Red Hat Enterprise Linux 7

논리 볼륨 관리자 관리

LVM 논리 볼륨 구성 및 관리

Last Updated: 2023-11-18

Red Hat Enterprise Linux 7 논리 볼륨 관리자 관리

LVM 논리 볼륨 구성 및 관리

Steven Levine
Red Hat Customer Content Services
slevine@redhat.com

법적 공지

초록

이 문서에서는 클러스터형 환경에서 LVM 실행 정보를 포함하여 LVM 논리 볼륨 관리자를 설명합니다.

차례

1장. LVM 논리 볼륨 관리자

이 장에서는 Red Hat Enterprise Linux 7의 초기 릴리스 이후 새로운 LVM 논리 볼륨 관리자의 기능에 대해 설명합니다. 이 장에서는 LVM(Logical Volume Manager)의 구성 요소에 대한 간략한 개요도 제공합니다.

1.1. 새로운 기능 및 변경된 기능

이 섹션에는 Red Hat Enterprise Linux 7의 초기 릴리스 이후 새로운 LVM 논리 볼륨 관리자의 기능이 나열됩니다.

1.1.1. Red Hat Enterprise Linux 7.1의 새로운 기능 및 변경된 기능

Red Hat Enterprise Linux 7.1에는 다음 문서 및 기능 업데이트 및 변경 사항이 포함되어 있습니다.

- 씬 프로비저닝된 볼륨 및 씬 프로비저닝된 스냅샷에 대한 문서가 명확히 되어 있습니다. LVM 씬 프로비저닝에 대한 추가 정보는 이제 **lvmthin**(7) 도움말 페이지에 있습니다. 씬 프로비저닝된 논리 볼륨에 대한 일반적인 정보는 2.3.4절. "씬 프로비저닝된 논리 볼륨(Thin Volumes)" 을 참조하십시오. 씬 프로비저닝된 스냅샷 볼륨에 대한 자세한 내용은 2.3.6절. "씬 프로비저닝된 스냅샷 볼륨" 을 참조하십시오.

- 이 설명서는 이제 B.2절. "**lvmconfig** 명령" 에 **lvm dumpconfig** 명령을 문서화합니다. Red Hat Enterprise Linux 7.2 릴리스에서는 이 명령의 이름이 **lvmconfig** 로 변경되었지만 이전 형식은 계속 작동합니다.

- 이 설명서는 B.3절. "LVM 프로필" 에 LVM 프로필을 문서화합니다.

- 이 설명서는 이제 3.6절. "**lvm** 명령을 사용하여 LVM 정보 표시" 에 **lvm** 명령을 문서화합니다.

- Red Hat Enterprise Linux 7.1 릴리스에서는 4.4.20절. "논리 볼륨 활성화 제어" 에 설명된 대로 **lvcreate** 및 **lvchange** 명령의 -k 및 -K 옵션을 사용하여 씬 풀 스냅샷 활성화를 제어할 수 있습니다.

- 이 매뉴얼은 Cryostat **import** 명령의 **--force** 인수를 문서화합니다. 이를 통해 누락된 물리 볼륨인 볼륨 그룹을 가져오고 나중에 Cryostatreduce **--removemissing** 명령을 실행할 수 있습니다. Cryostat **import 명령에 대한** 자세한 내용은 4.3.15절. "다른 시스템으로 볼륨 그룹 이동" 을 참조하십시오.

- 이 매뉴얼은 Cryostatreduce 명령의 **--mirrorsonly** 인수 를 문서화합니다. 이를 통해 실패한 물리 볼륨에서 미러 이미지인 논리 볼륨만 제거할 수 있습니다. 이 옵션 사용에 대한 자세한 내용은 4.3.15절. "다른 시스템으로 볼륨 그룹 이동" 을 참조하십시오.

또한 문서를 통해 소규모 기술 수정 및 설명이 수행되었습니다.

1.1.2. Red Hat Enterprise Linux 7.2의 새로운 기능 및 변경된 기능

Red Hat Enterprise Linux 7.2에는 다음 문서 및 기능 업데이트 및 변경 사항이 포함되어 있습니다.

- 많은 LVM 처리 명령에서 **-S** 또는 **--select** 옵션을 사용하여 해당 명령에 대한 선택 기준을 정의합니다. LVM 선택 기준이 새 부록 부록 C. *LVM 선택 기준* 에 설명되어 있습니다.

- 이 문서에서는 4.4.8절. "LVM 캐시 논리 볼륨 생성" 에서 캐시 논리 볼륨을 생성하기 위한 기본 절차를 설명합니다.

- 이 문서의 문제 해결 장에는 6.7절. "다중 경로 장치에 대한 PV 경고 중복" 새 섹션이 포함되어 있습니다.

- Red Hat Enterprise Linux 7.2 릴리스에서는 이전 형식이 계속 작동하지만 **lvm dumpconfig** 명령의 이름이 **lvmconfig** 로 변경되었습니다. 이러한 변경 사항은 이 문서 전체에 반영됩니다.

또한 문서를 통해 소규모 기술 수정 및 설명이 수행되었습니다.

1.1.3. Red Hat Enterprise Linux 7.3의 새로운 기능 및 변경된 기능

Red Hat Enterprise Linux 7.3에는 다음 문서 및 기능 업데이트 및 변경 사항이 포함되어 있습니다.

- LVM은 RAID0 세그먼트 유형을 지원합니다. RAID0은 스트라이프 크기 단위로 여러 데이터 하위 볼륨에 논리 볼륨 데이터를 분산합니다. RAID0 볼륨 생성에 대한 자세한 내용은 4.4.3.1절. "RAID0 볼륨 생성(Red Hat Enterprise Linux 7.3 및 later)" 을 참조하십시오.

- **lvm fullreport** 명령을 사용하여 물리 볼륨, 볼륨 그룹, 논리 볼륨 세그먼트, 물리 볼륨 세그먼트 및 논리 볼륨 세그먼트에 대한 정보를 한 번에 보고할 수 있습니다. 이 명령 및 해당 기능에 대한 자세한 내용은 **lvm-fullreport**(8) 도움말 페이지를 참조하십시오.

- LVM은 LVM 명령 실행 중에 수집된 전체 오브젝트 식별을 통해 작업 로그, 메시지 및 개체별 상태를 포함하는 로그 보고서를 지원합니다. LVM 로그 보고서의 예는 4.8.6절. "명령 로그 보고(Red Hat Enterprise Linux 7.3 이상)" 를 참조하십시오. LVM 로그 보고서에 대한 자세한 내용은 **lvmreport**(7) 도움말 페이지를 참조하십시오.

- LVM 표시 명령의 **--reportformat** 옵션을 사용하여 JSON 형식으로 출력을 표시할 수 있습니다. JSON 형식으로 표시된 출력 예는 4.8.5절. "JSON 형식 출력(Red Hat Enterprise Linux 7.3 이상)" 를 참조하십시오.

- 이제 **lvm.conf** 구성 파일에서 **record_lvs_history** 메타데이터 옵션을 활성화하여 제거된 thin snapshot 및 thin 논리 볼륨을 추적하도록 시스템을 구성할 수 있습니다. 이를 통해 원래 종속성 체인에서 제거되어 과거 논리 볼륨이 된 논리 볼륨이 포함된 전체 씬 스냅샷 종속성 체인을 표시할 수 있습니다. 기록 논리 볼륨에 대한 자세한 내용은 4.4.21절. "추적 및 표시 논리 볼륨 (Red Hat Enterprise Linux 7.3 및 later)" 을 참조하십시오.

또한 문서를 통해 소규모 기술 수정 및 설명이 수행되었습니다.

1.1.4. Red Hat Enterprise Linux 7.4의 새로운 기능 및 변경된 기능

Red Hat Enterprise Linux 7.4에는 다음 문서 및 기능 업데이트 및 변경 사항이 포함되어 있습니다.

- Red Hat Enterprise Linux 7.4는 RAID 인수 및 RAID 회전을 지원합니다. 이러한 기능에 대한 요약은 4.4.3.12절. "RAID takeover(Red Hat Enterprise Linux 7.4 및 later)" 및 4.4.3.13절. "RAID 논리 볼륨 조정(Red Hat Enterprise Linux 7.4 및 later)" 을 참조하십시오.

1.2. 논리 볼륨

볼륨 관리는 물리 스토리지에 대한 추상화 계층을 생성하여 논리 스토리지 볼륨을 생성할 수 있습니다. 이를 통해 물리적 스토리지를 직접 사용하는 것보다 여러 가지 면에서 유연성이 훨씬 향상됩니다. 논리 볼륨의 경우 물리 디스크 크기로 제한되지 않습니다. 또한 하드웨어 스토리지 구성은 소프트웨어에서 숨겨져 있으므로 애플리케이션을 중지하거나 파일 시스템을 마운트 해제하지 않고도 크기를 조정하고 이동할 수 있습니다. 이는 운영 비용을 줄일 수 있습니다.

논리 볼륨은 물리적 스토리지를 직접 사용하는 것보다 다음과 같은 이점을 제공합니다.

- 유연한 용량

  논리 볼륨을 사용하는 경우 디스크와 파티션을 단일 논리 볼륨으로 집계할 수 있으므로 파일 시스템은 여러 디스크에 걸쳐 확장될 수 있습니다.

- 크기 조정 가능한 스토리지 풀

  기본 디스크 장치를 다시 포맷하고 다시 파티션하지 않고도 논리 볼륨을 확장하거나 간단한 소프트웨어 명령으로 크기를 줄일 수 있습니다.

- 온라인 데이터 재배치

  최신의, 속도 또는 더 유연한 스토리지 하위 시스템을 배포하기 위해 시스템을 활성화하는 동안 데이터를 이동할 수 있습니다. 디스크가 사용 중인 상태에서 데이터를 다시 정렬할 수 있습니다. 예를 들어 제거하기 전에 hot-swappable 디스크를 비워 둘 수 있습니다.

- 편리한 장치 이름 지정

  논리 스토리지 볼륨은 사용자 정의 및 사용자 지정 명명된 그룹에서 관리할 수 있습니다.

- 디스크 스트라이핑

  두 개 이상의 디스크에 데이터를 스트라이프하는 논리 볼륨을 만들 수 있습니다. 이는 처리량을 크게 높일 수 있습니다.

- 볼륨 미러링

  논리 볼륨은 데이터의 미러를 구성하는 편리한 방법을 제공합니다.

- 볼륨 스냅샷

  논리 볼륨을 사용하면 일관된 백업에 대해 장치 스냅샷을 사용하거나 실제 데이터에 영향을 주지 않고 변경 효과를 테스트할 수 있습니다.

