LVM 管理者ガイド

設定と管理

5.2

ISBN: N/A
Publication date:

この文書は、クラスタ化した環境内で LVM を実行する為の情報を含んだ LVM 論理ボリューム マネージャについて説明しています。この文書の内容は、LVM2 リリースに特定してあります。

LVM 管理者ガイド: 設定と管理

製作著作 © Red Hat, Inc.

The GPG fingerprint of the security@redhat.com key is:

CA 20 86 86 2B D6 9D FC 65 F6 EC C4 21 91 80 CD DB 42 A6 0E

1801 Varsity Drive Raleigh, NC 27606-2072 USA Phone: +1 919 754 3700 Phone: 888 733 4281 Fax: +1 919 754 3701 PO Box 13588 Resea

はじめに

1. このガイドについて

この文書では、クラスター化した環境内で LVM を実行するための情報を含む、論理ボリュームマネージャ（LVM）を説明しています。この文書の内容は LVM2 リリースに特定しています。

2. 対象者

この文書は、Linux オペレーティングシステムで稼働するシステムを維持する システム管理者に使用されることを意図しています。Red Hat Enterprise Linux 5 と GFS ファイルシステム管理に熟知していることが要求されます。

3. ソフトウェアのバージョン

ソフトウェア	説明
RHEL5	RHEL5 とそれ以降を指します
GFS	RHEL5 とそれ以降用の GFS を指します

表 1. ソフトウェアのバージョン

4. 関連ドキュメント

Red Hat Enterprise Linux の使用に関する情報には、以下の資料を参照してください:

- Red Hat Enterprise Linux インストールガイド　Red Hat Enterprise Linux 5 の インストールについての情報を提供します。
- Red Hat Enterprise Linux 導入ガイド　Red Hat Enterprise Linux 5 の導入、設定、及び管理に関する情報を提供します。

Red Hat Enterprise Linux 5 の為の Red Hat クラスタセット に関する情報には、以下の資料を参照してください:

- Red Hat クラスタセット 概要　Red Hat クラスタセットのハイレベルの概要を提供します。
- Red Hat クラスタの設定と管理　Red Hat クラスタ コンポーネントのインストール、設定

、及び管理に関する情報を提供します。

- グローバルファイルシステム: 設定と管理　Red Hat GFS (Red Hat Global File System) のインストール、設定、及び維持に関する情報を提供します。
- デバイスマッパーマルチパスの使用　Red Hat Enterprise Linux 5 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ます。
- Global File System システムを持つ GNBD の使用　Red Hat GFS を持つ GNBD (Global Network Block Device) の使用についての概要を提供します。
- Linux 仮想サーバー管理　LVS (Linux Virtual Server) での ハイパフォーマンスシステムとサービスの設定に関する情報を提供します。
- Red Hat クラスタセットリリースノート　Red Hat クラスタセットの現在のリリースに関する情報を提供します。

Red Hat クラスタセット のドキュメントとその他の Red Hat ドキュメントは オンライン、http://www.redhat.com/docs/ 上と Red Hat Enterprise Linux ドキュメント CD 上で HTML、PDF、及び RPM バージョンで入手できます。

5. フィードバック

誤字/脱字を発見されたり、このドキュメントを改善する案をお持ちの場合は弊社に ご連絡下さい。その場合は、コンポーネント rh-cs に対して、Bugzilla (http://bugzilla.redhat.com/bugzilla/) 内でご報告くださるようにお願いします。

ドキュメントの識別子も忘れないように記入して下さい:

```
rh-clvm(EN)-5.1 (2007-10-30T15:15)
```

このドキュメント識別子を記入して頂くと、弊社でご使用のガイドのバージョンを 速やかに判断できます。

ドキュメントの改善案をお持ちの場合は、出来るだけ詳細に説明をお願いします。エラーを 発見された場合は、そのセクション番号と周辺の文章も含んで頂くと弊社で早く見つける ことができます。

6. ドキュメントの慣習

このドキュメント内の特定の用語は、異なるフォント、異なるスタイル、異なる太字で表現 されています。このような強調表示は、その用語が特定のカテゴリの一部であることを示します。 そのカテゴリには以下のような分類があります:

Courier フォント
: Courier フォントは コマンド、ファイル名とパス、 そして、プロンプト を表示します。

  When shown as below, it indicates computer output:

```
Desktop       about.html       logs      paulwesterberg.png
Mail          backupfiles      mail      reports
```

太字 Courier フォント
: 太字 Courier フォントは service jonas start など、ユーザーが入力すべき テキストを示します。

  root としてコマンドを実行する必要がある場合は、root プロンプト (#) が コマンドの前に付きます:

```
# gconftool-2
```

イタリック Courier フォント
: Italic Courier font represents a variable, such as an installation directory: install_dir/bin/

太字フォント
: 太字フォントは アプリケーションプログラム と グラフィカル インターフェイス上のテキスト を示します。

  このように表示がある場合: OK 、それは、グラフィカル アプリケーションインターフェイスにあるボタンを意味します。

更に、ドキュメントは異なる手法を使って、情報の重要な部分にユーザーの注意を引くようにします。 ユーザーに取って情報がどれほど重要かに応じて以下のようなマークが設定されています:

注記

注記は、システムの動作についてユーザーが理解すべき標準的な情報です。

Tip

ヒントは、一般的にタスクを実行する為の別の方法を示すものです。

重要

重要の情報は、再起動後に維持されなかった設定の変更など、必要でありながら予想されにくいことを 案内します。

注意

注意は、カーネルのリコンパイルなど、ユーザーのサポート同意に違反するような行為を示します。

警告

警告は、ハードウェアを最大パフォーマンス用にチューンしている場合のように、発生する可能性のある データロス等の危険状態を示します。

LVM 論理ボリュームマネージャ

この章では、LVM （Logical Volume Manager 論理ボリュームマネージャ）の コンポーネントについてハイレベルの概要を提供しています。

1. 論理ボリューム

ボリューム管理は物理ストレージに対して抽象的レイヤーを作成します。これが 論理ストレージボリュームの作成を可能にします。それにより直接物理ストレージを 使用することに比較して多種のより高い柔軟性を提供します。

論理ボリュームはストレージ仮想化を提供するものです。論理ボリュームがあると、 物理ディスクのサイズに制限されません。更には、ハードウェアストレージ設定は ソフトウェアには見えませんので、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。これにより運営コストを 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の直接使用に対して以下のような利点を持っています:

- 柔軟な機能

  論理ボリュームを使用している場合、ディスクやパーティションを１つの論理ボリュームに収束できるため、ファイルシステムは複数ディスクに渡って拡張が可能になります。

- サイズ変更可能なストレージプール

  背後に有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 コマンドにより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。

- オンラインデータ移動

  より新しくて、より速くて、より強靭な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時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。例えば、ホットスワップ可能なディスクを 先に空にしてからそれを削除するようにできます。

- 便利なデバイスの命名

  論理ストレージボリュームはユーザー定義のグループで管理することができて、 好みに応じて命名することができます。

- ディスクのストライプ化

  ２つ又はそれ以上のディスクに渡ってデータをストライプ化する論理ボリュームを 作成することが出来ます。これにより、スループットは劇的に向上します。

- ボリュームのミラー化

  論理ボリュームはデータのミラー化の設定に便利な手法を提供します。

- 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、安定したバックアップの為の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する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はこのドキュメントの残りの部分で説明されています。

2. LVM アーキテクチャの概要

Linux オペレーティングシステムの RHEL 4 リリース用には、オリジナルの LVM1 論理ボリューム マネージャは LVM2 に入れ替わっています。これは LVM1 に比較してより汎用のカーネルフレーム ワークを持っています。LVM2 は LVM1 に対して以下のような改良を提供します:

- 柔軟な能力
- より効率的なメタデータストレージ
- より良い復元の形式
- 新規の ASCII メタデータ形式
- メタデータのアトミック変更
- メタデータの冗長コピー

LVM2 は LVM1 に対して、スナップショットとクラスタサポート以外は下方互換性を持っています。 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については、 vgconvert(8) man ページをご覧下さい。

LVM 論理ボリュームの背後の物理ストレージユニットは、パーティションか、あるいはディスク全体の ブロックデバイスです。このデバイスは LVM physical volume (PV) (物理ボリューム) として初期化されます。

LVM 論理ボリュームを作成する為に、物理ボリュームは volume group (VG) (ボリュームグループ) に統合されます。これがディスク領域の集合体を構成し、そこから LVM logical volumes (LV) (論理ボリューム) が割り当てられます。この工程は、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にもアプリケーション (データベースなど) にも使用されます。

図 1.1. 「LVM 論理ボリュームのコンポーネント」 簡単な LVM 論理ボリュームのコンポーネントを示しています:

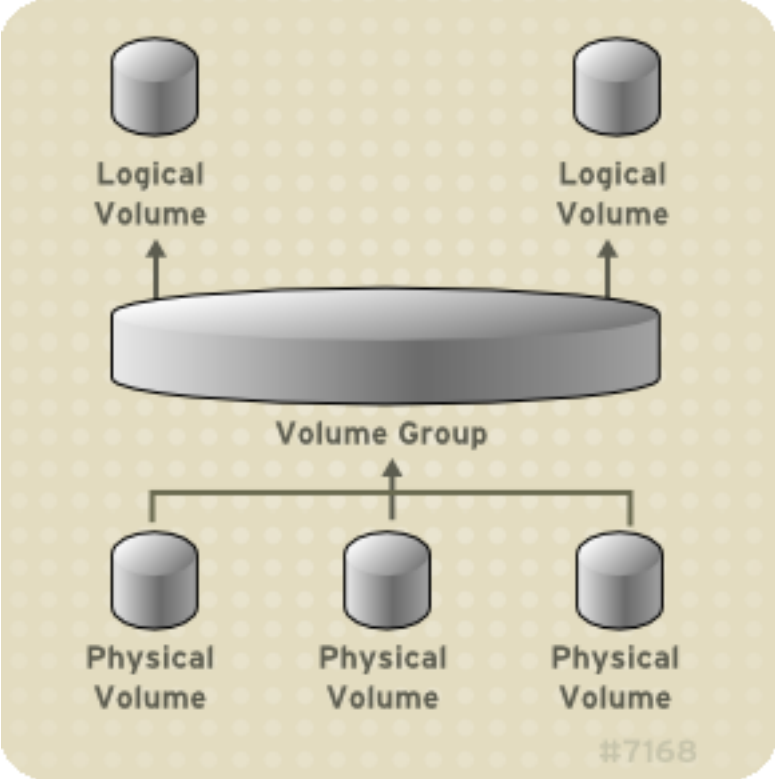

図 1.1. LVM 論理ボリュームのコンポーネント

LVM 論理ボリュームのコンポーネントに関する詳細情報については、章 2. LVM コンポーネント を ご覧下さい。

3. クラスタ内で LVM の実行

CLVM （Clustered Logical Volume Manager）は LVM のクラスタリング拡張の集合です。 これらの拡張により、コンピュータのクラスタが LVM を使用した共有ストレージ（例えば、SAN）を 管理できるようになります。

clvmd デーモン は LVM の基幹クラスタリング拡張です。 clvmd デーモン は各クラスタコンピュータ内で稼働し、 クラスタ内で LVM メタデータ更新を分配することにより、各クラスタコンピュータに論理 ボリュームの同一認識を提示します。

図 1.2. 「CLVM の概要」 Red Hat cluster 内での CLVM 概要を示しています。

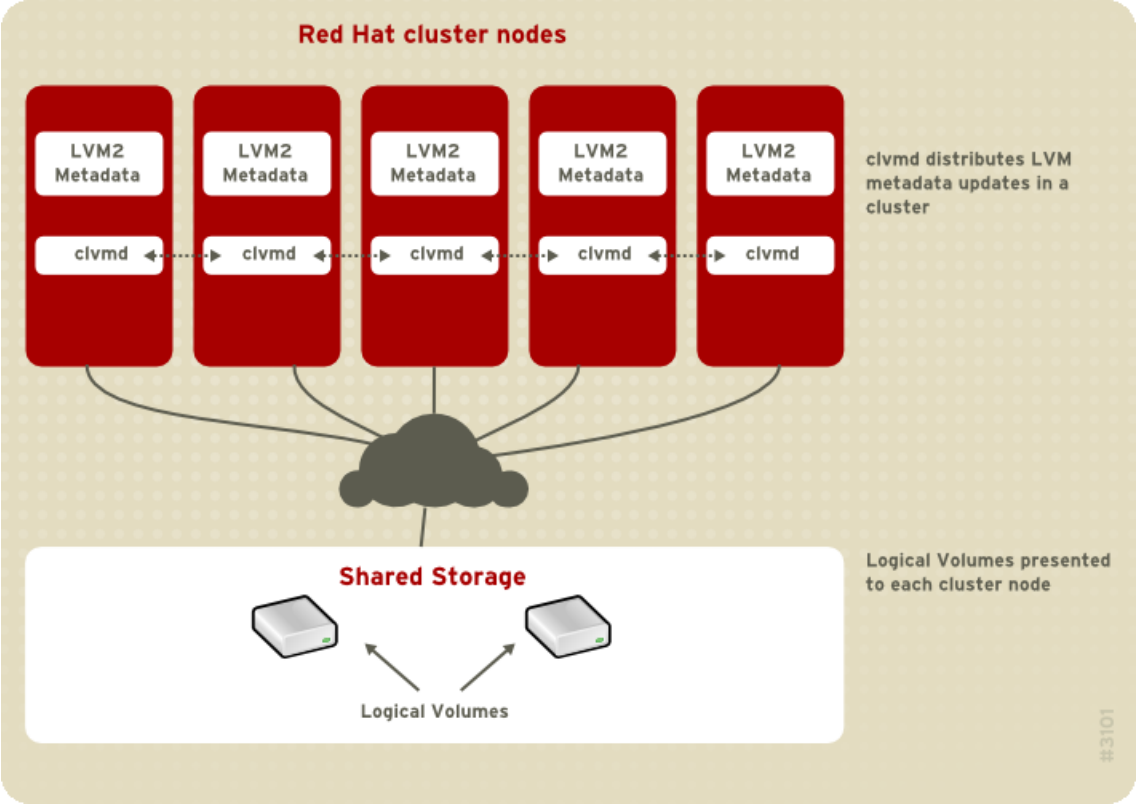

図 1.2. CLVM の概要

共有ストレージ上に CLVM で作成された論理ボリュームは、共有ストレージに アクセスできるコンピュータすべてに見えています。

CLVM の使用により、ユーザーは論理ボリュームが設定されている間に物理ストレージへの アクセスをロックすることにより、共有ストレージ上で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 はハイアベイラビリティ対称型のインフラストラクチャにより提供された ロッキングサービスを使用します。

注意

クラスタ全域のロッキングの為には CLVM は lvm.conf ファイルへの 変更を必要とします。CLVM をサポートする為の lvm.conf ファイルの設定に関する情報は、項1. 「クラスタ内で LVM ボリュームを作成」 でご覧下さい。

章 4. CLI コマンドでの LVM 管理 と 章 7. LVM GUI での LVM 管理 で説明してあるようにして、 LVM コマンドの標準的セットか、又は LVM グラフィカルユーザーインターフェイスでクラスタ内での 使用の為に LVM ボリュームを設定します。

Red Hat Cluster 内に LVM をインストールする方法については、Red Hat Cluster の設定と管理 をご覧ください。

4. ドキュメントの概要

このドキュメントの残りの部分には、以下のような章が含まれています:

