いただくようお願いします。

```
Logical_Volume_Manager_Administration(EN)-6 (2014-10-8-15:20)
```

本ガイドを改善するためのご意見/ご提案をお持ちの場合は、可能な限り具体的にご説明いただくようお願いします。また、エラーを発見された場合は、弊社で対象箇所を容易に見つけることができるように、そのセクション番号と周辺の文章も含めてご報告いただくようお願いします。

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 6 の初期リリースおよびそれ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。その後に、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大まかに説明します。

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 6 の初期リリース以降に組み込まれている LVM 論理ボリュームマネージャーの新機能および変更された機能について説明します。

1.1.1. Red Hat Enterprise Linux 6.0 の新機能および変更された機能

Red Hat Enterprise Linux 6.0 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.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して、デバイスに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に動作するかを定義することができます。このパラメーターが **remove** に設定されている場合、システムは障害の発生したデバイスを削除し、そのデバイスなしでの実行を試みます。このパラメーターが **allocate** に設定されている場合、システムは障害の発生したデバイスを削除し、そのデバイスの代わりとなる新たなデバイス上にスペースの割り当てを試みます。代わりとなる適切なデバイスとスペースを割り当てることができない場合、このポリシーは **remove** ポリシーと同様に機能します。LVM のミラー障害ポリシーに関する情報は、「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。

- Red Hat Enterprise Linux 6 リリースでは、Linux I/O スタックが強化され、ベンダーの提供する I/O 制限情報を処理できるようになりました。これによって、LVM を含むストレージ管理ツールでデータの配置とアクセスを最適化することができます。このサポートは、**lvm.conf** ファイル内の **data_alignment_detection** と **data_alignment_offset_detection** のデフォルト値を変更することによって無効にすることができます。ただし、このサポートの無効化は推奨されません。

  LVM のデータ配置に関する情報および **data_alignment_detection** と **data_alignment_offset_detection** のデフォルト値の変更に関する情報は、**/etc/lvm/lvm.conf** ファイルのインラインドキュメントを参照してください。これは、付録B *LVM 設定ファイル* にも記載されています。Red Hat Enterprise Linux 6 における I/O スタックおよび I/O 制限のサポートに関する一般的な情報は、『ストレージ管理ガイド』 を参照してください。

- Red Hat Enterprise Linux 6 では、デバイスマッパーが **udev** 統合に対する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 デバイスを含むすべてのデバイスマッパーデバイスに関連したすべての **udev** 処理と同期化されます。**udev** デバイスマネージャに対するデバイスマッパーのサポートに関する情報は、「デバイスマッパーによる udev デバイスマネージャーのサポート」 を参照してください。

- Red Hat Enterprise Linux 6 リリースでは、ディスク障害発生後に、**lvconvert --repair** コマンドを使用してミラーを修復することができます。これによって、ミラーが一貫性のある状態に戻ります。**lvconvert --repair** コマンドに関する情報は、「ミラー化論理ボリュームの修復」 を参照してください。

- Red Hat Enterprise Linux 6 リリースでは、**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。スナップショットのマージに関する情報は、「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。

- Red Hat Enterprise Linux 6 リリースでは、**lvconvert** コマンドの **--splitmirrors** 引数を使用して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。このオプションの使用に関する情報は、「ミラー化論理ボリュームの冗長イメージの分割」 を参照してください。

- ミラー化論理デバイスを作成する際に、**lvcreate** コマンドの **--mirrorlog mirrored** 引数を使用して、それ自体がミラー化されるミラー化論理デバイスのミラーログを作成できるようになりました。このオプションの使用に関する情報は、「ミラー化ボリュームの作成」 を参照してください。

1.1.2. Red Hat Enterprise Linux 6.1 の新機能および変更された機能

Red Hat Enterprise Linux 6.1 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。

- Red Hat Enterprise Linux 6.1 リリースは、ミラー化論理ボリュームのスナップショット論理ボリューム作成をサポートしています。リニアまたはストライプ化された論理ボリュームの作成と同様に、ミラー化ボリュームのスナップショットを作成することができます。スナップショットボリュームの作成方法については、 「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。

- 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スペースがなく、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。

  lvextend コマンドで **--alloc cling** オプションを使用した LVM ミラー化ボリュームの拡張については、「**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。

- **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るようになりました。オブジェクトタグの追加と削除については、「オブジェクトタグの追加と削除」 を参照してください。

- LVM オブジェクトタグで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。LVM オブジェクトタグについては、付録C *LVM オブジェクトタグ* を参照してください。

- 単一の論理ボリューム内で RAID0 (ストライピング) と RAID1 (ミラーリング) の併用ができるようになりました。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。ミラー化論理ボリュームの作成については、「ミラー化ボリュームの作成」 を参照してください。

- Red Hat Enterprise Linux 6.1 リリースでは、クラスター化された論理ボリューム上で一貫性のあるデータバックアップを作成する必要がある場合、ボリュームを排他的にアクティブ化してから、スナップショットを作成することができます。ノード上における論理ボリュームを排他的にアクティブ化する方法についての情報は、「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。

1.1.3. Red Hat Enterprise Linux 6.2 の新機能および変更された機能

Red Hat Enterprise Linux 6.2 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。

- Red Hat Enterprise Linux 6.2 リリースは、**lvm.conf** 設定ファイルの **issue_discards** パラメーターをサポートしています。このパラメーターを設定すると、論理ボリュームが物理ボリューム上の領域を使用しなくなった場合、LVM はその物理ボリュームの配下の物理ボリュームに対して破棄を実行します。このパラメーターの詳細は、**/etc/lvm/lvm.conf** ファイルのインラインドキュメントを参照してください。これは、付録B *LVM 設定ファイル* にも記載されています。

1.1.4. Red Hat Enterprise Linux 6.3 の新機能および変更された機能

Red Hat Enterprise Linux 6.3 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。

- Red Hat Enterprise Linux 6.3 リリースは、LVM は RAID4/5/6 およびミラーリングの新実装をサポートしています。RAID 論理ボリュームの詳細は、「RAID 論理ボリューム」 を参照してください。

- 回復する必要のないミラーを新規に作成する場合は、最初のデバイスからの初期同期が不要であることを示す **--nosync** 引数を指定できます。ミラー化ボリュームの作成方法については、「ミラー化ボリュームの作成」 を参照してください。

- 本ガイドには、スナップショット機能の **autoextend** についての説明が加わりました。スナップショットボリュームの作成方法については、「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。

1.1.5. Red Hat Enterprise Linux 6.4 の新機能および変更された機能

Red Hat Enterprise Linux 6.4 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。

- 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、アプリケーションにより必要な場合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、シンプールにバインド可能なデバイスを後の割り当て用に作成できます。シンプールは、コスト効率が高いストレージ領域の割り当てに必要な場合に動的に拡張できます。

  シンプロビジョニングされた論理ボリュームの全般情報については、「シンプロビジョニングされた論理ボリューム (シンボリューム)」 を参照してください。シンボリュームの作成についての詳細は、「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。

- Red Hat Enterprise Linux 6.4 バージョンの LVM 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により、多くの仮想デバイスを同じデータボリューム上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でのデータシェアが可能になります。

  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についての全般情報は、「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。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。

- 本書には、LVM の割り当てポリシーについて説明した 「LVM の割り当て」 のセクションが新たに追加されました。

- LVM は **raid10** 論理ボリュームのサポートを提供するようになりました。RAID 論理ボリュームの詳細は、「RAID 論理ボリューム」 を参照してください。

- Red Hat Enterprise Linux 6.4 リリースは、LVM メタデータデーモンである **lvmetad** をサポートしています。このデーモンを有効にすることで、多くのブロックデバイスを持つシステムのスキャンの量を減らすことができます。**lvmetad** デーモンは、クラスターのノード間では現在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である必要があります。

  メタデータデーモンの詳細は、「メタデータデーモン (lvmetad)」 を参照してください。

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が行われています。

1.1.6. Red Hat Enterprise Linux 6.5 の新機能および変更された機能

Red Hat Enterprise Linux 6.5 では、ドキュメントと機能が以下のように更新され、変更されています。

- **lvchange** コマンドの **--writemostly** および **--writebehind** パラメーターを使って RAID1 論理ボリューム上の I/O 操作を制御できます。これらのパラメーターの情報は、「RAID1 論理ボリュームでの I/O 操作の制御」 を参照してください。

- **lvchange** コマンドは、デバイスを再アクティブ化せずに一時的に失敗したデバイスを復元できる **--refresh** パラメーターに対応するようになりました。この機能は 「「allocate」RAID 障害ポリシー」 で説明されています。

- 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。この機能の情報は、「RAID 論理ボリュームのスクラビング」 を参照してください。

- **lvs** コマンドが対応するフィールドは更新されています。**lvs** コマンドについての情報は、表 4.4「lvs 表示フィールド」 を参照してください。

- **lvchange** コマンドは、新規の **--maxrecoveryrate** および **--minrecoveryrate** パラメーターに対応します。これらのパラメーターにより、**sync** 操作が実行される速度を制御できます。これらのパラメーターについての情報は、「RAID 論理ボリュームのスクラビング」 を参照してください。

- 復旧スロットルを実装することにより、RAID 論理ボリュームが初期化される速度を制御することができます。**sync** 操作が実施される速度は、「RAID 論理ボリュームの作成」 に説明されているように **lvcreate** コマンドの **--minrecoveryrate** および **--maxrecoveryrate** オプションを使っ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って実行できます。

- 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できるようになりました。外部ボリュームとして知られるこれらのボリュームの作成方法についての情報は、「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。

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が行われています。

1.1.7. Red Hat Enterprise Linux 6.6 の新機能および変更された機能

Red Hat Enterprise Linux 6.6 には、以下のドキュメントと機能の更新および変更内容が含まれます。

- 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ングされたスナップショットについての文書が明確化されました。LVM シンプロビジョニングについての追加情報は、**lvmthin**(7) man ページに記載されています。シンプロビジョングされた論理ボリュームについての情報は、「シンプロビジョニングされた論理ボリューム (シンボリューム)」 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ついての情報は、「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。

- 本ガイドでは、**lvm dumpconfig** コマンドについて 「**lvm dumpconfig** コマンド」 で説明しています。
- 本ガイドでは、LVM プロファイルについて 「LVM プロファイル」 で説明しています。
- 本ガイドでは、**lvm** コマンドについて 「**lvm** コマンドの使用による LVM 情報の表示」 で説明しています。
- Red Hat Enterprise Linux 6.6 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** コマンドの -k と -K オプションを使ってシンプールスナップショットのアクティブ化を制御できます。
- 本マニュアルは、**vgimport** コマンドの **--force** 引数について説明します。このコマンドを使うと、物理ボリュームの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。

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が行われています。

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。これは、物理ストレージディレクトリーを使用するよりもはるかに高い柔軟性を数々の方法で提供します。論理ボリュームが作成されると、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で、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せずに、サイズ変更や移動が可能になります。これにより運用コストを削減することができます。

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している場合、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またがって拡張することが可能です。

- 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、再パーティションせずに、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したり縮小したりすることができます。

- オンラインでのデータの再配置

  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することができます。データはディスクが使用中の場合でもディスク上で再配置できます。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。

- 便宜上のデバイスの命名

  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、便宜上命名することができます。

- 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することができます。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上で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、一貫性のあるバックアップのために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効果をテストすることができます。

LVM でのこれらの機能の実装については、本ガイドの後半で説明しています。

1.3. LVM アーキテクチャーの概要

Linux オペレーティングシステムの Red Hat Enterprise Linux 4 リリースでは、元の LVM1 論理ボリュームマネージャーが LVM2 に置き換えられました。LVM2 は、LVM1 よりも汎用性の高いカーネルフレームワークを採用しています。また、LVM2 では、LVM1 と比べて、以下のような点が改善されています。

- 容量の柔軟性
- より効率的なメタデータストレージ
- より優れたリカバリー形式
- 新たな ASCII メタデータ形式
- メタデータのアトミックな変更
- メタデータの冗長コピー

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 論理ボリュームの配下の物理ストレージユニットは、パーティションまたは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、物理ボリュームは **ボリュームグループ** (Volume Group: VG) に統合されます。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) が割り当てられます。このプロセスは、ディスクがパーティションに分割され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、簡易 LVM 論理ボリュームのコンポーネントを示しています。

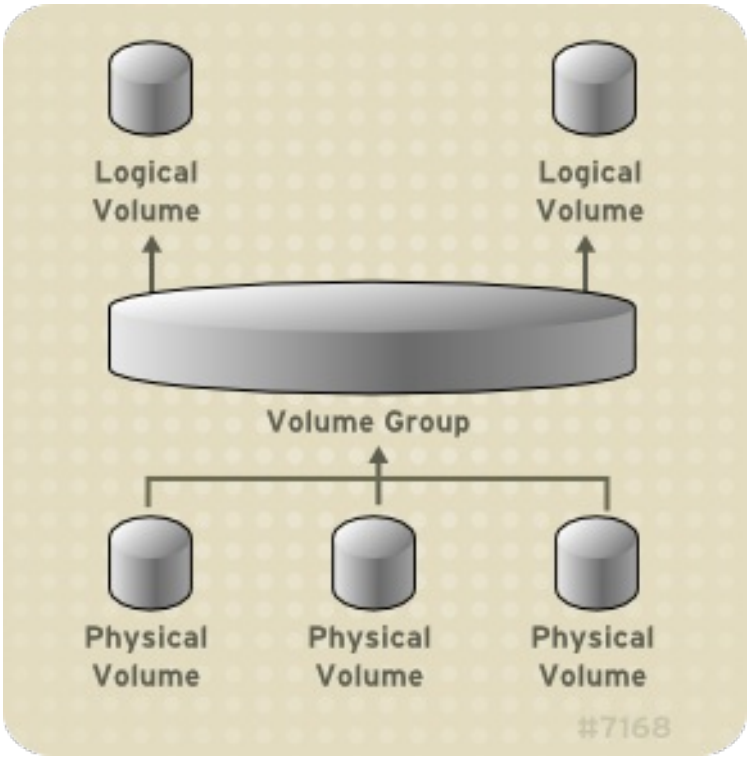


図1.1 LVM 論理ボリュームのコンポーネント

LVM 論理ボリュームのコンポーネントに関する詳細情報は、*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。これらの拡張機能により、コンピューターのクラスターは、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 アドオンの一部です。

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。

- ご使用のシステムの 1 つのノードのみが、論理ボリュームとして設定する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 が使用可能であり、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。
- 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ており、ストレージにアクセスする単一のノードのみが常にアクティブである場合は、High Availability Logical Volume Management (HA-LVM) エージェントの使用をお勧めします。
- ご使用のクラスターの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 の使用により、ユーザーは論理ボリュームの設定中に物理ストレージへのアクセスをロックすることによって、共有ストレージ上の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。CLVM 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 を使用するには、**clvmd** デーモンを含む High Availability アドオンおよび Resilient Storage アドオン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 の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内で稼働し、クラスター内で LVM メタデータ更新を配布して、各クラスターコンピューターに論理ボリュームの同一ビューを提供します。High Availability アドオンのインストールと管理の詳細は、『クラスターの管理』 を参照してください。

clmvd が起動時に確実に開始するようにするためには、**chkconfig ... on** コマンドを**clvmd** サービスに対して以下のように実行します。

```
# chkconfig clvmd on
```

clvmd デーモンが開始していない場合は、以下のように **clvmd** サービスを **service ... start** コマンドで実行します。

```
# service clvmd start
```

クラスター環境で LVM 論理ボリュームを作成することは、単一ノード上に LVM 論理ボリュームを作成することと同じです。*4章CLI コマンドでの LVM 管理* と *7章LVM GUI での LVM 管理* で説明されているように、LVM コマンド自体や LVM グラフィカルユーザーインターフェースに相違はありません。クラスター内に作成する LVM ボリュームを有効にするためには、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に達している必要があります。

デフォルトでは、共有ストレージ上に CLVM で作成され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対して可視となっています。ただし、全ストレージデバイスがクラスター内の 1 つのノードのみに可視となる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。

図1.2「CLVM の概観」 は、クラスター内の CLVM の概観を示しています。

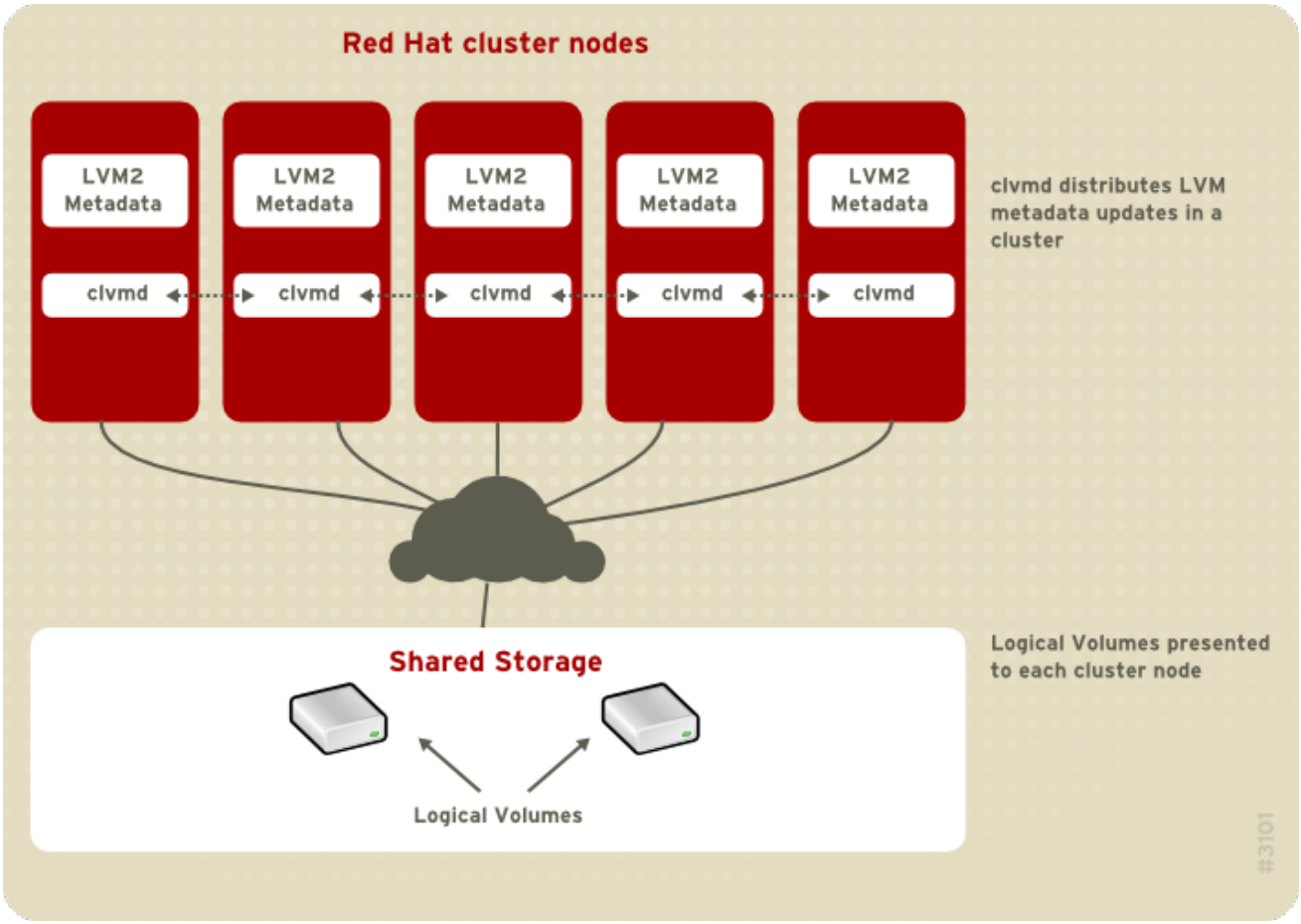


図1.2 CLVM の概観

注記

CLVM で、クラスター全体をロッキングするには、**lvm.conf** ファイルを変更する必要があります。クラスターロッキングをサポートするための **lvm.conf** ファイルの設定に関する情報は、**lvm.conf** ファイル自体に記載されています。**lvm.conf** ファイルについての情報は 付録B *LVM 設定ファイル* を参照してください。

1.5. ドキュメントの概要

本書の後半には、以下のような章が含まれています。

- 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ています。
- 3章*LVM 管理の概要* では、LVM コマンドラインインターフェース (CLI) コマンドを使用するか、LVM グラフィカルユーザーインターフェース (GUI) を使用するかにかかわらず、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ています。
- 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ています。

- *7章LVM GUI での LVM 管理* では、LVM GUI の操作についてまとめています。
- 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ています。
- 付録C *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ています。
- 付録D *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メタデータについて説明しており、これには LVM ボリュームグループ用のメタデータのサンプルコピーも含まれます。

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 論理ボリュームの配下の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、デバイスは物理ボリューム (PV) として初期化されなければなりません。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。

デフォルトでは、LVM ラベルは 2 番目の 512 バイトセクターに配置されます。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、このデフォルトを書き換えることができます。これにより、LVM ボリュームは、必要であれば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。

システムの起動時にデバイスは任意の順序で立ち上がることがあります。そのため、LVM ラベルでは、物理デバイスの正確な ID とデバイスの順序を提供します。LVM ラベルは再起動後にもクラスター全域で永続化します。

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上で保存される位置も記録します。

LVM メタデータには、システム上の LVM ボリュームグループの設定詳細が含まれます。デフォルトでは、メタデータの同一コピーが、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で維持されています。LVM メタデータは小規模であり、ASCII 形式で保存されます。

現在 LVM により、各物理ボリューム上でメタデータの 1 つまたは 2 つの同一コピーを保存することができます。デフォルトでは、コピーは 1 つです。物理ボリューム上のメタデータのコピー数をいったん設定すると、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の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、それはデバイスの最終位置に配置されます。意図したものとは異なるディスクに書き込みをして、ディスクの先頭位置に誤って上書きしてしまった場合でも、デバイス後部にあるメタデータの 2 つ目のコピーでメタデータの復元が可能となります。

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D *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 ラベルは 2 番目のセクターにあり、その後にメタデータ領域とデバイスの使用可能なスペースが順に続きます。