LVM에서 이러한 기능의 구현은 이 문서의 나머지 부분에서 설명합니다.

1.3. LVM 아키텍처 개요

참고

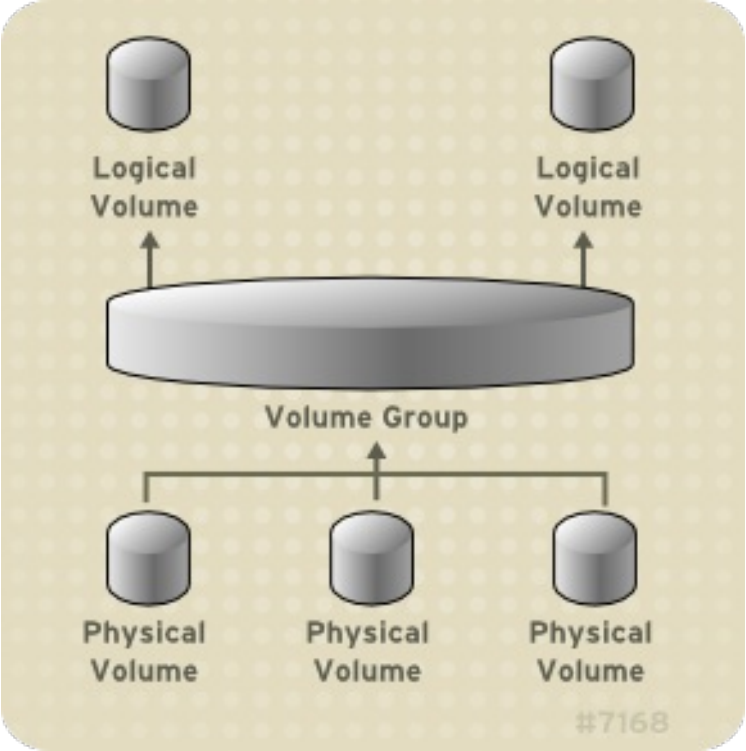
LVM2는 스냅샷 및 클러스터 지원을 제외하고 LVM1과 이전 버전과 호환됩니다. Cryostatconvert 명령을 사용하여 볼륨 그룹을 LVM1 형식에서 LVM2 형식으로 **변환할** 수 있습니다. LVM 메타데이터 형식을 변환하는 방법에 대한 자세한 내용은 Cryostat **convert**(8) 도움말 페이지를 참조하십시오.

LVM 논리 볼륨의 기본 물리 저장 단위는 파티션 또는 전체 디스크와 같은 블록 장치입니다. 이 장치는 LVM *물리 볼륨* (PV)으로 초기화됩니다.

LVM 논리 볼륨을 만들기 위해 물리 볼륨은 *볼륨 그룹* (VG)으로 결합됩니다. 이렇게 하면 LVM 논리 볼륨(LV)을 할당할 수 있는 디스크 공간 풀이 생성됩니다. 이 프로세스는 디스크를 파티션으로 분할하는 방식과 유사합니다. 논리 볼륨은 파일 시스템 및 애플리케이션(예: 데이터베이스)에서 사용합니다.

그림 1.1. "LVM 논리 볼륨 구성 요소" 는 간단한 LVM 논리 볼륨의 구성 요소를 표시합니다.

그림 1.1. LVM 논리 볼륨 구성 요소



[D]

LVM 논리 볼륨의 구성 요소에 대한 자세한 내용은 2장. *LVM 구성 요소* 을 참조하십시오.

1.4. RED HAT HIGH AVAILABILITY CLUSTER의 LVM 논리 볼륨

Red Hat High Availability Add-On에서는 두 가지 고유한 클러스터 구성에서 LVM 볼륨을 지원합니다.

- 클러스터의 단일 노드만 한 번에 스토리지에 액세스하는 활성/수동 장애 조치 구성의 HA-LVM(고가용성 LVM 볼륨)입니다.
- 클러스터의 두 개 이상의 노드가 동시에 스토리지에 액세스해야 하는 활성/활성 구성에서 클러스터형 논리 볼륨(CLVM) 확장을 사용하는 LVM 볼륨. CLVM은 스토리지 장애 복구 애드온의 일부입니다.

1.4.1. CLVM 또는 HA-LVM 선택

CLVM 또는 HA-LVM을 사용하는 경우 배포 중인 애플리케이션 또는 서비스 요구 사항을 기반으로 해야 합니다.

- 클러스터의 여러 노드에 활성/활성 시스템에서 LVM 볼륨에 대한 동시 읽기/쓰기 액세스 권한이 필요한 경우 CLVMD를 사용해야 합니다. CLVMD는 클러스터의 노드 간에 LVM 볼륨의 활성화 및 변경 사항을 조정하고 변경할 수 있는 시스템을 제공합니다. CLVMD 클러스터형 잠금 서비스는 클러스터의 다양한 노드와 볼륨과 상호 작용하고 레이아웃을 변경할 때 LVM 메타데이터를 보호합니다. 이러한 보호는 **lvm.conf** 파일에서 **locking_type** 을 3으로 설정하고 CLVMD에서 관리하고 여러 클러스터 노드에서 동시에 활성화할 모든 볼륨 그룹에 클러스터형 플래그를 설정하는 등 해당 볼륨 그룹을 적절하게 구성할 때 발생합니다.
- 고가용성 클러스터가 한 번에 지정된 LVM 볼륨에 대한 액세스 권한이 하나만 있는 활성/수동 방식으로 공유 리소스를 관리하도록 구성된 경우 CLVMD 클러스터형 잠금 서비스 없이 HA-LVM을 사용할 수 있습니다.

대부분의 애플리케이션은 다른 인스턴스와 동시에 실행되도록 설계되거나 최적화되지 않으므로 활성/수동 구성에서 더 잘 실행됩니다. 클러스터형 논리 볼륨에서 클러스터가 인식되지 않는 애플리케이션을 실행하도록 선택하면 논리 볼륨이 미러링된 경우 성능이 저하될 수 있습니다. 이러한 인스턴스에 논리 볼륨 자체의 클러스터 통신 오버헤드가 있기 때문입니다. 클러스터 인식 애플리케이션은 클러스터 파일 시스템 및 클러스터 인식 논리 볼륨에서 도입한 성능 손실보다 성능이 향상될 수 있어야 합니다. 이는 일부 애플리케이션과 워크로드에서 다른 애플리케이션 및 워크로드보다 쉽게 수행할 수 있습니다. 클러스터의 요구사항이 무엇인지, 활성/활성 클러스터 최적화를 위한 추가 노력은 두 LVM 변형 중 하나를 선택하는 방법입니다. 대부분의 사용자는 HA-LVM을 사용하여 최상의 HA 결과를 얻을 수 있습니다.

HA-LVM 및 CLVM은 LVM 메타데이터와 해당 논리 볼륨이 손상되는 것을 방지하기 때문에 여러 시스템에서 중복을 변경할 수 있는 경우 발생할 수 있습니다. HA-LVM은 논리 볼륨을 독점적으로만 활성화할 수 있는 제한을 적용합니다. 즉, 한 번에 하나의 시스템에서만 활성화됩니다. 즉, 스토리지 드라이버의 로컬(클러스터되지 않음) 구현만 사용됩니다. 이러한 방식으로 클러스터 조정 오버헤드를 방지하면 성능이 향상됩니다. CLVM은 이러한 제한 사항을 적용하지 않으며 사용자는 클러스터의 모든 시스템에서 논리 볼륨을 자유롭게 활성화할 수 있습니다. 따라서 클러스터 인식 스토리지 드라이버를 강제로 사용하므로 클러스터 인식 파일 시스템 및 애플리케이션을 맨 위에 배치할 수 있습니다.

1.4.2. 클러스터에서 LVM 볼륨 구성

Red Hat Enterprise Linux 7에서 클러스터는 Pacemaker를 통해 관리됩니다. HA-LVM 및 CLVM 논리 볼륨 모두 Pacemaker 클러스터와 함께 지원되며 클러스터 리소스로 구성해야 합니다.

- Pacemaker 클러스터의 일부로 HA-LVM 볼륨을 구성하는 절차는 High Availability Add-On Administration 의 Red Hat High Availability Cluster의 Active/passive Apache HTTP Server 를 참조하십시오. 이 절차에는 다음 단계가 포함됩니다.
  - LVM 논리 볼륨 구성
  - 클러스터만 볼륨 그룹을 활성화할 수 있는지 확인
  - LVM 볼륨을 클러스터 리소스로 구성
- 클러스터에서 CLVM 볼륨을 구성하는 절차는 Global File System 2 의 클러스터에서 GFS2 파일 시스템 구성을 참조하십시오.

2장. LVM 구성 요소

이 장에서는 LVM 논리 볼륨의 구성 요소에 대해 설명합니다.

2.1. 물리 볼륨

LVM 논리 볼륨의 기본 물리 저장 단위는 파티션 또는 전체 디스크와 같은 블록 장치입니다. LVM 논리 볼륨에 장치를 사용하려면 장치를 물리 볼륨(PV)으로 초기화해야 합니다. 블록 장치를 물리 볼륨으로 초기화하면 장치 시작 근처에 레이블이 배치됩니다.

기본적으로 LVM 레이블은 두 번째 512바이트 섹터에 배치됩니다. 물리 볼륨을 생성할 때 처음 4개 섹터에 라벨을 배치하여 이 기본값을 덮어쓸 수 있습니다. 이렇게 하면 필요한 경우 LVM 볼륨이 이러한 섹터의 다른 사용자와 공존할 수 있습니다.

LVM 레이블은 물리적 장치에 대해 올바른 식별 및 장치 순서를 제공합니다. 시스템이 부팅될 때 장치가 임의의 순서로 표시될 수 있기 때문입니다. LVM 레이블은 재부팅 및 클러스터 전체에서 유지됩니다.

LVM 레이블은 장치를 LVM 물리 볼륨으로 식별합니다. 물리 볼륨에 대한 임의의 고유 식별자(UUID)를 포함합니다. 또한 블록 장치의 크기(바이트)를 저장하고 LVM 메타데이터가 장치에 저장될 위치를 기록합니다.

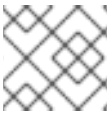
LVM 메타데이터에는 시스템의 LVM 볼륨 그룹의 구성 세부 정보가 포함되어 있습니다. 기본적으로 메타데이터의 동일한 사본은 볼륨 그룹 내의 모든 물리 볼륨의 모든 메타데이터 영역에서 유지 관리됩니다. LVM 메타데이터는 크기가 작고 ASCII로 저장됩니다.