- 章 2. LVM コンポーネント LVM 論理ボリュームを構成するコンポーネントを説明しています。
- 章 3. LVM 管理の概要 LVM コマンドラインインターフェイス（CLI）コマンドか、又は LVM グラフィカルユーザーインターフェイス（GUI）を使用している場合のいずれにも、LVM 論理ボリュームを設定する為に実行する基本手順の概要を提供します。
- 章 4. CLI コマンドでの LVM 管理 論理ボリュームを作成し、維持する為に LVM CLI コマンドで実行できる個別の管理タスクを 要約しています。
- 章 5. LVM 設定の例 各種 LVM 設定の例を提供します。
- 章 6. LVM トラブルシューティング 各種 LVM 問題のトラブルシューティング用の指導を提供します。
- 章 7. LVM GUI での LVM 管理 LVM GUI の運用を要約しています。
- 付録 A. デバイスマッパー 論理ボリュームと物理ボリュームをマップする為に LVM が使用するデバイスマッパーを説明します。
- 付録 B. LVM 設定ファイル LVM 設定ファイルを説明します。
- 付録 C. LVM オブジェクトタグ LVM オブジェクトタグとホストタグを説明します。
- 付録 D. LVM ボリュームグループメタデータ LVM ボリュームグループメタデータを説明して、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を含んでいます。

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ています。

1. 物理ボリューム

LVM 論理ボリュームの背後にある物理ストレージユニットは、パーティションや ディスク全体のようなブロックデバイスです。LVM 論理ボリューム用にデバイスを 使用するには、デバイスは物理ボリューム（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には、デバイスの先頭位置に ラベルを付けます。

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使用と共に共存でき ようになります。

デバイスがシステム起動時にまちまちの順序で立ち上がるために、LVM ラベルは、 物理デバイスの正しい識別とデバイス順序を提供します。LVM ラベルは再起動後にも クラスタ全域に渡って固執した状態で残ります。

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。これは、 物理ボリューム用のランダムで独特の識別子（UUID）を含んでいます。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ています。デフォルトでは、メタデータの複製コピーが、ボリュームグループ内で全ての物理ボリュームの全てのメタデータエリアで維持されています。LVM メタデータは小規模で ASCII 形式で格納されます。

現在、LVM により、各物理ボリューム上のメタデータの１つ又は２つの同一コピーの保存が 可能になっています。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 コピー数を設定してしまうと、後でその数量を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。意図したことと異なる間違えたディスクに書き込むことでディスクの先頭位置を誤って書き換えた場合でも、デバイス後部のメタデータの2つ目のコピーが メタデータの復元を可能にします。

LVM メタデータとメタデータパラメータの変更に関する詳細については、付録 D. LVM ボリュームグループメタデータ を ご覧下さい。

1.1. LVM 物理ボリュームのレイアウト

図 2.1. 「物理ボリュームレイアウト」 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とデバイスの使用可能な領域が順に続いています。

注意

Linux カーネル（及び、このドキュメント内）では、セクターは 512 バイトのサイズと 考慮されまています。

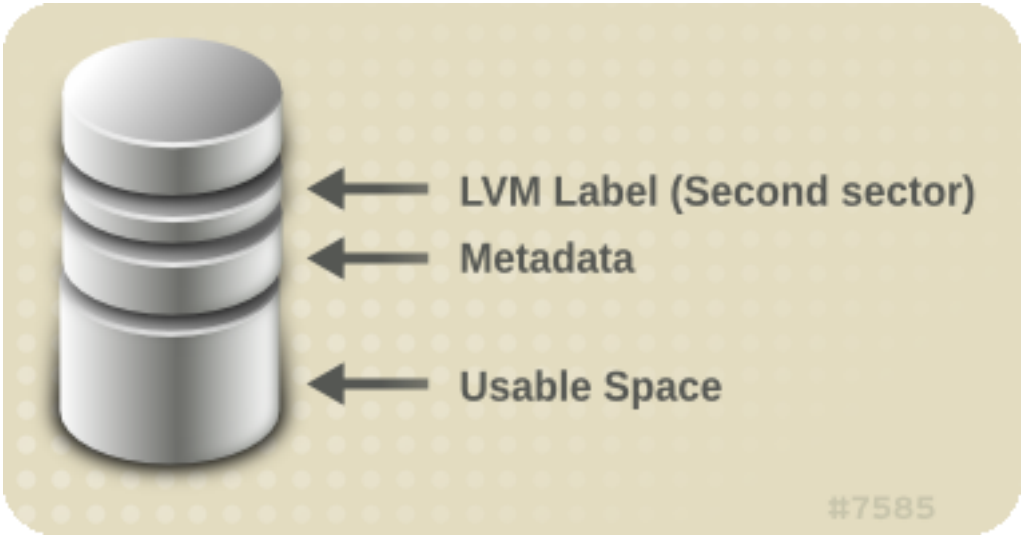

図 2.1. 物理ボリュームレイアウト

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可能になります。 その場合、通常、ディスク全体をカバーする１つのパーティションを作り、それを以下の理由の為に、 １つの LVM 物理ボリュームとしてラベルを付けることが推奨されます:

- 管理上の便宜

  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するのが簡単になります。これはディスクが故障した場合に、特に役に立ちます。更には、単独ディスク上の 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物理ディスクであることを認知できません。２つの 物理ボリュームが同一物理ディスク上にある時に、ストライプ化した論理ボリュームを 作成する

場合、ストライプは同じディスク上の異なるパーティションにあることに なります。これはパフォーマンスの向上ではなく、低下になる結果となります。

これは推奨できることではありませんが、あるディスクを別々の LVM 物理ボリュームに分割する 必要がある特定の状況が考えられます。例えば、数個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する必要が出てくるでしょう。更には、大容量のディスクが存在し、管理目的のために複数のボリュームグループを欲しい場合、そのディスクでパーティション設定する必要が出てきます。２つの パーティションを持つディスクがあって、その両方のパーティションが同じボリュームグループに 存在する場合、ストライプ化ボリュームを作成する時には論理ボリュームに含めるパーティションの 指定に注意する必要があります。

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集合体が作成されます。

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つのエクステントが割り当てできる領域の 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 呼称されます。

論理ボリュームは物理エスクテントと同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、ボリュームグループ内の全ての論理ボリュームにとってエクステント サイズは同じになります。ボリュームグループは論理エクステントを物理エクステントに マップします。

3. LVM 論理ボリューム

LVM では、ボリュームグループは論理ボリュームに分割されます。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、及び ミラー化（mirrored） ボリュームです。これらは以下のセクションで説明されています。

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を１つの論理ボリュームとして統合したものです。例えば、２つの 60GB ディスクがある場合、120GB の論理ボリュームが作成できます。 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順番に論理ボリュームの 領域に割り当てることになります。例えば、図 2.2. 「エクステントマッピング」 に示してあるように、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プして、 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プ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。

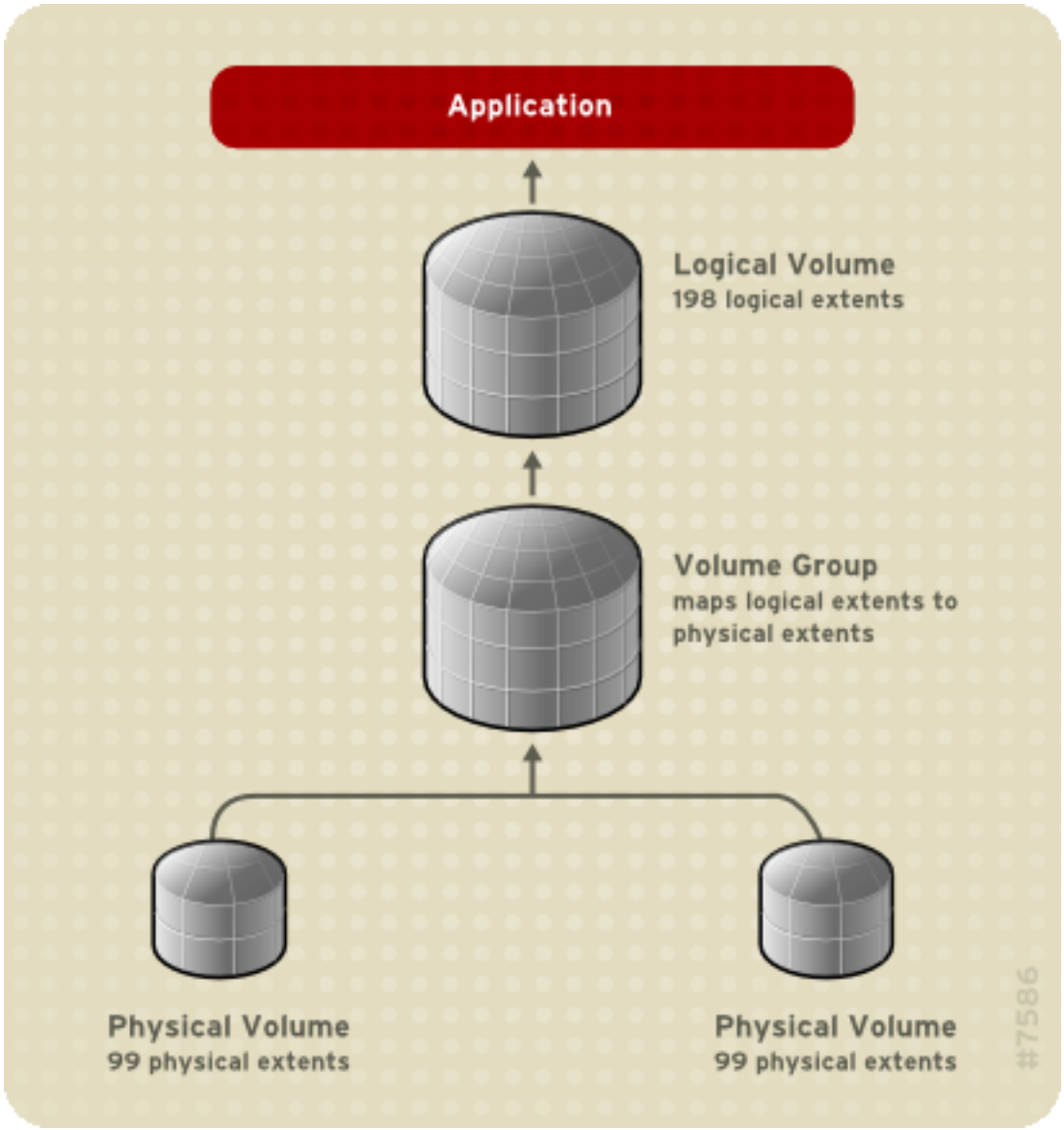

図 2.2. エクステントマッピング

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. 「不平等物理ボリュームを持つリニアボリューム」 では、4MB の物理エクステントサイズを持つ 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 と言う２つの物理ボリュームが含まれています。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 100 エクステントのサイズ（400MB）です。そして PV2 は、200 エクステントの サイズ（800MB）です。1 から 300 エクステント(4MB から 1200MB)までの自由な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 言うリニアボリュームは 300 エクステントのサイズを持っています。

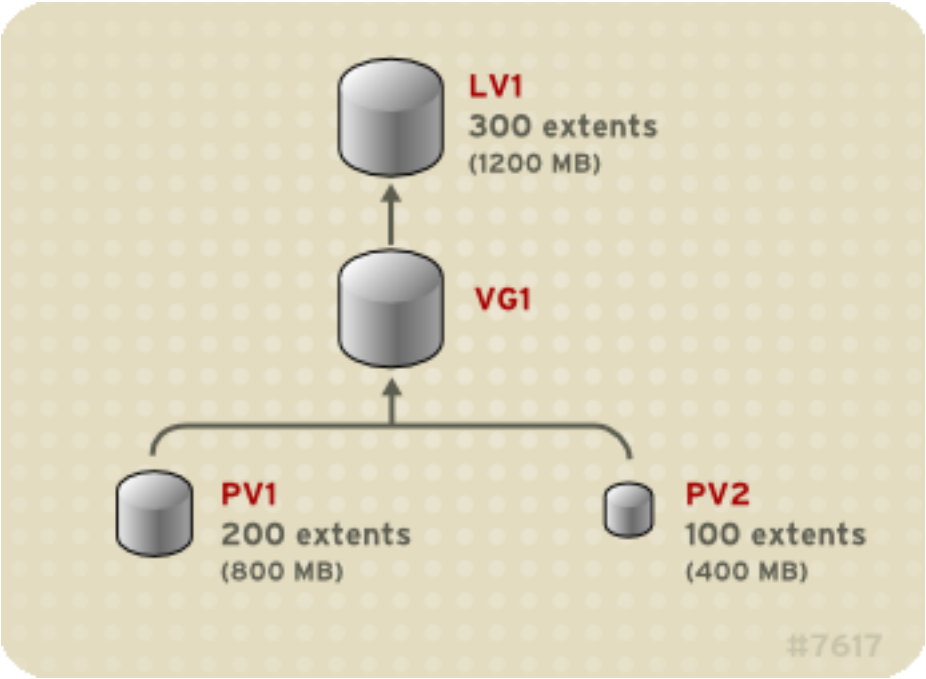

図 2.3. 不平等物理ボリュームを持つリニアボリューム

物理エクステントの集合体から好みのサイズのリニア論理ボリュームを複数作成する ことができます。図 2.4. 「複数論理ボリューム」 では、図 2.3. 「不平等物理ボリュームを持つリニアボリューム」 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 論理ボリュームが作り出されています。LV1 は 250 のエクステントサイズ（1000MB）を 持ち、LV2 は 50 のエクステントサイズ （200MB）を持っています。

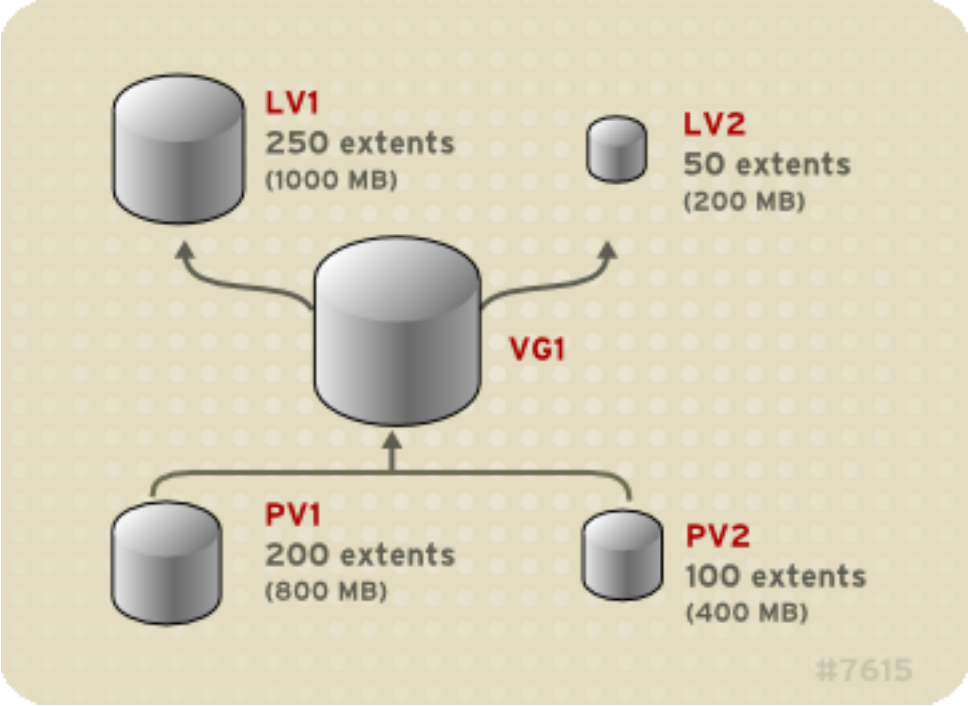

図 2.4. 複数論理ボリューム

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、背後にある 物理ボリュームすべてに渡って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。 大量の連続的読み込みと書き込みには、この方法がデータ I/O の効率を向上します。