注記

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

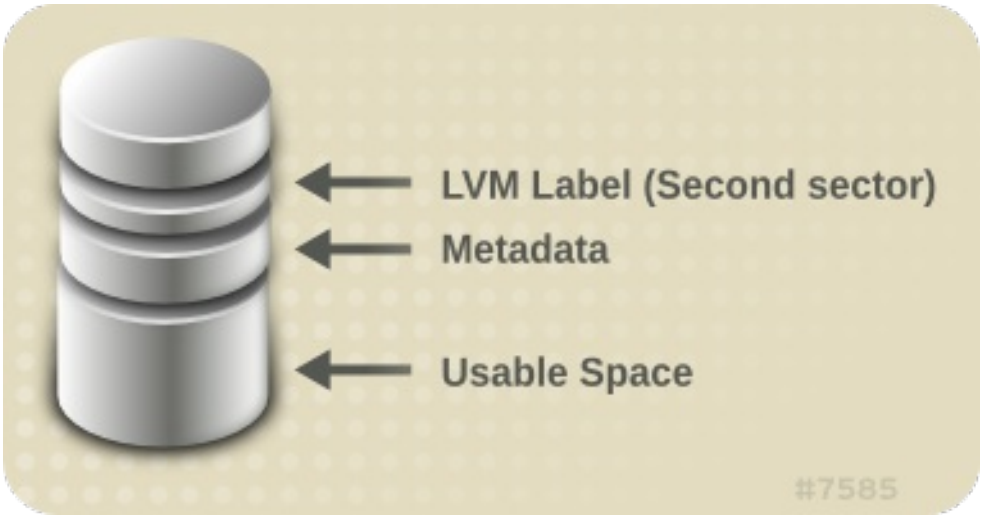


図2.1 物理ボリュームレイアウト

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することができます。それを以下のような理由により、ディスク全体をカバーする 1 つのパーティションを作成して、単一の LVM 物理ボリュームとしてラベルを付けることが一般的に推奨されます。

- 管理上の利便性

  それぞれの実ディスクが 1 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。これはディスクに障害が発生した場合に、特に役に立ちます。更には、単一のディスク上の複数物理ボリュームは、起動時にカーネルによって不明なパーティションとして警告を受ける原因とな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一の物理ディスクにあることは認識しません。2 つの物理ボリュームが同一の物理ディスク上にある場合にストライプ化された論理ボリュームを作成すると、ストライプ化されたボリュームは同じディスク上の異なるパーティション上にある可能性があります。これはパフォーマンスの向上ではなく、低下をもたらします。

推奨されることではありませんが、1 つのディスクを別々の LVM 物理ボリュームに分割しなければならない状況が考えられます。たとえば、少数のディスクしかないシステム上では、既存システムを LVM ボリュームに移行する場合にデータをパーティション間で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、そのディスクで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、それらのパーティションがいずれも同じボリュームグループ内に存在する場合に、ストライプ化ボリュームを作成する際は論理ボリュームに含め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。

論理ボリュームは物理エスクテントと同じサイズの論理エクステント割り当てることができます。そのため、エクステントは、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サイズになります。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複数のボリュームに分割されます。LVM 論理ボリュームには 3 つのタイプがあります。**リニア (linear)** ボリューム、**ストライプ化 (striped)** ボリューム、および **ミラー化 (mirrored)** ボリュームです。これらについては、以下のセクションで説明します。

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、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、物理エクステントの範囲を論理ボリュームの領域に順番に割り当てることになります。たとえば、図2.2「エクステントのマッピング」 に示されているように、1 から 99 までの論理エクステントを 1 つの物理ボリュームにマッピングして、100 から 198 までの論理エクステントを 2 番目の物理ボリュームにマッピングすることができます。アプリケーションの観点からは、デバイスには 198 のエクステントのサイズのデバイスが 1 つあることになります。

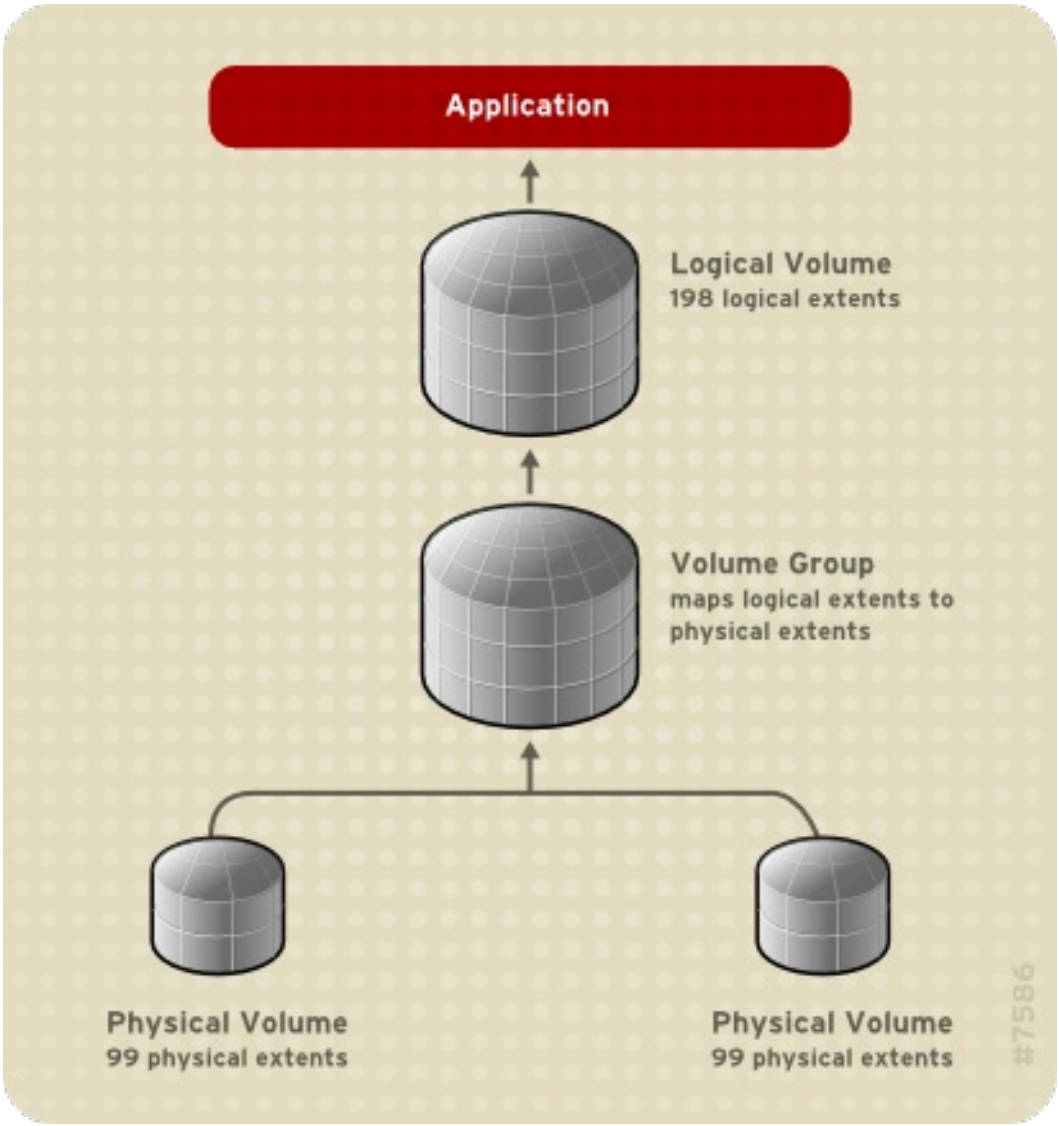


図2.2 エクステントのマッピング

論理ボリュームを構成している物理ボリュームは同じサイズ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含まれます。エクステントサイズが 4MB であることから、物理ボリュームは 4MB 単位に分割されます。この例では、**PV1** は 200 エクステントのサイズ (800MB) です。さらに **PV2** は、100 エクステントの サイズ (400MB) です。1 から 300 エクステント (4MB から 1200MB) までの任意のサイズのリニアボリュームを作成することができます。この例では、**LV1** というリニアボリュームのサイズは 300 エクステントです。

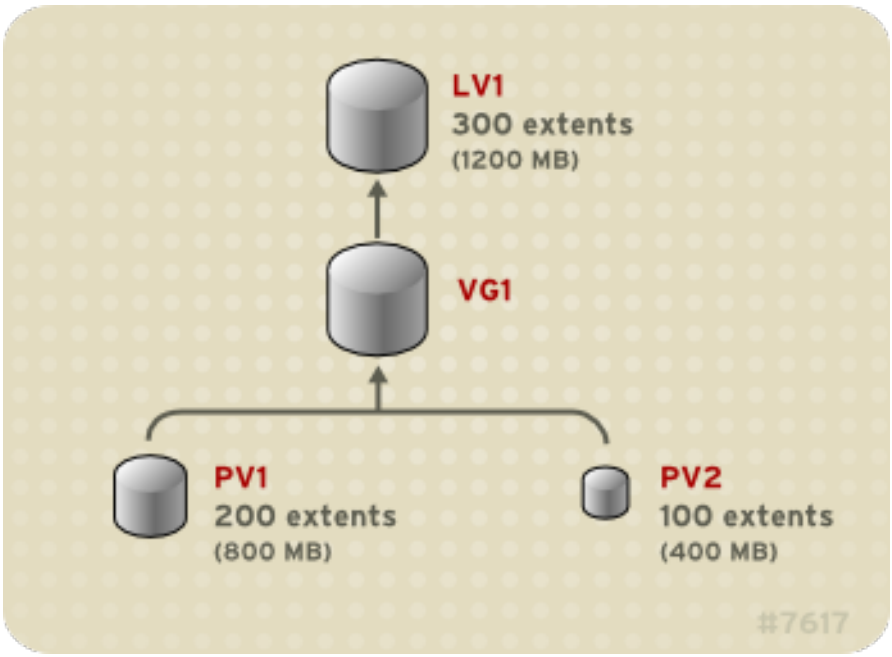


図2.3 サイズの異なる物理ボリュームを用いたリニアボリューム

物理エクステントのプールから必要なサイズの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、この場合は、そのボリュームグループから 2 つの論理ボリュームが構築されています。**LV1** は 250 エクステントでサイズは 1000MB です。**LV2** は 50 エクステントでサイズは 200MB です。

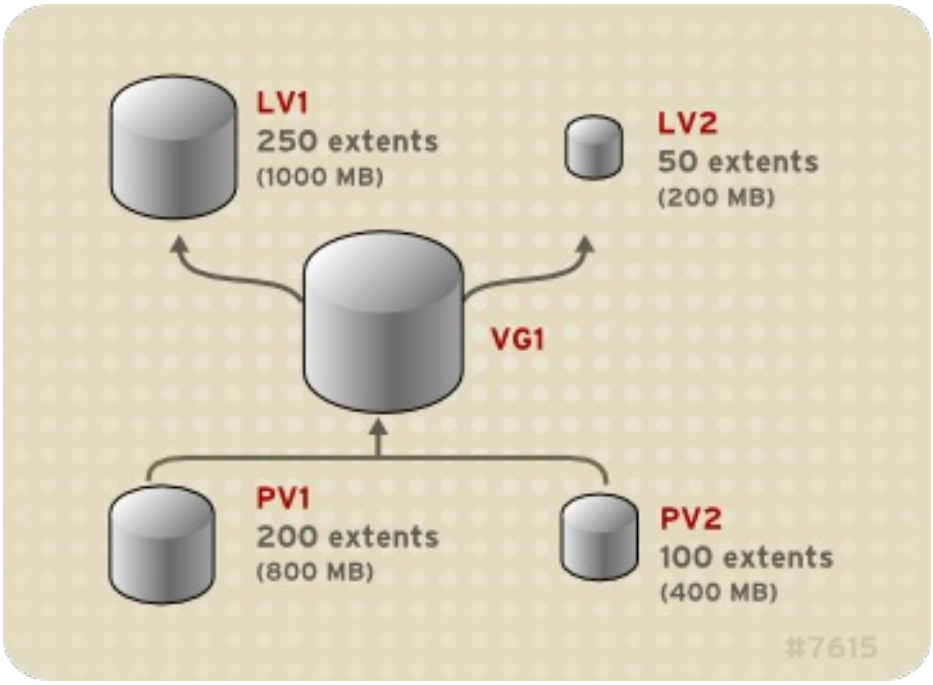


図2.4 複数の論理ボリューム

2.3.2. ストライプ化論理ボリューム

LVM 論理ボリューム上にデータを書き込む場合、ファイルシステムは、配下の物理ボリューム全体にデータを分配します。その場合、ストライプ化論理ボリュームを作成することにより、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。大量の順次読み取りと書き込みの場合は、この方法でデータ I/O の効率を改善することができます。

ストライピングは、所定数の物理ボリュームにデータをラウンドロビン式に書き込んでいくことにより、パフォーマンスを向上させます。ストライピングでは、I/O は並行して実行されます。場合によっては、ストライプ内に追加する各物理ボリュームに対して、ほぼ直線的なパフォーマンス向上をもたらします。

以下の図では、3 つの物理ボリューム全体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 PV1 に書き込まれます。
- データの 2 番目のストライプは PV2 に書き込まれます。
- データの 3 番目のストライプは PV３ に書き込まれます。
- データの 4 番目のストライプは PV1 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を超えることはできません。

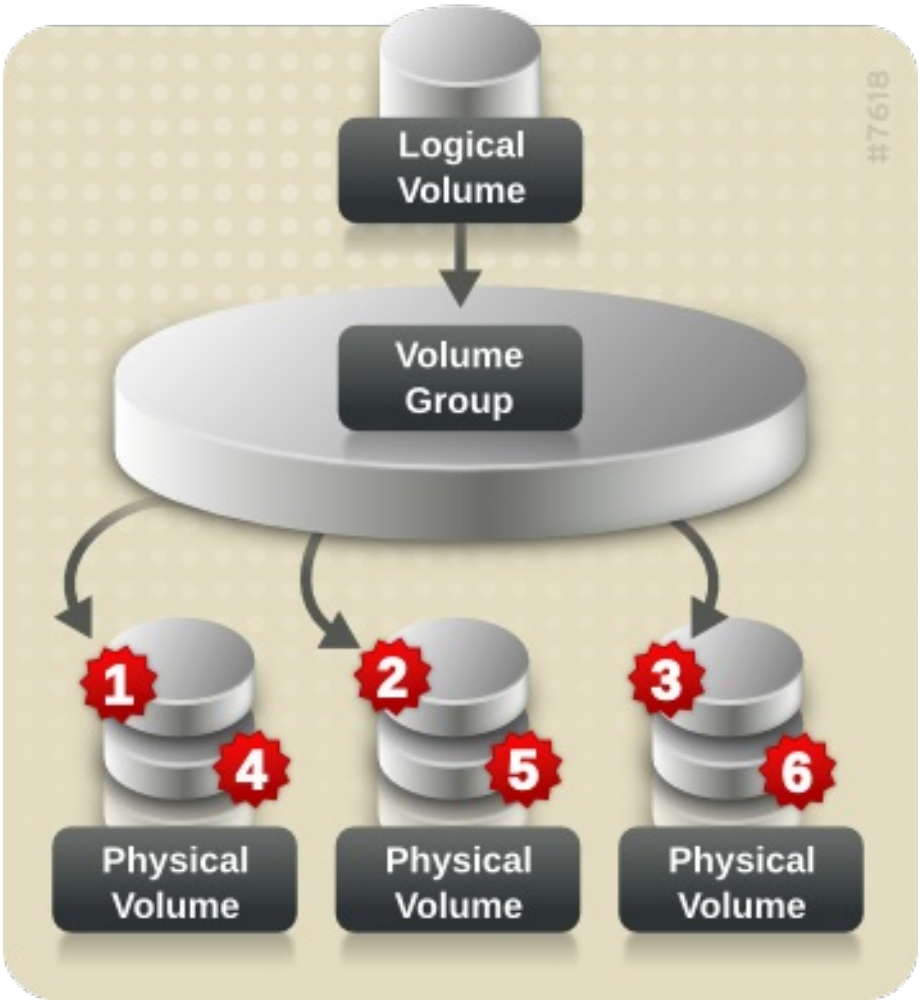


図2.5 3 つの PV にまたがるデータのストライピング

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ることにより拡張することができます。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ためにボリュームグループを構成する配下の物理ボリュームに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している 2 層型のストライプがある場合、そのボリュームグループにもう 1 つの物理ボリュームを追加するだけでは、ストライプを拡張することにはなりません。代わりに、2 つ以上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。ストライプ化ボリュームの拡張に関する詳細は、「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. ミラー化論理ボリューム

ミラーはデータの同一コピーを異なるデバイスに維持します。データが 1 つのデバイスに書き込まれると、それは 2 つ目のデバイスにも書き込まれ、データのミラー化が行われます。この重複保存は、デバイス障害に対する保護になります。ミラーレッグの 1 つに障害が発生した場合、論理ボリュームはリニアボリュームとなりますが、依然としてアクセスすることができます。

LVM はミラー化ボリュームに対応しています。ミラー化論理ボリュームを作成する場合、LVM は、配下の物理ボリュームに書き込まれるデータが、別の物理ボリュームに確実にミラー化されるようにします。LVM では、複数のミラーを持つミラー化論理ボリュームを作成することができます。

LVM ミラーは、複製されるデバイスを標準的な 512KB サイズのリージョンに分割します。LVM はミラーと同期してい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。このログは、再起動後も永続化するようにディスク上に保持したり、メモリー内で維持したりすることができます。

図2.6「ミラー化論理ボリューム」 は、単一のミラーを用いたミラー化論理ボリュームを示しています。この設定では、ログはディスク上で維持されます。

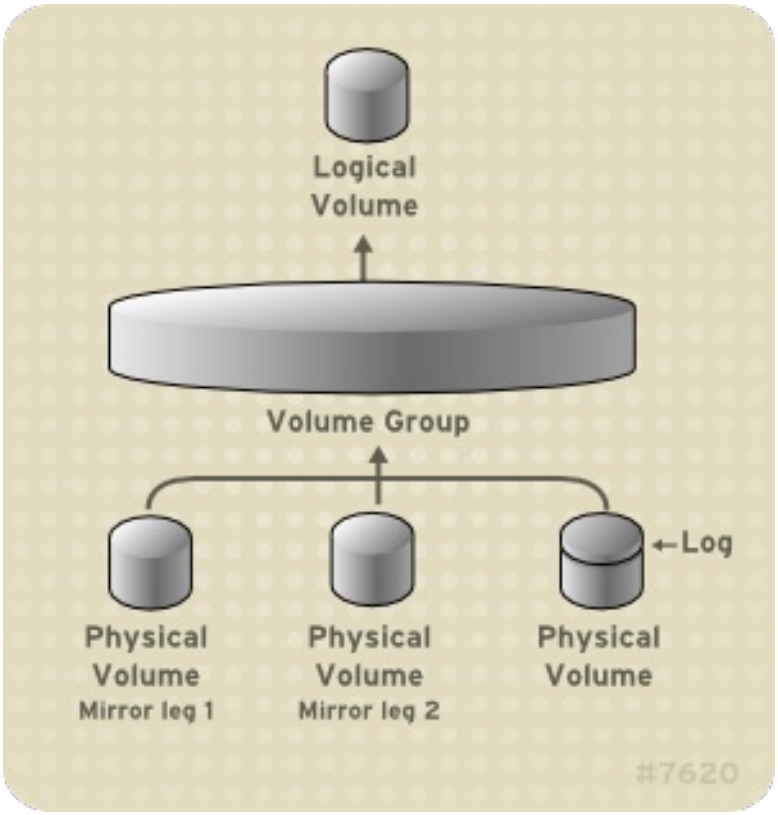


図2.6 ミラー化論理ボリューム

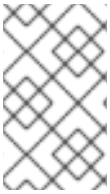
ミラーの作成と修正に関する情報は、 「ミラー化ボリュームの作成」 を参照してください。

2.3.4. RAID 論理ボリューム

Red Hat Enterprise Linux 6.3 リリースの時点では、LVM は RAID 論理ボリュームに対応しています。LVM が対応する RAID 実装についての情報は、「RAID 論理ボリューム」 を参照してください。

2.3.5. 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)

Red Hat Enterprise Linux 6.4 リリースでは、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、アプリケーションが必要とする場合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論理ボリュームに実際に書き込みを行う際の、後の割り当てが可能になるようにシンプールにバインドできる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の割り当てに必要な場合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 人のユーザーが各アプリケーション用にそれぞれ 100GB ファイルシステムを要求した場合、ストレージ管理者は各ユーザーに 100GB ファイルシステムと思われるもの (しかし実際は 100GB より少ないストレージによ

り利用可能で必要な場合にのみ使用される) を作成できます。シンプロビジョニングの使用時には、ストレージ管理者は、ストレージプールをモニターして、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。

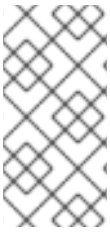
すべての利用可能な領域が使用できるようにするために、LVM ではデータの破棄に対応します。これにより、破棄されたファイルやその他のブロックの範囲で以前に使用された領域を再利用できます。

シンボリュームの作成についての詳細は、「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。

シンボリュームは、新実装のコピーオンライト (COW) スナップショット論理ボリュームのサポートを提供します。これにより、多くの仮想デバイスがシンプール内の同一データを共有することができます。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。

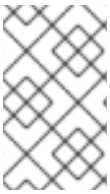
2.3.6. スナップショットボリューム

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することができます。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た場合、このスナップショット機能により、データが変更される前に変更部分のコピーを作成してデバイスの状態を再構築できます。



注記

Red Hat Enterprise Linux 6.4 リリースでは、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

LVM スナップショットは、クラスター内のノード全体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た後に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、複製元がまれにしか更新されない場合、その容量の 3-5 % だけで十分にスナップショットを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、ファイルシステム用の実際のメディアバックアップではありません。スナップショットはバックアップ手順の代替となるものを提供する訳ではありません。

スナップショットのサイズによって、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の量が左右されます。たとえば、スナップショットを作成後に複製元を完全に上書きした場合に、その変更を保管するには、スナップショットが最低でも複製元のボリュームと同じサイズである必要があります。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** などの、ほとんど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ほどにはスペースを必要としません。

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。これは、複製元のボリューム上の変更を追跡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、スナップショットは完全にサイズ変更可能のため、ストレージに余裕があれば、スナップショットのボリュームサイズを拡大してそれが停止することを防止できます。逆に、スナップショットボリュームのサイズが必要以上に大きい場合は、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用に必要となる領域を開放することができます。

スナップショット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が変更された場合、そのチャンクはマーク付けされ、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動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上で **fsck** コマンドを実行すると、ファイルシステムの整合性をチェックして、元のファイルシステムが修復を必要とするかどうかを判断することができます。
- スナップショットは読み取り/書き込み用のため、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することができます。これら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。

Red Hat Enterprise Linux 6 リリースでは、**lvconvert** 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あり、データやファイルを紛失した場合やその他の場合にシステムを以前の状態に復元する必要がある場合に実行されます。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についての情報は、「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。

2.3.7. 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

Red Hat Enterprise Linux 6.4 バージョンの LVM 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により、多くの仮想デバイスを同じデータボリューム上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でのデータ共有が可能になります。

すべてのシンボリュームだけでなくすべての LVM スナップショットボリュームの場合、シン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は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のみ排他的にアクティブ化する必要があります。

シン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。

- 同じ複製元ボリュームからのスナップショットが複数ある場合、シンスナップショットボリュームは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、大幅な速度の低下は発生しないはずです。

- シン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複製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は、シン論理ボリュームを作成することもできます。これにより、COW 操作が必要になるまでか、また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、その複製元によりアクティブ化する必要はありません。そのため、複製元のアクティブでないスナップショットボリュームが多くある間は、ユーザーは複製元のみをアクティブにしておくことができます。

- 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場合、複製元のボリュームの各スナップショットは、独立したシンプロビジョニングされたボリュームになります。これは、ある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て、その独立したボリュームを新しいスナップショットの複製元のボリュームとして使用して新しい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を選択できることを意味しています。