현재 LVM을 사용하면 각 물리 볼륨에서 0, 1 또는 2의 동일한 메타데이터 복사본을 저장할 수 있습니다. 기본값은 1 복사입니다. 물리 볼륨에 메타데이터 복사본 수를 구성하면 나중에 해당 번호를 변경할 수 없습니다. 첫 번째 사본은 라벨 직후에 장치 시작에 저장됩니다. 두 번째 복사본이 있으면 장치 끝에 배치됩니다. 의도와 다른 디스크에 작성하여 디스크의 시작 부분에 영역을 실수로 덮어 쓰기하면 장치 끝에 두 번째 메타데이터 복사본을 복구 할 수 있습니다.

LVM 메타데이터 및 메타데이터 매개변수 변경에 대한 자세한 내용은 부록 E. *LVM 볼륨 그룹 메타데이터*을 참조하십시오.

2.1.1. LVM 물리 볼륨 레이아웃

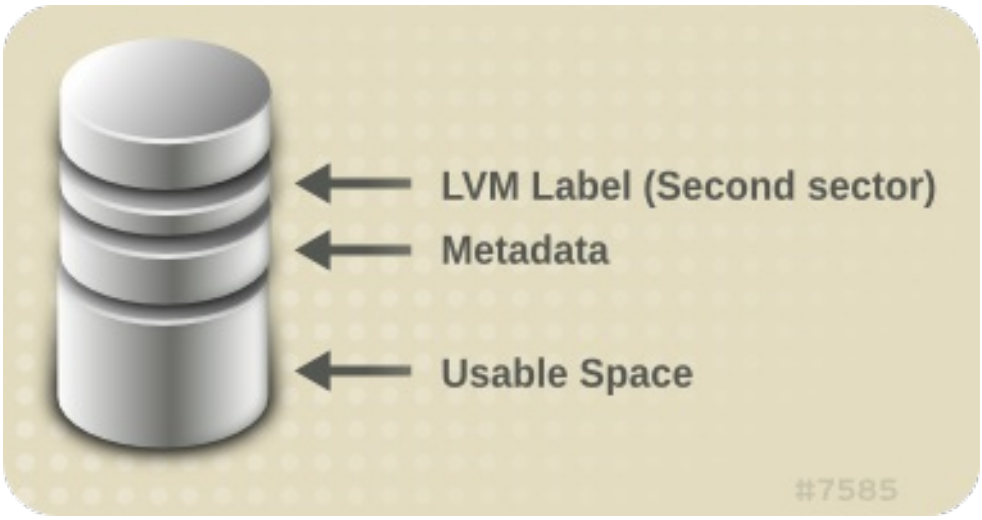
그림 2.1. "물리 볼륨 레이아웃" LVM 물리 볼륨의 레이아웃을 표시합니다. LVM 레이블은 두 번째 섹터에 있으며, 메타데이터 영역 다음에 장치에서 사용 가능한 공간이 뒤에 있습니다.



참고

Linux 커널(및 이 문서 전체에서) 섹터는 크기가 512바이트인 것으로 간주됩니다.

그림 2.1. 물리 볼륨 레이아웃



[D]

2.1.2. 디스크의 여러 파티션

LVM을 사용하면 디스크 파티션에서 물리 볼륨을 만들 수 있습니다. 다음과 같은 이유로 LVM 물리 볼륨으로 레이블하기 위해 전체 디스크를 다루는 단일 파티션을 생성하는 것이 좋습니다.

- 관리 편의성

  각 실제 디스크가 한 번만 표시되는 경우 시스템에서 하드웨어를 추적하는 것이 더 쉽습니다. 디스크가 실패하면 특히 그렇습니다. 또한 단일 디스크의 여러 물리 볼륨이 부팅 시 알 수 없는 파티션 유형에 대한 커널 경고를 유발할 수 있습니다.

- 성능 제거

  LVM에서 두 개의 물리 볼륨이 동일한 물리 디스크에 있음을 확인할 수 없습니다. 두 개의 물리 볼륨이 동일한 물리 디스크에 있는 경우 스트라이프가 동일한 디스크의 다른 파티션에 있을 수 있습니다. 이로 인해 증가하지 않고 성능이 저하됩니다.

권장되지는 않지만 디스크를 별도의 LVM 물리 볼륨으로 분할해야 하는 경우 특정 상황이 있을 수 있습니다. 예를 들어 디스크가 거의 없는 시스템에서는 기존 시스템을 LVM 볼륨으로 마이그레이션할 때 파티션 주위에 데이터를 이동해야 할 수 있습니다. 또한 매우 큰 디스크가 있고 관리 목적으로 두 개 이상의 볼륨 그룹이 필요한 경우 디스크를 분할해야 합니다. 두 개 이상의 파티션이 있고 두 파티션 모두 동일한 볼륨 그룹에 있는 디스크가 있는 경우 스트립 볼륨을 생성할 때 논리 볼륨에 포함할 파티션을 지정합니다.

2.2. 볼륨 그룹

물리 볼륨은 볼륨 그룹(VG)으로 결합됩니다. 그러면 논리 볼륨을 할당할 수 있는 디스크 공간 풀이 생성됩니다.

볼륨 그룹 내에서 할당에 사용 가능한 디스크 공간이 확장 영역이라는 고정 크기의 단위로 나뉩니다. 익스텐트는 할당할 수 있는 가장 작은 공간 단위입니다. 물리 볼륨 내에서 확장 영역을 물리 확장 영역으로 지정합니다.

논리 볼륨은 물리 확장 영역과 동일한 크기의 논리 확장 영역에 할당됩니다. 따라서 확장 크기는 볼륨 그룹의 모든 논리 볼륨에 대해 동일합니다. 볼륨 그룹은 논리 확장 영역을 물리 확장 영역에 매핑합니다.

2.3. LVM 논리 볼륨

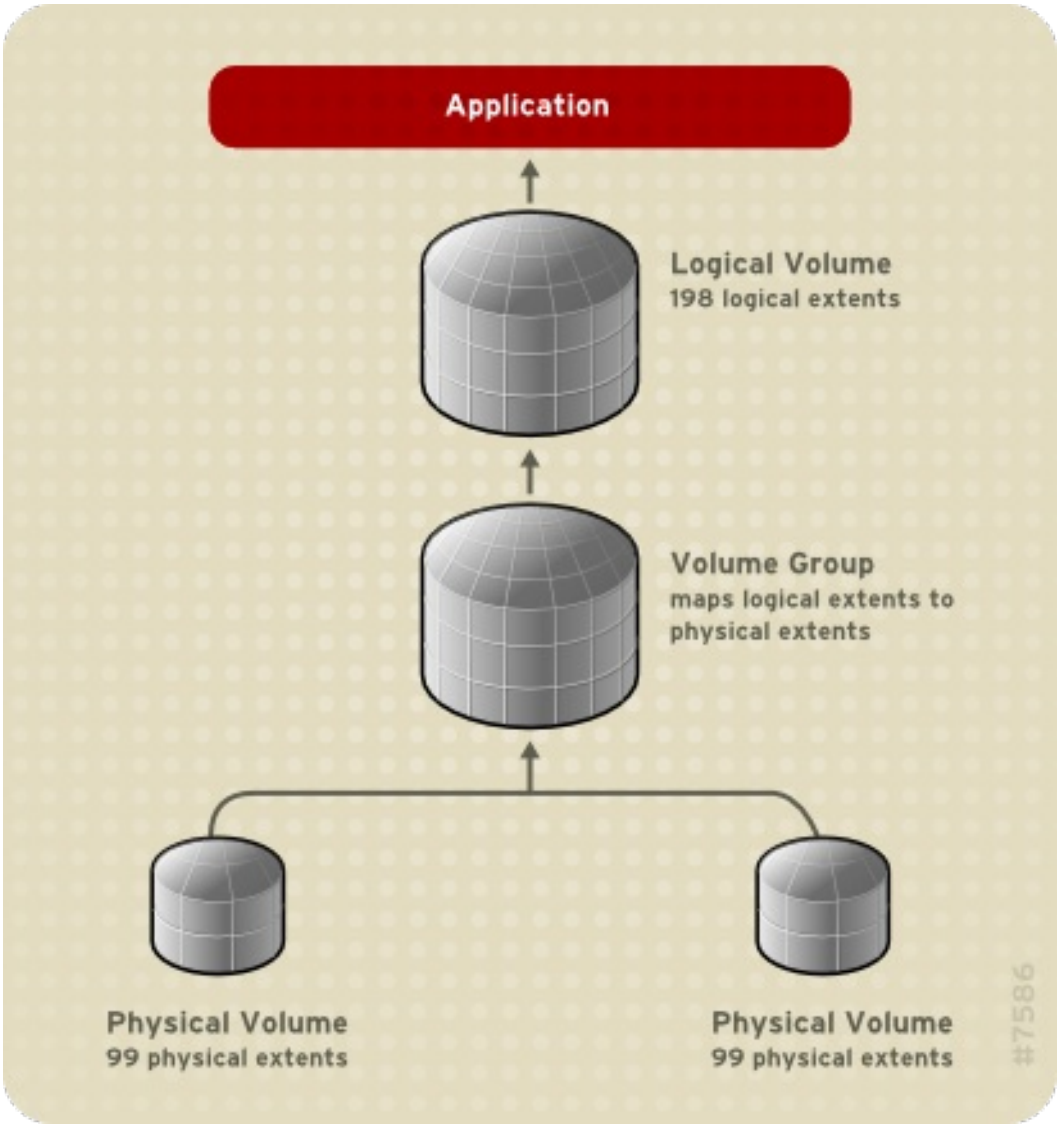
LVM에서 볼륨 그룹은 논리 볼륨으로 나뉩니다. 다음 섹션에서는 다양한 유형의 논리 볼륨에 대해 설명합니다.

2.3.1. 선형 볼륨

선형 볼륨은 하나 이상의 물리 볼륨에서 하나의 논리 볼륨으로 공간을 집계합니다. 예를 들어, 두 개의 60GB 디스크가 있는 경우 120GB 논리 볼륨을 생성할 수 있습니다. 물리적 스토리지가 연결됩니다.

선형 볼륨을 만들면 물리 확장 영역의 범위를 순서대로 논리 볼륨 영역에 할당합니다. 예를 들어 그림 2.2. "범위 매핑" 논리 확장 영역 1에서 99에 표시된 것처럼 논리 확장 영역 100을 198에 매핑하면 두 번째 물리 볼륨에 매핑할 수 있습니다. 애플리케이션의 관점에서는 198개의 확장 영역 크기의 장치가 한 개 있습니다.