ストライピングは、事前設定数の物理ボリュームにデータを総当たり式に書き込んでいくことに より、パフォーマンスを向上します。ストライピングでは、I/O は並行して実行されます。幾つかの状況では、これはストライプ内に追加する物理ボリューム毎にほぼ直線的な パフォーマンス上昇となり得ます。

以下のイラストでは、３つの物理ボリュームに渡ってデータがストライプ化されている 状態を示しています。この表では:

- 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。

ストライプ化し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 超過することはできません。

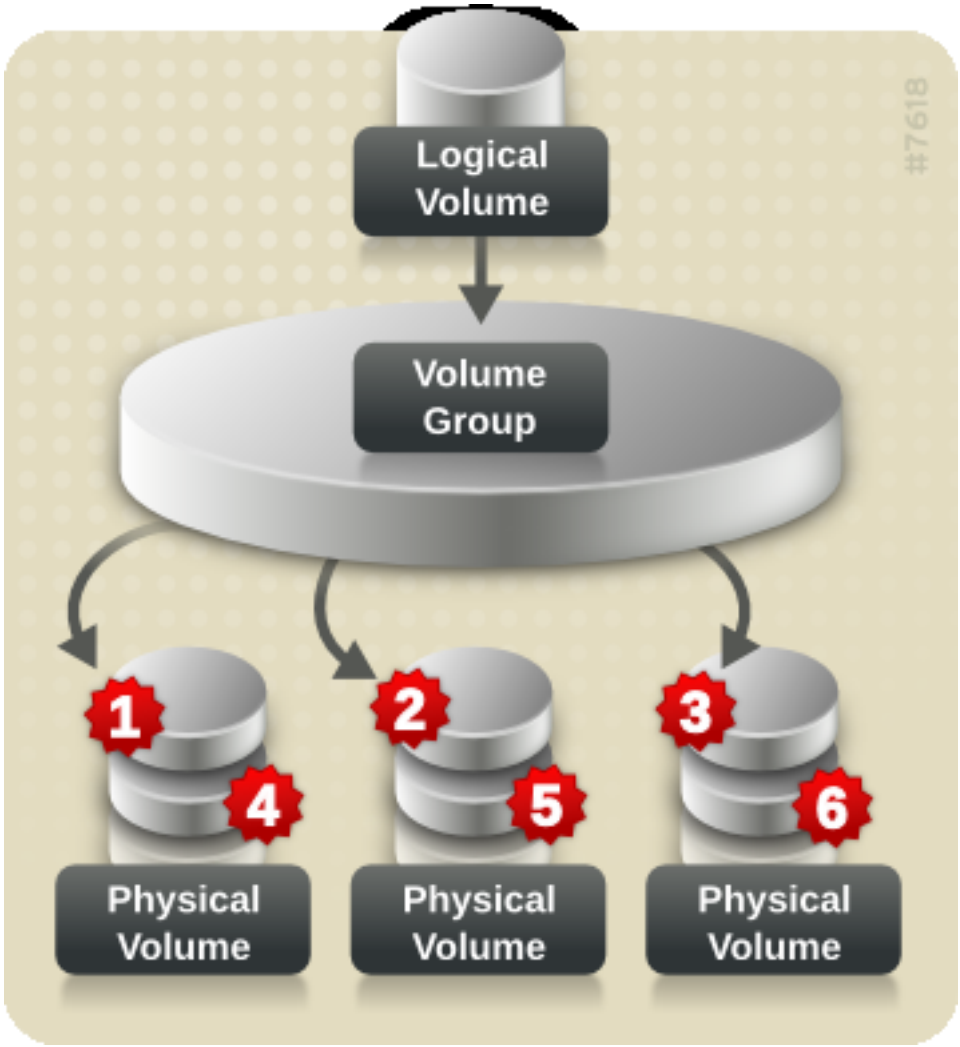

図 2.5. ３つの PV に渡るストライプのデータ

ストライプ化論理ボリュームでは、別のデバイスセットを最初のセットの末尾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為にボリュームグループを構成している背後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。例えば、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。そうではなく、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。ストライプ化ボリュームの拡張に関する詳細は 項 4.9. 「ストライプ化ボリュームの拡大化」 でご覧下さい。

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、 それはまた２つ目のデバイスにも書き込まれて、データのミラー化をします。この重複保存により、 デバイス故障に対する保護を提供します。ミラーの一脚が故障した場合、論理ボリュームはリニア ボリュームとなりますが、それでもまだアクセスは可能です。

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 は背後にある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 ミラー化されるように確認します。LVM では、複数ミラーでミラー化論理ボリュームを 作成することができます。

LVM ミラーはコピーを受けるデバイスを、標準的な 512KB サイズの区域に分割します。 LVM はミラーと同期している区域を追従確認するのに使用する小さなログを維持します。 このログは、再起動後にも残るように固執としてディスク上に保存されるか、あるいは メモリー内で維持されます。

図 2.6. 「ミラー化論理ボリューム」 ミラー１つを持つミラー化した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。

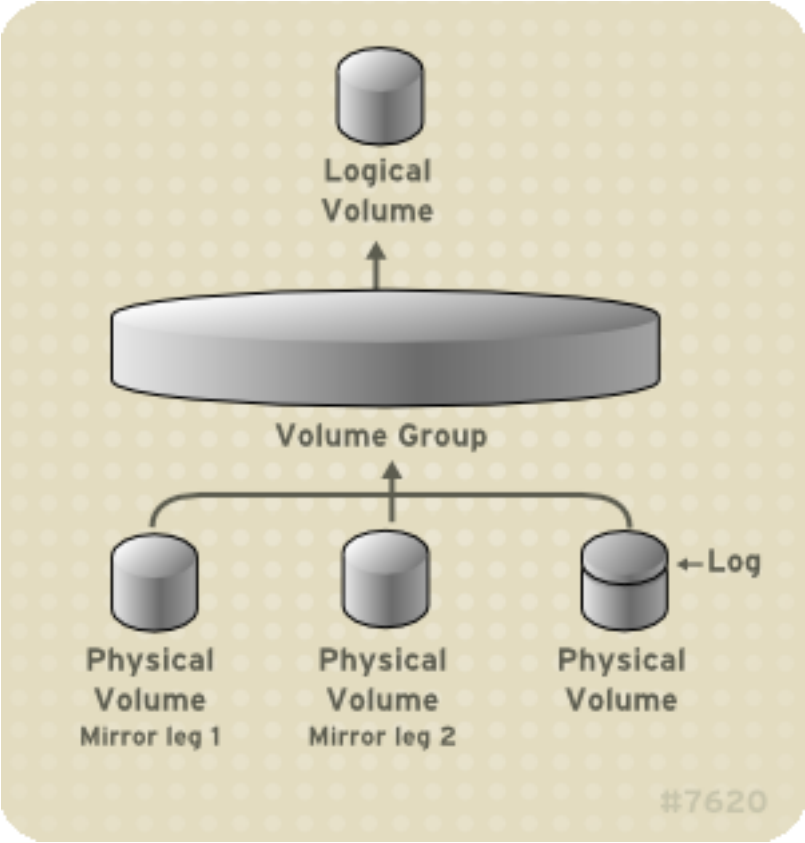

図 2.6. ミラー化論理ボリューム

注意

ミラー化論理ボリュームはクラスタ内では、現在サポートされていません。

ミラーの作成と修正に関する情報は 項4.1.3. 「ミラー化ボリュームの作成」 でご覧下さい。

3.4. スナップショットボリューム

LVM スナップショット機能は、サービスを妨害せずに特定の時期のデバイスの 仮想イメージを作成する能力を提供します。スナップショットを取った後に オリジナルデバイスに変更がなされた場合、スナップショット機能は、変更部分の変更以前の状態のコピーを作成してデバイスの状態の再構成ができる ようにします。

注意

LVM スナップショットにはクラスタ内のノード全域に渡るサポートはありません。

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は最小限のストレージだけを必要とします。 例えば、稀にしか更新されないオリジナルでは、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。

注意

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続きの 代替を提供するものではありません。

スナップショットが満杯になると、スナップショットは停止します。これは、 オリジナルのファイルシステムのために十分な領域が残っていることを確実にする 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増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サイズが必要以上に大きければ、そのボリュームの サイズを縮小して、他の論理ボリューム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場合、オリジナルへの完全な読み込み/書き込みの アクセスがそのまま残ります。スナップショットの一部が変更された場合、その変更部分にはマークが 付けられて、そこにはオリジナル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数種の使用法があります:

- 最も標準的なものとして、スナップショットは、継続的にデータを更新している ライブシステムを停止することなく、論理ボリューム上でバックアップを実行する時に 取り入れられます。
- 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統合性をチェックして、オリジナルのファイルシステムが修理を必要とする かどうかを判定することができます。
- スナップショットは読み込み/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、実際のデータを触ることなく実稼働データに対するアプリケーションの テストができます。
- Xen 仮想マシンモニターとの使用の為にボリュームを作成することができます。 スナップショット機能を使用してディスクイメージを作成し、それをスナップショットにして 特定の domU インスタンス用にそのスナップショットを修正することができます。もう一つの スナップショットを作成して、それをもう一つの domU インスタンス用に修正できます。 使用されるストレージは、オリジナルファイル上か、スナップショット上で変更された一塊のみですから、ボリュームの大部分は共有できます。

LVM 管理の概要

この章では、LVM 論理ボリュームの設定の為の管理手続きの概要を提供しています。 この章は、必要となる手順についての全般的な認識を深めることを目的としています。 一般的な LVM 設定工程における特定の段階別例については、章 5. LVM 設定の例 をご覧ください。

LVM 管理を執行するために使用できる CLI コマンドの説明については、章 4. CLI コマンドでの LVM 管理 を ご覧ください。別の方法として、章 7. LVM GUI での LVM 管理 で説明のある LVM GUI を使用することも できます。

1. クラスタ内で LVM ボリュームを作成

クラスタ環境内で LVM 論理ボリューム作成することは、単独ノード上に LVM 論理ボリュームを 作成することと同じです。LVM コマンドそのものや LVM GUI インターフェイスにはなんの相違も ありません。ただ、クラスタ内に作成している LVM ボリュームを有効にするために、クラスタインフラ ストラクチャが稼働している必要があり、クラスタは定員数以上である必要があります。

クラスタインフラストラクチャをセットアップする方法についての情報には、 Red Hat クラスタの設定と管理 をご覧下さい。

2. 論理ボリューム作成の概要

LVM 論理ボリュームを作成するために実行すべき手順の要約を以下に示します。

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。 （この操作がラベルを付けます）
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。 この文書内の例では、GFS ファイルシステムを使用します。

1. gfs_mkfs コマンドを使用して、論理ボリューム上に GFS 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。 クラスタ化したシステムでは、そのクラスタ内の全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の fstab 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 GFS ファイルシステムを作成してマウントする 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って、 ボリュームが作成されたマシンではないため、LVM ボリュームの作成はマシンから独立しています。 ストレージを使用するサーバーがローカルコピーを持っていますが、それは物理ボリューム上にある ものから復元できます。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 添えつけることが出来ます。

3. 論理ボリューム上でファイルシステムの増大

論理ボリューム上でファイルシステムを増大するには、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含めるように、増大するファイルシステムを持つ論理ボリュームを収納している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含めます。
4. ファイルシステムを増大します。

ボリュームグループ内に十分な未割り当ての領域がある場合は、手順１と２ の代わりに 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。

4. 論理ボリュームバックアップ

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、全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、メタデータバックアップは /etc/lvm/backup ファイルに保存され、メタデータアーカイブは /etc/lvm/archive ファイルに保存されます。 メタデータアーカイブが /etc/lvm/archive ファイルに保存される期間と保存されるアーカイブファイルの数量は、lvm.conf ファイル内で設定するパラメータにより決定されます。毎日のシステムバックアップは、バックアップ内に /etc/lvm ディレクトリの内容を含んでいる必要があります。

メタデータバックアップは、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 バックアップしません。

メタデータは、 vgcfgbackup コマンドを使用して、手動で /etc/lvm/backup ファイルにバックアップできます。また、 vgcfgrestore コマンドを使用すれば、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 項3.11.「ボリュームグループメタデータのバックアップ」 で説明があります。

5. ロギング

全てのメッセージ出力は、以下のロギングレベル用に独立した選択を持つ ロギングモジュールを経由して渡されます。

- 標準出力/エラー
- システムログ
- ログファイル
- 外部ログ機能

ロギングのレベルは /etc/lvm/lvm.conf ファイル内にセットされており、 これに関しては、付録 B. LVM 設定ファイル で説明があります。

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、維持するために LVM CLI （Command Line Interface）コマンドで実行できる個別の管理タスクを要約しています。

1. CLI コマンドの使用

全ての LVM CLI コマンドには数種の全般的な機能があります。

コマンドライン引数でサイズを必要とする時に、単位は常に明示的に指定することができます。 単位を指定しない場合、デフォルトが想定されて通常、 KB か MB になります。LVM CLI コマンドは 分数は受け付けません。

コマンドライン引数内で単位を指定している場合、LVM は大文字/小文字を区別しない為、 例えば、M か m の 指定は同等になります。そして 2 の乗数（1024 の倍数）が使用されます。しかし、コマンド内で --units 引数を指定している場合、 単位は 小文字では 1024 の倍数であり、大文字では 1000 の倍数の単位を示します。

コマンドが引数として、ボリュームグループ名か、論理ボリューム名を取る場合、 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、全てのボリュームグループの一覧が 代用されます。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提示されている場合、 そのボリュームグループ内の全ての論理ボリュームの一覧が代用されます。例えば、 lvdisplay vg0 コマンドは、ボリュームグループ vg0 内の 全ての論理ボリュームを表示します。

全ての LVM コマンドは -v 引数を受け付けますので、これを複数回数入力して 出力の冗長性を増加できます。例えば、次の例では 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 引数を持つ 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、あるいは、-vvvv の 引数を使用して、コマンドの実行に対してより進展した詳細を表示することができます。-vvvv 引数は、現時点で最大級の詳細情報を提供します。以下の例では、-vvvv 引数が指定 された lvcreate コマンドの出力の最初の数行を示すものです。

```
# 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 コマンドのいずれにも --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
```

あるコマンドの 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。

全ての LVM オブジェクトは、内部で UUID によって照合されます。これはオブジェクトを 作成した時に割り当てられます。これは、例えばボリュームグループの一部である 物理ボリューム /dev/sdf を削除して、そしてそれを入れ戻して それが今度は、/dev/sdk であることを発見した場合に便利になります。 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、物理ボリュームを 発見することができます。物理ボリュームの作成時に物理ボリュ

ームの UUID の指定の方法に 関する情報は、項4. 「物理ボリュームメタデータの復元」 でご覧下さい。

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各種事項を実行するコマンドを説明しています。

2.1. 物理ボリュームの作成

以下のサブセクションでは、物理ボリューム作成に使用するコマンドの説明をしています。

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全域デバイス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いけません。DOS のディスクパーティションには、fdisk や cfdisk コマンドか、又は同等を使用して、パーティション id が 0x8e に セットしてある必要があります。ディスク全域デバイスの為にだけは、パーティションテーブルを削除する必要があり、これは実質的にそのディスクの全てのデータを破壊します。以下のコマンドを使用して、 最初のセクターをゼロにすると既存のパーティションテーブルを外すことができます:

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

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。初期化とは、ファイルシステムのフォーマッティングと同義に なります。

以下のコマンドは、LVM 物理ボリュームとして使用する為に /dev/sdd1、 /dev/sde1、及び /dev/sdf1 を初期化します。

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

ディスク全域でなく、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、LVM 論理ボリュームの 一部として後の使用の為に LVM 物理ボリュームとして /dev/hdb1 を初期化しています。

```
pvcreate /dev/hdb1
```

2.1.3. ブロックデバイスのスキャン

以下の例に示してあるように、lvmdiskscan コマンドを使用して 物理ボリュームとして使用可

能な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
```

2.2. 物理ボリュームの表示

LVM 物理ボリュームのプロパティを表示するのに三つのコマンドが使用できます: pvs、pvdisplay、及び pvscan です。

pvs コマンドは、物理ボリューム毎に 1 行ずつ表示して設定可能な形式で 物理ボリューム情報を提供しします。pvs コマンドは充実した形式制御を 装備するため、スクリプトに役に立ちます。出力をカスタマイズする為の pvs コマンドの使用法に関する情報は 項9. 「LVM 用のカス

タム報告」 でご覧ください。

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。固定形式で物理プロパティ（サイズ、エクステント、ボリュームグループなど）を 表示します。

以下の例では、単独物理ボリュームの 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-X08M-mcpsVe
```

pvscan コマンドは、物理ボリューム用にシステム内のサポートされた全ての LVM ブロックデバイスをスキャンします。

以下のコマンドは見つけた物理デバイスを全て表示します:

```
# 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]
```

lvm.conf 内でフィルターを定義して、このコマンドが特定の 物理ボリュームのスキャンを避けるようにすることが出来ます。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法については 項6. 「LVM デバイススキャンをフィルターで制御」 をご覧下さい。

2.3. 物理ボリューム上での割り当て防止

pvchange コマンドを使用すると、 1 つ、又は複数の物理ボリュームの 空き領域上で物理エクステントの割り当てを防止することができます。これは、ディスクエラーがある場合、 又は物理ボリュームを取り除く場合に必要になるかも知れません。

以下のコマンドは /dev/sdk1 上での物理エクステントの割り当てを 禁止します。

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

pvchange コマンドの -xy 引数を使用すると、 以前に禁止されていた割り当てを許可することができるようになります。

2.4. 物理ボリュームのサイズ変更

なんらかの理由で背後にあるブロックデバイスのサイズを変更する必要がある場合は、pvresize コマンドを使用して LVM を新規サイズに更新します。 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。

2.5. 物理ボリュームの削除

あるデバイスが LVM での使用に必要でなくなった場合、pvremove コマンドを使用して LVM ラベルを取り除くことができます。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、項3.5. 「ボリュームグループから物理ボリュームを削除」 で説明してあるように、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
```

3. ボリュームグループ管理

このセクションでは、ボリュームグループ管理の各種事項を実行する為のコマンドを説明 しています。

3.1. ボリュームグループの作成

１つ、又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 コマンドは名前別に新しいボリュームグループを作成し、最低１つの物理ボリュームをそれに追加します。

以下のコマンドは、vg1 と言う名前のボリュームグループを作成します。 これには、物理ボリュームである /dev/sdd1 と /dev/sde1 が含まれることになります。

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

ボリュームグループの構成に物理ボリュームが使用される時、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、論理ボリュームがそのサイズを増加 したり、縮小したりする為の最小単位です。大量のエクステントで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。

デフォルトの設定が適切でない場合、vgcreate コマンドに -s 引数を使用して、エクステント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が所有できる物理ボリューム、又は論理ボリュームの数量を 限定することができます。

デフォルトで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ことなど、 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 使用すると、contiguous、anywhere、あるいは cling の割り当てポリシーを指定できるようになります。

contiguous ポリシーは 新規のエクステントが既存エクステントに 隣接していることを要求します。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 割り当てポリシーがそれらを使用できない場合は、同じ物理ボリュームに２つのストライプを配置すると、パフォーマンスが 低下しますが、anywhere 割り当てポリシーが割り当てをします。cling ポリシーは、論理ボリュームの同じストライプ内の既存エクステントと同じ物理ボリューム上に新規エクステントを配置します。これらのポリシーは vgchange を使用して、変更することができます。

一般的に normal 以外の割り当てポリシーは、特異な非標準的な エクステント割り当てを必要とする特別なケースのみに必要となります。

LVM ボリュームグループと背後にある論理ボリュームは、以下のようなレイアウトをもって、/dev ディレクトリ内のデバイス特別ファイルディレクトリに収納 されています。

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

例えば、２つのボリュームグループ: myvg1 と myvg2 を作成して、それぞれが ３つの論理ボリューム: lvo1、lvo2、lvo3 を 持つようにすると、６つのデバイス特別ファイルを作成することになります:

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

LVM に於けるデバイス最大サイズは、64-bit CPU 上で 8 Exabyte です。

3.2. 物理ボリュームをボリュームグループに追加

新規に物理ボリュームを既存のボリュームグループに追加するには、vgextend コマンドを使用します。vgextend コマンドは、１つ、又は複数の 空き物理ボリュームを追加することで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
```

3.3. ボリュームグループの表示

LVM ボリュームグループのプロパティを表示するには、２つのコマンドが使用できます: vgs と vgdisplay です。

vgscan コマンドの主要目的はボリュームグループの全ての 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ことですが、ボリューム グループの表示もします。vgscan コマンドに関する情報は 項3.4. 「キャッシュファイル構築の為にボリュームグループのディスクスキャン」 でご覧下さい。

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、ボリュームグループ毎１行ずつ表示をします。vgs コマンドは 形式制御の情報を多く提供するため、スクリプティングの便利なものです。出力をカスタマイズするための vgs コマンドの使用法については、項9. 「LVM 用のカスタム報告」 をご覧下さい。

vgdisplay コマンドは、固定した形式でボリュームグループの プロパティ（サイズ、エクステント、物理ボリュームの数量など）を表示します。以下にある 例では、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、全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
```

3.4. キャッシュファイル構築の為にボリュームグループのディスクスキャン

vgscan コマンドは LVM 物理ボリュームとボリュームグループを検索しながら、システム内の全てのサポートされるデバイスをスキャンします。これにより、 /etc/lvm/.cache ファイル内に LVM キャッシュファイルが構築されて、これが現在の LVM デバイスのリストを維持します。

LVM は、システムが起動した時点と、vgcreate コマンドの実行時や、 LVM が一貫性問題を検知した時点などの、他の LVM 稼働期間中に自動的に vgscan コマンドを実行します。ハードウェアの設定を変更した場合は、システムの起動時に存在していなかったデバイス状態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する必要が出てきます。これは、例えば、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いてい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でしょう。

lvm.conf ファイル内でフィルターを定義することにより、 特定デバイスを避けるようにスキャンを限定できます。フィルターを使用したスキャンする デバイスの制御法に関する情報は項6. 「LVM デバイススキャンをフィルターで制御」 でご覧下さい。

次の例では、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
```

3.5. ボリュームグループから物理ボリュームを削除

ボリュームグループから使用しない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、又はそれ以上の 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。これが、 異なるボリュームグループで使用できるように、又はシステムから削除できるように物理ボリュームを 開放します。

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 確認することが出来ます

。

```
# 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 使用してそこにある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
```

3.6. ボリュームグループのパラメータ変更

vgchange コマンドで既存のボリュームグループ用に 変更できるボリュームグループパラメータは数種類あります。しかし、項3.7. 「ボリュームグループのアクティベート（始動）/ディアクティベート（停止）」 に 示してあるように、このコマンドは主にボリュームグループのアクティベートとディアクティベート用に 使用されます。

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の 説明については vgchange(8) man ページをご覧下さい。

3.7. ボリュームグループのアクティベート（始動）/ディアクティベート（停止）

ボリュームグループを作成すると、それはデフォルトでアクティベートされます。 このことは、そのグループ内の論理ボリュームはアクセス可能で変更の影響を受けると 言う意味になります。

ボリュームグループを活動停止する必要のある、カーネルには未知の状況がで種々存在します。 ボリュームグループをディアクティベート、又はアクティベートするには、vgchange コマンドで -a（--available）引数を使用します。

以下の例では、ボリュームグループ my_volume_group を ディアクティベートします。

```
vgchange -a n my_volume_group
```

クラスタ化したロッキングが有効である場合、 ’ e’ を追加すると１つのノード上で専用にボリューム グループをアクティベート、又はディアクティベートし、’ l’ を追加すると、ローカルノード上の ボリュームグループをアクティベート、又はディアクティベートします。単独ホストスナップショットを 持つ論理ボリュームは、１度に１つのノードだけを使用するため、常に専用にアクティベートされます。

項4.4. 「論理ボリュームグループのパラメータ変更」 で説明してあるように、lvchange を使用して、個別の論理ボリュームをディアクティベートすることができます。クラスタ内の個別ノード上で 論理ボリュームをアクティベートする方法については、項8. 「クラスタ内の個別ノード上の論理ボリュームをアクティベート」 を ご覧下さい。

3.8. ボリュームグループの削除

論理ボリュームを含んでいる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
```

3.9. 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出来ません。それぞれの 既存の論理ボリュームは全面的に物理ボリューム上に存在し、既存の、又は新規の ボリュームグループを形成している必要があります。しかし必要であれば、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。

以下の例では、オリジナル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 を分離しています。

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

3.10. ボリュームグループの合成

２つのボリュームグループを合成して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 エクステントが同じ場合、そしてボリュームグループの物理、及び論理ボリューム要約が 「目標」ボリュームグループの制限内に適合するならば、停止中の 「起点」ボリュームを 活動中、又は停止中の「目標」ボリュームに融合することができます。

以下のコマンドは、停止中のボリュームグループ my_vg を 動作中、又は停止中の ボリュームグループ databases に統合して 冗長なランタイム情報を提供します。

```
vgmerge -v databases my_vg
```

3.11. ボリュームグループメタデータのバックアップ

メタデータのバックアップとアーカイブは、lvm.conf ファイル内で 無効になっていない限り、毎回のボリュームグループと論理ボリューム設定の変更毎に、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 に 保存されており、メタデータアーカイブは /etc/lvm/archives に保存されています。 vgcfgbackup コマンドを使用すると、メタデータを手動で /etc/lvm/backup ファイルにバックアップすることができます。

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全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取り戻す為の vgcfgrestore コマンドの 使用に関する例は、項 4. 「物理ボリュームメタデータの復元」 でご覧下さい。

3.12. 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。

以下のコマンドのいずれかで、既存ボリュームグループ vg02 を my_volume_group に改名できます。

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

```
vgrename vg02 my_volume_group
```

3.13. ボリュームグループを別のシステムに移動

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。

vgexport コマンドはシステムが停止中のボリュームグループにアクセスできないようにします。これが物理ボリュームの取り外しを可能にします。vgimport コマンドは vgexport コマンドで 停止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でユーザーがファイルにアクセスしていない 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停止中としてマークします。これはボリュームグループのこれ以降の 活動を防止します。

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、ボリュームグループを削除中のシステムからのアクセスを防止することになります。

   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れます。

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

   システムが次にシャットダウンされる時に、ボリュームグループを構成していたディスクを切断して、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、新しいシステムによるアクセスを可能にします。

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ベートします。

6. ファイルシステムをマウントしてそれを使用可能にします。

3.14. ボリュームグループディレクトリの再作成

ボリュームグループディレクトリと論理ボリューム特別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 ディレクトリ内の LVM2 特別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。このコマンドは欠如している特別ファイルを作成し、使用しないものを削除します。

--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出来ます。

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各種事項を実行するコマンドを説明しています。

4.1. 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。 以下のサブセクションで説明してあるように、リニアボリューム、ストライプ化ボリューム、及び ミラー化ボリュームが作成できます。

論理ボリューム用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)

以下のセクションでは、LVM で作成できる３つの論理ボリュームタイプの為の 論理ボリューム作成の例を提供します。

4.1.1. リニアボリュームの作成

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。通常は論理ボリュームは背後にある物理ボリューム上で次に使用可能な空きを基準にして空き領域を占有します。論理ボリュームを修正することで、物理ボリュームの領域の開放と再割り当てができます。

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 言う 1500 メガバイトのリニア論理ボリュームを作成し、ブロックデバイス /dev/testvg/testlv を 作ります。

```
lvcreate -L1500 -ntestlv 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 と言う 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 コマンドの -l 引数を使用して、 論理ボリュームのエキステント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、ボリュームグループ内で使用する論理グループのパーセントも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 testvol 内で全体の 60% を 使用する mylv と言う論理ボリュームを作成します。

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

lvcreate コマンドの -l 引数を使用して、 ボリュームグループ内の空き領域のパーセント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全て使用する yourlv と言う論理グループを作成します。

```
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の作成の別の方法としては、 vgdisplay コマンドを使用して、「合計 PE」サイズを見つけて、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 と言うボリュームグループを全使用する mylv と言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
```

論理ボリュームを作成するのに使用した背後にある物理ボリュームは、物理ボリュームが削除される場合に重要になります。そのため、論理ボリュームを作成する時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、 項3.5.「ボリュームグループから物理ボリュームを削除」 でご覧下さい。

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られる論理ボリュームを 作成するには、lvcreate コマンドの行の末尾にその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 testlv と言う論理ボリュームを作成します。

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

論理ボリュームとして使用される物理ボリュームのエクステントを指定することが出来ます。以下の例では、ボリュームグループ内の物理ボリューム /dev/sda1 の エクステント 0から 25 まで、 及び物理ボリューム /dev/sdb1 のエクステント 50 から 125 までで構成されるリニア論理ボリュームを作成します。

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

以下の例では、物理ボリューム /dev/sda1 のエクステント 0 から 25 までのリニア論理ボリュームを作成して、それからエクステント 100 で論理ボリュームの レイアウトを継続します。

```
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（相続）であり、これはボリュームグループ用の同じポリシーにも 適用されます。これらのポリシーは lvchange コマンドの使用で 変更できます。割り当てポリシーの詳細情報については 項3.1.「ボリュームグループの作成」 をご覧下さい。

4.1.2. ストライプ化ボリュームの作成

大量の連続的な読み込みと書き込みには、ストライプ化論理ボリュームの作成が データ I/O を効率的にします。ストライプ化ボリュームに関する一般情報には、 項3.2.「ストライプ化論理ボリューム」 をご覧下さい。

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量を指定します。これが、幾つの物理ボリュームで 論理ボリュームがストライプ化されるかを決定します。ストライプの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越えることは出来ません（--alloc anywhere 引数が 使用さる場合は例外）。

ストライプサイズは 4kB と 512kB の間で 2 の乗数にチューンされる必要があり、 ストライプ化ボリュームを使用しているアプリケーションの I/O に一致する必要があります。lvcreate コマンドの -I 引数はストライプサイズを キロバイトで指定します。

ストライプ化論理ボリュームの背後にある物理ボリュームデバイスが異なるサイズを持つ場合、 ストライプ化ボリュームの最大サイズはその背後のデバイスの最小サイズで決定されます。例えば、二脚のストライプがある場合、最大サイズは最小デバイスのサイズの二倍になります。 三脚ストライプの場合、最大サイズは最小デバイスのサイズの三倍になります。

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渡ってストライプ化論理

ボリュームを作成します。論理ボリュームは 50 ギガバイトのサイズで、gfslv と言う名前を持ち、ボリュームグループ vg0 から作り出されます。