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ご使用のニーズに合うケースもあります。

- シンプールのチャンクサイズは変更できません。シンプー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、古いスナップショット機能の使用を選択することができます。

- 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ール内の全領域を使用するため、ご使用のニーズに合わ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。

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ため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の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 設定手順における具体的なステップごとの例については、*5章LVM 設定の例* を参照してください。

LVM 管理を実行するために使用できる CLI コマンドの詳細は、*4章CLI コマンドでの LVM 管理* を参照してください。別の方法として、*7章LVM GUI での LVM 管理* で説明されている LVM GUI を使用することもできます。

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、クラスター論理ボリュームマネージャー (CLVM) 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。CLVM を使用するには、「クラスター論理ボリュームマネージャー (CLVM)」 で説明されているように **clmvd** デーモンを含む High Availability アドオンおよび Resilient Storage アオドンソフトウェアを起動時に開始する必要があります。

クラスター環境で LVM 論理ボリュームを作成することは、単一ノード上に LVM 論理ボリュームを作成することと同じです。LVM コマンド自体や LVM GUI インターフェースに相違はありません。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あり、かつクラスターが定足数に達している必要があります。

CLVM では、クラスター全体をロッキングするには **lvm.conf** ファイルを変更する必要があります。クラスターロッキングをサポートするために **lvm.conf** ファイルを設定する方法に関する情報は、**lvm.conf** ファイル自体に記載されています。**lvm.conf** ファイルについての情報は 付録B *LVM 設定ファイル* を参照してください。

デフォルトでは、共有ストレージ上に CLVM で作成され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がクラスター内の 1 つのノードのみに表示されるようにボリュームグループを作成することもできます。また、ボリュームグループの状態をローカルボリュームグループからクラスターボリュームグループに変更することもできます。詳細、「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アクセスできることを確認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。

High Availability アドオンのインストールとクラスターインフラストラクチャーのセットアップ方法についての情報は、『クラスターの管理』 を参照してください。

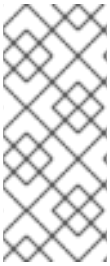
クラスター内でミラー化された論理ボリュームを作成する例は、「クラスター内でのミラー化 LVM 論理ボリュームの作成」 を参照してください。

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ために実行する必要のあるステップの概要です。

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後は、ファイルシステムを作成してマウントできます。本書の例では、GFS2 ファイルシステムを使用しています。



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の一部として実装することが可能ですが、Red Hat Enterprise Linux 6 リリースでは、Red Hat は単一ノードのファイルシステムとしての GFS2 の使用をサポートしていません。クラスターファイルシステムのスナップショット (例: バックアップ用) をマウントするための単一ノードの GFS2 ファイルシステムは引き続きサポートします。

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。クラスターシステムでは、そのクラスター内のすべてのノード上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。システム内の各ノード用に **`fstab`** ファイルに一行追加することもできます。

別の方法として、LVM GUI を使用して GFS2 ファイルシステムを作成し、マウントすることもできます。

LVM セットアップ情報の保存エリアは物理ボリューム上にあり、ボリュームが作成されたマシンにはないため、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、それを物理ボリューム上にあるものから再作成できます。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することができます。

3.3. 論理ボリューム上におけるファイルシステムの拡張

論理ボリューム上で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. 新規の物理ボリュームを作成します。
2. 新規の物理ボリュームを組み込むために拡張しようとしているファイルシステム付きの論理ボリュームを含むボリュームグループを拡張します。
3. 論理ボリュームを拡張して新規の物理ボリュームを組み込みます。
4. ファイルシステムを拡張します。

ボリュームグループ内に未割り当ての領域が十分にある場合は、手順 1 と 2 を実行する代わりに、その領域を使用して論理ボリュームを拡張することができます。

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** ファイル内で無効になっていない限り、すべてのボリュームグループで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アーカイブの保持期間と保持されるアーカイブファイルの数は、**lvm.conf** ファイルに設定できるパラメーターによって決定されます。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る必要があります。

メタデータバックアップでは、論理ボリュームに含まれている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、**/etc/lvm/backup** ファイルにメタデータを手動で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のメタデータのバックアップ」 で説明しています。

3.5. ロギング

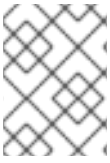
すべてのメッセージ出力は、以下についてのロギングレベルを独立して選択できるロギングモジュールを通過します。

- 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る時点で **udev** によってそれらを自動的にアクティブ化できるようにすることです。



注記

lvmetad デーモンは、クラスターのノード間では現在サポートされておらず、ロッキングタイプはローカルのファイルベースである必要があります。

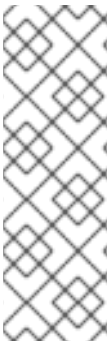
このデーモンの利点を活用するには、以下を実行してください。

- **lvm2-lvmetad** サービスを使ってデーモンを起動します。デーモンを起動時に自動的に起動するには、**chkconfig lvm2-lvmetad on** コマンドを使用します。デーモンを手動で起動するには、**service lvm2-lvmetad start** コマンドを使用します。

- デーモンを使用するように LVM を設定するには、**lvm.conf** 設定ファイル内で **global/use_lvmetad** 変数を 1 に設定します。**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。

通常、各 LVM コマンドはディスクスキャンを実行して、すべての関連する物理ボリュームを検索し、ボリュームグループのメタデータを読み取ります。ただし、メタデータデーモンが実行中で有効な場合は、この負荷がかかるスキャンは省略できます。代わりに、**lvmetad** デーモンは各デバイスが利用可能になる時点で 1 度のみ **udev** ルールによりデバイスをスキャンします。これにより I/O の量を大幅に削減でき、とくに多くのディスクがあるシステム上で 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。**udev** デバイスマネージャーと **udev** ルールについての情報は、「デバイスマッパーによる udev デバイスマネージャーのサポート」 を参照してください。

新規のボリュームグループがランタイム時に利用可能な場合 (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前にアクティブ化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化する必要のあるボリュームグループおよび/または論理ボリュームの一覧を設定できます。**lvmetad** デーモンが有効でない場合は、手動でアクティブ化する必要があります。デフォルトでは、この一覧は定義されません。つまり、すべてのボリュームはすべての物理ボリュームが配置されると自動的にアクティブ化されます。自動アクティブ化は、イベントベースのため、他のデバイスの先頭にスタックされる LVM に対して再帰的に機能します。



注記

lvmetad デーモンが実行中の場合、**/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際に適用されません。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、このデバイスのスキャンは一切行われません。VM で LVM を使用する場合で 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には、グローバルフィルターを使用する必要がある場合があります。

3.7. LVM コマンドの使用による LVM 情報の表示

lvm コマンドは、LVM サポートおよび設定についての情報を表示するため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 dumpconfig**

  /etc/lvm/lvm.conf ファイルおよびその他の設定ファイルのロード後に LVM 設定情報を表示します。LVM 設定ファイルの情報は、付録B *LVM 設定ファイル* を参照してください。

- **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`**

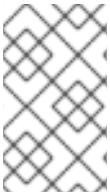
  このホストに定義されているタグを表示します。LVM オブジェクトタグの情報は、付録C *LVM オブジェクトタグ* を参照してください。

- **`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、これを保守するために 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 ボリュームを作成するか、または変更する場合、**clvmd** デーモンが稼働していることを確認する必要があります。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。

4.1. CLI コマンドの使用

すべての LVM CLI コマンドには一般的な特性がいくつかあります。

コマンドライン引数でサイズが必要な場合は、単位を常に明示的に指定することができます。単位を指定しないと、デフォルト (通常 KB または 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。

コマンドライン引数内で単位を指定する場合、LVM は大文字/小文字を区別せず (たとえば、M または m を指定しても同じことになります)、2 の累乗 (1024 の倍数) が使用されます。しかし、コマンド内で **--units** 引数を指定する場合、小文字の場合は単位が 1024 の倍数であり、大文字の場合は 1000 の倍数であることを示します。

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、完全なパス名はオプションとなります。ボリュームグループ **vg0** 内の論理ボリューム **lvol0** は **vg0/lvol0** と指定できます。ボリュームグループの一覧が必要であるのに空のままの場合、すべてのボリュームグループの一覧が代用されれます。論理ボリュームの一覧が必要な状態で、 1 つのボリュームグループだけ指定されている場合、そのボリュームグループ内のすべての論理ボリュームの一覧が代用されます。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 コマンドは **-v** 引数を使用できるため、これを複数回入力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を示しています。

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

以下の例は、**-v** 引数を使用した **lvcreate** コマンドの出力を示しています。

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

また、**-vv**、**-vvv**、または **-vvvv** の引数を使用して、コマンドの実行内容を徐々に詳しく表示することができます。**-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、**-vvvv** 引数が指定された **lvcreate** コマンドの出力の最初の数行のみを示しています。

```
# 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 CLI コマンドのいずれかに **--help** 引数を付けると、そのコマンドのヘルプを表示することができます。

```
# 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 lvm 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、オブジェクトを作成する時点で割り当てられる UUID によって、内部で参照されます。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** を削除した後で、接続し直した際に **/dev/sdk** になっている場合などに役立ちます。LVM は物理ボリュームをそのデバイス名ではなく、UUID で識別するため、依然として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要素を実行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

以下のサブセクションは、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

物理ボリューム用にディスクデバイス全体を使用している場合、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。DOS のディスクパーティションの場合は、**fdisk**、**cfdisk** またはそれらと同等のコマンドを使用して、パーティション ID を 0x8e に設定している必要があります。ディスクデバイス全体のみの場合は、パーティションテーブルを消去する必要があり、これによって、そのディスク上のすべてのデータが実質的に破棄されます。以下のコマンドを使用して、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することができます。

```
# 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、LVM 論理ボリュームの一部として後で使用するために、**/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。

```
# 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上で **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 論理ボリュームの一部として後で使用するために、LVM 物理ボリュームとして初期化します。

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

4.2.1.3. ブロックデバイスのスキャン

以下の例に示されるように、**lvmdiskscan** コマンドを使用し、物理ボリュームとして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ー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、物理ボリューム情報を設定可能な形式で提供し、1 つの物理ボリュームごとに 1 行ずつ表示します。**pvs** コマンドは形式の制御をかなり行うため、スクリプト作成に役立ちます。出力をカスタマイズするために **pvs** コマンドを使用する方法に関する情報は 「LVM のカスタム報告」 を参照してください。

pvdisplay コマンドは、各物理ボリュームについて複数行の詳細出力を提供します。一定の形式で物理プロパティー (サイズ、エクステント、ボリュームグループなど) を表示します。

以下の例は、単一物理ボリュームについての **pvdisplay** コマンドの出力を示しています。

```
# 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-XO8M-mcpsVe
```

pvscan コマンドは、物理ボリュームについて、システム内のサポートされたすべての LVM ブロックデバイスをスキャンします。

以下のコマンドは、検出される物理デバイスをすべて表示します。

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

このコマンドが特定の物理ボリュームをスキャンしないように **lvm.conf** 内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方法については、「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。

4.2.3. 物理ボリューム上での割り当ての防止

pvchange コマンドを使用すると、1 つ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を割り当てることを防止できます。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** 上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、以前に無効にされていた割り当てを許可することができます。

4.2.4. 物理ボリュームのサイズ変更

何かの理由で配下の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 を新規サイズで更新します。このコマンドは LVM が物理ボリュームを使用している間に実行することができます。

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 ラベルを削除することができます。**pvremove** コマンドを実行すると、空の物理ボリューム上の 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現在ボリュームグループの一部である場合、「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要素を実行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新しいボリュームグループを作成し、それに対して最低 1 つの物理ボリュームを追加します。

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

```
# 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場合、ディスク領域はデフォルトでは 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/縮小するための最小単位です。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、**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 **-l** の引数を使用すると、ボリュームグループに追加できる物理ボリュームまたは論理ボリュームの数を限定することができます。

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その配下の論理ボリュームは、以下のような配置で **/dev** ディレクトリー内のデバイス特有のファイルがある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** の 2 つのボリュームグループを作成して、それぞれに **lv01**、**lv02**、**lv03** の 3 つの論理ボリュームがある場合、6 つのデバイス特殊ファイルが作成されることになります。

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

デバイス特殊ファイルは、対応する論理ボリュームが現在アクティブでない場合には表示されません。

LVM での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 1 つ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内の未割り当ての物理エクステントが割り当て用に生成されます。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した場合は、指定した物理ボリューム上のその範囲内では、未割り当ての物理エクステントのみが割り当て用に考慮にされます。

- 各割り当てポリシーは、最も厳格なポリシー (**contiguous**) から始まり、最後は **--alloc** オプションを使用して指定されるか、または特定の論理ボリュームやボリュームグループ用にデフォルトとして設定される割り当てポリシーへと順番に試行されます。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、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。領域がさらに必要な場合は、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 1 番目の論理エクステントではない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させる必要があります。

  論理ボリュームがストライプ化またはミラー化されると、**contiguous** の割り当て制限が、領域を必要とするそれぞれの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。

- **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される任意の論理エクステントに使用される物理ボリュームは、その論理ボリューム内で 1 つ以上の論理エクステントによって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いる場合で、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、これらの両方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、割り当て用に、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、これら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、その並列論理ボリュームにすでに割り当て済みの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は最初にログやデータに対して異なる物理ボリュームの選択を試行します。それが不可能で、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、ログにより物理ボリュームとデータの一部を共有できるようになります。

  同様に、シンプールメタデータを割り当てる場合、**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注意事項に従い、**allocation/thin_pool_metadata_require_separate_pvs** 設定パラメーターの値に基づきます。

- 割り当て要求を満たすだけの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がそれらを使用しない場合は、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れらを使用します。

割り当てポリシーは **vgchange** コマンドを使用して変更できます。

注記

定義された割り当てポリシーに基づいてこのセクションで文書化されている以上のレイアウトの動作が必要な場合、今後のバージョンのコードで変更がある可能性がある点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同一番号を持つ 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、LVM はそれらが表示されている順番でそれぞれを使用するように認識します。ただし、今後のリリースでこのプロパティーが引き続き維持される保証はありません。特定の論理ボリュームに特定のレイアウトを取得することが重要な場合は、各ステップ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** の一連のステップでレイアウトを構築するようにしてください。

割り当てプロセスが特定ケースで実行される現在の方法を表示するには、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読み取ることができます。

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。単一ノードで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、共有ストレージ上の CLVM で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対して可視になります。ただし、クラスター内の 1 つのノードのみに可視となる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。**vgcreate** コマンドの **-c n** オプションを使用して、これを実行します。

クラスター環境内で以下のコマンドを実行すると、コマンドの実行元であ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がローカルか、またはクラスターかを変更できます。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** 見出しの下にある **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 コマンドに関する詳細は、「ボリュームグループの表示 」 「LVM 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新規に追加するには、**vgextend** コマンドを使用します。**vgextend** コマンドは、 1 つまたは複数の空き物理ボリュームを追加することによってボリュームグループの容量を拡大します。

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ー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** コマンドは形式の制御をかなり行うため、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** コマンドの使用方法の情報は、「LVM のカスタム報告」 を参照してください。

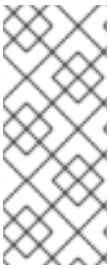
vgdisplay コマンドは、一定の形式でボリュームグループのプロパティー (サイズ、エクステント、物理ボリュームの数など) を表示します。以下の例は、ボリュームグループ **new_vg** 用の **vgdisplay** コマンドの出力を示しています。ボリュームグループを指定しないと、すべての既存ボリュームグループが表示されます。

```
# 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-OpMO-0118-nlwO-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、システム内のすべてのサポートされる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** ファイル内に LVM キャッシュファイルが構築され、ここで現在の LVM デバイスの一覧が維持されます。

LVM は、システムの起動時や、**vgcreate** コマンドの実行時や 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になる可能性があります。

lvm.conf ファイル内でフィルターを定義することで、特定デバイスを避けるようにスキャンを限定できます。スキャンするデバイスを制御するためのフィルターの使用方法については、「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。

次の例は、**vgscan** コマンドの出力を示しています。

```
# 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、 1 つまたは複数の空の物理ボリュームを削除することにより、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によって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-MOkn-76iP-iftz-OVSen7
```

物理ボリュームがまだ使用されている場合、**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。その後に、**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、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するには、見つからない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、**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。

4.3.8. 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ついていくつかの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。

4.3.9. 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化されます。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
```

クラスターロッキングが有効な場合には、「e」を追加すると 1 つのノード上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、ローカルノード上のみで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、1 度に 1 つのノード上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の変更」 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。

4.3.10. 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。

```
# vgremove 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結合

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トサイズが同じで、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の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** ファイル内で無効になっていない限り、すべての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/archives** ファイルに保存されます。**vgcfgbackup** コマンドを使用するとメタデータを **/etc/lvm/backup** ファイルに手動でバックアップできます。

vgcfrestore コマンドは、アーカイブからボリュームグループのメタデータを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ための **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。

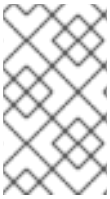
以下のいずれかのコマンドで、既存ボリュームグループ **vg02** の名前を **my_volume_group** に変更できます。

```
# 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

Red Hat Enterprise Linux 6.5 の時点では、**vgimport** コマンドの **--force** 引数が利用可能です。これにより、物理ボリュームの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できます。

vgexport コマンドは、システムが非アクティブのボリュームグループに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され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動するには、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の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。これは、削除中のシステムからのボリュームグループへのアクセスを防止します。

   ボリュームグループをエクスポートした後に、**pvscan** コマンドを実行すると、以下の例のように物理ボリュームがエクスポート先のボリュームグループ内に表示されます。

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

   システムが次にシャットダウンされる際に、ボリュームグループを構成していたディスクを取り外し、それらを新しいシステムに接続することができます。

4. ディスクが新しいシステムに接続されると、**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化します。

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殊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 特殊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。このコマンドは足りない特殊ファイルを作成し、未使用のファイルを削除します。

vgscan コマンドに **mknodes** 引数を指定することにより、**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場合、論理ボリュームはボリュームグループを構成する物理ボリューム上の空きエクステントを使用してボリュームグループから構築されます。通常、論理ボリュームは配下の物理ボリューム上で次に使用可能な空き領域を基準にして空き領域を占有します。論理ボリュームを変更することで、物理ボリューム内の領域の開放と再割り当てが可能になります。

Red Hat Enterprise Linux 6.3 リリースでは、LVM を使用して RAID 論理ボリュームの作成、表示、名前変更、使用、削除を行うことができます。RAID 論理ボリュームの詳細は、「RAID 論理ボリューム」を参照してください。

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

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

次のコマンドは、ボリュームグループ **testvg** 内に **testlv** という 1500 MB 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** 内の空きエクステントから **gfslv** という 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、エクステント内の論理ボリュームのサイズを指定することができます。この引数を使用すると、論理ボリュームに使用するボリュームグループのパーセンテージも指定できます。以下のコマンドは、ボリュームグループ **testvg** 内で全体の領域の 60% を使用する **mylv** と呼ばれる論理ボリュームを作成します。

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

lvcreate コマンドで **-l** 引数を使用すると、ボリュームグループ内で残っている空き領域のパーセンテージを、論理ボリュームのサイズとして指定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 **testvol** 内の未割り当て領域をすべて使用する **yourlv** と呼ばれる論理ボリュームを作成します。

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

lvcreate コマンドで、**-l** 引数を使用して、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。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別の方法としては、**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することです。

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

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

論理ボリュームの作成に使用した配下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についての情報は、「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内の特定の物理ボリュームから割り当てる論理ボリュームを作成するには、**lvcreate** コマンドラインの末尾に物理ボリュームを指定する必要があります。以下のコマンドは、物理ボリューム **/dev/sdg1** から割り当てられるボリュームグループ **testvg** 内に論理ボリューム **testlv** を作成します。

```
# lvcreate -L 1500 -ntestlv 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。以下の例は、ボリュームグループ **testvg** 内にエクステントが 0 から 24 の物理ボリューム **/dev/sda1** およびエクステントが 50 から 124 の物理ボリューム **/dev/sdb1** の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た後に、エクステントが 100 の論理ボリュームを配置しています。

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

論理ボリュームのエクステントが割り当てられる方法についてのデフォルトポリシーは **inherit** であり、ボリュームグループの場合と同じポリシーに適用されます。これらのポリシーは **lvchange** コマンドを使用して変更できます。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

大量の連続的な読み取りと書き込みを行う場合、ストライプ化論理ボリュームを作成すると、データ I/O が効率化されます。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時には、**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。これは、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を決定します。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** 引数が使用される場合は例外)。

ストライプ化論理ボリュームを構成する配下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その配下の最小デバイスで決定されます。たとえば、2 レッグのストライプがある場合、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場合、最大サイズは最小デバイスの 3 倍になります。

以下のコマンドは、64KB 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。論理ボリュームのサイズは 50 ギガバイトで、**gfslv** と呼ばれ、ボリュームグループ **vg0** から構築されます。

```
# lvcreate -L 50G -i2 -I64 -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することができます。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、**stripelv** と呼ばれる 100 エクステントのストライプ化ボリュームを **testvg** のボリュームグループ内に作成します。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 -i2 -nstripelv testvg /dev/sda1:0-49 /dev/sdb1:50-99
  Using default stripesize 64.00 KB
  Logical volume "stripelv" created
```

4.4.3. ミラー化ボリュームの作成

注記

Red Hat Enterprise Linux 6.3 リリースは、LVM は RAID4/5/6 およびミラーリングの新たな実装をサポートしています。この新しい実装の詳細については、「RAID 論理ボリューム」 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に達しており、かつクラスターロッキングを有効化するように **lvm.conf** ファイル内の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作成の例については、「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。

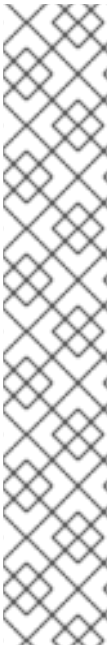
単一クラスター内の複数のノードから短時間に連続して複数の LVM ミラー作成および変換コマンドを実行しようとすると、これらのコマンドのバックログが生じる場合があります。これによって、要求した動作がタイムアウトになり、その後に失敗する可能性があります。この問題を回避するために、クラスターミラー作成コマンドをそのクラスターの単一ノードから実行することを推奨します。

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、作成するデータのコピー数を指定します。**-m1** を指定すると、1 つのミラーが作成され、ファイルシステムのコピーが合計 2 つとなります (1 つのリニア論理ボリュームと 1 つのコピー)。同様に **-m2** を指定すると、2 つのミラーが作成され、ファイルシステムのコピーが合計 3 つになります。

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。ボリュームは、サイズは 50 ギガバイトで、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。

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

デフォルトで LVM ミラーデバイスは、コピー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** コマンドで **-R** 引数を使用して、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** ファイル内の **mirror_region_size** 設定を編集して、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が 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。リージョンサイズを大きくしておかないと、LVM の作成がハングしてしまい、またその他の LVM コマンドもハングしてしまう可能性もあります。