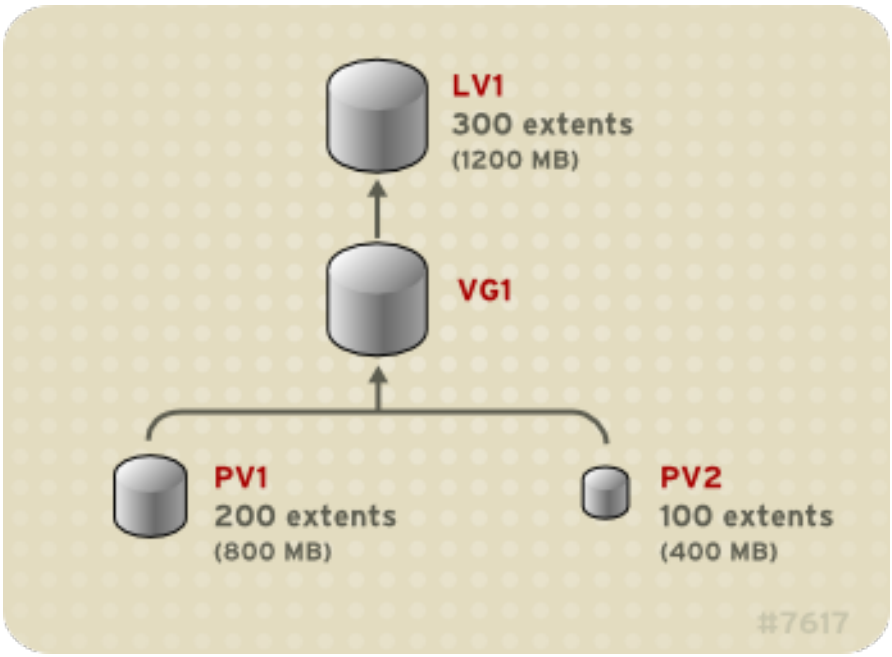
그림 2.2. 범위 매핑



[D]

논리 볼륨을 구성하는 물리 볼륨은 크기가 같을 필요가 없습니다. 그림 2.3. "Unequal 물리 볼륨이 있는 선형 볼륨" 는 물리 확장 영역 크기가 4MB인 **VG1** 볼륨 그룹을 표시합니다. 이 볼륨 그룹에는 **PV1** 및 **PV2** 라는 두 개의 물리 볼륨이 포함되어 있습니다. 물리 볼륨은 확장 크기이므로 4MB 단위로 나뉩니다. 이 예에서 **PV1** 은 200개의 확장 영역 크기(800MB)이고 **PV2** 는 크기가 100개(400MB)입니다. 1에서 300개의 확장 영역(MB에서 1200MB) 사이의 모든 크기를 만들 수 있습니다. 이 예에서 **LV1** 이라는 선형 볼륨은 크기가 300개입니다.

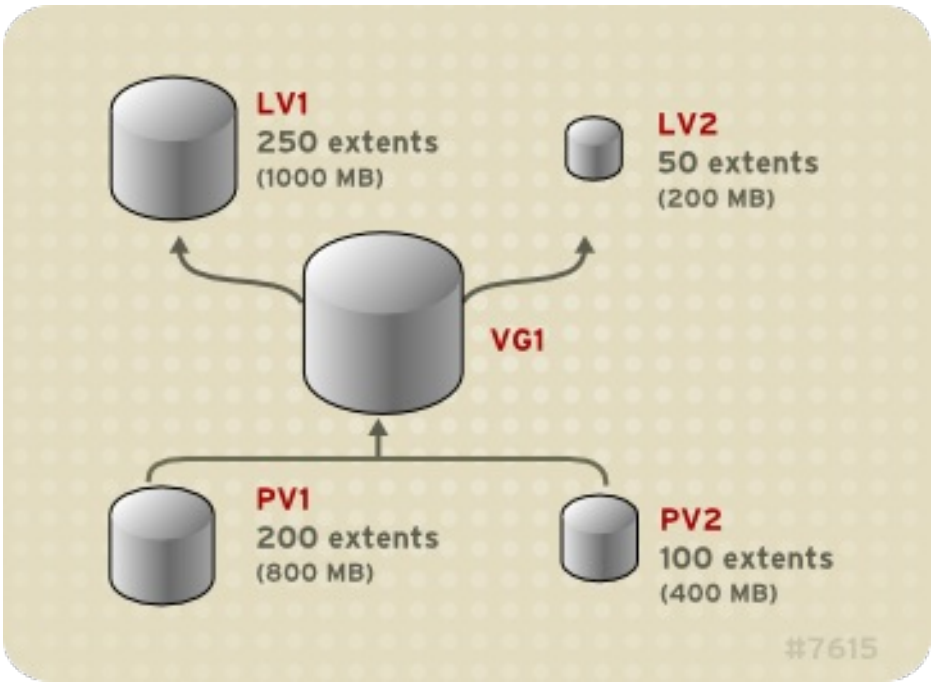
그림 2.3. Unequal 물리 볼륨이 있는 선형 볼륨



[D]

물리 확장 영역 풀에서 필요한 크기에 대해 두 개 이상의 선형 논리 볼륨을 구성할 수 있습니다. 그림 2.4. "다중 논리 볼륨" 는 그림 2.3. "Unequal 물리 볼륨이 있는 선형 볼륨" 와 동일한 볼륨 그룹을 표시하지만, 이 경우 두 개의 논리 볼륨이 볼륨 그룹에서 분리되었습니다. **LV1** 은 크기(1000MB) 및 **LV2** 의 크기(200MB) 및 50개의 확장 영역입니다.

그림 2.4. 다중 논리 볼륨



[D]

2.3.2. 제거된 논리 볼륨

LVM 논리 볼륨에 데이터를 쓸 때 파일 시스템은 기본 물리 볼륨에 걸쳐 데이터를 배치합니다. 스트라이핑된 논리 볼륨을 생성하여 데이터를 물리 볼륨에 쓰는 방식을 제어할 수 있습니다. 대규모 순차적 읽기 및 쓰기의 경우 데이터 I/O의 효율성을 향상시킬 수 있습니다.

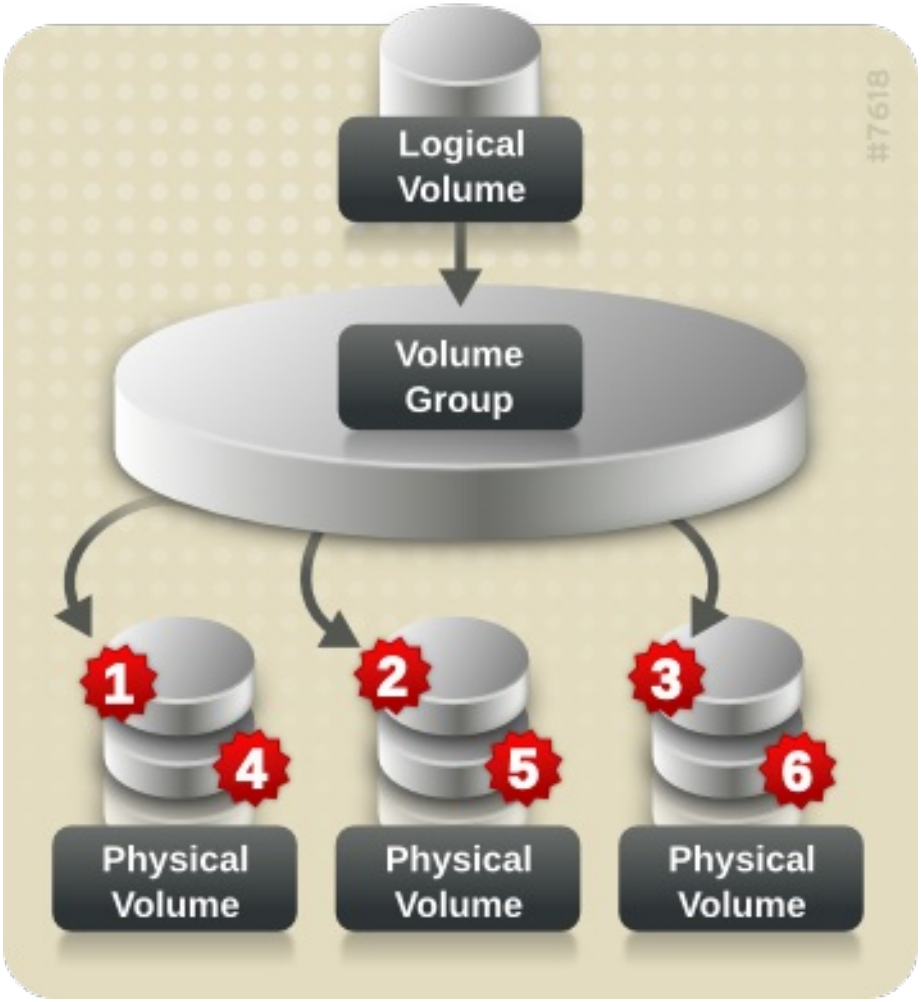
스트리핑은 라운드 로빈 방식으로 미리 정해진 수의 물리 볼륨에 데이터를 작성하여 성능을 향상시킵니다. 스트라이핑을 통해 I/O를 병렬로 수행할 수 있습니다. 경우에 따라 스트라이프의 추가 물리 볼륨에 대해 거의 선형 성능이 저하될 수 있습니다.

다음 그림에서는 세 개의 물리 볼륨에 걸쳐 있는 데이터를 보여줍니다. 이 그림에서 다음을 수행합니다.

- 데이터의 첫 번째 스트라이프는 첫 번째 물리 볼륨에 기록됩니다.
- 데이터의 두 번째 스트라이프는 두 번째 물리 볼륨에 기록됩니다.
- 세 번째 데이터 스트라이프는 세 번째 물리 볼륨에 기록됩니다.
- 데이터의 네 번째 스트라이프는 첫 번째 물리 볼륨에 기록됩니다.

스트라이핑된 논리 볼륨에서 스트라이프 크기는 확장 영역 크기를 초과할 수 없습니다.