```
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、ボリュームグループ testvg の中で、 2 つの物理ボリュームに渡ってストライプ化する 100 エクステントのサイズの、stripelv と 言う名前のストライプ化ボリュームを作成します。ストライプは /dev/sda1 の セクター 0-50 と、/dev/sdb1 のセクター 50-100 を使用します。

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

4.1.3. ミラー化ボリュームの作成

ミラー化ボリュームを作成する時には、lvcreate コマンドの -m 引数を使用して、データのコピー数量を指定します。-m1 と指定すると、ミラー１つが作成され、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）。同じように -m2 と指定すると、ミラー２つが作成され、ファイルシステムのコピーが合計３つとなります。

以下のコマンドは、単独ミラーのミラー化論理ボリュームを作成します。ボリュームは 50 ギガバイトの サイズで、mirrorlv と言う名前で、ボリュームグループ vg0 から 作り出されます：

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

LVM ミラーは、コピーされるデバイスをデフォルトで 512KB のサイズとなっている区画（regions）に 分割します。この区画サイズは -R 引数を使用することにより、MB のサイズで指定することが出来ます。LVM は、ミラーと同期している区画を追跡記録するために小さなログを 維持します。デフォルトでは、このログはディスク上に保存され、再起動後も固執するようになっています。 --corelog 引数を使用すると、それを変更してログがメモリー上で保存されるように 指定でき、余分なログデバイスの必要性を解消します。しかし、これには再起動の度にミラー全体を再度 同期化することが要求されます。

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 ondiskmirvol と言う名前で、１つのミラーを 持ちます。このボリュームは 12MB のサイズで、ミラーログをメモリーに保存します。

```
# lvcreate -L 12MB -m1 --corelog -n ondiskmirvol bigvg
```

```
  Logical volume "ondiskmirvol" created
```

ミラーが作成されると、ミラーの区画は同期化されます。大きなミラーのコンポーネントには、 同期プロセスは長時間かかる可能性があります。再生され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 初期同期化は必要ないことを示すことができます。

ミラーログとログ用に使用するデバイス、及びそのデバイスの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精確なエクステントを指定します。 LVM は、コマンドラインでは、必ずしもデバイスの一覧順序を配慮しません。物理ボリュームが一覧にあれば、 それが割り当て実行の唯一の場所です。既に割り当てされている物理エクステントが一覧にあれば、それは 無視されます。

以下のコマンドは、単独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 500 メガバイトのサイズで、mirrorlv と言う名前であり、ボリュームグループ vg0 から作り出されます。ミラーの最初の脚はデバイス /dev/sda1 上で、 2 番目の脚が /dev/sdb1 上で、 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独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 500 メガバイトのサイズで、mirrorlv と言う名前を持ち、ボリューム グループ vg0 から作り出されます。ミラーの最初の脚はデバイス /dev/sda1 のエクステント 0 から 499 までにあり、ミラーの二番目の脚はデバイス /dev/sdb1 のエクステント 0 から 499 までにあります。そしてミラーログは デバイス /dev/sdc1 のエクステント 0 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既に割り当てられている場合、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
```

4.1.4. ミラー化ボリューム設定の変更

論理ボリュームは、lvconvert を使用して、ミラー化ボリュームから リニアボリュームに、又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 などの既存論理ボリュームの他のミラーパラメータも再設定できます。

論理ボリュームをミラー化ボリュームに変換する時には、基本的に既存ボリューム用に ミラー脚を作成することになります。これは、ボリュームグループがミラー脚とミラーログの 為にデバイスと領域を持っている必要があると言う意味です。

ミラーの 1 脚が欠如した場合、LVM はボリュームをリニアボリュームに変換して、 ミラーの冗

長なしでもまだ ボリュームへのアクセスが可能であるようにします。その脚を入れ替えた後は、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 項3. 「LVM ミラー障害からの復元」 で説明してあります。

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、ミラー脚を取り除きます。

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

4.2. 固執デバイスの番号

メジャーとマイナーの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。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）番号でアクティベートされている場合に、一部のアプリケーションは 最も良く機能を発揮します。そのような状態は lvcreate と lvchange コマンドで以下の引数を使用してこれらのデバイスを指定することで 達成できます:

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

大きめのマイナー番号を使用することで、その番号が既に別のデバイスに動的に割り当て られていないことを確実にします。

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に fsid パラメータを指定すると、LVM 内で固執のデバイス番号を セットする必要がなくなるでしょう。

4.3. 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変更するには、lvreduce コマンドを 使用します。論理ボリュームがファイルシステムを含んでいる場合、最初にファイルシステムを 縮小して（又は LVM GUI を使用して）、論理ボリュームが常に少なくとも期待されるファイルシステムの 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 論理エクステントだけ削減します。

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

4.4. 論理ボリュームグループのパラメータ変更

論理ボリュームのパラメータを変更するには、lvchange を 使用します。変更できるパラメータの一覧を見るには、lvchange(8) 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ベートと ディアクティベートができます。ボリュームグループ内の全ての論理ボリュームのアクティベートと ディアクティベートを同時に達成するには、項3.6. 「ボリュームグループのパラメータ変更」 で説明してある 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 の権限を変更して読み込み専用にします。

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

4.5. 論理ボリュームの名前変更

既存の論理ボリュームの名前を変更するには、lvrename コマンドを 使用します。

以下のコマンドのいずれもボリュームグループ vg02 内の 論理ボリューム lvold を lvnew に改名します。

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

```
lvrename vg02 lvold lvnew
```

クラスタ内の個別ノード上で論理ボリュームをアクティベートする方法に関する情報は 項8. 「クラスタ内の個別ノード上の論理ボリュームをアクティベート」 でご覧下さい。

4.6. 論理ボリュームの削除

活動していない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。ただし削除する前に、umount コマンドで論理ボリュームを 閉じる必要があります。更には、クラスタ環境では、その削除の前に論理ボリュームを停止する必要が あります。

論理ボリュームが現在マウントされている場合、ボリュームを削除する前にそれを アンマウントしてください。

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、論理ボリュームはディアクティベートされていないことに注意してください。

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

lvchange -an コマンドの使用で、削除する前に明示的に 論理ボリュームをディアクティベート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の確認プロンプトは表示されません。

4.7. 論理ボリュームの表示

LVM 論理ボリュームのプロパティを表示するのに使用できるコマンドが３つあります: lvs、lvdisplay、及び lvscan です。

lvs コマンドは設定可能な形式で論理ボリューム情報を提供して、 論理ボリューム毎に１行ずつ表示します。lvs コマンドは多大な 形式制御を提供するため、スクリプティングに便利です。出力をカスタマイズする為の lvs コマンドの使用法についての詳細は 項9. 「LVM 用のカスタム報告」 で ご覧下さい。

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（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オリジナル論理ボリューム用に作成 されている場合、このコマンドは全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 （活動中か、停止中か）の一覧も表示します。

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

lvscan コマンドは、システム内の全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
```

4.8. 論理ボリュームの増大化

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

論理ボリュームを拡大した後は、それに合うように関連したファイルシステムのサイズも 拡大する必要があります。

論理ボリュームを拡大する場合、そのボリュームをどれだけ拡大したいか、すなわち 拡大後のサイズがどれくらいかを指示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 ギガバイトまで拡大します。

```
# 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 更に 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 コマンドに -l 引数を使用して論理ボリュームのサイズを拡大する量の エクステント数を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のパーセント、 又はボリュームグループの残りの空き領域のパーセントを指定することも出来ます。以下のコマンドは、 ボリュームグループ myvg 内の全ての未割り当て領域を取り込むように testlv と言う論理ボリュームを拡大します。

```
[root@tng3-1 ~]# 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大した後は、それに適合するようにファイルシステムサイズも拡大する 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 サイズを基になる論理ボリュームのサイズまで拡大しますので、２つのコマンドそれぞれの為に同じサイズを指定する煩わしさはありません。

4.9. ストライプ化ボリュームの拡大化

ストライプ化論理ボリュームのサイズを拡大するためには、ボリュームグループを構成している 物理ボリュームに、ストライプをサポートする為の十分な空き領域がなければなりません。

例えば、 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、ストライプの拡大にはなりません。その為には、 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。

例えば、以下の vgs コマンドで表示される２つの背後にある物理 ボリュームから構成されるボリュームグループ vg を考えてみます。

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

ボリュームグループの全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た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
```

以下のコマンドは、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、これが 135G の追加領域を与えます。

```
# 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全サイズまで 拡大することは出来ません。データをストライプ化するのに背後にあるデバイスが ２つ必要です。

```
# 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大するには、もう１つの物理ボリュームを追加して、それで論理ボリュームを拡大します。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 5A をボリュームグループの全サイズまで 拡大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大するのに十分な物理デバイスがない場合でも、 その拡大部分がストライプでなくても良い場合には、ボリュームの拡大はとにかく可能です。但しこれは不均一なパフォーマンスになります。論理ボリュームに領域を追加している時、 デフォルトの動作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プパラメータを使用する ことですが、これらのパラメータは書き換えることができます。以下の例では、初期の 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大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
```

4.10. 論理ボリュームの縮小化

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。 それから、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度マウントします。

注意

ボリューム自身を縮小する前に、ファイルシステム、又はボリューム内に存在するものの サイズを縮小することが重要です。そうしないとデータ喪失の恐れがあります。

論理ボリュームを縮小すると、ボリュームグループ内で他の論理ボリュームに割り当てることになる ボリュームグループの一部を開放することになります。

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論理エクステントだけ縮小しています。

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

5. スナップショットボリュームの作成

vgchange コマンドで -s 引数を使用すると、 スナップショットボリュームを作成します。スナップショットボリュームは 書き込み可能です。

LVM スナップショットには クラスタ認識がありません。そのため、ボリュームへの専用アクセスを 必要とします。クラスタ内の個別ノード上で論理ボリュームをアクティベートする方法に関する情報は 項8. 「クラスタ内の個別ノード上の論理ボリュームをアクティベート」 でご覧下さい。

以下のコマンドは、/dev/vg00/snap と言う名の 100 メガバイトサイズの スナップショット論理ボリュームを作成します。原点となる論理ボリュームに、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上でスナップショット論理ボリュームをマウントして、その ファイルシステムの内容にアクセスして、原点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、 原点のボリュームを指定すると、全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 （活動中か、停止中か）の 一覧を出力します。

以下の例では、そのスナップショットボリューム /dev/new_vg/newvgsnap が作成されてた論理ボリューム /dev/new_vg/lvol0 の ステータスを示しています。

```
# 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 lvs コマンドは、原点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トを表示します。以下の例では、 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
```

注意

Because the snapshot increases in size as the origin volume changes, it is important to monitor the percentage of the snapshot volume regularly with the lvs command to be sure it does not fill. A snapshot that is 100% full is lost completely, as a write to unchanged parts of the origin would be unable to succeed without corrupting the snapshot.

6. LVM デバイススキャンをフィルターで制御

起動時に、vgscan コマンドは、LVM ラベルを調査するために、 システム上のブロックデバイスをスキャンします。そしてどれが物理ボリュームかを判定し、 メタデータを読んで、ボリュームグループの一覧を構成します。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 に保存されています。 それ以後のコマンドがそのファイルを読み込んで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。このフィルターは簡単な正規表現の 連続で構成されており、/dev ディレクトリ内のデバイス名に適用されて、 各ブロックデバイスを受理するか、拒否するかの判定をします。

以下の例では、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。 正規表現は開放的に完全パス名に対して照合されるため、これらの例の一部は、必ずしも 最善の実践を示すものではないことに注意して下さい。例えば、 a/loop/ は a/.*loop.*/ と同等であり、/dev/solooperation/lvol1 と 適合してしまいます。

以下のフィルターは、設定ファイル内に設定されたフィルターがないため、 デフォルトの動作として、全ての発見デバイスを追加します:

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

以下のフィルターは、ドライブがメディアを含んでいない場合に、遅延を 防止する為に cdrom デバイスを削除します:

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

以下のフィルターは全てのループを追加して、全ての他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全てのループと IDE を追加して、全ての他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一番目の IDE ドライブ上にパーティション８ のみを追加して、 そして他の全てのブロックデバイスを削除します:

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

lvm.conf ファイルに関する情報には、付録 B. LVM 設定ファイル 及び、lvm.conf(5) man ページを ご覧下さい。

7. オンラインデータ移動

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

pvmove コマンドは、移動する予定のデータをセクション単位に分割して、 一時的にミラーを作成して各セクションを移動します。pvmove コマンドの 機能運用に関する情報には、pvmove(8) man ページをご覧下さい。

pvmove コマンドはミラー設定を使用するため、クラスタ認識がなく、 ボリュームへの専用アクセスを必要とします。クラスタ内の個別ノード上の論理ボリュームを アクティベートする方法については、項8. 「クラスタ内の個別ノード上の論理ボリュームをアクティベート」 をご覧下さい。

以下のコマンドは全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へ移動します:

```
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、前面の 進行状況表示を避けるようにバックグラウンドでコマンドを実行する方が良いでしょう。 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 全てのエクステントを、バックグランドで /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態を 5 秒間隔でパーセントで報告します。

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

8. クラスタ内の個別ノード上の論理ボリュームをアクティベート

クラスタ環境に LVM をインストールしている状態では、 1 つのノード上で専用に論理ボリュームを 時々アクティベートする必要があるかも知れません。例えば、pvmove コマンドは クラスタ認識がなく、ボリュームに専用のアクセスを必要とします。LVM スナップショットもボリュームに 専用のアクセスを必要とします。

論理ボリュームを１つのノード上で専用にアクティベートするには、lvchange -aey コマンドを使用します。別の方法としては、lvchange -aly コマンドを使用して 専用ではなくても、そのローカルノードのみ上で論理ボリュームをアクティベートすることも出来ます。 その後では、他のノード上で同時にそれらをアクティベートすることができます。

また、付録 C. LVM オブジェクトタグ で説明してあるように、LVM タグを使用することに よって個別ノード上で論理ボリュームをアクティベートすることもできます。更に、 設定ファイル内でノードのアクティベーションを指定することも出来ます。これは、付録 B. LVM 設定ファイル で説明してあります。

9. LVM 用のカスタム報告

LVM オブジェクトの簡潔で、カスタム可能な報告は、pvs、 lvs、及び vgs コマンドを使用して 作成することができます。これらのコマンドが生成する報告は各オブジェクト毎に１行ずつの 出力を含んでいます。それぞれの行は、オブジェクトに関連したプロパティのフィールドについて順列一覧を 持っています。オブジェクトが報告される方式には５種類があります: 物理ボリューム毎、ボリュームグループ毎、 論理ボリューム毎、物理ボリュームセグメント毎、及び論理ボリュームセグメント毎があります。

次のセクションでは以下を提供します:

- 生成された報告の形式を制御するのに使用できるコマンド引数の要約
- 各 LVM オブジェクト用に選択できるフィールドのリスト
- 生成された報告を分別する為に使用できるコマンド引数の要約。
- 報告出力の単位を指定する為の指示

9.1. 形式制御

pvs、lvs、又は 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列記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 制御することができます。

- -o 引数を使用すると、デフォルト以外を表示するフィールドに変更する ことができます。例えば、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出来るプラスサイン（+）を使用して、出力にフィールドを追加することが できます。

  以下の例では、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-tUqkCS
```

- コマンドに -v 引数を追加すると、幾つかのフィールドを含むように なります。例えば、pvs -v コマンドは、デフォルトフィールドに 加えて、DevSize と PV UUID のフィールドも表示します。