1.5TB を超えるミラー用のリージョンサイズを指定するための一般的なガイドラインとして、ミラーサイズをテラバイト単位で捉えて、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できます。たとえば、ミラーサイズが 1.5TB の場合、**-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** を指定することができます。

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 -m1 -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。大きなミラーコンポーネントの場合は、同期プロセスに長い時間がかかる可能性があります。Red Hat Enterprise Linux 6.3 では、回復させる必要のない新規のミラーを作成している場合は、**--nosync** 引数を指定して、最初のデバイスからの初期の同期が不要であることを示すことができます。

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ため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上に保持され、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上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。しかし、これには再起動のたびにミラー全体を再同期することが必要になります。

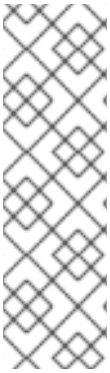
以下のコマンドは、ボリュームグループ **bigvg** からミラー化論理ボリュームを作成します。論理ボリュームの名前は **ondiskmirvol** であり、これには単一のミラーがあります。ボリュームのサイズは 12MB で、ミラーログをメモリーに保持します。

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

このミラーログは、いずれかのミラーレッグが作成されるデバイスとは異なるデバイス上で作成されます。しかし、**vgcreate** コマンドに **--alloc anywhere** 引数を使用することにより、ミラーレッグの 1 つと同じデバイス上にミラーログを作成することが可能です。これはパフォーマンスを低下させる可能性がありますが、配下のデバイスが 2 つしかない場合で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ッグの 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 つのデバイスのみ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 **mirrorlv** という名前で、サイズが 500 MB のボリュームが作成されます。

```
# lvcreate -L 500M -m1 -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 -L 12MB -m1 --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ッ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が、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ッ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。大きなミラーコンポーネントの場合は、同期プロセスには長い時間がかかる可能性があります。回復させる必要のない新規のミラーを作成している場合は、**--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示すことができます。

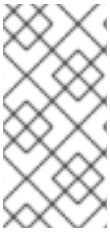
ミラーレッグとログ用に使用するデバイス、およびそのデバイスで使用するエクステントを指定することができます。ログを特定のディスクに強制するには、それが配置されるディスク上のエクステントを正確に 1 つ指定します。LVM は、コマンドラインでデバイスが一覧表示される順序を必ずしも優先しません。物理ボリュームが一覧にあれば、それが割り当てが実行される唯一のスペースになります。割り当て済みの物理エクステントが一覧にある場合、そのエクステントは無視されます。

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 のミラーレッグはデバイス **/dev/sda1** 上にあり、第 2 のミラーレッグはデバイス **/dev/sdb1** 上にあり、そのミラーログは **/dev/sdc1** 上にあります。

```
# lvcreate -L 500M -m1 -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 のミラーレッグは、エクステントが 0 から 499 のデバイス **/dev/sda1** にあり、第 2 のミラーレッ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。これらは 1MB のエクステントです。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 -L 500M -m1 -n mirrorlv vg0 /dev/sda1:0-499 /dev/sdb1:0-499
/dev/sdc1:0
```



注記

Red Hat Enterprise Linux 6.1 リリースでは、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できます。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。

4.4.3.1. 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ます。このパラメーターが **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スペース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スペースがない場合、このポリシーは **remove** ポリシーと同様に機能します。

デフォルトでは、**mirror_log_fault_policy** パラメーターは **allocate** に設定されています。ログにこのポリシーを使用するとプロセスが速まり、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、ミラーはクラッシュとシステムの再起動時の同期ステータスは記憶せず、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。

4.4.3.2. 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 つのミラーレッグが含まれます。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できます。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、**/dev/sdc1** と **/dev/sde1** のデバイスで構成される、2 つのミラーレッグが含まれます。

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

4.4.3.3. 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** パラメーターによっ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。

4.4.3.4. 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、基本的に既存ボリューム用にミラーレッ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ッグ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。

ミラーレッグを 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して、ミラーの冗長性なしにボリュームに依然としてアクセスできます。そのレッ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」 に説明されています。

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、ミラーレッグを削除します。

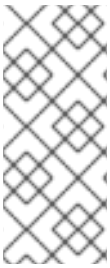
```
# lvconvert -m0 vg00/lvol1
```

以下のコマンドは、既存の論理ボリューム **vg00/lvol1** にミラーレッグを追加します。この例は、**lvconvert** コマンドがそのボリュームを 2 つのミラーレッ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)
```

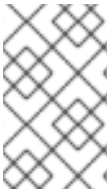
4.4.4. シンプロビジョニングされた論理ボリュームの作成

Red Hat Enterprise Linux 6.4 リリースでは、論理ボリュームのシンプロビジョニングが可能です。これにより、利用可能なエクステントより大きい論理ボリュームを作成することができます。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、アプリケーションが必要とする場合に、これを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際など後で割り当てられるように、シンプールにバインドできるデバイスを作成できます。シンプールでは、ストレージ領域のコスト効率の良い割り当てに必要となる動的な拡張が可能です。



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ーを使用する方法についての情報は、**lvmthin**(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内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。

シンボリュームを作成するには、以下のタスクを実行してください。

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シンプールとプール内のシン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** コマンドに **-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。コマンドが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を推定します。

```
# 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 -V1G -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、シンプールとプール内にシン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** 内に **mythinpool** という名前のシンプールを作成し、そのプール内に **thinvolume** という名前のシンボリュームも作成します。

```
# lvcreate -L 100M -T vg001/mythinpool -V1G -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 -L100M -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。

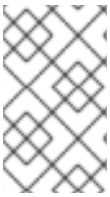
以下のコマンドは、既存のシンプールのサイズ (100M) を変更し、100M 拡張します。

```
# 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** は数式 (Pool_LV_size / Pool_LV_chunk_size * 64) に沿って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。現時点ではメタデータボリュームのサイズを変更することはできませんが、後でシンプールのサイズが大幅に拡大されることが予測される場合には、**lvcreate** コマンドの **--poolmetadatasize** パラメーターを使ってこの値を増やしておくことをお勧めします。シンプールのメタデータ論理ボリュームのサポートされる値は、2MiB から 16GiB の間です。

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** 内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ています。また、ボリュームグループ **vg001** 内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ています。

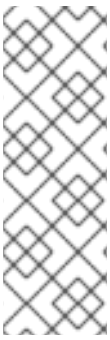
```
# lvconvert --thinpool vg001/lv1 --poolmetadata vg001/lv2
  Converted vg001/lv1 to thin pool.
```

4.4.5. スナップショットボリュームの作成

注記

Red Hat Enterprise Linux 6.4 リリースでは、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。

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、Red Hat Enterprise Linux 6.1 リリースでは、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。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についての情報は、「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

Red Hat Enterprise Linux 6.1 リリースでは、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。

Red Hat Enterprise Linux 6.3 リリースでは、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 論理ボリュームの詳細は、「RAID 論理ボリューム」 を参照してください。

Red Hat Enterprise Linux 6.5 リリースの時点では、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 LVM では作成できません。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を作成します。

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についての情報は、「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、**/dev/vg00/snap** という名前でサイズが 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、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-OjsI-LT03-nHLC-y8XW-EhCl78
  LV Write Access        read/write
  LV snapshot status     source of
                         /dev/new_vg/newvgsnap1 [active]
  LV Status              available
  # open                 0
  LV Size                52.00 MB
  Current LE             13
  Segments               1
  Allocation             inherit
  Read ahead sectors     0
  Block device           253:2
```

デフォルトでは **lvs** コマンドは、複製元のボリュームと、各スナップショットボリューム用に使用されているスナップショットボリュームの現在のパーセンテージを表示します。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%
  lvol0      new_vg owi-a- 52.00M
  newvgsnap1 new_vg swi-a-  8.00M lvol0    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% 一杯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。

Red Hat Enterprise Linux 6.2 リリースでは、スナップショット関連の新機能が 2 つあります。1 つ目は、スナップショットが満杯になったときに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が強制的にマウント解除される機能です。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。2 つ目は、**lvm.conf** ファイル内で **snapshot_autoextend_threshold** オプション

を指定できる点です。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。

Red Hat Enterprise Linux 6.5 リリースの時点では、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 LVM では作成できません。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ほど十分拡大されると、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。

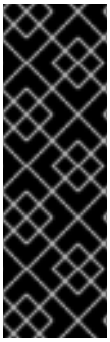
snapshot_autoextend_threshold と **snapshot_autoextend_percent** の設定についての情報は、**lvm.conf** ファイルに記載されています。**lvm.conf** ファイルの詳細は、付録B *LVM 設定ファイル* を参照してください。

4.4.6. 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の作成

Red Hat Enterprise Linux 6.4 バージョンの LVM 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利点と制限についての情報は、「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。

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についての情報は、**lvmthin**(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せん。たとえば、**lvcreate -s vg/thinvolume -L10M** コマンドは、複製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複製元ボリューム用に作成できます。Red Hat Enterprise Linux 6.5 リリースの時点では、シンスナップショットは、シンプロビジョニングされない複製元ボリューム用にも作成できます。

lvcreate コマンドの **--name** オプションを使って、スナップショットボリュームの名前を指定することができます。このオプションは、論理ボリュームを作成する際に使用することをお勧めします。これにより、**lvs** コマンドを使って論理ボリュームを表示する際に、作成したボリュームを簡単に表示できるようにするためです。

以下のコマンドは、シンプロビジョニングされた論理ボリューム **vg001/thinvolume** の、**mysnapshot1** という名前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 mysnapshot1 vg001/thinvolume
```

```
  Logical volume "mysnapshot1" created
# lvs
  LV          VG       Attr     LSize   Pool       Origin     Data%
Move Log Copy%  Convert
  mysnapshot1 vg001    Vwi-a-tz   1.00g mythinpool thinvolume   0.00
  mythinpool  vg001    twi-a-tz 100.00m                         0.00
  thinvolume  vg001    Vwi-a-tz   1.00g mythinpool             0.00
```

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についての情報は、「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。

Red Hat Enterprise Linux 6.5 リリースの時点では、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ができます。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、これは *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。論理ボリュームの **origin_volume** は、その後に既存のシンプール **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になければならないため、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
```

4.4.7. スナップショットボリュームのマージ

Red Hat Enterprise Linux 6 リリースでは、**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開いている状態でなければ、マージはただちに開始されます。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最初にアクティブ化され、かつ両方が閉じられている状態で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化されるまで先延ばしされます。マージが開始すると、結果として生じる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。マージの進行中は、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** はすべて **@some_tag** でタグ付けされます。以下のコマンドは、3 つすべて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C *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。

4.4.8. 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大きいマイナー番号を使用します。

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。

4.4.9. 論理ボリュームのサイズ変更

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 (または LVM GUI を使用して)、論理ボリュームが常に、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** 内の論理ボリューム **lvol1** のサイズを 3 つ論理エクステント分縮小します。

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

4.4.10. 論理ボリュームグループのパラメーターの変更

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 **lvol1** のパーミッションを読み取り専用に変更します。

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

4.4.11. 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に関する情報は 「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。

4.4.12. 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。

4.4.13. 論理ボリュームの表示

LVM 論理ボリュームのプロパティー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** コマンドは形式の制御をかなり行うため、スクリプト作成に役立ちます。出力をカスタマイズするための **lvs** コマンドの使用方法に関する情報は 「LVM のカスタム報告」 を参照してください。

lvdisplay コマンドは、固定した形式で、論理ボリュームのプロパティー (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
```

4.4.14. 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、そのボリュームの追加容量または拡張後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** にさらに 1 ギガバイトを追加します。

```
# 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内の残りの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** 内の未割り当て領域をすべて満たすようにします。

```
# 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後は、それに適合するようにファイルシステムのサイズも拡張する必要があります。

デフォルトでは、ほとんどのファイルシステムサイズ変更ツールは、ファイルシステムのサイズを配下の論理ボリュームのサイズに拡大するので、2 つのコマンドのそれぞれで同じサイズを指定する必要はありません。

4.4.14.1. 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配下の物理ボリュームに、ストライプをサポートするための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してしまう 2 方向ストライプがある場合、ボリュームグループに 1 つの物理ボリュームを追加しただけでは、ストライプを拡張できるようにはなりません。そのためには、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る、2 つの配下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すべての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には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
```

以下のコマンドにより、ボリュームグループにもう 1 つの物理ボリュームを追加し、135G の領域を追加します。

```
# 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 つの配下の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 2 つの物理ボリュームを追加することにより、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配下の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
```

4.4.14.2. ミラー化ボリュームの拡張

Red Hat Enterprise Linux 6.3 リリースでは、新しいミラーリージョンの同期を実行しなくても **lvextend** コマンドでミラー化論理ボリュームを拡張することができます。

lvcreate コマンドでミラー化論理ボリュームを作成する場合、**--nosync** オプションを指定すると、ミラー作成時にミラーリージョンは同期されません。詳細は、「ミラー化ボリュームの作成」 を参照してください。後ほど **--nosync** オプションを使用して作成したミラーを拡張する場合も、ミラーの拡張部分は同期されません。

--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論理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ミラー化ボリュームの場合、その論理ボリュームには「M」の属性ビット 1 が設定されます。論理ボリュームが初期同期により作成された場合は、「m」の属性ビット 1 が設定されます。

以下のコマンドは、初期同期を行わずに作成された **lv** という名前のミラー化論理ボリュームの属性を表示します。属性ビット 1 は「M」と表示されます。属性ビット 7 は「m」であり、**mirror** が対象のタイプであることを示します。属性ビットの意味についての情報は、表4.4「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     Copy%  Convert
  lv   vg   Mwi-a-m- 5.00g                         lv_mlog 100.00
```

このミラー化論理ボリュームを **lvextend** コマンドで拡張する場合、ミラーの拡張部分は再同期されません。

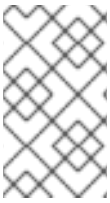
lvcreate コマンドで **--nosync** オプションを指定せずにミラー化論理ボリュームを作成した場合、**lvextend** コマンドで **--nosync** オプションを指定することで、ミラーを再同期することなく論理ボリュームを拡張することができます。

以下の例では、**--nosync** オプションなしで作成された論理ボリュームを拡張し、ミラー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はミラーが同期されないように指定しています。ボリュームには「m」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「M」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     Copy%
Convert
  lv   vg   mwi-a-m- 20.00m                          lv_mlog 100.00
# lvextend -L +5G vg/lv --nosync
  Extending 2 mirror images.
  Extending logical volume lv to 5.02 GiB
  Logical volume lv successfully resized
# lvs vg
  LV   VG   Attr     LSize Pool Origin Snap%  Move Log     Copy%  Convert
  lv   vg   Mwi-a-m- 5.02g                         lv_mlog 100.00
```

ミラーが非アクティブの場合、ミラー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ミラー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

ミラーがリカバリーを実行する場合、**--nosync** オプションを指定してボリュームの作成/拡張を行うと、ミラー化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、リカバリー中にミラーを拡張できます。

4.4.14.3. 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領域が、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、それらの場所に応じて、物理ボリュームにタグ付けすることができます。この場合、物理ボリュームに @site1 や @site2 というタグを付けて、**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C *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、ミラーレッ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]1
  PV         VG   Fmt  Attr PSize   PFree   PV Tags
  /dev/sdb1  taft lvm2 a-   135.66g 135.66g A
  /dev/sdc1  taft lvm2 a-   135.66g 135.66g B
  /dev/sdd1  taft lvm2 a-   135.66g 135.66g B
  /dev/sde1  taft lvm2 a-   135.66g 135.66g C
  /dev/sdf1  taft lvm2 a-   135.66g 135.66g C
  /dev/sdg1  taft lvm2 a-   135.66g 135.66g A
  /dev/sdh1  taft lvm2 a-   135.66g 135.66g A
```

以下のコマンドは、ボリュームグループ **taft** から 100GB のミラー化ボリュームを作成します。

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

以下のコマンドは、ミラーレッグおよびミラーログ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         Copy%  Devices
  mirror            taft      Mwi-a- 100.00g mirror_mlog 100.00
mirror_mimage_0(0),mirror_mimage_1(0)
  [mirror_mimage_0] taft      iwi-ao 100.00g
/dev/sdb1(0)
  [mirror_mimage_1] taft      iwi-ao 100.00g
/dev/sdc1(0)
  [mirror_mlog]     taft      lwi-ao   4.00m
/dev/sdh1(0)
```

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ッグが拡張される必要があることを示します。

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

以下に表示したコマンドは、レッグと同一のタグが付いた物理ボリュームを使用してミラーレッ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         Copy%  Devices
  mirror            taft      Mwi-a- 200.00g mirror_mlog  50.16
mirror_mimage_0(0),mirror_mimage_1(0)
  [mirror_mimage_0] taft      Iwi-ao 200.00g
/dev/sdb1(0)
  [mirror_mimage_0] taft      Iwi-ao 200.00g
/dev/sdg1(0)
  [mirror_mimage_1] taft      Iwi-ao 200.00g
/dev/sdc1(0)
  [mirror_mimage_1] taft      Iwi-ao 200.00g
/dev/sdd1(0)
  [mirror_mlog]     taft      lwi-ao   4.00m
/dev/sdh1(0)
```

4.4.15. RAID 論理ボリューム

Red Hat Enterprise Linux 6.3 リリースでは、LVM は RAID4/5/6 およびミラーリングの新たな実装をサポートしています。最新のミラーリングの実装と以前のミラーリングの実装 (「ミラー化ボリュームの作成」 で説明) が異なる点は以下のとおりです。

- ミラーリングの新たな実装でのセグメントタイプは **raid1** です。以前の実装でのセグメントタイプは **mirror** です。
- RAID 4/5/6 実装のように、ミラーリングの新たな実装は MD ソフトウェア RAID を活用します。
- ミラーリングの新たな実装は、各ミラーイメージの完全な冗長性を備えたビットマップ領域を維持し、これにより、その障害処理機能を拡大します。これは、このセグメントタイプで作成されるミラーには **--mirrorlog** や **--corelog** オプションはないことを意味しています。
- ミラーリングの新実装は、一時的な障害を処理することができます。
- ミラー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。
- ミラーリングの新実装はスナップショットをサポートします (高いレベルの RAID の実装も同様)。
- RAID の新たな実装はクラスター対応ではありません。クラスターボリュームグループ内に LVM RAID 論理ボリュームを作成することはできません。

RAID 論理ボリュームが障害に対応する方法については、「RAID 障害ポリシーの設定」 を参照してください。

このセクションの残りの部分では、LVM RAID デバイスで実行できる以下の管理タスクについて説明します。

- 「RAID 論理ボリュームの作成」
- 「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」
- 「既存の RAID1 デバイス内のイメージ数の変更」
- 「別々の論理ボリュームとしての RAID イメージの分割」
- 「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」
- 「RAID1 論理ボリュームでの I/O 操作の制御」

4.4.15.1. RAID 論理ボリュームの作成

RAID 論理ボリュームを作成するには、**lvcreate** コマンドの **--type** 引数として raid タイプを指定します。通常、**lvcreate** コマンドを使用して論理ボリュームを作成する場合、**--type** 引数は暗黙的になります。たとえば、**-i *stripes*** 引数を指定する場合、**lvcreate** コマンドは **--type stripe** オプションを想定します。**-m *mirrors*** 引数を指定する場合は、**lvcreate** コマンドは **--type mirror** オプションを想定します。ただし、RAID 論理ボリュームを作成する場合は、任意のセグメントタイプを明示的に指定する必要があります。使用される可能性のある RAID のセグメントタイプは 表 4.1「RAID のセグメントタイプ」 に記載されています。

表4.1 RAID のセグメントタイプ

セグメントタイプ	説明
raid1	RAID1 ミラーリング
raid4	RAID4 専用のパリティーディスク
raid5	**raid5_ls** と同様
raid5_la	RAID5 左非対称 パリティー 0 をローテートしてデータを継続
raid5_ra	RAID5 右非対称 パリティー N をローテートしてデータを継続
raid5_ls	RAID5 左対称 パリティー 0 をローテートしてデータを再起動
raid5_rs	RAID5 右対称 パリティー N をローテートしてデータを再起動
raid6	**raid6_zr** と同様
raid6_zr	RAID6 ゼロの再起動 パリティーゼロを (左から右に) ローテートしてデータを再起動)
raid6_nr	RAID6 N 再起動 パリティー N を (左から右に) ローテートしてデータを再起動

セグメントタイプ	説明
raid6_nc	RAID6 N 継続 パリティー N を (左から右に) ローテートしてデータを継続
raid10 (Red Hat Enterprise Linux 6.4 以降)	ストライプ化ミラー ミラーセットのストライピング

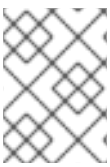
大半のユーザーの場合、5 つのプライマリータイプ (**raid1**、**raid4**、**raid5**、**raid6**、**raid10**) の中から 1 つを指定するだけで十分です。RAID 5/6 が使用する各種アルゴリズムの詳細は、http://www.snia.org/sites/default/files/SNIA_DDF_Technical_Position_v2.0.pdf の 『Common RAID Disk Data Format Specification』 の第 4 章を参照してください。

RAID 論理ボリュームを作成する場合、LVM は各データまたはアレイ内のパリティー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、2 つのメタデータサブボリューム (**lv_rmeta_0** および **lv_rmeta_1**) と 2 つのデータサブボリューム (**lv_rimage_0** および **lv_rimage_1**) が生じます。同様に、3 方向のストライプ (＋ 1 つの暗黙的なパリティーデバイス) RAID4 を作成すると、4 つのメタデータサブボリューム (**lv_rmeta_0**、**lv_rmeta_1**、**lv_rmeta_2**、および **lv_rmeta_3**) と 4 つのデータサブボリューム (**lv_rimage_0**、**lv_rimage_1**、**lv_rimage_2**、および **lv_rimage_3**) が生じます。

以下のコマンドは、ボリュームグループ **my_vg** 内にサイズが 1G の **my_lv** という名前の 2 方向の RAID1 アレイを作成します。

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

-m 引数に指定する値に沿って、異なる数のコピーで RAID1 アレイを作成することができます。**-m** 引数は、以前のミラー実装用のコピー数を指定するために使用される同じ引数ですが、この場合はセグメントタイプを **raid1** と明示的に指定することで、デフォルトのセグメントタイプ **mirror** を上書きします。同様に、**-i argument** を使用して RAID 4/5/6 論理ボリュームのストライプ数を指定して、デフォルトのセグメントタイプを必要な RAID タイプで上書きします。また、ストライプサイズも **-I** 引数で指定できます。



注記

デフォルトのミラーセグメントタイプを **raid1** に設定するには、**lvm.conf** ファイルの **mirror_segtype_default** を変更します。

以下のコマンドは、ボリュームグループ **my_vg** 内のサイズが 1G の **my_lv** という RAID5 アレイ (3 つのストライプ + 1 つの暗黙的なパリティードライブ) 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に対して指定するのと同じように行います。パリティードライブの正確な数は自動

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** 内にサイズが 1G の **my_lv** と呼ばれる RAID6 アレイ (3 つのストライプ + 2 つの暗黙的なパリティードライブ) を作成します。

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

LVM によって RAID 論理ボリュームを作成した後には、ボリュームのアクティブ化、変更、削除、表示、使用を他の LVM 論理ボリュームと同じように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。これはとくに数多くの RAID 論理ボリュームを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。

Red Hat Enterprise Linux 6.5 の時点では、復旧スロットルを実装することにより、RAID 論理ボリュームが初期化される速度を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** オプションを使用したそれらの操作の最小および最大 I/O 速度を設定して実行できます。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、アレイ内のそれぞれのデバイスに対して 1 秒あたりの量として指定されます。サフィックスが指定されない場合、kiB/sec/device が想定されます。復旧速度を 0 に設定すると、これが無制限になります。

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

  RAID 論理ボリュームの最小復旧速度を設定し、**sync** 操作の I/O が、負荷の高い通常の I/O がある場合でも最小スループットを達成できるようにします。*速度* はアレイ内のそれぞれのデバイスに対して 1 秒あたりの量として指定されます。サフィックスが指定されない場合、kiB/sec/device が想定されます。

以下のコマンドは、最大復旧速度が 128 kiB/sec/device で、サイズが 10G の 3 つのストライプのある 2-way RAID10 アレイを作成します。このアレイは **my_lv** という名前で、ボリュームグループ **my_vg** にあります。