그림 2.5. 세 개의 PV 간 데이터 제거



[D]

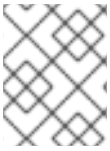
줄인 논리 볼륨은 다른 장치 집합을 첫 번째 세트의 끝에 연결하여 확장할 수 있습니다. 그러나 스트라이핑된 논리 볼륨을 확장하려면 스트라이프를 지원하기 위해 기본 물리 볼륨 세트에 충분한 여유 공간이 있어야 합니다. 예를 들어 전체 볼륨 그룹을 사용하는 양방향 스트라이프가 있는 경우 볼륨 그룹에 단일 물리 볼륨을 추가하면 스트라이프를 확장할 수 없습니다. 대신 볼륨 그룹에 두 개 이상의 물리 볼륨을 추가해야 합니다. 스트라이핑된 볼륨 확장에 대한 자세한 내용은 4.4.17절. "스트립 볼륨 확장" 을 참조하십시오.

2.3.3. RAID 논리 볼륨

LVM은 RAID0/1/4/5/6/10을 지원합니다. LVM RAID 볼륨에는 다음과 같은 특징이 있습니다.

- LVM을 통해 생성 및 관리하는 RAID 논리 볼륨은 MD 커널 드라이버를 활용합니다.
- RAID1 이미지는 일시적으로 배열에서 분할하고 나중에 다시 배열로 병합할 수 있습니다.
- LVM RAID 볼륨은 스냅샷을 지원합니다.

RAID 논리 볼륨 생성에 대한 자세한 내용은 4.4.3절. "RAID 논리 볼륨" 을 참조하십시오.



참고

RAID 논리 볼륨은 클러스터가 인식되지 않습니다. RAID 논리 볼륨은 한 컴퓨터에서만 생성하고 활성화할 수 있지만 두 개 이상의 컴퓨터에서 동시에 활성화할 수 없습니다.

2.3.4. 씬 프로비저닝된 논리 볼륨(Thin Volumes)

논리 볼륨은 씬 프로비저닝할 수 있습니다. 이를 통해 사용 가능한 확장 영역보다 큰 논리 볼륨을 만들 수 있습니다. 씬 프로비저닝을 사용하면 씬 풀이라는 여유 공간 스토리지 풀을 관리할 수 있으며, 애플리케이션에 필요할 때 임의의 수의 장치에 할당할 수 있습니다. 그런 다음 애플리케이션이 논리 볼륨에 실제로 쓸 때 나중에 할당하기 위해 씬 풀에 바인딩될 수 있는 장치를 생성할 수 있습니다. 스토리지 공간을 비용 효율적으로 할당하는 데 필요할 때 씬 풀을 동적으로 확장할 수 있습니다.

참고

클러스터의 노드에서는 씬 볼륨이 지원되지 않습니다. thin 풀과 모든 씬 볼륨은 하나의 클러스터 노드에서만 독점적으로 활성화되어야 합니다.

스토리지 관리자는 씬 프로비저닝을 사용하여 물리적 스토리지를 과다 할당할 수 있으므로 추가 스토리지를 구매할 필요가 없는 경우가 많습니다. 예를 들어, 10명의 사용자가 애플리케이션에 대해 100GB 파일 시스템을 요청하는 경우 스토리지 관리자는 각 사용자에 대해 100GB 파일 시스템으로 표시되는 항목을 생성할 수 있지만 필요한 경우에만 사용되는 실제 스토리지에서는 지원되지 않습니다.

참고

씬 프로비저닝을 사용할 때는 스토리지 관리자가 스토리지 풀을 모니터링하고 용량을 늘리기 시작하는 것이 중요합니다.

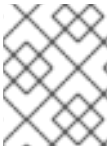
사용 가능한 모든 공간을 사용할 수 있도록 LVM에서 데이터 삭제 기능을 지원합니다. 이를 통해 삭제한 파일 또는 다른 블록 범위에서 이전에 사용한 공간을 다시 사용할 수 있습니다.

씬 볼륨 생성에 대한 자세한 내용은 4.4.5절. "씬 프로비저닝된 논리 볼륨 생성" 에서 참조하십시오.

씬 볼륨은 여러 가상 장치가 씬 풀에서 동일한 데이터를 공유할 수 있도록 새로운 COW(Copy-On-Write) 스냅샷 논리 볼륨을 구현할 수 있도록 지원합니다. 씬 스냅샷 볼륨에 대한 자세한 내용은 2.3.6절. "씬 프로비저닝된 스냅샷 볼륨" 에서 참조하십시오.

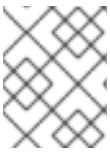
2.3.5. 스냅샷 볼륨

LVM 스냅샷 기능을 사용하면 서비스 중단 없이 특정 즉시 장치의 가상 이미지를 생성할 수 있습니다. 스냅샷을 만든 후 원래 장치(원본)를 변경하면 스냅샷 기능을 통해 장치의 상태를 재구성할 수 있도록 변경 전과 마찬가지로 변경된 데이터 영역을 복사합니다.



참고

LVM은 씬 프로비저닝된 스냅샷을 지원합니다. 씬 프로비저닝된 스냅샷 볼륨에 대한 자세한 내용은 2.3.6절. "씬 프로비저닝된 스냅샷 볼륨" 을 참조하십시오.



참고

LVM 스냅샷은 클러스터의 노드 전반에서 지원되지 않습니다. 클러스터형 볼륨 그룹에서 스냅샷 볼륨을 생성할 수 없습니다.

스냅샷은 스냅샷을 만든 후에 변경된 데이터 영역만 복사하므로 스냅샷 기능에는 최소한의 스토리지가 필요합니다. 예를 들어 거의 업데이트되지 않는 오리진을 사용하면 원본 용량의 3 ~ 3 %가 스냅샷을 유지하기에 충분합니다.

참고

파일 시스템의 스냅샷 복사본은 가상 복사본이며, 파일 시스템의 실제 미디어 백업이 아닙니다. 스냅샷은 백업 프로시저를 대체하는 기능을 제공하지 않습니다.

스냅샷의 크기는 원본 볼륨에 대한 변경 사항을 저장하기 위해 별도로 설정된 공간 크기를 제어합니다. 예를 들어 스냅샷을 만든 후 원본을 완전히 덮어쓰는 경우 스냅샷이 변경 사항을 유지하는 원본 볼륨만큼 커야 합니다. 예상되는 변경 수준에 따라 스냅샷의 차원이 필요합니다. 예를 들어 **/usr** 과 같은 읽기 볼륨 스냅샷의 수명이 짧은 스냅샷에는 **/home** 과 같이 더 많은 쓰기 수가 표시되는 볼륨의 긴 스냅샷보다 적은 공간이 필요합니다.

스냅샷이 완전히 실행되면 원본 볼륨에서 변경 사항을 더 이상 추적할 수 없으므로 스냅샷이 유효하지 않습니다. 스냅샷의 크기를 정기적으로 모니터링해야 합니다. 그러나 스냅샷은 완전히 재구성할 수 있으므로 스토리지 용량이 있으면 스냅샷 볼륨의 크기를 늘려 드롭되지 않도록 할 수 있습니다. 반대로 스냅샷 볼륨이 필요한 것보다 크면 볼륨의 크기를 줄여 다른 논리 볼륨에 필요한 공간을 확보할 수 있습니다.

스냅샷 파일 시스템을 생성하면 원본에 대한 전체 읽기 및 쓰기 액세스가 가능합니다. 스냅샷의 청크가 변경되면 해당 청크가 표시되고 원래 볼륨에서 복사되지 않습니다.

스냅샷 기능에는 다음과 같은 몇 가지 용도가 있습니다.

- 일반적으로 데이터를 지속적으로 업데이트하는 라이브 시스템을 중단하지 않고 논리 볼륨에서 백업을 수행해야 하는 경우 스냅샷이 수행됩니다.
- 스냅샷 파일 시스템에서 **fsck** 명령을 실행하여 파일 시스템의 무결성을 확인하고 원래 파일 시스템에 파일 시스템 복구가 필요한지 여부를 확인할 수 있습니다.
- 스냅샷은 읽기/쓰기이므로 스냅샷을 만들고 스냅샷에 대한 테스트를 실행하여 프로덕션 데이터에 대해 애플리케이션을 테스트하여 실제 데이터를 그대로 둘 수 있습니다.
- Red Hat Virtualization과 함께 사용할 LVM 볼륨을 생성할 수 있습니다. LVM 스냅샷을 사용하여 가상 게스트 이미지의 스냅샷을 생성할 수 있습니다. 이러한 스냅샷을 사용하면 기존 게스트를 쉽게 수정하거나 추가 스토리지를 최소화하여 새 게스트를 만들 수 있습니다. Red Hat Virtualization으로 LVM 기반 스토리지 풀 생성에 대한 자세한 내용은 가상화 *관리 가이드* 를 참조하십시오.

스냅샷 볼륨 생성에 대한 자세한 내용은 4.4.6절. "스냅샷 볼륨 생성" 을 참조하십시오.

lvconvert 명령의 **--merge** 옵션을 사용하여 스냅샷을 원본 볼륨에 병합할 수 있습니다. 데이터 또는 파일이 손실되었거나 시스템을 이전 상태로 복원해야 하는 경우 이 기능에 사용하는 기능 중 하나는 시스템 롤백을 수행하는 것입니다. 스냅샷 볼륨을 병합하면 결과 논리 볼륨의 이름, 마이너 번호, UUID가 제거되고

병합된 스냅샷이 제거됩니다. 이 옵션 사용에 대한 자세한 내용은 4.4.9절. "스냅샷 볼륨 병합" 을 참조하십시오.

2.3.6. 씬 프로비저닝된 스냅샷 볼륨

Red Hat Enterprise Linux는 씬 프로비저닝된 스냅샷 볼륨을 지원합니다. 씬 스냅샷 볼륨을 사용하면 여러 가상 장치를 동일한 데이터 볼륨에 저장할 수 있습니다. 이를 통해 관리를 단순화하고 스냅샷 볼륨 간에 데이터를 공유할 수 있습니다.

모든 LVM 스냅샷 볼륨 및 모든 씬 볼륨에 대해 씬 스냅샷 볼륨은 클러스터의 노드에서 지원되지 않습니다. 스냅샷 볼륨은 하나의 클러스터 노드에서만 독점적으로 활성화되어야 합니다.

씬 스냅샷 볼륨은 다음과 같은 이점을 제공합니다.