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

- --noheadings 引数は、頭書きの行を抑制します。これはスクリプトを 書くときに便利です。

  以下の例は pv_name 引数と共に --noheadings 引数を 使用して、全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。 これは、出力上で grep コマンドを使用している場合、 スクリプトで役に立ちます。

  次の例では、pvs コマンドのデフォルト出力フィールドを イコールサイン（=）で分離しています。

```
# 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 引数を使用している時にフィールドを整列させるには、 separator 引数と共に --aligned 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 か vgs コマンドで -P 引数を使用して、他の方法では出力には出ないような故障ボリュームの 情報を表示することができます。この引数が生み出す出力に関する情報は 項2.「故障デバイスの情報表示」 で ご覧下さい。

表示引数の完全な一覧には、pvs(8)、vgs(8)、 及び lvs(8) の man ページをご覧下さい。

ボリュームグループフィールドは物理ボリューム（及び物理ボリュームセグメント）フィールド、 又は 論理ボリューム（及び論理ボリュームセグメント）フィールドとの混合となる可能性がありますが、 物理ボリュームと論理ボリュームのフィールドは混合出来ません。例えば、以下のコマンドは各物理 ボリューム毎に 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
```

9.2. オブジェクト選択

このセクションでは、LVM オブジェクトについて表示できる情報を pvs、vgs、及び lvs の コマンドを使って一覧表示する表のセットを提供します。

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。 例えば、pvs コマンドでは、name は pv_name の意味で、vgs コマンドでは、 name は vg_name と解釈されます。

以下のコマンドを実行することは、pvs -o pv_free の実行と 同等なものです。

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

pvs コマンド

表 4.1. 「pvs 表示フィールド」pvs コマンドの表示引数を一覧表示し、ヘッダ表示に出るフィールド名と フィールドの説明も表示します。

引数	ヘッダ	説明
dev_size	DevSize	物理ボリュームが作成される元となる背後のデバイスのサイズ
pe_start	1st PE	背後にある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 又は e(x)ported
pv_fmt	Fmt	物理ボリュームのメタデータ形式 (lvm2 又は 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	使用される物理エクステントの数量
pv_pe_count	PE	物理エクステントの数量
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物理エクステントの開始点
pv_size	PSize	物理ボリュームのサイズ
pv_tags	PV Tags	物理ボリュームに付けられた 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

引数	ヘッダ	説明
pv_uuid	PV UUID	物理ボリュームの UUID

表 4.1. pvs 表示フィールド

pvs コマンドは、デフォルトで以下のようなフィールドを表示します: pv_name, vg_name, pv_fmt, pv_attr, pv_size, pv_free。表示は pv_name で分別されます。

```
# 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以下のフィールドを追加します: dev_size, pv_uuid。

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

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。セグメントとは、エクステントの集合のことです。 セグメントの表示は、論理ボリュームがフラグメント化（分散）しているかどうかを見るのに 便利になります。

pvs --segments コマンドは、デフォルトで以下のフィールドを 表示します: pv_name, vg_name, pv_fmt, pv_attr, pv_size, pv_free, pvseg_start, pvseg_size。この表示は、物理ボリューム内で pv_name と pvseg_size で分別されます。

```
# 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デバイスのうち、LVM が検出したデバイスを見ることができます。

```
# 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 4.2. 「vgs 表示フィールド」vgs コマンドの表示引数を一覧表示し、ヘッダ表示に出るフィールド名と フィールドの説明も表示します。

引数	ヘッダ	説明
lv_count	#LV	ボリュームグループが含んでいる論理ボリュームの数量
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

引数	ヘッダ	説明
		量（無制限には 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量（無制限には 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の数量
snap_count	#SN	ボリュームグループが含むスナップショットの数量
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、及び (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量
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、又は 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量
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付けられた LVM タグ
vg_uuid	VG UUID	ボリュームグループの UUID

表 4.2. vgs 表示フィールド

vgs コマンドは、デフォルトで、以下のフィールドを表示します: vg_name, pv_count, lv_count, snap_count, vg_attr, vg_size, vg_free。この表示は、vg_name で 分別されます。

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

vgs コマンドで -v 引数を使用すると、 デフォルトの表示に以下のフィールドを追加します : vg_extent_size, vg_uuid。

```
# vgs -v
    Finding all volume groups
    Finding volume group "new_vg"
```

```
  VG      Attr   Ext   #PV #LV #SN VSize  VFree  VG UUID
  new_vg wz--n- 4.00M   3   1   1 51.42G 51.36G jxQJ0a-ZKk0-OpMO-0118-nlwO-wwqd-fD5D32
```

lvs コマンド

表 4.3. 「lvs 表示フィールド」lvs コマンドの表示引数を一覧表示します。ヘッダ表示内に出る フィールド名とフィールドの説明も表示されます。

引数	ヘッダ	説明
chunksize chunk_size	Chunk	スナップショットボリュームの単位サイズ
copy_percent	Copy%	ミラー化論理ボリュームの同期パーセント。これは、また 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要因として背後にあるデバイス: 物理ボリューム、論理ボリューム、 そして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部分は以下のように なります: Bit 1: ボリュームタイプ: ミラー化(m)、初期同期のないミラー化(M)、基礎(o)、 pv移動(p)、スナップショット(s)、無効スナップショット(S)、仮想(v) Bit2: 権限: 書き込み(w)、読み込みのみ(r) Bit 3: 割り当てポリシー: 隣接(c)、通常(n)、場所不問(a)、相続(i)。 これは、例えば、pvmove コマンドの実行時など、 ボリュームが現在の割り当て変更に対してロックされている場合は、大文字化されます。 Bit 4: 修正済み マイナー(m) Bit 5 状態: アクティブ(a)、休止中(s)、無効スナップショット(I)、無効休止中スナップショット(S)、 表のないマップ化デバイスの存在(d)、休止中の表を持つマップ化デバイスの存在(i) Bit 6: デバイス 開放(o)
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（停止中の場合、-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（停止中の場合、-1）
lv_major	Maj	論理ボリュームの固執メジャーデバイス番号（無指定の場合

引数	ヘッダ	説明
		、-1)
lv_minor	Min	論理ボリュームの固執マイナーデバイス番号（無指定の場合、-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	論理ボリュームに添付された 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論理ボリュームの ソース物理ボリューム
origin	Origin	スナップショットボリュームの原点デバイス
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添付されている 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の現在のパーセント
stripes	#Str	論理ボリューム内のストライプ、又はミラーの数量
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

表 4.3. lvs 表示フィールド

lvs コマンドはデフォルトで、以下のようなフィールドを表示します: lv_name, vg_name, lv_attr, lv_size, origin, snap_percent, move_pv, mirror_log, copy_percent。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 により分別されます。

```
# 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
```

lvs コマンドで -v 引数を使用すると、 デフォルトの表示に以下のようなフィールドを追加します:seg_count, lv_major, lv_minor, lv_kernel_major, lv_kernel_minor, lv_uuid。

```
# lvs -v
    Finding all logical volumes
  LV         VG     #Seg Attr   LSize  Maj Min KMaj KMin Origin Snap%  Move Copy%  Log LV
UUID
  lvol0      new_vg    1 owi-a- 52.00M  -1  -1 253  3
LBy1Tz-sr23-OjsI-LT03-nHLC-y8XW-EhCl78
  newvgsnap1 new_vg    1 swi-a-  8.00M  -1  -1 253  5    lvol0    0.20
1ye1OU-1cIu-o79k-20h2-ZGF0-qCJm-CfbsIx
```

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。lvs --segments コマンドはデフォルトで以下のフィールドを表示します: lv_name, vg_name, lv_attr, stripes, segtype, seg_size。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 で分別されて、論理ボリューム内では seg_start で分別 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がそれを示します。

```
# 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以下のフィールドを追加します: seg_start, stripesize, chunksize。

```
# lvs -v --segments
    Finding all logical volumes
  LV         VG     Attr   Start SSize  #Str Type   Stripe Chunk
  lvol0      new_vg owi-a-    0  52.00M    1 linear     0     0
  newvgsnap1 new_vg swi-a-    0   8.00M    1 linear     0  8.00K
```

以下の例では、１つの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おり、その後に segments 引数を付けた lvs コマンドのデフォルト出力を示しています。

```
# 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
```

9.3. LVM 報告の分別

通常、lvs、vgs、又は pvs のコマンドの全出力は、生成して保存した後に 分別して正しくコラムに列記されるものです。--unbuffered 引数を 指定すると、それを生成直後に分別されないままの出力で表示することができます。

別の順列でコラム一覧の分別を指定するには、報告コマンドのいずれかと一緒に -O 引数を使用します。出力自身の中にこれらのフィールドを含める必要はありません。

以下の例では、物理ボリュームの名前、サイズ、及び空き領域を表示する pvs コマンドの出力を示しています。

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

以下の例では、空き領域のフィールドで分別された同じ出力を示しています。

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

以下の例では、分別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
```

逆順で分別するには、 -O 引数の後で指定するフィールドの先頭に- 印を付けます。

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

9.4. 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)、及び 人間可読表示(h) を指定できます。デフォルトの 表示は人間可読表示です。このデフォルト設定は、lvm.conf ファイルの global セクションの中で units パラメータを設定する ことにより書き換えることができます。

以下の例では、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 の乗数（1024 の倍数）で表示されます。これらのユニットは 指定を大文字化（B, K, M, G, T, H）することにより 1000 の倍数として表示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）又はカスタム単位も指定できます。

以下の例では、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
```

次の例では、pvs コマンドの出力を 4 メガバイト単位で 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
```

LVM 設定の例

この章では、一部の基本的な LVM 設定の例を提供しています。

1. LVM 論理ボリュームを ３つのディスク上に作成

この例では、new_logical_volume と言う LVM 論理ボリュームを作成しており、これは、/dev/sda1、/dev/sdb1、及び /dev/sdc1 で構成されています。

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 物理ボリュームとして ラベルを付けます。

注意

このコマンドは、/dev/sda1、/dev/sdb1、及び /dev/sdc1 上のデータを破壊します。

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

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。

```
[root@tng3-1 ~]# 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。

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

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。

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

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 GFS ファイルシステムを作成します。

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

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

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

Syncing...
All Done
```

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

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

2. ストライプ化論理ボリュームの作成

この例では、striped_logical_volume と言う LVM ストライプ化論理ボリュームを 作成しており、これは /dev/sda1、/dev/sdb1、及び /dev/sdc1 のディスクに渡ってデータをストライプ化しています。

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 物理ボリュームとしてラベルを付けます。

注意

このコマンドは、/dev/sda1、/dev/sdb1、及び /dev/sdc1 上のデータを破壊します。

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

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 striped_vol_group を作成します。

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

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

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 striped_vol_group 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。

```
[root@tng3-1 ~]# lvcreate -i3 -I4 -L2G -nstriped_logical_volume striped_vol_group
  Rounding size (512 extents) up to stripe boundary size (513 extents)
  Logical volume "striped_logical_volume" created
```

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 GFS ファイルシステムを作成します。

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

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

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

Syncing...
All Done
```

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

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

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、新しくディスクを追加しないで新規の 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 から取り込まれていて、ボリュームグループ自身は３つの物理ボリューム /dev/sda1、 /dev/sdb1、及び /dev/sdc1 で構成されています。

この手順を完了した後には、ボリュームグループ myvg は /dev/sda1 と /dev/sdb1 で構成され

ています。 ２つ目のボリュームグループ yourvg は /dev/sdc1 で構成されています。

3.1. 空き領域の判定

pvscan コマンドを使用すると、どの位の空き領域が現在ボリューム グループで利用可能かを判定することができます。

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

3.2. データの移動

pvmove コマンドを使用して、/dev/sdc1 内の 全ての使用中 物理エクステントを /dev/sdb1 に移動することができます。pvmove コマンドはその実行に長時間を要します。

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

データを移動した後は、/dev/sdc1 上の全ての領域が空きになっている ことが分ります。

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

3.3. ボリュームグループの分割

新規の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、論理ボリュームは停止している必要があります。ファイルシステムがマウントされている場合は、論理ボリュームを停止する前に 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。

ボリュームグループを停止るには lvchange コマンド、又は、 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 の活動を停止して、ボリュームグループ myvg からボリュームグループ yourvg を分離させて、物理ボリューム /dev/sdc1 をその新規のボリュームグループ yourvg に移動させます。

```
[root@tng3-1 ~]# lvchange -a n /dev/myvg/mylv
[root@tng3-1 ~]# 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。

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

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。

```
[root@tng3-1 ~]# lvcreate -L5G -n yourlv yourvg
  Logical volume "yourlv" created
```

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

新しい論理ボリューム上にファイルシステムを作成してそれをマウントすることが できます。

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

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

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

```
Syncing...
All Done

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

3.6. オリジナル論理ボリュームのアクティベーションとマウント

論理ボリューム mylv の活動を停止しましたので、マウントできるように する為には、それを再度アクティベートする必要があります。

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

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

4. 論理ボリュームからディスクの削除

この例では、ディスクを入れ替える為に、あるいは異なるボリュームの一部として ディスクを使用する為に、既存の論理ボリュームからディスクを取り外す方法を示して います。ディスクを取り外すには、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、又はディスク集合に移動しなければなりません。

4.1. エクステントを既存物理ボリュームへ移動

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 物理ボリュームに渡って分配されています。

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

/dev/sdb1 からエクステントを移動して、ボリュームグループからそれを 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のエクステントがある場合、 その削除したいデバイス上でオプション無しで pvmove コマンドを 実行すると、それらのエクステントは他のデバイスに分配されるようになります。

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

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

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。

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

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。

4.2. エクステントを新規ディスクに移動

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渡って分配されています。

```
[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/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。

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。

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

4.2.2. この新しい物理ボリュームをボリュームグループに追加します。

/dev/sdd1 を既存のボリュームグループ myvg に追加します。

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

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。

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

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

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します。

データを /dev/sdb1 から移動したら、それをボリュームグループから 削除することができます。

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

これで、このディスクを別のボリュームグループに移動するか、又はシステムから このディスクを削除することができます。

LVM トラブルシューティング

この章では、多種多様な LVM 問題に対するトラブルシューティングの案内を提供しています。

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ている場合、以下の方法で診断情報を収集できます:

- 出力の冗長レベルの順番に、-v、-vv、-vvv、 又は -vvvv 引数をいずれかのコマンドと一緒に使用します。
- 問題が論理ボリュームのアクティベーションに関連している場合は、設定ファイルの ログセクションで activation = 1 とセットして、-vvvv 引数を 付けてコマンドを実行します。この出力を検証し終わった後には、このパラメータを 0 にリセットして、低メモリー状況で起こりう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、診断目的の情報とダンプを提供します。詳細は lvmdump(8) man ページで ご覧下さい。
- 追加のシステム情報を得るには、lvs -v か pvs -a か dmsetup info -c コマンドを実行します。
- /etc/lvm/backup 内の最後のメタデータバックアップと /etc/lvm/archive 内のアーカイブバージョンを検証します。
- lvm dumpconfig コマンドを実行すると現在の設定情報を チェックします。
- どのデバイスが物理ボリュームを持っているかの記録を調べる為に /etc/lvm ディレクトリ内の .cache ファイルをチェックします。

2. 故障デバイスの情報表示

lvs か 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故障ボリュームに関する情報を表示することが出来ます。この引数により、メタデータに完全な内部統合性がなくても、 一部の操作が可能になります。例えば、ボリュームグループ vg を構成するデバイスの 1 つが故障した場合、vgs コマンドが以下のような出力を表示するでしょう。

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

vgs コマンドで -P オプションを指定すると、 ボリュームグループはまだ使用不可ですが、その故障デバイスについての情報をより多く見ることが出来ます。

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

この例では、故障デバイスはボリュームグループ内のリニアとストライプの 両方の論理ボリュームの障害原因になっています。-P 引数を付けない lvs コマンドでは、以下のような出力を出します。

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

-P 引数を使用すると、故障した論理ボリュームを表示します。

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

以下の例では、ミラー化論理ボリュームの 1 つの脚が故障した場合における、-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。

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

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