```
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。

4.4.15.2. 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** 内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、新しい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同じ物理ボリューム上の (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 アレイへの変換後、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になる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場合、**lvconvert** は失敗します。

4.4.15.3. 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/sda2** 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)
```

4.4.15.4. 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、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_***) の名前を RAID サブボリューム (***_rimage_***) に変更します。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上にある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)
```

4.4.15.5. 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を変更するには、LVM ミラーリングの初期実装でイメージ数を変更する際と同様に、**lvconvert** コマンドを使用して、追加/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できます。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場合、結果として生じるデバイス用のイメージの合計数を指定できます。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、オプションとして新しいメタデータ/データイメージのペアが存在する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 (名前は ***_rmeta_***) 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上に常に存在します。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 アレイ内の別のメタデータ/データサブボリュームのペアと同じ物理ボリューム上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、以下は 2 方向 RAID1 アレイの LVM デバイス **my_vg/my_lv** を示しています。

```
# 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、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 デバイスに変換して、物理ボリューム **/dev/sdd1** がアレイに使用されるように指定します。

```
# 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、結果として生じるデバイス用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、オプションとして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が削除されると、それよりも大きな番号のイメージがスロットを埋めるために切り替わります。**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** を削除する場合、**lv_rimage_0** と **lv_rimage_1** で構成される RAID1 アレイが生じます。サブボリューム **lv_rimage_2** の名前は変更され、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て、削除するイメージを含む物理ボリュームを **/dev/sde1** として指定します。

```
# 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)
```

4.4.15.6. 別々の論理ボリュームとしての RAID イメージの分割

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) をデバイスの中心から削除する場合、それより大きい番号のイメージはスロットを埋めるために切り替わります。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)
```

4.4.15.7. 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** 引数を使用することにより、すべての変更を追跡しながら、読み取り専用に RAID1 アレイのイメージを一時的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、そのイメージをアレイに後で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の名前を変更することはできません。また、結果として生じるボリュームには以下の制約があります。

- 作成する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。
- 新規のボリュームおよび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、その後の **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場合、イメージが分割されてから変更され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ています。

```
# lvcreate --type raid1 -m2 -L1G -n my_lv .vg
  Logical volume "my_lv" created
# lvs -a -o name,copy_percent,devices my_vg
  LV               Copy%  Devices
  my_lv            100.00
my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]        /dev/sdb1(1)
  [my_lv_rimage_1]        /dev/sdc1(1)
  [my_lv_rimage_2]        /dev/sdd1(1)
```

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

以下の例は、残りのアレイ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ています。その後に、ボリュームをアレイ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した後に、その分割を永続化するためには第 2 のコマンド **lvconvert --splitmirrors** を発行し、**--trackchanges** 引数を指定せずにイメージを分割する最初の **lvconvert** コマンドを繰り返します。これで **--trackchanges** 引数が作成したリンクが機能しなくなります。

--trackchanges 引数を使用してイメージを分割した後は、アレイ上でその後に **lvconvert --splitmirrors** コマンドを発行することはできません。ただし、追跡されるイメージを永久に分割する場合は例外です。

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
```

4.4.15.8. 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場合、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、ログはデバイスが失敗したことを示します。これにより、ユーザーは取るべき一連の動作を判別することができます。

その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。

4.4.15.8.1. 「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 デバイスが失敗した場合は、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-H20O-wM2M-qdMANy.
Jan 17 15:57:27 bp-01 lvm[8599]: raid1 array, my_vg-my_lv, is not in-sync.
Jan 17 15:57:36 bp-01 lvm[8599]: raid1 array, my_vg-my_lv, is now in-sync.
```

raid_fault_policy フィールドは **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-H20O-wM2M-qdMANy.
  LV            Copy%  Devices
  lv            100.00 lv_rimage_0(0),lv_rimage_1(0),lv_rimage_2(0)
  [lv_rimage_0]        /dev/sdh1(1)
  [lv_rimage_1]        /dev/sdf1(1)
  [lv_rimage_2]        /dev/sdg1(1)
  [lv_rmeta_0]         /dev/sdh1(0)
  [lv_rmeta_1]         /dev/sdf1(0)
  [lv_rmeta_2]         /dev/sdg1(0)
```

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ることができなかったことを引き続き表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は RAID 論理ボリュームから削除されていますが、ボリュームグループからはまだ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 は **allocate** に設定されているものの、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」 に説明されているように、オプションとしてドライブを修正することができ、その後に論理ボリュームを非アクティブ化/アクティブ化できます。別の方法として、「RAID デバイスの置き換え」 で説明されているるように、障害が発生したデバイスを置き換えることも可能です。

4.4.15.8.2. 「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** ファイル内で **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。代わりに、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 fbI0YO-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 fbI0YO-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ることができなかったことを引き続き表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は RAID 論理ボリュームから削除されていますが、ボリュームグループからはまだ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、Red Hat Enterprise Linux リリース 6.5 の時点では、**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
```

4.4.15.9. 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は削除する必要がありました。そうしないと、ミラー化論理ボリュームがハングしたためです。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 タイプの場合、デバイスを削除することはレベルが下の RAID に変換することを意味します (たとえば、RAID6 から RAID5、または 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のではなく、LVM により、**lvconvert** コマンドに **--replace** 引数を使用することで RAID ボリューム内のデバイスをワンステップ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 -m2 -L 1G -n my_lv my_vg
  Logical volume "my_lv" created
# lvs -a -o name,copy_percent,devices my_vg
  LV               Copy%  Devices
  my_lv            100.00
my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]        /dev/sdb1(1)
  [my_lv_rimage_1]        /dev/sdb2(1)
  [my_lv_rimage_2]        /dev/sdc1(1)
  [my_lv_rmeta_0]         /dev/sdb1(0)
  [my_lv_rmeta_1]         /dev/sdb2(0)
  [my_lv_rmeta_2]         /dev/sdc1(0)
# lvconvert --replace /dev/sdb2 my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
  LV               Copy%  Devices
  my_lv             37.50
my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]        /dev/sdb1(1)
  [my_lv_rimage_1]        /dev/sdc2(1)
  [my_lv_rimage_2]        /dev/sdc1(1)
  [my_lv_rmeta_0]         /dev/sdb1(0)
  [my_lv_rmeta_1]         /dev/sdc2(0)
  [my_lv_rmeta_2]         /dev/sdc1(0)
```

以下の例は、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 -m1 -L 100 -n my_lv my_vg
  Logical volume "my_lv" created
# lvs -a -o name,copy_percent,devices my_vg
  LV               Copy%  Devices
  my_lv            100.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]        /dev/sda1(1)
  [my_lv_rimage_1]        /dev/sdb1(1)
  [my_lv_rmeta_0]         /dev/sda1(0)
```

```
  [my_lv_rmeta_1]         /dev/sdb1(0)
# pvs
  PV          VG       Fmt  Attr PSize    PFree
  /dev/sda1   my_vg    lvm2 a--  1020.00m  916.00m
  /dev/sdb1   my_vg    lvm2 a--  1020.00m  916.00m
  /dev/sdc1   my_vg    lvm2 a--  1020.00m 1020.00m
  /dev/sdd1   my_vg    lvm2 a--  1020.00m 1020.00m
# lvconvert --replace /dev/sdb1 my_vg/my_lv /dev/sdd1
# lvs -a -o name,copy_percent,devices my_vg
  LV               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、置き換えるドライブは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上に存在させることができません。

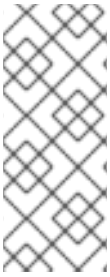
4.4.15.10. RAID 論理ボリュームのスクラビング

Red Hat Enterprise Linux 6.5 リリースの時点では、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ー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、それら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それらを修復します。

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* 操作は、**lvconvert --repair *vg/raid_lv*** 操作と同じ機能を実行しません。**lvchange --syncaction repair** 操作は、アレイ上でバックグラウンドの同期操作を開始しますが、**lvconvert --repair** 操作は、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。

新規の 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 ボリュームが実行している現在の同期操作を表示します。これには、以下の値のいずれか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ました (何も実行しない)
- **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 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ー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表示し、以下のインジケーターに対応す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す。

- (**r**)efresh (更新) は、LVM がデバイスラベルを読み取ることができ、かつデバイスが操作可能であると認識する場合でも、RAID アレイ内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は、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「(r)efresh」されるか、またはデバイスに障害が発生したことが疑われる場合はそれを置き換える「(r)eplace」必要があります。

lvs コマンドについての情報は、「オブジェクトの選択」 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 操作の速度が下が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することができます。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、アレイ内のそれぞれのデバイスに対して 1 秒あたりの量として指定されます。サフィックスが指定されない場合、kiB/sec/device が想定されます。復旧速度を 0 に設定すると、これが無制限になります。

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

  RAID 論理ボリュームの最小復旧速度を設定し、**sync** 操作の I/O が、負荷の高い通常の I/O がある場合でも最小スループットを達成できるようにします。*速度* はアレイ内のそれぞれのデバイスに対して 1 秒あたりの量として指定されます。サフィックスが指定されない場合は kiB/sec/device が想定されます。

4.4.15.11. RAID1 論理ボリュームでの I/O 操作の制御

Red Hat Enterprise Linux リリース 6.5 の時点では、**lvchange** コマンドの **--writemostly** および **--writebehind** パラメーターを使用して RAID1 論理ボリューム内のデバイスに対する I/O 操作を制御することができます。これらのパラメーターを使用するための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に **write-mostly** というマークを付けます。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。デフォルト動作では、論理ボリューム内の指定された物理ボリュームに **write-mostly** 属性を設定します。**:n** を物理ボリュームに追加して **write-mostly** フラグを削除することや、**:t** を指定して値を切り替えることができます。**--writemostly** 引数は、単一コマンドで 2 回以上指定することができ、1 回で論理ボリューム内のすべての物理ボリュームの write-mostly 属性を切り替えることが可能になります。

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

  write-mostly というマークが付けられる RAID1 論理ボリューム内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すべてが、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ことにより、設定がクリアされ、システムは値を任意に選択できるようになります。

4.4.16. 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後は、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解放されて、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 **lvol1** 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/lvol1
```

4.4.17. 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なしに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。

4.5. フィルターを使用した LVM デバイススキャンの制御

起動時に、**vgscan** コマンドが実行されて、システム上のブロックデバイスのスキャンによる 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、再スキャ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 がスキャ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、これらは **/dev** ディレクトリー内のデバイス名に適用されて、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 がスキャ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マッチングされるため、これらの例の一部は必ずしもベストプラクティスを示すもの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/.*/" ]
```

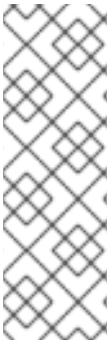
以下のフィルターはすべてのループと IDE を追加して、その他のすべてのブロックデバイスを削除します。

```
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、その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ない場合などには、グローバルフィルターを使用する必要がある場合があります。

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. オンラインでのデータの再配置

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ことを確認する必要があります。

以下のコマンドは、すべての割り当て領域を、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 コマンドは、その実行に長時間を要するため、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、移動の進捗状況をパーセンテージで 5 秒間ごとに報告します。

```
# 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に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C *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 のカスタム報告

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ー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。

以下のセクションでは、次のような内容を説明します。

- 生成されたレポートのフォーマットを制御するために使用できるコマンド引数の概要
- 各 LVM オブジェクトに選択できるフィールドの一覧
- 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
- レポート出力の単位を指定する手順

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-XO8M-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** コマンドで **-P** 引数を使用して、障害が発生したボリュームについて、出力では表示されないような情報を表示することができます。この引数によって得られる出力に関する情報は、「障害の発生したデバイスの情報表示」 を参照してください。

表示に関する引数の詳細の一覧は、**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. オブジェクトの選択

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加する可能性があります。既存の文字フィールドの位置を変更されませんが、新規フィールドは末尾に追加される可能性があります。特定の属性文字検索について、フィールドの終点に相対的な位置ではなく、フィールドの始点に相対的な位置に基づいて文字検索を行うスクリプトを作成する場合は、これを念頭に入れる必要があります。たとえば、**lv_attr** フィールドの 9 番目のビットの文字 **p** を検索するには、文字列 "^/.....p/" を検索することはできますが、文字列 "/*p$/" を検索することはできません。

pvs コマンド

表4.2「pvs 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、およびフィールドの説明を一覧にまとめています。

表4.2 pvs 表示フィールド

引数	ヘッダー	説明
dev_size	DevSize	物理ボリュームを作成する配下のデバイスのサイズ
pe_start	1st PE	配下のデバイス内の最初の物理エクステントの開始点までのオフセット
pv_attr	Attr	物理ボリュームのステータス: (a)llocatable または e(x)ported
pv_fmt	Fmt	物理ボリュームのメタデータ形式 (**lvm2** または **lvm1**)
pv_free	PFree	物理ボリュームにある残りの空き領域
pv_name	PV	物理ボリュームの名前
pv_pe_alloc_count	Alloc	使用されている物理エクステントの数
pv_pe_count	PE	物理エクステントの数
pvseg_size	SSize	物理ボリュームのセグメントサイズ
pvseg_start	Start	物理ボリュームセグメントの最初の物理エクステント
pv_size	PSize	物理ボリュームのサイズ
pv_tags	PV Tags	物理ボリュームに割り当てられた LVM タグ
pv_used	Used	物理ボリューム上で現在使用中の領域の量
pv_uuid	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-XO8M-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トの集合です。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 物理ボリュームとして初期化されていなかった LVM が検出した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4.3「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4.3 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)
pv_count	#PV	ボリュームグループを定義する物理ボリューム数
snap_count	#SN	ボリュームグループに含まれるスナップショット数
vg_attr	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
vg_extent_count	#Ext	ボリュームグループ内の物理エクステントの数
vg_extent_size	Ext	ボリュームグループ内の物理エクステントのサイズ
vg_fmt	Fmt	ボリュームグループのメタデータ形式 (**lvm2** または **lvm1**)
vg_free	VFree	ボリュームグループ内の残りの空き領域のサイズ
vg_free_count	Free	ボリュームグループ内の空き物理エクステントの数
vg_name	VG	ボリュームグループ名
vg_seqno	Seq	ボリュームグループの改訂を示す番号
vg_size	VSize	ボリュームグループのサイズ
vg_sysid	SYS ID	LVM1 システム ID
vg_tags	VG Tags	ボリュームグループに割り当てられた LVM タグ
vg_uuid	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-OpMO-0118-
nlwO-wwqd-fD5D32
```

lvs コマンド

表4.4「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4.4 lvs 表示フィールド

引数	ヘッダー	説明
chunksize **chunk_size**	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 **pv_move** 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_attr	Attr	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のようになります。

引数	ヘッダー	説明
		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 (I)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)
		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)
		ビット 3: 割り当てポリシー: (a)nywhere、(c)ontiguous、(i)nherited、c(l)ing、(n)ormal。これは、たとえば **pvmove** 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。
		ビット 4: 固定されたマイナー番号
		ビット 5: 状態: (a)ctive (アクティブ)、(s)uspended (サスペンド)、(I)nvalid 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)
		ビット 6: デバイス開放(o)
		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ーを使用する場合、それらは (m) と表示されます。md raid カーネルドライバーを使用する同等の 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ー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ー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。
		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。

引数	ヘッダー	説明
		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。デバイスは更新するか、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 **check** 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 **check** および **repair** は、**lvchange** 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。 ビット 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。
lv_kernel_major	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)
lv_kernel_minor	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)
lv_major	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)
lv_minor	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)
lv_name	LV	論理ボリュームの名前
lv_size	LSize	論理ボリュームのサイズ
lv_tags	LV Tags	論理ボリュームに割り当てられた LVM タグ
lv_uuid	LV UUID	論理ボリュームの UUID
mirror_log	Log	ミラーログが存在するデバイス
modules	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
move_pv	Move	**pvmove** 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

引数	ヘッダー	説明
origin	Origin	スナップショットボリュームの複製元のデバイス
regionsize **region_size**	Region	ミラー化論理ボリュームのユニットサイズ
seg_count	#Seg	論理ボリューム内のセグメント数
seg_size	SSize	論理ボリューム内のセグメントサイズ
seg_start	Start	論理ボリューム内のセグメントのオフセット
seg_tags	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
segtype	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)
snap_percent	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
stripes	#Str	論理ボリューム内のストライプ、またはミラーの数
stripesize **stripe_size**	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
  lvol0      new_vg owi-a- 52.00M
  newvgsnap1 new_vg swi-a-  8.00M lvol0    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
  lvol0      new_vg    1 owi-a- 52.00M  -1  -1 253  3
```

```
LBy1Tz-sr23-OjsI-LT03-nHLC-y8XW-EhCl78
  newvgsnap1 new_vg    1 swi-a-  8.00M  -1  -1 253  5    lvol0    0.20
1ye1OU-1cIu-o79k-20h2-ZGF0-qCJm-CfbsIx
```

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
  LogVol00 VolGroup00 -wi-ao    1 linear  36.62G
  LogVol01 VolGroup00 -wi-ao    1 linear 512.00M
  lv       vg         -wi-a-    1 linear 104.00M
  lv       vg         -wi-a-    1 linear 104.00M
  lv       vg         -wi-a-    1 linear 104.00M
  lv       vg         -wi-a-    1 linear  88.00M
```

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 **-** 文字を付けます。

```
# 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
```

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 つのディスク上に LVM 論理ボリュームを作成する

この例では、**/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 LVM 論理ボリュームを作成します。

5.1.1. 物理ボリュームの作成

ボリュームグループ内のディスクを使用するには、それらに LVM 物理ボリュームというラベルを付けます。

警告

このコマンドは、**/dev/sda1**、**/dev/sdb1**、および **/dev/sdc1** 上の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
```

5.1.2. ボリュームグループの作成

以下のコマンドはボリュームグループ **new_vol_group** を作成します。

```
# 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、新規ボリュームグループの属性を表示することができます。

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

5.1.3. 論理ボリュームの作成

以下のコマンドは、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 2GB を使用する論理ボリュームを作成しています。

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

5.1.4. ファイルシステムの作成

以下のコマンドは論理ボリューム上に GFS2 ファイルシステムを作成します。

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

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

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

Syncing...
All Done
```

以下のコマンドは、論理ボリュームをマウントして、ファイルシステムのディスク領域使用率を報告します。

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

5.2. ストライプ化論理ボリュームの作成

この例では、**/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 LVM ストライプ化論理ボリュームを作成します。

5.2.1. 物理ボリュームの作成

ボリュームグループ内で使用するディスクに LVM 物理ボリュームというラベルを付けます。

警告

このコマンドは、**/dev/sda1**、**/dev/sdb1**、および **/dev/sdc1** 上の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
```

5.2.2. ボリュームグループの作成

以下のコマンドはボリュームグループ **volgroup01** を作成します。

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

vgs コマンドを使用すると、新規ボリュームグループの属性を表示することができます。

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

5.2.3. 論理ボリュームの作成

以下のコマンドは、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** を作成します。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を 3 つを持つ論理ボリュームを作成します。

```
# lvcreate -i3 -I4 -L2G -nstriped_logical_volume volgroup01
  Rounding size (512 extents) up to stripe boundary size (513 extents)
  Logical volume "striped_logical_volume" created
```

5.2.4. ファイルシステムの作成

以下のコマンドは論理ボリューム上に GFS2 ファイルシステムを作成します。

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

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

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

Syncing...
All Done
```

以下のコマンドは、論理ボリュームをマウントして、ファイルシステムのディスク領域使用率を報告します。

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

```
tmpfs                    127880         0    127880   0% /dev/shm
/dev/volgroup01/striped_logical_volume
                        1969936        20   1969916   1% /mnt
```

5.3. ボリュームグループの分割

この例では、既存ボリュームグループは 3 つの物理ボリュームから構成されています。これらの物理ボリュームに十分な未使用領域があれば、新規のディスクを追加せずに新規のボリュームグループを作成することができます。

最初のセットアップでは、論理ボリューム **mylv** は、ボリュームグループ **myvol** から作成され、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。

この手順の完了後、ボリュームグループ **myvg** は **/dev/sda1** と **/dev/sdb1** で構成されます。2 つ目のボリュームグループ **yourvg** は **/dev/sdc1** で構成されます。

5.3.1. 空き領域の判別

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判別することができます。

```
# 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   ]
```

5.3.2. データの移動

pvmove コマンドを使用して、**/dev/sdc1** 内の すべての使用中の物理エクステントを **/dev/sdb1** に移動することができます。**pvmove** コマンドは実行するのに長い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%
```

データを移動した後は、**/dev/sdc1** 上のすべての領域が空き領域になっていることを確認できます。

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

5.3.3. ボリュームグループの分割

新規ボリュームグループ **yourvg** を作成するには、**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前に、論理ボリュームは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しなければなりません。

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。以下のコマンドは論理ボリューム **mylv** を非アクティブ化して、ボリュームグループ **myvg** からボリュームグループ **yourvg** を分割し、物理ボリューム **/dev/sdc1** を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
```

5.3.4. 新規論理ボリュームの作成

新規のボリュームグループを作成した後は、新規の論理ボリューム **yourlv** を作成することができます。

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

5.3.5. ファイルシステムの作成と新規論理ボリュームのマウント

新規の論理ボリューム上にファイルシステムを作成してから、その論理ボリュームをマウントすることができます。

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

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

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

Syncing...
All Done

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

5.3.6. 元の論理ボリュームのアクティブ化とマウント

論理ボリューム **mylv** を非アクティブ化する必要があったため、これをマウントできるようにするには、その前にこの論理ボリュームを再度アクティブ化する必要があります。

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

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

5.4. 論理ボリュームからディスクを削除する

この例は、ディスクを置き換えるか、またはディスクを別のボリュームの一部として使用するために、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には、まず LVM 物理ボリューム上のエクステントを異なるディスクまたはディスクセットに移動しなければなりません。

5.4.1. 既存物理ボリュームへのエクステントの移動

この例では、論理ボリュームは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
```

/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。

ボリュームグループ内の他の物理ボリューム上に十分な空きエクステントがある場合、他のオプションを指定せずに削除したいデバイスに対して **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した後は、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
```