- 씬 스냅샷 볼륨은 동일한 원본 볼륨의 스냅샷이 여러 개인 경우 디스크 사용량을 줄일 수 있습니다.
- 동일한 원본의 스냅샷이 여러 개인 경우 원본에 쓰기를 수행하면 하나의 COW 작업이 데이터를 보존합니다. 원본의 스냅샷 수를 늘리면 큰 속도 저하가 발생하지 않습니다.
- 씬 스냅샷 볼륨은 다른 스냅샷의 논리 볼륨 원본으로 사용할 수 있습니다. 이를 통해 임의 수준의 재귀 스냅샷(스냅샷 스냅샷의 하위 집합)을 사용할 수 있습니다.
- 씬 논리 볼륨의 스냅샷도 씬 논리 볼륨을 생성합니다. 이 경우 COW 작업이 필요하거나 스냅샷 자체를 작성할 때까지 데이터 공간이 사용되지 않습니다.
- 씬 스냅샷 볼륨은 원본으로 활성화할 필요가 없으므로 사용자는 원본의 비활성 스냅샷 볼륨만 있는 동안만 사용할 수 있습니다.
- 씬 프로비저닝된 스냅샷 볼륨의 원본을 삭제하면 해당 원본 볼륨의 각 스냅샷이 씬 프로비저닝된 볼륨이 됩니다. 즉, 원본 볼륨과 스냅샷을 병합하는 대신 원본 볼륨을 삭제한 다음 해당 독립 볼륨을 새 스냅샷의 원본 볼륨으로 사용하여 씬 프로비저닝된 새 스냅샷을 만들 수 있습니다.

씬 스냅샷 볼륨을 사용하는 데는 많은 이점이 있지만 이전 LVM 스냅샷 볼륨 기능이 필요에 따라 더 적합할 수 있는 몇 가지 사용 사례가 있습니다.

- 씬 풀의 청크 크기를 변경할 수 없습니다. 씬 풀에 큰 청크 크기(예: 1MB)가 있고 큰 청크 크기가 효율적이지 않은 청크 크기가 필요한 경우 이전 스냅샷 기능을 사용하도록 선택할 수 있습니다.
- 씬 스냅샷 볼륨의 크기를 제한할 수 없습니다. 스냅샷에서는 필요한 경우 씬 풀의 모든 공간을 사용합니다. 이것은 귀하의 필요에 적합하지 않을 수 있습니다.

일반적으로 사용할 스냅샷 형식을 결정할 때 사이트의 특정 요구 사항을 고려해야 합니다.

참고

씬 프로비저닝을 사용할 때는 스토리지 관리자가 스토리지 풀을 모니터링하고 용량을 늘리기 시작하는 것이 중요합니다. 씬 프로비저닝된 스냅샷 볼륨에 대한 정보를 구성 및 표시하는 방법에 대한 자세한 내용은 4.4.7절. "씬 프로비저닝된 스냅샷 볼륨 생성" 을 참조하십시오.

2.3.7. 캐시 볼륨

Red Hat Enterprise Linux 7.1 릴리스에서 LVM은 빠른 블록 장치(예: SSD 드라이브)를 더 큰 느린 블록 장치에 대한 나중 쓰기 또는 쓰기 캐시로 사용할 수 있도록 지원합니다. 사용자는 캐시 논리 볼륨을 생성하여 기존 논리 볼륨의 성능을 개선하거나 크고 느린 장치와 결합된 작고 빠른 장치로 구성된 새 캐시 논리 볼

륨을 생성할 수 있습니다.

LVM 캐시 볼륨 생성에 대한 자세한 내용은 4.4.8절. “LVM 캐시 논리 볼륨 생성” 을 참조하십시오.

3장. LVM 관리 개요

이 장에서는 LVM 논리 볼륨을 구성하는 데 사용하는 관리 절차에 대해 설명합니다. 이 장에서는 관련 단계에 대한 일반적인 이해를 제공하기 위한 것입니다. 일반적인 LVM 구성 절차의 특정 단계별 예는 5장. *LVM 설정 예* 를 참조하십시오.

LVM 관리를 수행하는 데 사용할 수 있는 CLI 명령에 대한 설명은 4장. *CLI 명령을 사용하여 LVM 관리* 을 참조하십시오.

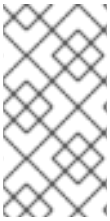
3.1. 논리 볼륨 생성 개요

다음은 LVM 논리 볼륨을 만들기 위한 단계를 요약한 것입니다.

1. LVM 볼륨에 사용할 파티션을 물리 볼륨으로 초기화합니다.
2. 볼륨 그룹을 만듭니다.
3. 논리 볼륨을 생성합니다.

논리 볼륨을 만든 후 파일 시스템을 생성하고 마운트할 수 있습니다. 이 문서의 예제에서는 GFS2 파일 시스템을 사용합니다.

1. **mkfs.gfs2** 명령을 사용하여 논리 볼륨에 polkit2 파일 시스템을 생성합니다.
2. Cryostat 명령을 사용하여 새 마운트 지점을 만듭니다. **클러스터형 시스템에서 클러스터의 모든 노드에 마운트 지점을 생성합니다.**
3. **파일 시스템을 마운트합니다. 시스템의 각 노드의 fstab 파일에 행을 추가할 수 있습니다.**



참고

GFS2 파일 시스템은 독립형 시스템이나 클러스터 구성의 일부로 구현할 수 있지만 Red Hat Enterprise Linux 7 릴리스의 경우 Red Hat은 단일 노드 파일 시스템으로 GFS2 사용을 지원하지 않습니다. Red Hat은 클러스터 파일 시스템의 스냅샷 마운트를 위해 단일 노드 GFS2 파일 시스템을 계속 지원합니다(예: 백업 목적으로).

LVM 설정 정보의 스토리지 영역이 물리 볼륨에 있고 볼륨이 생성된 시스템이 아니기 때문에 LVM 볼륨 생성은 시스템이 독립적입니다. 스토리지를 사용하는 서버에는 로컬 복사본이 있지만 물리 볼륨에 있는 서버에서 다시 생성할 수 있습니다. LVM 버전이 호환되는 경우 물리 볼륨을 다른 서버에 연결할 수 있습니다.

3.2. 논리 볼륨에서 파일 시스템 확장

논리 볼륨에서 파일 시스템을 확장하려면 다음 단계를 수행합니다.

1. **기존 볼륨 그룹에 할당되지 않은 공간이 충분한지 확인하여 논리 볼륨을 확장합니다. 그렇지 않은 경우 다음 절차를 수행합니다.**
   a. **pvcreate 명령을 사용하여 새 물리 볼륨을 생성합니다.**
   b. **Cryo stat extend 명령을 사용하여 새 물리 볼륨을 포함하도록 확장 중인 파일 시스템으로 논리 볼륨이 포함된 볼륨 그룹을 확장합니다.**
2. **볼륨 그룹이 큰 파일 시스템을 포함할 수 있을 만큼 커지면 lvresize 명령을 사용하여 논리 볼륨을 확장합니다.**

3. 논리 볼륨에서 파일 시스템의 크기를 조정합니다.

lvresize 명령의 **-r** 옵션을 사용하여 논리 볼륨을 확장하고 단일 명령으로 기본 파일 시스템의 크기를 조정할 수 있습니다.

3.3. 논리 볼륨 백업

lvm.conf 파일에서 이 기능을 비활성화하지 않는 한 볼륨 그룹 또는 논리 볼륨에 대한 구성 변경이 있을 때마다 메타데이터 백업 및 아카이브가 자동으로 생성됩니다. 기본적으로 메타데이터 백업은 **/etc/lvm/backup** 파일에 저장되고 메타데이터 아카이브는 **/etc/lvm/archive** 파일에 저장됩니다. **/etc/lvm/archive** 파일에 저장된 메타데이터 아카이브와 유지되는 아카이브 파일 수는 **lvm.conf** 파일에 설정할 수 있는 매개변수에 따라 결정됩니다. 일일 시스템 백업에는 백업에 **/etc/lvm** 디렉터리의 내용이 포함되어야 합니다.

메타데이터 백업은 논리 볼륨에 포함된 사용자 및 시스템 데이터를 백업하지 않습니다.

Cryostat **cfg** backup 명령을 사용하여 **/etc/lvm/backup** 파일에 메타데이터를 수동으로 백업할 수 있습니다. Cryostat **cfgrestore** 명령을 사용하여 메타데이터를 복원할 수 있습니다. Cryo **statcfgbackup** 및 Cryo **statcfgrestore** 명령은 4.3.13절. "볼륨 그룹 메타데이터 백업" 에 설명되어 있습니다.

3.4. 로깅

모든 메시지 출력은 다음과 같이 로깅 수준을 별도로 선택할 수 있는 로깅 모듈을 통해 전달됩니다.

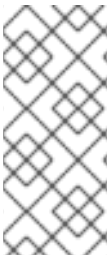
- 표준 출력/오류
- syslog
- 로그 파일
- 외부 로그 기능

로깅 수준은 부록 B. *LVM 구성 파일* 에 설명된 **/etc/lvm/lvm.conf** 파일에 설정됩니다.

3.5. 메타데이터 데몬(LVMETAD)

LVM에서는 데몬(**lvmetad**) 및 **udev** 규칙을 통해 구현된 중앙 메타데이터 캐시를 선택적으로 사용할 수 있습니다. 메타데이터 데몬은 두 가지 주요 용도가 있습니다. LVM 명령의 성능을 개선하고 **udev** 가 시스템에서 사용 가능하게 되면 논리 볼륨 또는 전체 볼륨 그룹을 자동으로 활성화할 수 있습니다.

lvm.conf 구성 파일에서 **global/use_lvmetad** 변수가 1로 설정된 경우 LVM은 데몬을 사용하도록 구성됩니다. 이는 기본값입니다. **lvm.conf** 구성 파일에 대한 자세한 내용은부록 B. *LVM 구성 파일* 을 참조하십시오.



참고

lvmetad 데몬은 현재 클러스터의 노드에서 지원되지 않으며 잠금 유형이 로컬 파일 기반 잠금이어야 합니다. **lvmconf --enable-cluster/--disable-cluster** 명령을 사용하면 **use_lvmetad** 설정(lock _type=3의 경우 0이어야 함)을 포함하여**lvm.conf** 파일이 적절하게 구성됩니다. 그러나 Pacemaker 클러스터에서 **ocf:heartbeat:clvm** 리소스 에이전트 자체는 이러한 매개변수를 시작 절차의 일부로 설정합니다.

다음 명령을 사용하여 **use_lvmetad** 의 값을 1에서 0으로 변경하는 경우**lvmetad** 서비스를 수동으로 재부팅하거나 중지해야 합니다.