```
/dev/sdb1(0)
  [my_mirror_mlog]      corey lwi-ao   4.00M
/dev/sdd1(0)
```

3. LVM ミラー障害からの復元

このセクションでは、物理ボリュームの背後にあるデバイスが停止したことにより、 LVM ミラー化ボリュームの１つの脚が障害を起こした状態から復元する例を提供してます。ミラー脚が障害を起こすと、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。それは以前と同じように稼働しますが、ミラーの冗長がなくなります。この時点で、新規のディスクデバイスをシステムに追加して代替の物理デバイスとして使用し、ミラーを再構成できます。

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。

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

以下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。

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

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 ミラー脚用に背後にある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

注意して下さい。Copy% フィールドが 100.00 に なるまで待ってから継続する必要があります。

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

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

この例では、ミラーの主要脚 /dev/sda1 が故障しています。ミラー化 ボリュームへの書き込み行為はいずれも 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 はミラーを単独のリニアボリュームに変換します。この場合、 この変換を起こすためには、dd コマンドを実行します。

```
[root@link-08 ~]# 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、デバイスが現在リニアデバイスで あることを確認することができます。故障したディスクが原因で I/O エラーが発生します。

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

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、ミラー冗長がなくなります。

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取り替えて、物理ボリュームを再度作成し直します。新規ディスクに取り替えないで同じディスクを使用すると、 pvcreate コマンドを実行した時に、”inconsistent”の警告が 表示されることになります。

```
[root@link-08 ~]# pvcreate /dev/sda[12]
  Physical volume "/dev/sda1" successfully created
  Physical volume "/dev/sda2" successfully created

[root@link-08 ~]# pvscan
  PV /dev/sdb1   VG vg   lvm2 [67.83 GB / 67.10 GB free]
  PV /dev/sdb2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdc1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdc2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdd1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdd2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sde1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sde2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdf1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdf2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdg1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdg2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdh1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdh2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sda1           lvm2 [603.94 GB]
  PV /dev/sda2           lvm2 [603.94 GB]
  Total: 16 [2.11 TB] / in use: 14 [949.65 GB] / in no VG: 2 [1.18 TB]
```

次に、新規物理ボリュームでオリジナルボリュームグループを拡張します。

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

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

```
  PV /dev/sdg2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdh1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdh2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sda1   VG vg   lvm2 [603.93 GB / 603.93 GB free]
  PV /dev/sda2   VG vg   lvm2 [603.93 GB / 603.93 GB free]
  Total: 16 [2.11 TB] / in use: 16 [2.11 TB] / in no VG: 0 [0   ]
```

リニアボリュームをその元のミラー状態に変換し戻します。

```
[root@link-08 ~]# lvconvert -m 1 /dev/vg/groupfs /dev/sda1 /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。

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

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メタデータが間違えて書き換えられたり、破損したり した場合は、メタデータエリアが正しくないこと、又はシステムが特定の UUID を持つ物理 ボリュームを見つけることが出来ないことを示すエラーメッセージが出ます。物理ボリュームのデータを復元するには、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エリアを書き込みます。

注意

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下さい。間違えた UUID を指定するとデータ損失の原因となります。

以下の例では、メタデータエリアが欠如していたり、破損している場合に出る 出力の種類を示しています。

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

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

上書きされている物理ボリュームの UUID は、/etc/lvm/archive ディレクトリを 開けば見つけることができます。そのボリュームの最後の有効なアーカイブ化した LVM メタデータの VolumeGroupName_xxxx.vg ファイルを 確認します。

別の方法としては、そのボリュームを停止して、partial（-P）引数をセットすると欠如/破損した物理ボリュームの UUID を見つけることが出来ます。

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

pvcreate コマンドで、--uuid と --restorefile 引数を使用して、物理ボリュームの復元をします。 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を付けます。このコマンドがボリュームグループ用の最も最近の正しい 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 ボリュームグループラベルを復元します。 restorefile 引数は、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 pvcreate コマンドに 指示をして、新規のメタデータは、古い物理ボリュームが含んで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、例えば、オリジナルの pvcreate コマンドが、メタデータの 配置制御をするコマンドライン引数を使用していた場合や、物理ボリュームが本来、異なるデフォルトを 使用するソフトウェアの異なるバージョンを使用して作成されていた場合などに発生可能です）。 pvcreate コマンドは LVM メタデータエリアのみを上書きし、既存の データエリアには影響を与えません。

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

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。

```
[root@link-07 backup]# 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。

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

以下のコマンドはボリュームをアクティベートしてそのアクティブになったボリュームを表示します。

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

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、このコマンドは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 メタデータエリアを越えた場合、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と良いでしょう。

5. 紛失した物理ボリュームの入れ替え

物理ボリュームが障害を持つか、他の理由で入れ替えを必要とする場合、項4. 「物理ボリュームメタデータの復元」 に説明してあるように、物理ボリュームメタデータの復元のための手順と同じ方法に従って、既存ボリュームグループ内の紛失した物理ボリュームを入れ替えるために、新しい物理ボリュームにラベルを付けることができます。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 及び サイズを表示することができます。もう 1 つ同じサイズの物理ボリュームを入れ替えたい場合は、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. 紛失した物理ボリュームをボリュームグループから削除

物理ボリュームが無くなった場合、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ベート 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全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、何を破壊しようとしているのかを先に検

証することをお薦めします。

ほとんどの LVM 操作と同じく、vgcfgrestore コマンドを直後に使用して、 ボリュームグループ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 コマンドは反転可能です。例えば、--test 引数なしで vgreduce コマンドで、--removemissing 引数を使用して、保存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、まだ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以前の状態に戻すことができます。

7. 論理ボリュームでの不十分な空きエクステント

論理ボリュームを作成している時に、"Insufficient free extents" と言うエラーメッセージを 受けることがあります。これは vgdisplay や vgs の コマンドの出力を基にして十分なエクステントがあると思っている時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二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からです。実際のサイズを 指定するには、物理ボリュームのサイズ決定にバイトの倍数を使用せずに、空き物理エクステントカウントを 使用します。

デフォルトでは、vgdisplay コマンドは空き物理エクステントを、以下の出力行を含んで表示します。

```
# 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。

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

8780 の空き物理エクステントに、次のコマンドの実行で、小文字 l の引数を使って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
```

これが、ボリュームグループ内のすべてのエクステントを使用します。

```
# 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 することで、ボリュームグループ内の残りの空き領域のパーセント指定で使用できる論理ボリュームを 拡大することができます。

LVM GUI での LVM 管理

コマンドラインインターフェイス（CLI）の他にも、LVM にはグラフィカルユーザーインターフェイス（GUI）が 提供されており、それを LVM 論理ボリュームの設定に使用することができます。このユーティリティを立ち上げるには system-config-lvm を入力します。Red Hat Enterprise Linux 導入 ガイド の LVM の章で、このユーティリティを使用した LVM 論理ボリュームの設定に関する 段階的手順が説明してあります。

更に、LVM GUI は Conga 管理インターフェイスの一部としても利用できます。Conga での LVM GUI 使用に関する情報には、オンラインで Conga 用のヘルプをご覧下さい。

付録 A. デバイスマッパー

The Device Mapper is a kernel driver that provides a framework for volume management. It provides a generic way of creating mapped devices, which may be used as logical volumes. It does not specifically know about volume groups or metadata formats.

The Device Mapper provides the foundation for a number of higher-level technologies. In addition to LVM, Device-Mapper multipath and the dmraid command use the Device Mapper. The user interface to the Device Mapper is the ioctl system call.

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

1. Device Table Mappings

A mapped device is defined by a table that specifies how to map each range of logical sectors of the device using a supported Device Table mapping. The table for a mapped device is constructed from a list of lines of the form:

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

In the first line of the Device Mapper, the start parameter must equal 0. The start + length parameters on one line must equal the start on the next line. Which mapping parameters are specified in a line of the mapping table depends on which mapping type is specified on the line.

Sizes in the Device Mapper are always specified in sectors (512 bytes).

When a device is specified as a mapping parameter in the Device Mapper, it can be referenced by the device name in the filesystem (for example, /dev/hda) or by the major and minor numbers in the format major:minor. The major:minor format is preferred because it avoids pathname lookups.

The following shows a sample mapping table for a device. In this table there are four

linear targets:

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

The first 2 parameters of each line are the segment starting block and the length of the segment. The next keyword is the mapping target, which in all of the cases in this example is linear. The rest of the line consists of the parameters for a linear target.

The following subsections describe the format of the following mappings:

- linear
- striped
- mirror
- snapshot and snapshot-origin
- error
- zero
- multipath
- crypt

1.1. The linear Mapping Target

A linear mapping target maps a continuous range of blocks onto another block device. The format of a linear target is as follows:

```
start length linear device offset
```

start
    starting block in virtual device

length
    length of this segment

device
: block device, referenced by the device name in the filesystem by the major and minor numbers in the format major:minor

offset
: starting offset of the mapping on the device

The following example shows a linear target with a starting block in the virtual device of 0, a segment length of 1638400, a major:minor number pair of 8:2, and a starting offset for the device of 41146992.

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

The following example shows a linear target with the device parameter specified as the device /dev/hda.

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

1.2. The striped Mapping Target

The striped mapping target supports striping across physical devices. It takes as arguments the number of stripes and the striping chunk size followed by a list of pairs of device name and sector. The format of a striped target is as follows:

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

There is one set of device and offset parameters for each stripe.

start
: starting block in virtual device

length
: length of this segment

#stripes
: number of stripes for the virtual device

chunk_size
: number of sectors written to each stripe before switching to the next; must be power of 2 at least as big as the kernel page size

device
: block device, referenced by the device name in the filesystem by the major and minor numbers in the format major:minor.

offset
: starting offset of the mapping on the device

The following example shows a striped target with three stripes and a chunk size of 128:

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

0
: starting block in virtual device

73728
: length of this segment

striped 3 128
: stripe across three devices with chunk size of 128 blocks

8:9
: major:minor numbers of first device

384
: starting offset of the mapping on the first device

8:8
: major:minor numbers of second device

384
: starting offset of the mapping on the second device

8:7
: major:minor numbers of of third device

9789824
: starting offset of the mapping on the third device

The following example shows a striped target for 2 stripes with 256 KiB chunks, with the device parameters specified by the device names in the file system rather than by the major and minor numbers.

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

1.3. The mirror Mapping Target

The mirror mapping target supports the mapping of a mirrored logical device. The format of a mirrored target is as follows:

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

start
: starting block in virtual device

length
: length of this segment

log_type
: The possible log types and their arguments are as follows:

  core
  : The mirror is local and the mirror log is kept in core memory. This log type takes 1 - 3 arguments:

    regionsize [[no]sync] [block_on_error]

  disk
  : The mirror is local and the mirror log is kept on disk. This log type takes 2 - 4 arguments:

    logdevice regionsize [[no]sync] [block_on_error]

  clustered_core
  : The mirror is clustered and the mirror log is kept in core memory. This log type takes 2 - 4 arguments:

    regionsize UUID [[no]sync] [block_on_error]

  clustered_disk
  : The mirrored is clustered and the mirror log is kept on disk. This log type takes 3 - 5 arguments:

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

LVM maintains a small log which it uses to keep track of which regions are in sync with the mirror or mirrors. The regionsize argument specifies the size of these regions.

In a clustered environment, the UUID argument specifies a unique number associated with the mirror log device so that the log state can be maintained throughout the cluster.

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

#log_args
number of log arguments that will be specified in the mapping

logargs
the log arguments for the mirror; the number of log arguments provided is specified by the #log-args parameter and the valid log arguments are determined by the log_type parameter.

#devs
the number of legs in the mirror; a device and an offset is specifed for each leg.

device
block device for each mirror leg, referenced by the device name in the filesystem or by the major and minor numbers in the format major:minor. A block device and offset is specified for each mirror leg, as indicated by the #devs parameter.

offset
starting offset of the mapping on the device. A block device and offset is specified for each mirror leg, as indicated by the #devs parameter.

The following example shows a mirror mapping target for a clustered mirror with a mirror log kept on disk.

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

0
starting block in virtual device

52428800

length of this segment

mirror clustered_disk
: mirror target with a log type specifying that mirror is clustered and the mirror log is maintained on disk

4
: 4 mirror log arguments will follow

253:2
: major:minor numbers of log device

1024
: region size the mirror log uses to keep track of what is in sync

UUID
: UUID of mirror log device to maintain log information throughout a cluster

block_on_error
: mirror should respond to errors

3
: number of legs in mirror

253:3 0 253:4 0 253:5 0
: major:minor numbers and offset for devices constituting each leg of mirror

1.4. The snapshot and snapshot-origin Mapping Targets

When you create the first LVM snapshot of a volume, four Device Mapper devices are used:

1. A device with a linear mapping containing the original mapping table of the source volume.
2. A device with a linear mapping used as the copy-on-write (COW) device for the source volume; for each write, the original data is saved in the COW device of each snapshot to keep its visible content unchanged (until the COW device fills up).
3. A device with a snapshot mapping combining #1 and #2, which is the visible snapshot volume
4. The "original" volume (which uses the device number used by the original source volume), whose table is replaced by a "snapshot-origin" mapping from device #1.

A fixed naming scheme is used to create these devices, For example, you might use the following commands to create an LVM volume named base and a snapshot volume named snap based on that volume.

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

This yields four devices, which you can view with the following commands:

```
# 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
```

The format for the snapshot-origin target is as follows:

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

start
: starting block in virtual device

length
: length of this segment

origin
: base volume of snapshot

The snapshot-origin will normally have one or more snapshots based on it. Reads will be mapped directly to the backing device. For each write, the original data will be saved in the COW device of each snapshot to keep its visible content unchanged until the COW device fills up.

The format for the snapshot target is as follows:

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

start
: starting block in virtual device

length
: length of this segment

origin
: base volume of snapshot

COW-device
: Device on which changed chunks of data are stored

P|N
: P (Persistent) or N (Not persistent); indicates whether snapshot will survive after reboot. For transient snapshots (N) less metadata must be saved on disk; they can be kept in memory by the kernel.

chunksize
: Size in sectors of changed chunks of data that will be stored on the COW device

The following example shows a snapshot-origin target with an origin device of 254:11.

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

The following example shows a snapshot target with an origin device of 254:11 and a COW device of 254:12. This snapshot device is persistent across reboots and the chunk size for the data stored on the COW device is 16 sectors.

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

1.5. The error Mapping Target

With an error mapping target, any I/O to the mapped sector fails. This target is used to fill holes in devices.

Note to reviewers: This explanation needs a little beefing up. Can somebody provide a little more explanation of how an error target "fills holes in devices"? Also, What does it mean to "Swap in to discard any queued I/O" -- which another use for an error

target.

The error mapping target takes no additional parameters besides the start and length parameters.

The following example shows an error target.

```
0 65536 error
```

1.6. The zero Mapping Target

The zero mapping target is a block device equivalent of /dev/zero. A read operation to this mapping returns blocks of zeros. Data written to this mapping is discarded, but the write succeeds. The zero mapping target takes no additional parameters besides the start and length parameters.

The following example shows a zero target for a 16Tb Device.