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

これでディスクは物理的に削除可能となり、他のユーザーへの割り当ても可能になります。

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、論理ボリュームは、以下のようにボリュームグループ **myvg** 内の 3 つの物理ボリュームに分配されます。

```
# 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
```

/dev/sdb1 のエクステントを新しいデバイス **/dev/sdd1** に移動してみましょう。

5.4.2.1. 新規物理ボリュームの作成

/dev/sdd1 から新規の物理ボリュームを作成します。

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

5.4.2.2. ボリュームグループへの新規物理ボリュームの追加

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

5.4.2.3. データの移動

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%

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

5.4.2.4. 古い物理ボリュームをボリュームグループから削除する

データを **/dev/sdb1** から移動した後に、この物理ボリュームをボリュームグループから削除することができます。

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

これで、このディスクの別のボリュームグループへの再割り当てや、システムからの削除が可能になります。

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

クラスター内にミラー化 LVM 論理ボリュームを作成するには、単一ノード上にミラー化 LVM 論理ボリュームを作成するのと同一の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働中であり、クラスターが定足数に達しており、かつクラスターのロッキングを有効にするように **lvm.conf** ファイル内の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。これは、「クラスター内での LVM ボリューム作成」 で説明しているように、直接または **lvmconf** コマンドを使用して実行します。

クラスター内でミラー化された LVM ボリュームを作成するには、以下の手順で行います。まず、この手順ではクラスターサービスがインストール済みで実行されているかどうかを確認し、次にミラー化ボリュームを作成します。

1. クラスター内のすべてのノードが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内の **lvm.conf** ファイルにロッキングタイプを正しく設定されている必要があります。デフォルトでは、ロッキングタイプは、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、クラスターロッキングを有効にします。

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

2. クラスター論理ボリュームを作成するには、クラスター内のすべてのノード上でクラスターインフラストラクチャーが稼動中である必要があります。以下の例では **clvmd** デーモンが発行されたノード上で稼働中であることを検証します。

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

以下のコマンドは、クラスター状態のローカルビューを表示します。

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

dlm lockspaces
name          clvmd
id            0x4104eefa
flags         0x00000000
change        member 3 joined 1 remove 0 failed 0 seq 1,1
members       1 2 3
```

3. **cmirror** パッケージがインストールされていることを確認します。

4. **cmirrord** サービスを開始します。

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

5. 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、ミラーレッグとして使用され、3 つ目の物理ボリュームにはミラーログが格納されます。

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

6. 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された 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/xvdb1 /dev/xvdb2 /dev/xvdc1
  Clustered volume group "vg001" successfully created
```

vgcreate コマンドの出力がボリュームグループがクラスター化されていることを示し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
```

7. 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ッグは 1 つです。この例では、物理ボリュームのどのエクステントが論理ボリュームに使用されるかを指定します。

```
# lvcreate -l 1000 -m1 vg001 -n mirrorlv /dev/xvdb1:1-1000
/dev/xvdb2:1-1000 /dev/xvdc1:0
  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することができます。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% 同期になる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
[root@doc-07 log]# lvs vg001/mirrorlv
  LV       VG       Attr   LSize Origin Snap%  Move Log
Copy%  Convert
  mirrorlv vg001    mwi-a- 3.91G                    vg001_mlog
91.00
[root@doc-07 ~]#  lvs vg001/mirrorlv
  LV       VG       Attr   LSize Origin Snap%  Move Log
Copy%  Convert
  mirrorlv vg001    mwi-a- 3.91G                    vg001_mlog
100.00
```

ミラーの完了は、システムログに記録されます。

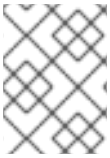
```
May 10 14:52:52 doc-07 [19402]: Monitoring mirror device vg001-
mirrorlv for events
May 10 14:55:00 doc-07 lvm[19402]: vg001-mirrorlv is now in-sync
```

8. **lvs** を **-o +devices** オプションと共に使用すると、ミラーの設定を表示することができます。これには、ミラーレッ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/xvdb1(1)
  [mirrorlv_mimage_1] vg001      iwi-ao  3.91G
/dev/xvdb2(1)
  [mirrorlv_mlog]     vg001      lwi-ao  4.00M
/dev/xvdc1(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して、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可能です。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/xvdb1:1-1000
  /dev/xvdb2:1-1000
  /dev/xvdc1:0-0
```



注記

LVM ミラー化ボリュームのいずれかのレッグに障害が発生した際の回復方法の情報は、「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

この章では、様々な LVM 問題のトラブルシューティング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後は、このパラメーターを 0 にリセットして、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。
- **lvmdump** 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 **lvmdump**(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。
- **lvm dumpconfig** 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** ディレクトリー内の **.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まだ使用不可ですが、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、障害デバイスは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、物理ボリュームの配下のデバイスが停止したために LVM ミラー化ボリュームのレッグの 1 つに障害が発生し、かつ **mirror_log_fault_policy** パラメーターが **remove** に設定されているためにミラーを手動で再構築する必要がある状態から復旧する例を説明します。**mirror_log_fault_policy** パラメーターの設定に関する情報は、「LVM ミラー障害からの回復」 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
[root@link-08 ~]# lvcreate -L 750M -n groupfs -m 1 vg /dev/sda1 /dev/sdb1
/dev/sdc1
  Rounding up size to full physical extent 752.00 MB
  Logical volume "groupfs" created
```

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)

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

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

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せずに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

[root@link-08 ~]# pvscan
  PV /dev/sdb1   VG vg   lvm2 [67.83 GB / 67.10 GB free]
  PV /dev/sdb2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdc1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdc2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdd1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdd2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sde1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sde2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdf1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdf2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdg1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdg2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdh1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdh2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdi1           lvm2 [603.94 GB]
  PV /dev/sdi2           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   ]
```

リニアボリュームが元のミラー化された状態に戻るように変換します。

```
# 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、またはシステムが特定の UUID で物理ボリュームを見つけることができないことを示すエラーメッセージが出されます。物理ボリュームのデータの復元は、紛失し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

機能している LVM 論理ボリュームについては、この手順を試みないでください。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。該当するボリュームの最後にアーカイブ化された有効な LVM メタデータについて ***VolumeGroupName_xxxx*.vg** ファイルを確認します。

別の方法としては、そのボリュームを非アクティブ化止して、**partial** (**-P**) 引数を設定することにより、見つからないかまたは破損した物理ボリュームの UUID を見つけることができます。

```
# 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新規物理ボリュームを作るように指示し、新規のメタデータが古い物理ボリュームに含まれていたデータの場所に配置されないように確認します。(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
```

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

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、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」 で説明されているような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、**pvcreate** 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論理ボリュームのすべてをボリュームグループから取り除くには **vgreduce** コマンドで **--removemissing** 引数を使用します。

vgreduce コマンドで **--test** 引数を使用することで、破棄されようとしているものを先に検証することを推奨します。

ほとんどの LVM 操作と同じく、**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ならば、ある意味で **vgreduce** コマンドは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、その物理ボリュームの入れ替えは可能であり、**vgcfgrestore** コマンドを使用してボリュームグループを直前の状態に戻すことができます。

6.7. 論理ボリュームの不十分な空きエクステント

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** や **vgs** のコマンドの出力に基づいて十分なエクステント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第 2 小数点まで四捨五入して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トのカウントを使用します。

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ト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、**vgs** 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について、小文字 l の引数を使って次のコマンドを実行し、バイトの代わりにエクステントを使用できます。

```
# lvcreate -l8780 -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
```

別の方法として、**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。

第7章 LVM GUI での LVM 管理

コマンドラインインターフェース (CLI) のほかにも、LVM はグラフィカルユーザーインターフェース (GUI) を提供しており、ユーザーはこれを使って LVM 論理ボリュームを設定することができます。このユーティリティーを立ち上げるには **system-config-lvm** を入力します。**ストレージ管理ガイド**の LVM の章では、このユーティリティーを使用した LVM 論理ボリュームの設定の手順をステップごとに説明しています。

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化されます。それぞれの論理ボリュームは、マップ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内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、および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。2 つのディスクは、各ディスクに対して 1 つのリニアマッピングが指定される一対のリニアマッピングを維持することで 1 つの論理ボリュームとして連結することができます。LVM2 がボリュームを作成する場合、**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ための **dmsetup** コマンドの使用方法についての情報は、「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。

以下のサブセクションでは次のマッピングの形式について説明します。

- linear
- striped
- mirror
- snapshot および snapshot-origin
- error
- zero
- multipath
- crypt
- device-mapper RAID
- thin
- thin-pool

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されるか、または ***major*:*minor*** 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

ミラー内のレッグ数。デバイスとオフセットが各レッグに指定されます。

device

それぞれのレッ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ッグに指定されます。

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ッ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

同期される部分を追跡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

クラスター全体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のようになります。

```
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キング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のようになります。

```
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

変更され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) には、多くのメタデータをディスク上に保存できません。メタデータはカーネルによってメモリー内に保持できます。

chunksize

COW デバイスに保存されるデータの変更したチャンクのセクターサイズ

次の例は、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にも永続し、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、マップされたセクターへの I/O 操作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、またはミラーレッグをスワップアウトして、そのレッ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には、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*** パラメーターがあります。

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。

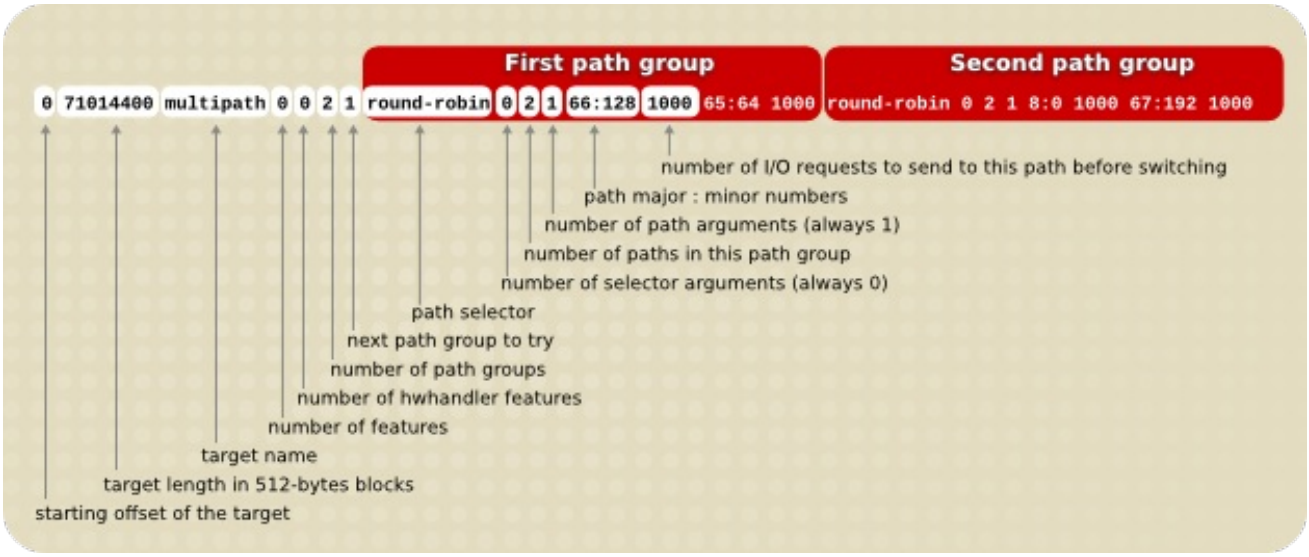
device

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、すべてのパスを含む 1 つのパスグループのみが存在します。このセットアップでは、マルチパスはすべてのパスに対して負荷を均等に分散します。

```
0 71014400 multipath 0 0 1 1 round-robin 0 4 1 66:112 1000 \
 67:176 1000 68:240 1000 65:48 1000
```

マルチパス化に関する詳細情報は、『DM マルチパスの使用 (Using Device Mapper Multipath)』 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するデータの受け渡しを暗号化します。これは、カーネル 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されるか、または ***major*:*minor*** 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.1.9. デバイスマッパー RAID マッピングターゲット

デバイスマッパー RAID (dm-raid) ターゲットは、DM から MD へのブリッジを提供します。これにより、MD RAID ドライバーには、デバイスマッパーインターフェースを使ってアクセスできます。dm-raid ターゲットの形式は以下のとおりです。

```
start length raid raid_type #raid_params raid_params #raid_devs
metadata_dev0 dev0 [.. metadata_devN devN]
```

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

raid_type

RAID タイプは以下のいずれかにすることができます。

raid1
RAID1 ミラーリング

raid4
RAID4 専用のパリティーディスク

raid5_la
RAID5 左非対称

— パリティー 0 をローテートしてデータを継続

raid5_ra
RAID5 右非対称

— パリティー N をローテートしてデータを継続

raid5_ls
RAID5 左対称

— パリティー 0 をローテートしてデータを再起動

raid5_rs
RAID5 右対称

— パリティー N をローテートしてデータを再起動

raid6_zr
RAID6 ゼロの再起動

— パリティーゼロを (左から右に) ローテートしてデータを再起動

raid6_nr
RAID6 N の再起動

— パリティー N を (左から右に) ローテートしてデータを再起動

raid6_nc
RAID6 N の継続

— パリティー N を (右から左に) ローテートしてデータを継続

raid10
追加のオプション引数で選択される各種の RAID10 型アルゴリズム

— RAID 10: ストライプ化ミラー (ミラー上部でのストライピング)

— RAID 1E: 統合された隣接のストライプ化ミラーリング

— RAID 1E: 統合されたオフセットのストライプ化ミラーリング

— 上記以外の同様の RAID10 のバリエーション

#raid_params

続くパラメーターの数

raid_params

必須パラメーター:

chunk_size

セクター単位のチャンクサイズです。このパラメーターは「ストライプサイズ」として知られることがよくあります。これは必須パラメーターのみであり、最初に配置されます。

オプションパラメーターが続きます (任意の順序):

[sync|nosync]

RAID の初期化を強制または回避します。

rebuild *idx*

ドライブ番号 ***idx*** (最初のドライブは 0) を再構築します。

daemon_sleep *ms*

ビットをクリアする bitmap デーモンの実行間の間隔です。間隔が長くなると、ビットマップ I/O が少なくなりますが、失敗後の再同期により長い時間がかかる可能性があります。

min_recovery_rate *KB/sec/disk*

RAID スロットルの初期化

max_recovery_rate *KB/sec/disk*

RAID スロットルの初期化

write_mostly *idx*

ドライブインデックスに ***idx*** write-mostly というマークを付けます。

max_write_behind *sectors*

mdadm man ページの **--write-behind** についての説明を参照してください。

stripe_cache *sectors*

ストライプキャッシュのサイズ (RAID 4/5/6 のみ)

region_size *sectors*

リージョンの数を乗じた **region_size** はアレイの論理サイズです。ビットマップは各リージョンのデバイスの同期状態を記録します。

raid10_copies *#copies*

RAID10 コピーの数です。このパラメーターは、RAID10 設定のデフォルトレイアウトを変更するために **raid10_format** パラメーターと併用されます。デフォルト値は 2 です。

raid10_format near|far|offset

このパラメーターは、RAID10 設定のデフォルトレイアウトを変更するために **raid10_copies** パラメーターと併用されます。デフォルト値は **near** で、この値は標準のミラーレイアウトを指定します。

raid10_copies と **raid10_format** が指定されないままか、または **raid10_copies 2** および/または **raid10_format near** が指定されている場合に、2、3 および 4 デバイスのレイアウトは以下のようになります。

```
2 drives    3 drives      4 drives
--------    ----------    --------------
A1  A1      A1  A1  A2    A1  A1  A2  A2
A2  A2      A2  A3  A3    A3  A3  A4  A4
A3  A3      A4  A4  A5    A5  A5  A6  A6
A4  A4      A5  A6  A6    A7  A7  A8  A8
..  ..      ..  ..  ..    ..  ..  ..  ..
```

2 デバイスレイアウトは 2 方向の RAID1 と同等です。4 デバイスレイアウトは従来の RAID10 のレイアウトのようになります。3 デバイスレイアウトは「RAID1E - 統合された隣接のストライプミラーリング」のようになります。

raid10_copies 2 および **raid10_format far** が指定される場合、2、3 および 4 デバイスのレイアウトは以下のようになります。

```
2 drives    3 drives        4 drives
--------   -----------      ------------------
A1  A2      A1  A2  A3      A1   A2   A3   A4
A3  A4      A4  A5  A6      A5   A6   A7   A8
A5  A6      A7  A8  A9      A9   A10  A11  A12
..  ..      ..  ..  ..      ..   ..   ..   ..
A2  A1      A3  A1  A2      A2   A1   A4   A3
A4  A3      A6  A4  A5      A6   A5   A8   A7
A6  A5      A9  A7  A8      A10  A9   A12  A11
..  ..      ..  ..  ..      ..   ..   ..   ..
```

raid10_copies 2 および **raid10_format offset** が指定されている場合、2、3 および 4 デバイスのレイアウトは以下のようになります。

```
2 drives    3 drives       4 drives
--------    --------       ------------------
A1  A2      A1  A2  A3     A1   A2   A3   A4
A2  A1      A3  A1  A2     A2   A1   A4   A3
A3  A4      A4  A5  A6     A5   A6   A7   A8
A4  A3      A6  A4  A5     A6   A5   A8   A7
A5  A6      A7  A8  A9     A9   A10  A11  A12
A6  A5      A9  A7  A8     A10  A9   A12  A11
..  ..      ..  ..  ..     ..   ..   ..   ..
```

これらのレイアウトは、RAID1E - 統合されたオフセットのストライプミラーリングのレイアウトに非常によく似ています。

#raid_devs

アレイを構成するデバイスの数

それぞれのデバイスは 2 つのエントリーから構成されます。最初のデバイスは、メタデータ (ある場合) を含むデバイスで、2 つ目のデバイスは、データを含むデバイスです。

ドライブが失敗するか、または作成時にない場合、'-' を所定位置のメタデータとデータドライバーの両方に指定できます。

以下の例は、始点ブロックが 0 でセグメントの長さが 1960893648 の RAID4 ターゲットを示しています。4 データドライブ、1 パリティーがあり、スーパーブロック/ビットマップ情報を保持するためにメタデータデバイスは指定されておらず、チャンクサイズは 1MiB になっています。

```
0 1960893648 raid raid4 1 2048 5 - 8:17 - 8:33 - 8:49 - 8:65 - 8:81
```

以下の例は、始点ブロックが 0 でセグメントの長さが 1960893648 の RAID4 ターゲットを示しています。4 データドライバー、1 パリティーがあり、メタデータデバイスの指定あるほか、チャンクサイズは 1MiB、RAID 初期化の強制、および **min_recovery** レートは 20 kiB/sec/disks に指定されています。

```
0 1960893648 raid raid4 4 2048 sync min_recovery_rate 20 5 8:17 8:18 8:33
8:34 8:49 8:50 8:65 8:66 8:81 8:82
```

A.1.10. シンマッピングターゲットおよびシンプールマッピングターゲット

シンプールターゲットの形式は次のようになります。

```
start length thin-pool metadata_dev data_dev data_block_size
low_water_mark [#feature_args [arg*] ]
```

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

metadata_dev

メタデータデバイス

data_dev

データデバイス

data_block_size

データブロックサイズ (セクター単位)。データブロックサイズは、512 バイトセクターで表される単位で割り当てられるディスク領域の最も小さな単位を提供します。データブロックサイズは 64KB (128 セクター) から 1GB (2097152 セクター) までの範囲になければならず、128 (64KB) の倍数でなければなりません。

low_water_mark

サイズ ***data_block_size*** のブロックで表わされる低ウォーターマークです。データデバイスの空き領域がこのレベルを下回ると、ユーザースペースデーモンが取得するデバイスマッパーイベントがトリガーされ、プールデバイスを拡張できるようになります。このイベントは 1 回のみ送信され

ます。新規テーブル自体を使ってデバイスを再開すること自体でイベントがトリガーされるので、ユーザースペースデーモンはこれを使って、新規テーブルがしきい値を超えている状況を検出することができます。

メタデータデバイス用の低ウォーターマークはカーネルに保持され、メタデータデバイスの空き容量がこのレベルを下回ると、デバイスマッパーイベントをトリガーします。

#feature_args

feature 引数の数

arg

シンプール feature 引数は以下のとおりです。

skip_block_zeroing

新規にプロビジョニングされたブロックのゼロ化をスキップします。

ignore_discard

破棄サポートを無効にします。

no_discard_passdown

破棄内容を配下のデータデバイスに渡しません。マッピングを削除するだけです。

read_only

プールメタデータに対する変更を許可しません。

error_if_no_space

領域がない場合のキューに登録する代わりのエラー IO です。

以下の例では、仮想デバイス内の始点ブロックが 0、セグメントの長さが 1638400 のシンプールターゲットを示しています。**/dev/sdc1** は小規模なメタデータデバイスであり、**/dev/sdc2** はより大きなデータデバイスです。チャンクサイズは 64k であり、low_water_mark は 0 で、feature はありません。

```
0 16384000 thin-pool /dev/sdc1 /dev/sdc2 128 0 0
```

シンターゲットの形式は以下のようになります。

```
start length thin pool_dev dev_id [external_origin_dev]
```

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

pool_dev

シンプールデバイスです。たとえば、**/dev/mapper/my_pool** または 253:0 になります。

dev_id

アクティブ化されるデバイスの内部デバイス識別子です。

external_origin_dev

読み取り専用スナップショットの複製元として処理されるプール外のオプションのブロックデバイスです。シンターゲットのプロビジョニングされていない領域の読み取りがこのデバイスにマップされます。

以下の例は、**/dev/mapper/pool** をそのバッキングストア (シンプール) として使用する 1 GiB thinLV を示しています。ターゲットには、仮想デバイス内の始点ブロック 0 とセグメント長さ 2097152 があります。

```
0 2097152 thin /dev/mapper/pool 1
```

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、以下のサブ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

このカテゴリーで使用できる状態は **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

受信されたイベントの現在の数。**dmsetup wait *n*** コマンドを発行すると、ユーザーは 'n' 番めのイベントを待つことができ、それが受理されるまでコールをブロックします。

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、3 つのディスクにまたがるリニアデバイスには 3 つのターゲットがあります。ディスクの先頭と終点で構成され、中間を持たないリニアデバイスの場合は 2 つのターゲットになります。

UUID

デバイスの UUID

以下の例では、**dmsetup info** コマンドの部分的な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-tOcS1kqFV9drb0X1Vr8sxeYP0tqcrpdegyqj5lZxe45JMGlmvtqLmbLpBcenh2L3
```

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*** コマンドを使用して一覧表示できます。 **dmsetup ls** の他の操作については、**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* コマンドは指定されたデバイス内の各ターゲットについての状態情報を提供します。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になります。**dmsetup status --target *target_type*** コマンドを使用する

と、1 つの指定されたタイプの 1 つ以上のターゲットを持つデバイスの状態のみを一覧表示することができます。

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によって参照されるデバイス用の (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デバイスマッパーデバイスすべて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. デバイスマッパーによる UDEV デバイスマネージャーのサポート