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

일반적으로 각 LVM 명령은 디스크 검사를 실행하여 관련 물리 볼륨을 찾아 볼륨 그룹 메타데이터를 읽습니다. 그러나 메타데이터 데몬이 실행 중이고 활성화된 경우 이 비용이 많이 드는 검사를 건너뛸 수 있습니다. 대신 **lvmetad** 데몬은 **udev** 규칙을 사용하여 사용 가능하게 되면 각 장치를 한 번만 검사합니다. 이를 통해 상당한 양의 I/O를 절약하고 특히 많은 디스크가 있는 시스템에서 LVM 작업을 완료하는 데 필요한 시간을 줄일 수 있습니다.

런타임 시(예: 핫플러그 또는 iSCSI를 통해) 새 볼륨 그룹을 사용할 수 있게 되면 사용할 수 있도록 논리 볼륨을 활성화해야 합니다. **lvmetad** 데몬이 활성화되면 **lvm.conf** 구성 파일의 **activation/auto_activation_volume_list** 옵션을 사용하여 자동으로 활성화해야 하는 볼륨 그룹 또는 논리 볼륨 목록을 구성할 수 있습니다. **lvmetad** 데몬이 없으면 수동 활성화가 필요합니다.

참고

lvmetad 데몬이 실행 중인 경우 **pvscan --cache *device*** 명령을 실행할 때 **/etc/lvm/lvm.conf** 파일의 **filter =** 설정이 적용되지 않습니다. 장치를 필터링하려면 **global_filter =** 설정을 사용해야 합니다. 글로벌 필터를 장애가 발생한 장치는 LVM에서 열지 않으며 스캔하지 않습니다. 예를 들어 VM에서 LVM 장치를 사용하는 경우 글로벌 필터를 사용해야 할 수 있으며 VM의 장치 내용이 물리적 호스트에서 스캔되지 않도록 해야 합니다.

3.6. LVM 명령을 사용하여 LVM 정보 표시

lvm 명령은 LVM 지원 및 구성에 대한 정보를 표시하는 데 사용할 수 있는 몇 가지 기본 제공 옵션을 제공합니다.

- **LVM devtypes**

  알려진 빌드 블록 장치 유형(Red Hat Enterprise Linux 릴리스 6.6 이상)을 표시합니다.

- **LVM 형식**

  알려진 메타데이터 형식을 표시합니다.

- **LVM 도움말**

  LVM 도움말 텍스트를 표시합니다.

- **LVM 분리**

  알려진 논리 볼륨 세그먼트 유형을 표시합니다.

- **LVM 태그**

  이 호스트에 정의된 태그를 표시합니다. LVM 개체 태그에 대한 자세한 내용은 부록 D. *LVM 오브젝트 태그*을 참조하십시오.

- **LVM 버전**

  현재 버전 정보를 표시합니다.

4장. CLI 명령을 사용하여 LVM 관리

이 장에서는 LVM CLI(명령줄 인터페이스) 명령으로 수행할 수 있는 개별 관리 작업을 요약하여 논리 볼륨을 생성하고 유지 관리합니다.

LVM CLI(명령줄 인터페이스) 외에도 SSM(System Storage Manager)을 사용하여 LVM 논리 볼륨을 구성할 수 있습니다. LVM과 함께 SSM을 사용하는 방법에 대한 자세한 내용은 *스토리지 관리 가이드*를 참조하십시오.

4.1. CLI 명령 사용

모든 LVM CLI 명령에는 몇 가지 일반적인 기능이 있습니다.

명령줄 인수에 크기가 필요한 경우 항상 단위를 명시적으로 지정할 수 있습니다. 단위를 지정하지 않으면 기본값은 일반적으로 KB 또는 MB입니다. LVM CLI 명령은 분수를 허용하지 않습니다.

명령줄 인수에 단위를 지정하는 경우 LVM은 대소문자를 구분하지 않습니다. M 또는 m은 동일합니다. 예를 들어 2(여러s의 1024)의 전원이 사용됩니다. 그러나 명령에 **--units** 인수를 지정하면 소문자가 1024인 단위로 장치 수가 1024임을 나타냅니다. 대문자는 단위가 1000개임을 나타냅니다.

명령에서 볼륨 그룹 또는 논리 볼륨 이름을 인수로 사용하는 경우 전체 경로 이름은 선택 사항입니다. Cryostat0이라는 볼륨 그룹에서 **lvol0** 이라는 논리 볼륨을 Cryostat0 /**lvol0** 으로 지정할 수 있습니다. 볼륨 그룹 목록이 필요하지만 비어 있는 경우 모든 볼륨 그룹 목록을 대체합니다. 논리 볼륨 목록이 필요하지만 볼륨 그룹이 제공되는 경우 해당 볼륨 그룹의 모든 논리 볼륨 목록을 대체합니다. 예를 들어 **lvdisplay Cryostat0** 명령은 볼륨 그룹 Cryostat0 의 모든 논리 볼륨을 표시합니다.

모든 LVM 명령은 **-v** 인수를 허용하며, 출력 세부 정보 표시 수준을 높이기 위해 여러 번 입력할 수 있습니다. 예를 들어 다음 예제에서는 **lvcreate** 명령의 기본 출력을 보여줍니다.

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

다음 명령은 **-v** 인수와 함께 **lvcreate** 명령의 출력을 보여줍니다.

```
# 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
```

-vv,-vvv 또는 **-vvvv** 인수를 사용하여 명령 실행에 대한 자세한 내용을 표시할 수도 있습니다. 현재 **-vvv** 인수는 최대 정보 양을 제공합니다. 다음 예제에서는 **-vvvv** 인수가 지정된 **lvcreate** 명령에 대한 처음 몇 줄의 출력만 보여줍니다.

```
# 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
...
```

명령의 **--help** 인수를 사용하여 LVM CLI 명령에 대한 도움말을 표시할 수 있습니다.

```
# commandname --help
```

명령의 도움말 페이지를 표시하려면 **man** 명령을 실행합니다.

```
# man commandname
```

man lvm 명령은 LVM에 대한 일반 온라인 정보를 제공합니다.

모든 LVM 오브젝트는 오브젝트를 만들 때 할당된 UUID로 내부적으로 참조됩니다. 이 기능은 볼륨 그룹에 속하는 **/dev/sdf** 라는 물리 볼륨을 제거하고 다시 연결할 때**/dev/sdk** 임을 확인할 수 있습니다. UUID로 물리 볼륨을 식별하고 장치 이름이 아니므로 LVM은 물리 볼륨을 계속 찾습니다. 물리 볼륨을 생성할 때 물리 볼륨의 UUID를 지정하는 방법에 대한 자세한 내용은 6.3절. "물리 볼륨 메타데이터 복구"을 참조하십시오.

4.2. 물리 볼륨 관리

이 섹션에서는 물리 볼륨 관리의 다양한 측면을 수행하는 명령에 대해 설명합니다.

4.2.1. 물리 볼륨 생성

다음 하위 섹션에서는 물리 볼륨을 만드는 데 사용되는 명령을 설명합니다.

4.2.1.1. 파티션 유형 설정

물리 볼륨에 전체 디스크 장치를 사용하는 경우 디스크에 파티션 테이블이 없어야 합니다. Cryostat 디스크 파티션의 경우 partition id는 Cryostat 또는 **cfdisk** 명령 또는 이와 동등한 를 사용하여 0x8e로 설정해야 합니다. 전체 디스크 장치의 경우 파티션 테이블만 지워야 하므로 해당 디스크의 모든 데이터를 효과적으로 삭제합니다. 다음 명령을 사용하여 첫 번째 섹터를 0으로 기존 파티션 테이블을 제거할 수 있습니다.

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

4.2.1.2. 물리 볼륨 초기화

pvcreate 명령을 사용하여 물리 볼륨으로 사용할 블록 장치를 초기화합니다. 초기화는 파일 시스템의 포맷과 유사합니다.

다음 명령은 LVM 논리 볼륨의 일부로 나중에 사용할 수 있도록 **/dev/sdd**,**/dev/sde**, **/dev/sdf** 를 LVM 물리 볼륨으로 초기화합니다.

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

전체 디스크가 아닌 파티션을 초기화하려면 파티션에서 **pvcreate** 명령을 실행합니다. 다음 예제에서는 LVM 논리 볼륨의 일부로 나중에 사용할 수 있도록 **/dev/hdb1** 파티션을 LVM 물리 볼륨으로 초기화합니다.

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

4.2.1.3. 블록 장치 검색

다음 예와 같이 **lvmdiskscan** 명령을 사용하여 물리 볼륨으로 사용할 수 있는 블록 장치를 스캔할 수 있습니다.

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

4.2.2. 물리 볼륨 표시

LVM 물리 볼륨의 속성을 표시하는 데 사용할 수 있는 명령은 **pvs**,**pvdisplay**, **pvscan** 입니다.

pvs 명령은 물리 볼륨 정보를 구성 가능한 형식으로 제공하여 물리 볼륨당 한 행을 표시합니다**pvs** 명령은 많은 형식 제어를 제공하며 스크립팅에 유용합니다. **pvs** 명령을 사용하여 출력을 사용자 지정하는 방법에 대한 자세한 내용은 4.8절. "LVM에 대한 사용자 정의 보고"을 참조하십시오.

pvdisplay 명령은 각 물리 볼륨에 대한 자세한 다중 줄 출력을 제공합니다. 고정된 형식으로 물리 속성(크기, 확장 영역, 볼륨 그룹 등)을 표시합니다.

다음 예제에서는 단일 물리 볼륨에 대한 **pvdisplay** 명령의 출력을 보여줍니다.

```
# 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-ldwM-01S9-XO8M-mcpsVe
```

pvscan 명령은 물리 볼륨에 대해 시스템에서 지원되는 모든 LVM 블록 장치를 검사합니다.

다음 명령은 발견된 모든 물리적 장치를 표시합니다.

```
# 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]
```

이 명령은 특정 물리 볼륨을 스캔하지 않도록 **lvm.conf** 파일에 필터를 정의할 수 있습니다. 필터를 사용하여 스캔할 장치를 제어하는 방법에 대한 자세한 내용은 4.5절. "필터를 사용하여 LVM 장치 스캔 제어"을 참조하십시오.

4.2.3. 물리 볼륨의 할당 방지

pvchange 명령을 사용하여 하나 이상의 물리 볼륨의 사용 가능한 공간에 물리 확장 영역을 할당하지 못할 수 있습니다. 디스크 오류가 있거나 물리 볼륨을 제거하는 경우 이 작업이 필요할 수 있습니다.

다음 명령은 **/dev/sdk1** 의 물리 확장 영역 할당을 허용하지 않습니다.

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

pvchange 명령의 **-xy** 인수를 사용하여 이전에 허용되지 않은 위치에서 할당을 허용할 수도 있습니다.