```
0 65536 zero
```

1.7. The multipath Mapping Target

The following illustration shows the format of a multipath target with two path groups.

Note to reviewers: The following ascii representation will be an actual figure in the final document.

```
[----------- 1st path group -----------] [--------- 2nd path group -----------]
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
^     ^       ^      ^ ^ ^ ^      ^      ^ ^ ^   ^      ^
|     |       |      | | | |      |      | | |   |      nb of io to send to this path
before switching
|     |       |      | | | |      |      | | |   path major:minor numbers
|     |       |      | | | |      |      | | number of path arguments
|     |       |      | | | |      |      | number of paths in this path group
|     |       |      | | | |      |      number of selector arguments
|     |       |      | | | |      path selector
|     |       |      | | | next path group to try
|     |       |      | | number of path groups
|     |       |      | number of hwhandlers
|     |       |      number of features
|     |       target name
```

```
|     target length in 512-bytes blocks
starting offset of the target
```

Note to reviewers: What is a "feature" in the "number of features" parameter?

The following example shows a pure failover target definition for the same LU:

Note to reviewers: What does it mean here to say "pure failover target"? How does this differ from the first example? What is an LU? (Is that a typo?)

```
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
```

The following example shows a full spread (multibus) target definition for the same LU:

Note to reviewers: I need a little more explanation of what how does this differs from the previous example, in terms of what is configured. Maybe just a sentence.

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

For further information about multipathing, see the Using Device Mapper Multipath document.

1.8. The crypt Mapping Target

The crypt target encrypts the data passing through the specified device. It uses the kernel Crypto API.

The format for the crypt target is as follows:

```
start length crypt cipher key IV-offset device offset
```

start
    starting block in virtual device

length
    length of this segment

cipher

Note to reviewers: I need a bit of help with the explanation. The information I have says that this parameter consists of cipher-chainmode-ivmode:iv options, but the example I have give aes-plain as the parameter, and I'm not sure how that corresponds. Is "chainmode" optional? Are there iv options only if the ivmode requires them (as in essiv:hash)?

Cipher consists of cipher-chainmode-ivmode:iv options.

cipher
Ciphers available are listed in /proc/crypto (for example, aes).

chainmode
Always use cbc. Do not use ebc; it does not use an initial vector (IV).

ivmode:iv options
IV is an initial vector used to vary the encryption. The IV mode is plain or essiv:hash. An ivmode of -plain uses the sector number (plus IV offset) as the IV. An ivmode of -essiv is an enhancement avoiding a watermark weakness

key
Encryption key, is supplied in hex

IV-offset
Initial Vector (IV) offset

device
block device, referenced by the device name in the filesystem by the major and minor numbers in the format major:minor

offset
starting offset of the mapping on the device

The following is an example of a crypt target.

Note to reviewers: The following example does not include start and length parameters, which are required. What would reasonable start and length parameters for this example be?

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

2. The dmsetup Command

The dmsetup command is a command line wrapper for communication with the Device

Mapper. For general system information about LVM devices, you may find the info, ls, status, and deps options of the dmsetup command to be useful, as described in the following subsections.

For information about additional options and capabilities of the dmsetup command, see the dmsetup(8) man page.

2.1. The dmsetup info Command

The dmsetup info device command provides summary information about Device Mapper devices. If you do not specify a device name, the output is information about all of the currently configured Device Mapper devices. If you specify a device, then this command yields information for that device only.

The dmsetup info command provides information in the following categories:

Note to reviewers: I need some more explanation of some of these headings.

Name
: The name of the device

State
: SUSPENDED or ACTIVE, READ-ONLY

Read Ahead
: Note to reviewers: What does this category mean?

Tables present
: LIVE and/or INACTIVE

  Note to reviewers: What does this category mean? What is a live or inactive table? What is the table being referenced?

Open count
: Open reference count

  Note to reviewers: What does this mean?

Event number
: Last event sequence number (used by the wait call)

  Note to reviewers: This is another category which I think needs clarification.

Major, minor
: Major and minor device number

Number of targets
: Number of targets in the live table

UUID
: UUID of mirror log device to maintain log information throughout a cluster

The following example shows the partial output for the dmsetup info that does not specify a device. This shows the output for a local and for a clustered logical volume.

```
[root@ask-07 ~]# dmsetup info
Name:              testgfsvg-testgfslv3
State:             ACTIVE
Read Ahead:        256
Tables present:    LIVE
Open count:        0
Event number:      0
Major, minor:      253, 4
Number of targets: 1
...
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
```

2.2. The dmsetup ls Command

You can list the device names of mapped devices with the dmsetup ls command. You can list devices that have at least one target of a specified type with the dmsetup ls --target target_type command. For other options of the dmsetup ls, see the dmsetup man page.

The following example shows the command to list the device names of currently configured mapped devices.

```
[root@ask-07 ~]# dmsetup ls
testgfsvg-testgfslv3    (253, 4)
testgfsvg-testgfslv2    (253, 3)
```

```
testgfsvg-testgfslv1    (253, 2)
VolGroup00-LogVol01     (253, 1)
VolGroup00-LogVol00     (253, 0)
```

The following example shows the command to list the devices names of currently configured mirror mappings.

Note to reviewers: We'll come up with better names for the devices here.

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

2.3. The dmsetup status Command

The `dmsetup status device` command provides status information for each target in a specified device. If you do not specify a device name, the output is information about all of the currently configured Device Mapper devices. You can list the status only of devices that have at least one target of a specified type with the `dmsetup status --target target_type` command.

The following example shows the command to list the status of the targets in all currently configured mapped devices.

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

2.4. The dmsetup deps Command

The `dmsetup deps device` command provides a list of (major, minor) pairs for devices

referenced by the mapping table for the specified device. If you do not specify a device name, the output is information about all of the currently configured Device Mapper devices.

The following example shows the command to list the dependencies of all currently configured mapped devices.

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

The following example shows the command to list the dependencies only of the device lock_stress-grant--02.1722:

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

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からロードされます。このディレクトリはデフォルトでは /etc/lvm にセットしてあります。

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。全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 項2. 「ホストタグ」 でご覧下さい。

1. LVM 設定ファイル

LVM 設定には以下のようなファイルが使用されます:

/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
: 各ホストタグ用に余分の設定ファイルが存在すれば、それが読み込まれます。（lvm_hosttag.conf） そのファイルが新規のタグを定義する場合、更なる設定ファイルが読み込みの為にファイルの一覧に追記されます。ホストタグに関する情報には、項2. 「ホストタグ」 をご覧下さい。

LVM 設定ファイルの他にも、LVM を稼働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
: 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/
: 自動 グループメタデータバックアップ用ディレクトリ（設定可能）

/etc/lvm/archive/
: 自動ボリュームグループメタデータアーカイブ用ディレクトリ（ディレクトリパスと アーカイブ履歴の範囲に関して設定可能）

/var/lock/lvm/
: 単独ホストの設定では、並行ツール実行がメタデータ破損することを防止するロックファイルが使用され、 そしてクラスタでは、クラスタ全域の DLM が使用されます。

2. サンプルの lvm.conf ファイル

サンプルの lvm.conf 設定ファイルを以下に示します。

```
[root@tng3-1 lvm]# cat lvm.conf
# This is an example configuration file for the LVM2 system.
# It contains the default settings that would be used if there was no
# /etc/lvm/lvm.conf file.
#
# Refer to 'man lvm.conf' for further information including the file layout.
#
# To put this file in a different directory and override /etc/lvm set
# the environment variable LVM_SYSTEM_DIR before running the tools.


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

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

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

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

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

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

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

```
    # By default we accept every block device:
    # Steel Toe installed filter -- use this line while running tests
    filter = [ "r/hda/", "r/disk/", "a/.*/" ]
    # Steel Toe installed filter -- use this line when installing new kernels
    #filter = [ "r/disk/", "a/.*/" ]

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

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

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

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

    # The results of the filtering are cached on disk to avoid
    # rescanning dud devices (which can take a very long time).  By
    # default this cache file is hidden in the /etc/lvm directory.
    # It is safe to delete this file: the tools regenerate it.
    cache = "/etc/lvm/.cache"

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

    # Advanced settings.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
    # activation = 0
}

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

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

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

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

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

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

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

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

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



# Miscellaneous global LVM2 settings
global {
    library_dir = "/usr/lib"
```

```
# 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

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

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

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

# Location of proc filesystem
proc = "/proc"

# Type of locking to use. Defaults to local file-based locking (1).
# Turn locking off by setting to 0 (dangerous: risks metadata corruption
# if LVM2 commands get run concurrently).
# Type 2 uses the external shared library locking_library.
# Type 3 uses built-in clustered locking.
locking_type = 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"

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

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

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

activation {
    # Device used in place of missing stripes if activating incomplete volume.
    # For now, you need to set this up yourself first (e.g. with 'dmsetup')
    # For example, you could make it return I/O errors using the 'error'
    # target or make it return zeros.
    missing_stripe_filler = "/dev/ioerror"

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

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

    # Nice value used while devices suspended
    process_priority = -18
```

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

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

    # 'mirror_image_fault_policy' and 'mirror_log_fault_policy' define
    # how a device failure affecting a mirror is handled.
    # A mirror is composed of mirror images (copies) and a log.
    # A disk log ensures that a mirror does not need to be re-synced
    # (all copies made the same) every time a machine reboots or crashes.
    #
    # In the event of a failure, the specified policy will be used to
    # determine what happens:
    #
    # "remove" - Simply remove the faulty device and run without it.  If
    #            the log device fails, the mirror would convert to using
    #            an in-memory log.  This means the mirror will not
    #            remember its sync status across crashes/reboots and
    #            the entire mirror will be re-synced.  If a
    #            mirror image fails, the mirror will convert to a
    #            non-mirrored device if there is only one remaining good
    #            copy.
    #
    # "allocate" - Remove the faulty device and try to allocate space on
    #            a new device to be a replacement for the failed device.
    #            Using this policy for the log is fast and maintains the
    #            ability to remember sync state through crashes/reboots.
    #            Using this policy for a mirror device is slow, as it
    #            requires the mirror to resynchronize the devices, but it
    #            will preserve the mirror characteristic of the device.
    #            This policy acts like "remove" if no suitable device and
    #            space can be allocated for the replacement.
    #            Currently this is not implemented properly and behaves
    #            similarly to:
    #
    # "allocate_anywhere" - Operates like "allocate", but it does not
    #            require that the new space being allocated be on a
    #            device is not part of the mirror.  For a log device
    #            failure, this could mean that the log is allocated on
```

```
    #           the same device as a mirror device.  For a mirror
    #           device, this could mean that the mirror device is
    #           allocated on the same device as another mirror device.
    #           This policy would not be wise for mirror devices
    #           because it would break the redundant nature of the
    #           mirror.  This policy acts like "remove" if no suitable
    #           device and space can be allocated for the replacement.

    mirror_log_fault_policy = "allocate"
    mirror_device_fault_policy = "remove"
}


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

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

    # pvmetadatacopies = 1

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

    # pvmetadatasize = 255

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

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

# Event daemon
#
# dmeventd {
    # mirror_library is the library used when monitoring a mirror device.
    #
    # "libdevmapper-event-lvm2mirror.so" attempts to recover from failures.
    # It removes failed devices from a volume group and reconfigures a
    # mirror as necessary.
    #
    # mirror_library = "libdevmapper-event-lvm2mirror.so"
#}
```

付録 C. LVM オブジェクトタグ

LVM タグとは、同じタイプの LVM2 オブジェクトを一緒にグループ化する為に使用する言葉です。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 オブジェクトに付けることができます。タグはクラスタ設定ではホストに添付されます。スナップショットにはタグを付けることが出来ません。

タグは、コマンドラインで引数 PV、VG、や LV の代わりに表示することができます。 タグは混乱を防ぐために @ を前に付ける必要があります。各タグは、コマンドライン上のその場所で想定されるタイプのタグを持つ全てのオブジェクトで、そのタグを入れ替えて拡張できます。

LVM タグは、最長 128 文字までの [A-Za-z0-9_+.-] を使用した文字列です。これは ハイフンを最初に持つことは出来ません。

ボリュームグループ内のオブジェクトだけ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排除された場合は、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ており、物理ボリュームが排除 された時にはそれが無くなることが理由です。スナップショットにはタグを付けられません。

以下のコマンドは、database タグを持つ全ての 論理ボリュームを一覧表示します。

```
lvs @database
```

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 か、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 か、lvcreate コマンドで --addtag や --deltag オプションを使用します。

2. ホストタグ

クラスタ構成では、設定ファイル内でホストタグを定義することができます。tags セクションで hosttags = 1 とセットした場合、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。これにより全ての使用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は異なる動作を するようになります。

設定ファイルに関する情報には、付録 B. LVM 設定ファイル をご覧下さい。

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込みの為にファイルの一覧に追記されます。

例えば、設定ファイル内の以下のエントリは常に、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"] } }
```

3. タグでアクティベーションの制御

特定の論理ボリュームのみがホスト上でアクティベートされるように設定ファイルで指定することが出来ます。例えば、以下のエントリはアクティベーション要求（vgchange -ay など）のフィルターとして動作して、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、又はボリュームグループのみを アクティベートします。

```
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、クラスタ内の各マシンがその設定ファイル内に以下のエントリを 持っている状況を考慮します:

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

ホスト db2 上のみで vg1/lvol2 を アクティベートしたい場合は、以下のようにします:

1. クラスタ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、メタデータの同一コピーが、ボリュームグループ内の全ての物理ボリュームの全ての メタデータエリアで維持されています。LVM ボリュームグループメタデータは小容量の 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無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で達成できます。物理ボリュームが含むべきメタデータコピーの 数量を選択すると、もうそれは後で変更できません。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、注意することは、全てのボリュームグループは常時、メタデータエリア 1 つを持つ 物理ボリュームを最低 1 つ含む必要があることです。（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 ファイルシステム内に保存できる場合を除く）将来、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 つのメタデータが必要となります。

核となるメタデータは ASCII 形式で保存されます。メタデータエリアは循環バッファです。新規のメタデータは古いメタデータに追記され、それからその開始点への指標が更新されます。

メタデータエリアのサイズは、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、多数の論理ボリュームや物理ボリュームを持つボリュームグループには小さすぎます。

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 番めの 512-byte 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が含むもの:

- 物理ボリューム UUID
- ブロックデバイスの容量 （バイト）
- データエリアロケーションの NULL-終了一覧
- メタデータエリアロケーションの NULL-終了一覧

メタデータロケーションはオフセット及びサイズ（バイト）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

データエリアと最大で２つのメタデータエリアです。

2. メタデータの内容

ボリュームグループメタデータが含むもの:

- それが作成された手段と時期の情報
- 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が含むもの:

- 名前と独自の識別子
- メタデータが更新される度に上昇するバージョン番号
- プロパティ: 読み込み/書き込み？ サイズ変更可能？
- 含まれる物理ボリュームと論理ボリュームの数量に対する管理制限
- エクステントのサイズ（512 byte 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自由配列一覧、それぞれ以下を含む:
  - UUID：それを含有するブロックデバイスの決定に使用
  - プロパティ：物理ボリュームの割り当て可能性など
  - 物理ボリューム内の一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）
  - エクステントの数量
- 論理ボリュームの自由配列一覧、それぞれ以下を含む
  - 論理ボリュームセグメントの順番配列一覧。それぞれのセグメント用にメタデータは 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番配列一覧に適用 するマッピングを含んでいます。

3. サンプルのメタデータ

myvg と呼ばれる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
                        ]
                }
        }
}
```

```
}
```

目次

シンボル

C

L

M

P

V