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、ユーザースペースにおける **udev** ルールを適用することによって指示されます。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** ディレクトリー内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によって指定される任意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使用することが可能です。

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

Red Hat Enterprise Linux 6 では、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、ユーザーが前回の変更イベントの結果として **udev** ルールで処理された引き続きオープンなデバイスをユーザーが削除しようとすると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 6 リリースは、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ように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
10-dm.rules	基本的/一般的なデバイスマッパールールを格納し、**/dev/mapper** 内に **/dev/dm-*N*** 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。 (**/dev/dm-*N*** はノードです) 注意: **/dev/dm-*N*** ノードは、デバイスにアクセスするスクリプトには **決して** 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、デバイスがアクティブ化される順序によって変化するためです。したがって、**/dev/mapper** ディレクトリー内の実際の名前を使用する必要があります。このレイアウトは、ノード/シンボリックリンクを作成する方法についての **udev** の要件をサポートします。

ファイル名	説明
11-dm-lvm.rules	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、**/dev/*vgname*** ディレクトリーに、**/dev/dm-*N*** をターゲットとして作成されます。 注意: デバイスマッパーサブシステムの今後のすべてのルールの命名基準に一致させるには、udev ルールは、**11-dm-*subsystem_name*.rules** の形式に従う必要があります。**udev** ルールも提供する **libdevmapper** ユーザーはいずれも、この基準に従う必要があります。
13-dm-disk.rules	すべてのデバイスマッパーデバイス用に適用されるルールを格納し、**/dev/disk/by-id**、**/dev/disk/by-uuid**、および **/dev/disk/by-uuid** ディレクトリー内にシンボリックリンクを作成します。
95-dm-notify.rules	**libdevmapper** 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 **dmsetup** と同様)。これまでのすべてのルールが適用された後に通知が行われ、**udev** 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。
69-dm-lvm-metad.rules	システム内の新たに表示されるブロックデバイス上での 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、**lvmetad** デーモンをサポートし、**lvm.conf** ファイル内の **use_lvmetad=1** で設定されます。**lvmeetad** デーモンと自動アクティブ化はクラスター環境ではサポートされません。

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
dmsetup udevcomplete	udev がルールの処理を完了し、待機中のプロセスをロック解除したことを通知するために使用されます (**95-dm-notify.rules** 内の **udev** ルールの中から呼び出されます)。
dmsetup udevcomplete_all	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。
dmsetup udevcookies	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。
dmsetup udevcreatecookie	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。
dmsetup udevreleasecookie	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 **udev** 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。**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す。

```
export DM_UDEV_COOKIE=$(dmsetup udevcreatecookie)
  dmsetup command ...
  dmsetup command ...
  ...
  dmsetup command ...
```

--noudevrules

udev ルールを無効にします。Nodes/symlinks は、**libdevmapper** 自体によって作成されます (旧式の方法)。このオプションは、**udev** が適正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 とそのオプション関する情報は、**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に関する詳細情報は、**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、**lvm dumpconfig** 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 プロファイル

LVM プロファイルは、特定の環境に実装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。LVM プロファイルの設定は、既存の設定を上書きするために使用できます。LVM プロファイルの情報は、「LVM プロファイル」 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. LVM DUMPCONFIG コマンド

lvm コマンドの **dumpconfig** オプションを使用して、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。**lvm dumpconfig** コマンド は、以下を含む各種の機能を提供します。

- 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、すべてのプロファイル可能な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の情報は、「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詳細一覧と **lvm dumconfig** オプションの指定方法についての情報は、**lvm-dumpconfig** 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され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、**lvm dumpconfig** 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。

```
lvm dumpconfig --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。

```
lvm dumpconfig --file ProfileName.profile --type profilable-metadata
section
```

セクションが指定されない場合、すべてのプロファイル可能な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。ご使用の設定ファイルとは若干異なる可能性があります。

```
# This is an example configuration file for the LVM2 system.
# It contains the default settings that would be used if there was no
# /etc/lvm/lvm.conf file.
#
# Refer to 'man lvm.conf' for further information including the file
layout.
#
# To put this file in a different directory and override /etc/lvm set
# the environment variable LVM_SYSTEM_DIR before running the tools.
#
# N.B. Take care that each setting only appears once if uncommenting
# example settings in this file.

# This section allows you to set the way the configuration settings are
handled.
config {
```

```
    # If enabled, any LVM2 configuration mismatch is reported.
    # This implies checking that the configuration key is understood
    # by LVM2 and that the value of the key is of a proper type.
    # If disabled, any configuration mismatch is ignored and default
    # value is used instead without any warning (a message about the
    # configuration key not being found is issued in verbose mode only).
    checks = 1

    # If enabled, any configuration mismatch aborts the LVM2 process.
    abort_on_errors = 0

    # Directory where LVM looks for configuration profiles.
    profile_dir = "/etc/lvm/profile"
}

# This section allows you to configure which block devices should
# be used by the LVM system.
devices {

    # Where do you want your volume groups to appear ?
    dir = "/dev"

    # An array of directories that contain the device nodes you wish
    # to use with LVM2.
    scan = [ "/dev" ]

    # If set, the cache of block device nodes with all associated symlinks
    # will be constructed out of the existing udev database content.
    # This avoids using and opening any inapplicable non-block devices or
    # subdirectories found in the device directory. This setting is
applied
    # to udev-managed device directory only, other directories will be
scanned
    # fully. LVM2 needs to be compiled with udev support for this setting
to
    # take effect. N.B. Any device node or symlink not managed by udev in
    # udev directory will be ignored with this setting on.
    obtain_device_list_from_udev = 1

    # If several entries in the scanned directories correspond to the
    # same block device and the tools need to display a name for device,
    # all the pathnames are matched against each item in the following
    # list of regular expressions in turn and the first match is used.
    # preferred_names = [ ]

    # Try to avoid using undescriptive /dev/dm-N names, if present.
    preferred_names = [ "^/dev/mpath/", "^/dev/mapper/mpath",
"^/dev/[hs]d" ]

    # A filter that tells LVM2 to only use a restricted set of devices.
    # The filter consists of an array of regular expressions.  These
    # expressions can be delimited by a character of your choice, and
    # prefixed with either an 'a' (for accept) or 'r' (for reject).
    # The first expression found to match a device name determines if
    # the device will be accepted or rejected (ignored).  Devices that
    # don't match any patterns are accepted.
```

```
    # Be careful if there there are symbolic links or multiple filesystem
    # entries for the same device as each name is checked separately
against
    # the list of patterns.  The effect is that if the first pattern in
the
    # list to match a name is an 'a' pattern for any of the names, the
device
    # is accepted; otherwise if the first pattern in the list to match a
name
    # is an 'r' pattern for any of the names it is rejected; otherwise it
is
    # accepted.

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

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


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

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

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

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

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

    # Since "filter" is often overridden from command line, it is not
suitable
    # for system-wide device filtering (udev rules, lvmetad). To hide
devices
    # from LVM-specific udev processing and/or from lvmetad, you need to
set
    # global_filter. The syntax is the same as for normal "filter"
    # above. Devices that fail the global_filter are not even opened by
LVM.

    # global_filter = []

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

```
    # these new ones is present.)
    # N.B. If obtain_device_list_from_udev is set to 1 the list of
    # devices is instead obtained from udev and any existing .cache
    # file is removed.
    cache_dir = "/etc/lvm/cache"
    cache_file_prefix = ""

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

    # Advanced settings.

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

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

    # By default, LVM2 will ignore devices used as component paths
    # of device-mapper multipath devices.
    # 1 enables; 0 disables.
    multipath_component_detection = 1

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

    # By default, if a PV is placed directly upon an md device, LVM2
    # will align its data blocks with the md device's stripe-width.
    # 1 enables; 0 disables.
    md_chunk_alignment = 1

    # Default alignment of the start of a data area in MB.  If set to 0,
    # a value of 64KB will be used.  Set to 1 for 1MiB, 2 for 2MiB, etc.
    # default_data_alignment = 1

    # By default, the start of a PV's data area will be a multiple of
    # the 'minimum_io_size' or 'optimal_io_size' exposed in sysfs.
    # - minimum_io_size - the smallest request the device can perform
    #   w/o incurring a read-modify-write penalty (e.g. MD's chunk size)
    # - optimal_io_size - the device's preferred unit of receiving I/O
    #   (e.g. MD's stripe width)
    # minimum_io_size is used if optimal_io_size is undefined (0).
    # If md_chunk_alignment is enabled, that detects the optimal_io_size.
    # This setting takes precedence over md_chunk_alignment.
    # 1 enables; 0 disables.
    data_alignment_detection = 1

    # Alignment (in KB) of start of data area when creating a new PV.
    # md_chunk_alignment and data_alignment_detection are disabled if set.
    # Set to 0 for the default alignment (see: data_alignment_default)
    # or page size, if larger.
```

```
    data_alignment = 0

    # By default, the start of the PV's aligned data area will be shifted
by
    # the 'alignment_offset' exposed in sysfs.  This offset is often 0 but
    # may be non-zero; e.g.: certain 4KB sector drives that compensate for
    # windows partitioning will have an alignment_offset of 3584 bytes
    # (sector 7 is the lowest aligned logical block, the 4KB sectors start
    # at LBA -1, and consequently sector 63 is aligned on a 4KB boundary).
    # But note that pvcreate --dataalignmentoffset will skip this
detection.
    # 1 enables; 0 disables.
    data_alignment_offset_detection = 1

    # If, while scanning the system for PVs, LVM2 encounters a device-
mapper
    # device that has its I/O suspended, it waits for it to become
accessible.
    # Set this to 1 to skip such devices.  This should only be needed
    # in recovery situations.
    ignore_suspended_devices = 0

    # During each LVM operation errors received from each device are
counted.
    # If the counter of a particular device exceeds the limit set here, no
    # further I/O is sent to that device for the remainder of the
respective
    # operation. Setting the parameter to 0 disables the counters
altogether.
    disable_after_error_count = 0

    # Allow use of pvcreate --uuid without requiring --restorefile.
    require_restorefile_with_uuid = 1

    # Minimum size (in KB) of block devices which can be used as PVs.
    # In a clustered environment all nodes must use the same value.
    # Any value smaller than 512KB is ignored.

    # Ignore devices smaller than 2MB such as floppy drives.
    pv_min_size = 2048

    # The original built-in setting was 512 up to and including version
2.02.84.
    # pv_min_size = 512

    # Issue discards to a logical volumes's underlying physical volume(s)
when
    # the logical volume is no longer using the physical volumes' space
(e.g.
    # lvremove, lvreduce, etc).  Discards inform the storage that a region
is
    # no longer in use.  Storage that supports discards advertise the
protocol
    # specific way discards should be issued by the kernel (TRIM, UNMAP,
or
    # WRITE SAME with UNMAP bit set).  Not all storage will support or
```

```
benefit
    # from discards but SSDs and thinly provisioned LUNs generally do.  If
set
    # to 1, discards will only be issued if both the storage and kernel
provide
    # support.
    # 1 enables; 0 disables.
    issue_discards = 0
}

# This section allows you to configure the way in which LVM selects
# free space for its Logical Volumes.
allocation {

    # When searching for free space to extend an LV, the "cling"
    # allocation policy will choose space on the same PVs as the last
    # segment of the existing LV.  If there is insufficient space and a
    # list of tags is defined here, it will check whether any of them are
    # attached to the PVs concerned and then seek to match those PV tags
    # between existing extents and new extents.
    # Use the special tag "@*" as a wildcard to match any PV tag.

    # Example: LVs are mirrored between two sites within a single VG.
    # PVs are tagged with either @site1 or @site2 to indicate where
    # they are situated.

    # cling_tag_list = [ "@site1", "@site2" ]
    # cling_tag_list = [ "@*" ]

    # Changes made in version 2.02.85 extended the reach of the 'cling'
    # policies to detect more situations where data can be grouped
    # onto the same disks.  Set this to 0 to revert to the previous
    # algorithm.
    maximise_cling = 1

    # Set to 1 to guarantee that mirror logs will always be placed on
    # different PVs from the mirror images.  This was the default
    # until version 2.02.85.
    mirror_logs_require_separate_pvs = 0

    # Set to 1 to guarantee that thin pool metadata will always
    # be placed on different PVs from the pool data.
    thin_pool_metadata_require_separate_pvs = 0

    # Specify the minimal chunk size (in KB) for thin pool volumes.
    # Use of the larger chunk size may improve perfomance for plain
    # thin volumes, however using them for snapshot volumes is less
efficient,
    # as it consumes more space and takes extra time for copying.
    # When unset, lvm tries to estimate chunk size starting from 64KB
    # Supported values are in range from 64 to 1048576.
    # thin_pool_chunk_size = 64

    # Specify discards behavior of the thin pool volume.
    # Select one of  "ignore", "nopassdown", "passdown"
    # thin_pool_discards = "passdown"
```

```
    # Set to 0, to disable zeroing of thin pool data chunks before their
    # first use.
    # N.B. zeroing larger thin pool chunk size degrades performance.
    # thin_pool_zero = 1
}

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

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

    # Set to 1 to suppress all non-essential messages from stdout.
    # This has the same effect as -qq.
    # When this is set, the following commands still produce output:
    # dumpconfig, lvdisplay, 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

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

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

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

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

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

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

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

```
severity
    # of each message.
    prefix = "  "

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

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

    # Some debugging messages are assigned to a class and only appear
    # in debug output if the class is listed here.
    # Classes currently available:
    #   memory, devices, activation, allocation, lvmetad, metadata, cache,
    #   locking
    # Use "all" to see everything.
    debug_classes = [ "memory", "devices", "activation", "allocation",
        "lvmetad", "metadata", "cache", "locking" ]
}

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

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

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

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

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

    # What is the minimum number of archive files you wish to keep ?
    retain_min = 10

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

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

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


# Miscellaneous global LVM2 settings
global {
    # The file creation mask for any files and directories created.
    # Interpreted as octal if the first digit is zero.
    umask = 077

    # Allow other users to read the files
    #umask = 022

    # Enabling test mode means that no changes to the on disk metadata
    # will be made.  Equivalent to having the -t option on every
    # command.  Defaults to off.
    test = 0

    # Default value for --units argument
    units = "h"

    # Since version 2.02.54, the tools distinguish between powers of
    # 1024 bytes (e.g. KiB, MiB, GiB) and powers of 1000 bytes (e.g.
    # KB, MB, GB).
    # If you have scripts that depend on the old behaviour, set this to 0
    # temporarily until you update them.
    si_unit_consistency = 1

    # Whether or not to communicate with the kernel device-mapper.
    # Set to 0 if you want to use the tools to manipulate LVM metadata
    # without activating any logical volumes.
    # If the device-mapper kernel driver is not present in your kernel
    # setting this to 0 should suppress the error messages.
    activation = 1

    # If we can't communicate with device-mapper, should we try running
    # the LVM1 tools?
    # This option only applies to 2.4 kernels and is provided to help you
    # switch between device-mapper kernels and LVM1 kernels.
    # The LVM1 tools need to be installed with .lvm1 suffices
    # e.g. vgscan.lvm1 and they will stop working after you start using
    # the new lvm2 on-disk metadata format.
    # The default value is set when the tools are built.
    # fallback_to_lvm1 = 0

    # The default metadata format that commands should use - "lvm1" or
"lvm2".
    # The command line override is -M1 or -M2.
    # Defaults to "lvm2".
    # format = "lvm2"

    # Location of proc filesystem
    proc = "/proc"
```

```
    # Type of locking to use. Defaults to local file-based locking (1).
    # Turn locking off by setting to 0 (dangerous: risks metadata
corruption
    # if LVM2 commands get run concurrently).
    # Type 2 uses the external shared library locking_library.
    # Type 3 uses built-in clustered locking.
    # Type 4 uses read-only locking which forbids any operations that
might
    # change metadata.
    locking_type = 1

    # Set to 0 to fail when a lock request cannot be satisfied
immediately.
    wait_for_locks = 1

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

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

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

    # Whenever there are competing read-only and read-write access
requests for
    # a volume group's metadata, instead of always granting the read-only
    # requests immediately, delay them to allow the read-write requests to
be
    # serviced.  Without this setting, write access may be stalled by a
high
    # volume of read-only requests.
    # NB. This option only affects locking_type = 1 viz. local file-based
    # locking.
    prioritise_write_locks = 1

    # Other entries can go here to allow you to load shared libraries
    # e.g. if support for LVM1 metadata was compiled as a shared library
use
    #   format_libraries = "liblvm2format1.so"
    # Full pathnames can be given.
```

```
    # Search this directory first for shared libraries.
    #   library_dir = "/lib"

    # The external locking library to load if locking_type is set to 2.
    #   locking_library = "liblvm2clusterlock.so"

    # Treat any internal errors as fatal errors, aborting the process that
    # encountered the internal error. Please only enable for debugging.
    abort_on_internal_errors = 0

    # Check whether CRC is matching when parsed VG is used multiple times.
    # This is useful to catch unexpected internal cached volume group
    # structure modification. Please only enable for debugging.
    detect_internal_vg_cache_corruption = 0

    # If set to 1, no operations that change on-disk metadata will be
permitted.
    # Additionally, read-only commands that encounter metadata in need of
repair
    # will still be allowed to proceed exactly as if the repair had been
    # performed (except for the unchanged vg_seqno).
    # Inappropriate use could mess up your system, so seek advice first!
    metadata_read_only = 0

    # 'mirror_segtype_default' defines which segtype will be used when the
    # shorthand '-m' option is used for mirroring.  The possible options
are:
    #
    # "mirror" - The original RAID1 implementation provided by LVM2/DM.
It is
    #           characterized by a flexible log solution (core, disk,
mirrored)
    #   and by the necessity to block I/O while reconfiguring in the
    #   event of a failure.
    #
    #   There is an inherent race in the dmeventd failure handling
    #   logic with snapshots of devices using this type of RAID1 that
    #   in the worst case could cause a deadlock.
    #     Ref: https://bugzilla.redhat.com/show_bug.cgi?id=817130#c10
    #
    # "raid1"  - This implementation leverages MD's RAID1 personality
through
    #           device-mapper.  It is characterized by a lack of log
options.
    #   (A log is always allocated for every device and they are placed
    #   on the same device as the image - no separate devices are
    #   required.)  This mirror implementation does not require I/O
    #   to be blocked in the kernel in the event of a failure.
    #   This mirror implementation is not cluster-aware and cannot be
    #   used in a shared (active/active) fashion in a cluster.
    #
    # Specify the '--type <mirror|raid1>' option to override this default
    # setting.
    mirror_segtype_default = "mirror"

    # 'raid10_segtype_default' determines the segment types used by
```

```
default
    # when the '--stripes/-i' and '--mirrors/-m' arguments are both
specified
    # during the creation of a logical volume.
    # Possible settings include:
    #
    # "raid10" - This implementation leverages MD's RAID10 personality
through
    #            device-mapper.
    #
    # "mirror" - LVM will layer the 'mirror' and 'stripe' segment types.
It
    #            will do this by creating a mirror on top of striped sub-
LVs;
    #            effectively creating a RAID 0+1 array.  This is
suboptimal
    #            in terms of providing redunancy and performance.
Changing to
    #            this setting is not advised.
    # Specify the '--type <raid10|mirror>' option to override this default
    # setting.
    raid10_segtype_default = "mirror"

    # The default format for displaying LV names in lvdisplay was changed
    # in version 2.02.89 to show the LV name and path separately.
    # Previously this was always shown as /dev/vgname/lvname even when
that
    # was never a valid path in the /dev filesystem.
    # Set to 1 to reinstate the previous format.
    #
    # lvdisplay_shows_full_device_path = 0

    # Whether to use (trust) a running instance of lvmetad. If this is set
to
    # 0, all commands fall back to the usual scanning mechanisms. When set
to 1
    # *and* when lvmetad is running (it is not auto-started), the volume
group
    # metadata and PV state flags are obtained from the lvmetad instance
and no
    # scanning is done by the individual commands. In a setup with
lvmetad,
    # lvmetad udev rules *must* be set up for LVM to work correctly.
Without
    # proper udev rules, all changes in block device configuration will be
    # *ignored* until a manual 'pvscan --cache' is performed.
    #
    # If lvmetad has been running while use_lvmetad was 0, it MUST be
stopped
    # before changing use_lvmetad to 1 and started again afterwards.
    use_lvmetad = 0

    # Full path of the utility called to check that a thin metadata device
    # is in a state that allows it to be used.
    # Each time a thin pool needs to be activated or after it is
deactivated
```

```
    # this utility is executed. The activation will only proceed if the
utility
    # has an exit status of 0.
    # Set to "" to skip this check.  (Not recommended.)
    # The thin tools are available as part of the device-mapper-
persistent-data
    # package from https://github.com/jthornber/thin-provisioning-tools.
    #
    # thin_check_executable = "/usr/sbin/thin_check"

    # Array of string options passed with thin_check command. By default,
    # option "-q" is for quiet output.
    # With thin_check version 2.1 or newer you can add "--ignore-non-
fatal-errors"
    # to let it pass through ignoreable errors and fix them later.
    #
    # thin_check_options = [ "-q" ]

    # Full path of the utility called to repair a thin metadata device
    # is in a state that allows it to be used.
    # Each time a thin pool needs repair this utility is executed.
    # See thin_check_executable how to obtain binaries.
    #
    # thin_repair_executable = "/usr/sbin/thin_repair"

    # Array of extra string options passed with thin_repair command.
    # thin_repair_options = [ "" ]

    # Full path of the utility called to dump thin metadata content.
    # See thin_check_executable how to obtain binaries.
    #
    # thin_dump_executable = "/usr/sbin/thin_dump"

    # If set, given features are not used by thin driver.
    # This can be helpful not just for testing, but i.e. allows to avoid
    # using problematic implementation of some thin feature.
    # Features:
    #   block_size
    #   discards
    #   discards_non_power_2
    #   external_origin
    #   metadata_resize
    #
    # thin_disabled_features = [ "discards", "block_size" ]
}

activation {
    # Set to 1 to perform internal checks on the operations issued to
    # libdevmapper.  Useful for debugging problems with activation.
    # Some of the checks may be expensive, so it's best to use this
    # only when there seems to be a problem.
    checks = 0

    # Set to 0 to disable udev synchronisation (if compiled into the
binaries).
    # Processes will not wait for notification from udev.
```

```
    # They will continue irrespective of any possible udev processing
    # in the background.  You should only use this if udev is not running
    # or has rules that ignore the devices LVM2 creates.
    # The command line argument --nodevsync takes precedence over this
setting.
    # If set to 1 when udev is not running, and there are LVM2 processes
    # waiting for udev, run 'dmsetup udevcomplete_all' manually to wake
them up.
    udev_sync = 1

    # Set to 0 to disable the udev rules installed by LVM2 (if built with
    # --enable-udev_rules). LVM2 will then manage the /dev nodes and
symlinks
    # for active logical volumes directly itself.
    # N.B. Manual intervention may be required if this setting is changed
    # while any logical volumes are active.
    udev_rules = 1

    # Set to 1 for LVM2 to verify operations performed by udev. This turns
on
    # additional checks (and if necessary, repairs) on entries in the
device
    # directory after udev has completed processing its events.
    # Useful for diagnosing problems with LVM2/udev interactions.
    verify_udev_operations = 0

    # If set to 1 and if deactivation of an LV fails, perhaps because
    # a process run from a quick udev rule temporarily opened the device,
    # retry the operation for a few seconds before failing.
    retry_deactivation = 1

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

    # The linear target is an optimised version of the striped target
    # that only handles a single stripe.  Set this to 0 to disable this
    # optimisation and always use the striped target.
    use_linear_target = 1

    # How much stack (in KB) to reserve for use while devices suspended
    # Prior to version 2.02.89 this used to be set to 256KB
    reserved_stack = 64

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

    # Nice value used while devices suspended
    process_priority = -18

    # If volume_list is defined, each LV is only activated if there is a
    # match against the list.
```

```
    #
    #   "vgname" and "vgname/lvname" are matched exactly.
    #   "@tag" matches any tag set in the LV or VG.
    #   "@*" matches if any tag defined on the host is also set in the LV
or VG
    #
    # If any host tags exist but volume_list is not defined, a default
    # single-entry list containing "@*" is assumed.
    #
    # volume_list = [ "vg1", "vg2/lvol1", "@tag1", "@*" ]

    # If auto_activation_volume_list is defined, each LV that is to be
    # activated with the autoactivation option (--activate ay/-a ay) is
    # first checked against the list. There are two scenarios in which
    # the autoactivation option is used:
    #
    #   - automatic activation of volumes based on incoming PVs. If all
the
    #     PVs making up a VG are present in the system, the autoactivation
    #     is triggered. This requires lvmetad (global/use_lvmetad=1) and
udev
    #     to be running. In this case, "pvscan --cache -aay" is called
    #     automatically without any user intervention while processing
    #     udev events. Please, make sure you define
auto_activation_volume_list
    #     properly so only the volumes you want and expect are
autoactivated.
    #
    #   - direct activation on command line with the autoactivation
option.
    #     In this case, the user calls "vgchange --activate ay/-a ay" or
    #     "lvchange --activate ay/-a ay" directly.
    #
    # By default, the auto_activation_volume_list is not defined and all
    # volumes will be activated either automatically or by using --
activate ay/-a ay.
    #
    # N.B. The "activation/volume_list" is still honoured in all cases so
even
    # if the VG/LV passes the auto_activation_volume_list, it still needs
to
    # pass the volume_list for it to be activated in the end.

    # If auto_activation_volume_list is defined but empty, no volumes will
be
    # activated automatically and --activate ay/-a ay will do nothing.
    #
    # auto_activation_volume_list = []

    # If auto_activation_volume_list is defined and it's not empty, only
matching
    # volumes will be activated either automatically or by using --
activate ay/-a ay.
    #
    #   "vgname" and "vgname/lvname" are matched exactly.
    #   "@tag" matches any tag set in the LV or VG.
```

```
    #   "@*" matches if any tag defined on the host is also set in the LV
or VG
    #
    # auto_activation_volume_list = [ "vg1", "vg2/lvol1", "@tag1", "@*" ]

    # If read_only_volume_list is defined, each LV that is to be activated
    # is checked against the list, and if it matches, it as activated
    # in read-only mode.  (This overrides '--permission rw' stored in the
    # metadata.)
    #
    #   "vgname" and "vgname/lvname" are matched exactly.
    #   "@tag" matches any tag set in the LV or VG.
    #   "@*" matches if any tag defined on the host is also set in the LV
or VG
    #
    # read_only_volume_list = [ "vg1", "vg2/lvol1", "@tag1", "@*" ]

    # Each LV can have an 'activation skip' flag stored persistently
against it.
    # During activation, this flag is used to decide whether such an LV is
skipped.
    # The 'activation skip' flag can be set during LV creation and by
default it
    # is automatically set for thin snapshot LVs. The
'auto_set_activation_skip'
    # enables or disables this automatic setting of the flag while LVs are
created.
    # auto_set_activation_skip = 1

    # For RAID or 'mirror' segment types, 'raid_region_size' is the
    # size (in kiB) of each:
    # - synchronization operation when initializing
    # - each copy operation when performing a 'pvmove' (using 'mirror'
segtype)
    # This setting has replaced 'mirror_region_size' since version 2.02.99
    raid_region_size = 512

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

    # 'raid_fault_policy' defines how a device failure in a RAID logical
    # volume is handled.  This includes logical volumes that have the
following
    # segment types: raid1, raid4, raid5*, and raid6*.
    #
    # In the event of a failure, the following policies will determine
what
    # actions are performed during the automated response to failures
(when
    # dmeventd is monitoring the RAID logical volume) and when 'lvconvert'
is
    # called manually with the options '--repair' and '--use-policies'.
```

```
    #
    # "warn" - Use the system log to warn the user that a device in the
RAID
    #     logical volume has failed.  It is left to the user to run
    #    'lvconvert --repair' manually to remove or replace the failed
    #    device.  As long as the number of failed devices does not
    #    exceed the redundancy of the logical volume (1 device for
    #    raid4/5, 2 for raid6, etc) the logical volume will remain
    #    usable.
    #
    # "allocate" - Attempt to use any extra physical volumes in the volume
    #     group as spares and replace faulty devices.
    #
    raid_fault_policy = "warn"

    # 'mirror_image_fault_policy' and 'mirror_log_fault_policy' define
    # how a device failure affecting a mirror (of "mirror" segment type)
is
    # handled.  A mirror is composed of mirror images (copies) and a log.
    # A disk log ensures that a mirror does not need to be re-synced
    # (all copies made the same) every time a machine reboots or crashes.
    #
    # In the event of a failure, the specified policy will be used to
determine
    # what happens. This applies to automatic repairs (when the mirror is
being
    # monitored by dmeventd) and to manual lvconvert --repair when
    # --use-policies is given.
    #
    # "remove" - Simply remove the faulty device and run without it.  If
    #            the log device fails, the mirror would convert to using
    #            an in-memory log.  This means the mirror will not
    #            remember its sync status across crashes/reboots and
    #            the entire mirror will be re-synced.  If a
    #            mirror image fails, the mirror will convert to a
    #            non-mirrored device if there is only one remaining good
    #            copy.
    #
    # "allocate" - Remove the faulty device and try to allocate space on
    #            a new device to be a replacement for the failed device.
    #            Using this policy for the log is fast and maintains the
    #            ability to remember sync state through crashes/reboots.
    #            Using this policy for a mirror device is slow, as it
    #            requires the mirror to resynchronize the devices, but it
    #            will preserve the mirror characteristic of the device.
    #            This policy acts like "remove" if no suitable device and
    #            space can be allocated for the replacement.
    #
    # "allocate_anywhere" - Not yet implemented. Useful to place the log
device
    #            temporarily on same physical volume as one of the mirror
    #            images. This policy is not recommended for mirror devices
    #            since it would break the redundant nature of the mirror.
This
    #            policy acts like "remove" if no suitable device and space
can
```

```
    #            be allocated for the replacement.

    mirror_log_fault_policy = "allocate"
    mirror_image_fault_policy = "remove"

    # 'snapshot_autoextend_threshold' and 'snapshot_autoextend_percent'
define
    # how to handle automatic snapshot extension. The former defines when
the
    # snapshot should be extended: when its space usage exceeds this many
    # percent. The latter defines how much extra space should be allocated
for
    # the snapshot, in percent of its current size.
    #
    # For example, if you set snapshot_autoextend_threshold to 70 and
    # snapshot_autoextend_percent to 20, whenever a snapshot exceeds 70%
usage,
    # it will be extended by another 20%. For a 1G snapshot, using up 700M
will
    # trigger a resize to 1.2G. When the usage exceeds 840M, the snapshot
will
    # be extended to 1.44G, and so on.
    #
    # Setting snapshot_autoextend_threshold to 100 disables automatic
    # extensions. The minimum value is 50 (A setting below 50 will be
treated
    # as 50).

    snapshot_autoextend_threshold = 100
    snapshot_autoextend_percent = 20

    # 'thin_pool_autoextend_threshold' and 'thin_pool_autoextend_percent'
define
    # how to handle automatic pool extension. The former defines when the
    # pool should be extended: when its space usage exceeds this many
    # percent. The latter defines how much extra space should be allocated
for
    # the pool, in percent of its current size.
    #
    # For example, if you set thin_pool_autoextend_threshold to 70 and
    # thin_pool_autoextend_percent to 20, whenever a pool exceeds 70%
usage,
    # it will be extended by another 20%. For a 1G pool, using up 700M
will
    # trigger a resize to 1.2G. When the usage exceeds 840M, the pool will
    # be extended to 1.44G, and so on.
    #
    # Setting thin_pool_autoextend_threshold to 100 disables automatic
    # extensions. The minimum value is 50 (A setting below 50 will be
treated
    # as 50).

    thin_pool_autoextend_threshold = 100
    thin_pool_autoextend_percent = 20

    # While activating devices, I/O to devices being (re)configured is
```

```
    # suspended, and as a precaution against deadlocks, LVM2 needs to pin
    # any memory it is using so it is not paged out.  Groups of pages that
    # are known not to be accessed during activation need not be pinned
    # into memory.  Each string listed in this setting is compared against
    # each line in /proc/self/maps, and the pages corresponding to any
    # lines that match are not pinned.  On some systems locale-archive was
    # found to make up over 80% of the memory used by the process.
    # mlock_filter = [ "locale/locale-archive", "gconv/gconv-
modules.cache" ]

    # Set to 1 to revert to the default behaviour prior to version 2.02.62
    # which used mlockall() to pin the whole process's memory while
activating
    # devices.
    use_mlockall = 0

    # Monitoring is enabled by default when activating logical volumes.
    # Set to 0 to disable monitoring or use the --ignoremonitoring option.
    monitoring = 1

    # When pvmove or lvconvert must wait for the kernel to finish
    # synchronising or merging data, they check and report progress
    # at intervals of this number of seconds.  The default is 15 seconds.
    # If this is set to 0 and there is only one thing to wait for, there
    # are no progress reports, but the process is awoken immediately the
    # operation is complete.
    polling_interval = 15
}


####################
# Advanced section #
####################

# Metadata settings
#
# metadata {
    # Default number of copies of metadata to hold on each PV.  0, 1 or 2.
    # You might want to override it from the command line with 0
    # when running pvcreate on new PVs which are to be added to large VGs.

    # pvmetadatacopies = 1

    # Default number of copies of metadata to maintain for each VG.
    # If set to a non-zero value, LVM automatically chooses which of
    # the available metadata areas to use to achieve the requested
    # number of copies of the VG metadata.  If you set a value larger
    # than the the total number of metadata areas available then
    # metadata is stored in them all.
    # The default value of 0 ("unmanaged") disables this automatic
    # management and allows you to control which metadata areas
    # are used at the individual PV level using 'pvchange
    # --metadataignore y/n'.

    # vgmetadatacopies = 0
```

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

    # pvmetadatasize = 255

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

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

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

    mirror_library = "libdevmapper-event-lvm2mirror.so"

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

    snapshot_library = "libdevmapper-event-lvm2snapshot.so"

    # thin_library is the library used when monitoring a thin device.
    #
    # "libdevmapper-event-lvm2thin.so" monitors the filling of
    # pool and emits a warning through syslog when the use of
    # the pool exceeds 80%. The warning is repeated when 85%, 90% and
    # 95% of the pool is filled.

    thin_library = "libdevmapper-event-lvm2thin.so"

    # Full path of the dmeventd binary.
```

```
    #
    # executable = "/sbin/dmeventd"
}
```

付録C 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 つにグループ化するため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。スナップショットにはタグを付けることはできません。

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。

Red Hat Enterprise Linux 6.1 リリースでは、LVM タグは最長 1024 文字の文字列です (旧リリースでは、文字数制限は 128 文字でした)。LVM タグの先頭にハイフンを使用することはできません。

有効なタグは限定された範囲の文字のみで構成されます。Red Hat Enterprise Linux 6.0 リリースで使用可能な文字は [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。

ボリュームグループ内のオブジェクトのみにタグを付けられます。物理ボリュームは、ボリュームグループから削除された場合は、そのタグを失います。これは、タグがボリュームグループメタデータの一部として保存され、物理ボリュームが取り除かれる際に削除されるためです。スナップショットにタグを付けることはできません。

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
```

C.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。

Red Hat Enterprise Linux 6.1 リリースでは、**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、以下のコマンドはタグ **T9** と **T10** を削除し、タグ **T13** と **T14** をボリュームグループ **grant** に追加します。

```
vgchange --deltag T9 --deltag T10 --addtag T13 --addtag T14 grant
```

C.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することができます。**tags** セクションに **hosttags = 1** を設定した場合、ホストタグはマシンのホスト名を使用して自動的に定義されます。これにより、すべてのマシン上で複製できる共通設定ファイルを使用できるようになり、マシンがファ

イルの同一コピーを維持できますが、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エントリーは常に、**tag1** を定義して、 ホスト名が **host1** の場合は、**tag2** を定義します。

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

C.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、以下のエントリー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ー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D 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が多くの物理ボリュームを含む場合、それだけ多くのメタデータの冗長コピーを持つことは効率的ではありません。メタデータのコピーなしで物理ボリュームを作成するには、**pvcreate** コマンドで **--metadatacopies 0** オプションを使用することができます。物理ボリュームに含まれるメタデータコピーの数を一度選択すると、後で変更することはできません。0 コピーを選択すると、設定変更で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)。今後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、最低 1 つのメタデータコピーが必要になります。

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ーです。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** コマンドで **--metadatasize** オプションを使用して指定することができます。デフォルトのサイズは、数百の物理ボリュームや論理ボリュームを持つボリュームグループには小さすぎる場合があります。

D.1. 物理ボリュームラベル

デフォルトでは、**pvcreate** コマンドは 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。物理ボリュームラベルをスキャンする LVM ツールが 最初の 4 つのセクターをチェックするため、このラベルはオプションとしてそれら最初の 4 つのセクターのいずれかに配置することができます。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サイズ (バイト)
- データ領域ロケーションの NULL で終了する一覧
- メタデータ領域ロケーションの NULL で終了する一覧

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) として保存されます。ラベルには、15 ロケーション用のスペースがありますが、LVM ツールは現在 3 つしか使いません: 1 つのデータ領域と最大 2 つのメタデータ領域です。

D.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:

- メタデータが作成された方法と時期の情報
- 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:

- 名前と一意の ID

- メタデータが更新されるたびに増加するバージョン番号
- プロパティー: 読み取り/書き込み？ サイズ変更可能？
- ボリュームグループ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上の制限
- エクステントのサイズ (512 バイトとして定義されるセクターのユニットで表示)
- ボリュームグループを構成する物理ボリュームの順序が付けられていない一覧。それぞれ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
  - プロパティー: 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
- 論理ボリュームの順序が付けられていない一覧。それぞれは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順序が付けられている覧。各セグメントに対して、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の順序が付けられている一覧に適用されるマッピングがメタデータに含まれます。

D.3. サンプルのメタデータ

myvg というボリュームグループ用の LVM ボリュームグループメタデータの例を以下に示します。

```
# 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-jO6s-RAVx-UREW0G"
                        device = "/dev/sdc"     # Hint only

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

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

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

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

                        segment1 {
                                start_extent = 0
                                extent_count = 1280     # 5 Gigabytes

                                type = "striped"
                                stripe_count = 1        # linear

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

```
                                type = "striped"
                                stripe_count = 1        # linear

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

付録E 改訂履歴

改訂 7.0-11.4 **Wed May 6 2015** **Aiko Sasaki**
校閲完了

改訂 7.0-11.3 **Wed May 6 2015** **Aiko Sasaki**
翻訳ファイルを XML ソースバージョン 7.0-11 と同期

改訂 7.0-11.2 **Tue Apr 7 2015** **Aiko Sasaki**
翻訳完了

改訂 7.0-11.1 **Tue Apr 7 2015** **Aiko Sasaki**
翻訳ファイルを XML ソースバージョン 7.0-11 と同期

改訂 7.0-11 **Wed Dec 17 2014** **Steven Levine**
BZ#1175019 によりミラーサービスについての情報を修正。

改訂 7.0-10 **Tue Dec 16 2014** **Steven Levine**
RHEL 6 スプラッシュページを更新し、sort_order を実装

改訂 7.0-9 **Wed Oct 8 2014** **Steven Levine**
6.6 GA リリース向けバージョン

改訂 7.0-8 **Thu Aug 7 2014** **Steven Levine**
6.6 ベータリリース向けバージョン

改訂 7.0-7 **Mon Jul 21 2014** **Steven Levine**
6.6 向け最新ドラフト版

バグを修正: #1022850、#1071445
新規の **lvmthin**(7) man ページを反映し、シンプロビジョニングに関する文書を改良。

バグを修正: #1093227
LVM デバイスをデバイスマッパーデバイスとして直接指定し、LVM デバイスの説明を明確化。

バグを修正: #1103916
lvm tags コマンドの文書化。

バグを修正: #1093059
LVM プロファイルの文書化。

バグを修正: #1030639、#1009575
lvmetad デーモンについての文書を改良。

バグを修正: 1102840
シンプールスナップショットについての -k および -K オプションを文書化。

バグを修正: 987074
lvm コマンドについての文書を追加。

バグを修正: 969166
vgimport コマンドについての文書を更新。

改訂 6.0-14 **Wed Nov 13 2013** **Steven Levine**
6.5 GA リリース向けバージョン

改訂 6.0-13 **Thu Oct 17 2013** **Steven Levine**

外部の複製元ボリュームの文書化

改訂 6.0-11 **Thu Oct 10 2013** **Steven Levine**
バグを修正: #969166
vgimport コマンドへの更新を文書化。

改訂 6.0-10 **Fri Sep 27 2013** **Steven Levine**
6.5 ベータリリース向けバージョン

改訂 6.0-8 **Fri Sep 27 2013** **Steven Levine**
バグを修正: #889449、#973250、#997696、#997700
マイナーエラー修正のための若干の変更と明確化。

バグを修正: #1010413
lvchange コマンドの **--writemostly** および **--writebehind** オプションの文書化。

バグを修正: #986443
RAID スクラビング操作の文書化。

バグを修正: #987094
lvs コマンドへの更新を文書化。

バグを修正: #987107
lvchange コマンドに対する更新を文書化。

バグを修正: #1010411
外部ボリュームを文書化。

改訂 6.0-1 **Mon Sep 16 2013** **Steven Levine**
新規の RHEL 6.5 機能および更新についてのドラフト版。

改訂 5.0-19 **Mon Feb 18 2013** **John Ha**
GA 用に Author_Group.xml を含めてリビルド

改訂 5.0-16 **Fri Feb 15 2013** **Steven Levine**
6.4 GA リリース向けバージョン

改訂 5.0-12 **Mon Nov 27 2012** **Steven Levine**
6.4 ベータリリース向けのバージョン

改訂 5.0-10 **Wed Nov 21 2012** **Steven Levine**
複製元のボリュームとして使用されるシンスナップショットボリュームの詳細を明確化。

改訂 5.0-9 **Tue Nov 20 2012** **Steven Levine**

バグを修正: #810385
LVM のリニア論理ボリュームの定義を明確化。

バグを修正: #833491
シンプロビジョニングの機能について文書化。

バグを修正: #533057
シンスナップショットボリュームについて文書化。

バグを修正: #872778
lvchange の例の若干のエラーを修正。

バグを修正: #859561
シンスナップショット機能の詳細について明確化。

バグを修正: #846413
ボリューム作成の例の若干のエラーを修正。

バグを修正: #796978
LVM 割り当てポリシーについて文書化。

バグを修正: #853032
ミラー化ボリュームのスナップショットのサポートについて明確化。

バグを修正: #857530
raid10 のサポートについて文書化。

バグを修正: #787018
lvmetad デーモンについて文書化。

改訂 5.0-7 **Fri Nov 16 2012** **Steven Levine**
シンプロビジョニングに関する新しい章を更新および明確化。

改訂 5.0-2 **Tue Sep 25 2012** **Steven Levine**
割り当てポリシーについての新しいセクションを追加。

改訂 5.0-0 **Wed Sep 19 2012** **Steven Levine**
シンプロビジョニングされたボリュームについての初版ドラフト。

改訂 4.0-2 **Fri Jun 15 2012** **Steven Levine**
6.3 GA リリース向けのバージョン

改訂 4.0-1 **Fri Apr 13 2012** **Steven Levine**
バグを修正: #787018
スナップショット機能の autoextend について文書化。

バグを修正: #749932
lvextend の --nosync オプションについて文書化。

バグを修正: #758695
サンプルにあるプロンプトの一貫性を図る。

バグを修正: #729715
LVM RAID サポートについて文書化。

改訂 3.0-4 **Mon Nov 21 2011** **Steven Levine**

バグを修正: #755371、#755373、#755374
QE レビューについて文書化。

改訂 3.0-3 **Mon Nov 07 2011** **Steven Levine**

バグを修正: #749487
ディスク全体の pvcreate の例を明確化。

改訂 3.0-2 **Wed Oct 12 2011** **Steven Levine**

バグを修正: #744999
マイナーな誤字脱字を修正。

改訂 3.0-1 **Mon Sep 19 2011** **Steven Levine**

Red Hat Enterprise Linux 6.2 Beta リリースに向けた初回改訂

バグを修正: #730788
論理ボリュームの配下の物理ボリューム領域が使用されなくなった場合にその物理ボリューム領域に対して破棄を実行するサポートについて文書化。

バグを修正: #728361
非推奨の内容に対する古い言及を削除。

バグを修正: #714579
マイナーな誤字脱字を修正。

バグを修正: #664107
.cache ファイルの場所についての言及を修正。

改訂 2.0-1 **Thu May 19 2011** **Steven Levine**

Red Hat Enterprise Linux 6.1 の初期リリース

バグを修正: #694619
論理ボリュームを拡張する際の新たな **cling** 割り当てポリシーについて文書化。

バグを修正: #682649
クラスター化されたボリューム上での複数のミラー作成コマンドの連続した実行についての警告を追加。

バグを修正: #674100
dmsetup ls --tree コマンドの出力の例を追加。

バグを修正: #694607
単一のコマンドライン上での --addtag および --deltag の引数の複数使用のサポートを文書化。

バグを修正: #694604
タグにおける拡張文字一覧のサポートを文書化。

バグを修正: #694611
ミラー化されたストライプのサポートについて文書化。

バグを修正: #694616
ミラー化ボリュームのスナップショットのサポートを文書化。

バグを修正: #694618
排他的にアクティブ化されたクラスターボリュームのスナップショットのサポートを文書化。

バグを修正: #682648
ミラーレッグの再割り当てが行われると、ミラーログも移動する場合があることを文書化。

バグを修正: #661530
現行の機能を文書化した **cluster.conf** の例に更新。

バグを修正: #642400
クラスターログ管理が、最も低いクラスター ID のクラスターノードによって維持される点に関する注記を追加。

バグを修正: #663462
Xen 仮想マシンモニターについての古い言及を削除。

改訂 1.0-1 **Wed Nov 10 2010** **Steven Levine**

Red Hat Enterprise Linux 6 の初期リリース

索引

シンボル

M

P

R

U

V