4.2.4. 물리 볼륨 크기 조정

어떠한 이유로든 기본 블록 장치의 크기를 변경해야 하는 경우 **pvresize** 명령을 사용하여 LVM을 새 크기로 업데이트합니다. LVM에서 물리 볼륨을 사용하는 동안 이 명령을 실행할 수 있습니다.

4.2.5. 물리 볼륨 제거

LVM에서 장치를 더 이상 사용할 필요가 없는 경우 **pvremove** 명령을 사용하여 LVM 레이블을 제거할 수 있습니다. **pvremove** 명령을 실행하면 빈 물리 볼륨의 LVM 메타데이터가 제로됩니다.

제거하려는 물리 볼륨이 현재 볼륨 그룹의 일부인 경우 4.3.7절. "볼륨 그룹에서 물리 볼륨 제거"에 설명된 대로, 볼륨 그룹에서 해당 볼륨 을 제거해야 합니다.

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

4.3. 볼륨 그룹 관리

이 섹션에서는 볼륨 그룹 관리의 다양한 측면을 수행하는 명령에 대해 설명합니다.

4.3.1. 볼륨 그룹 만들기

하나 이상의 물리 볼륨에서 볼륨 그룹을 만들려면 Cryostat **create** 명령을 사용합니다. Cryo **statcreate** 명령은 이름으로 새 볼륨 그룹을 생성하고 하나 이상의 물리 볼륨을 추가합니다.

다음 명령은 물리 볼륨 **/dev/sdd 1** 및 **/dev/sde1** 을 포함하는 Cryostat1이라는 볼륨 그룹을 생성합니다.

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

물리 볼륨을 사용하여 볼륨 그룹을 만들 때 디스크 공간은 기본적으로 4MB 확장 영역으로 나뉩니다. 이 범위는 논리 볼륨의 크기를 늘리거나 줄일 수 있는 최소 크기입니다. 많은 수의 Extent는 논리 볼륨의 I/O 성능에 영향을 미치지 않습니다.

기본 범위 크기가 적합하지 않은 경우 **-s** 옵션으로 범위 크기를 지정할 수 있습니다. Cryostat **create** 명령의 **-p** 및 **-l** 인수를 사용하여 볼륨 그룹이 보유할 수 있는 물리 볼륨 또는 논리 볼륨 수에 제한을 설정할 수 있습니다.

기본적으로 볼륨 그룹은 동일한 물리 볼륨에 병렬 스트라이프를 배치하지 않는 등의 공통 규칙에 따라 물리 확장 영역을 할당합니다. 이는 일반적인 할당 정책입니다. Cryostat **create** 명령의 **--alloc** 인수를 사용하여 연속적인,어디에서나 또는 클링의 할당 정책을 지정할 수있습니다. 일반적으로 일반 이외의 할당 정책은 비정상적 또는 비표준 범위 할당을 지정해야 하는 특수한 경우에만 필요합니다. LVM에서 물리 확장 영역을 할당하는 방법에 대한 자세한 내용은 4.3.2절. “LVM Allocation” 을 참조하십시오.

LVM 볼륨 그룹과 기본 논리 볼륨은 다음 레이아웃을 사용하여 **/dev** 디렉터리의 장치 특수 파일 디렉터리 트리에 포함됩니다.

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

예를 들어 **myvg1** 및 **myvg2** 두 개의 볼륨 그룹을 생성하는 경우 각각**lv01,lv02, lv03** 이라는 세 개의 논리 볼륨이 있는 경우 6개의 장치 특수 파일이 생성됩니다.

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

해당 논리 볼륨이 현재 활성화되어 있지 않은 경우 장치 특수 파일이 존재하지 않습니다.

LVM의 최대 장치 크기는 64비트 CPU에서 8 Exabytes입니다.

4.3.2. LVM Allocation

LVM 작업에서 하나 이상의 논리 볼륨에 물리 확장 영역을 할당해야 하는 경우 할당은 다음과 같이 진행됩니다.

- 볼륨 그룹에서 할당되지 않은 물리 확장 영역의 전체 세트를 고려하여 생성됩니다. 명령줄 끝에 물리 확장 영역 범위를 제공하는 경우 지정된 물리 볼륨에 있는 해당 범위 내에서 할당되지 않은 물리 확장 영역만 고려해야 합니다.
- 각 할당 정책은 가장 엄격한 정책(연동)으로 시작하여 **--alloc** 옵션을 사용하여 지정된 할당 정책으로 끝나거나 특정 논리 볼륨 또는 볼륨 그룹의 기본값으로 설정됩니다. 각 정책에 대해 할당 정책에 따라 최대한 많은 공간을 할당해야 하는 빈 논리 볼륨 공간의 가장 낮은 수의 논리 확장에서 작업할 수 있습니다. 더 많은 공간이 필요한 경우 **LVM**은 다음 정책으로 이동합니다.

할당 정책 제한 사항은 다음과 같습니다.

- 연속적인 할당 정책을 사용하려면 논리 볼륨의 첫 번째 논리 범위가 아닌 논리 확장 영역의 물리적 위치가 바로 앞의 논리 확장 영역의 물리적 위치에 인접해야 합니다.

  논리 볼륨이 제거되거나 미러링되면 연속 할당 제한이 공간이 필요한 각 스트라이프 또는 미러 이미지(**leg**)에 독립적으로 적용됩니다.

- 복제의 할당 정책을 사용하려면 논리 볼륨에 사용된 물리 볼륨을 해당 논리 볼륨의 앞부분에서 하나 이상의 논리 범위에서 이미 사용 중인 기존 논리 볼륨에 추가해야 합니다. 구성 매개변수 **allocation/cling_tag_list** 가 정의된 경우 나열된 태그 중 하나가 두 물리 볼륨에 있는 경우 두 개의 물리 볼륨이 일치하는 것으로 간주됩니다. 이를 통해 유사한 속성(예: 물리 위치)이 있는 물리 볼륨 그룹에 태그를 지정하고 할당 목적으로 동일하게 처리될 수 있습니다. **LVM** 태그와 함께 복제 정책을 사용하여 **LVM** 볼륨을 확장할 때 사용할 추가 물리 볼륨을 지정하는 방법에 대한 자세한 내용은 **4.4.19절. “클링 할당 정책을 사용하여 논리 볼륨 확장 ”** 을 참조하십시오.

  논리 볼륨이 제거되거나 미러링되면 클링 할당 제한이 공간이 필요한 각 스트라이프 또는 미러 이미지(**leg**)에 독립적으로 적용됩니다.

- 일반 할당 정책은 병렬 논리 볼륨 내의 동일한 오프셋에서 병렬 논리 볼륨(즉, 다른 스트라이프 또는 미러 이미지/**leg**)에 이미 할당된 논리 확장 영역과 동일한 물리 볼륨을 공유하는 물리 범위를 선택하지 않습니다.

  미러 데이터를 유지하기 위해 논리 볼륨과 동시에 미러 로그를 할당할 때 **normal** 의 할당 정책은 먼저 로그 및 데이터에 대해 다른 물리 볼륨을 선택하려고 합니다. 이 옵션을 사용할 수 없고 **allocation/mirror_logs_require_separate_pvs** 구성 매개 변수가 **0**으로 설정된 경우 로그에서 데이터의 일부와 물리 볼륨을 공유할 수 있습니다.

  마찬가지로, 씬 풀 메타데이터를 할당할 때 **normal** 의 할당 정책은 할당 **/thin_pool_metadata_require_separate_pvs** 구성 매개변수 값에 따라 미러 로그 할당 과 동일한 고려 사항을 따릅니다.

- 할당 요청을 충족하기에 사용 가능한 확장 영역이 충분하지만 일반 할당 정책에서 사용하지 않는 경우 동일한 물리 볼륨에 두 개의 스트라이프를 배치하여 성능이 저하되는 경우에도 모든 위치에서 할당 정책을 사용합니다.

할당 정책은 **Cryostat change** 명령을 사용하여 변경할 수 있습니다.

참고

정의된 할당 정책에 따라 이 섹션에 설명된 레이아웃 이외의 레이아웃 동작에 의존하는 경우 이후 버전의 코드에서 변경될 수 있습니다.**If you rely on any layout behavior beyond that documented in this section according to the defined allocation policy, you should note that this might change in future versions of the code.** 예를 들어 할당에 사용할 수 있는 사용 가능한 물리 확장 영역 수가 동일한 두 개의 빈 물리 볼륨을 명령줄에 제공하는 경우 **LVM**은 현재 나열된 순서대로 각 볼륨을 사용하는 것을 고려하며 향후 릴리스에서는 해당 속성을 유지함을 보장하지 않습니다. 특정 논리 볼륨에 대한 특정 레이아웃을 얻는 것이 중요한 경우 일련의 **lvcreate** 및 **lvconvert** 단계를 통해 빌드해야 하므로 각 단계에 적용된 할당 정책은 **LVM**을 레이아웃에 의존하지 않습니다.

할당 프로세스가 특정 사례에서 작동하는 방식을 보려면, 예를 들어 명령에 **-vvv** 옵션을 추가하여 디버그 로깅 출력을 읽을 수 있습니다.

4.3.3. 클러스터에서 볼륨 그룹 만들기

단일 노드에서 생성하는 것처럼 **Cryostat create** 명령을 사용하여 클러스터 환경에서 **CLVM** 볼륨 그룹을 생성합니다.

참고

Red Hat Enterprise Linux 7에서 클러스터는 **Pacemaker**를 통해 관리됩니다. 클러스터형 **LVM** 논리 볼륨은 **Pacemaker** 클러스터와 함께만 지원되며 클러스터 리소스로 구성해야 합니다. 클러스터에서 **LVM** 볼륨 구성에 대한 일반적인 정보는 1.4절. “Red Hat High Availability Cluster의 LVM 논리 볼륨” 에서 참조하십시오.

클러스터의 멤버가 공유하는 볼륨 그룹은 **Cryostat create -cy** 또는 **Cryostat change -cy** 명령을 사용하여 클러스터형 특성을 사용하여 생성해야 합니다. **CLVMD**가 실행 중인 경우 클러스터형 특성이 자동으로 설정됩니다. 이 클러스터형 속성은 이 볼륨 그룹을 **CLVMD**에서 관리하고 보호해야 함을 나타냅니다. 클러스터에서 공유되지 않고 단일 호스트에만 표시되어야 하는 볼륨 그룹을 생성하는 경우, 이 클러스터형 특성은 **-cn** 또는 **Cryostat change -cn** 명령을 사용하여 비활성화해야 합니다.

기본적으로 공유 스토리지의 클러스터형 특성으로 생성된 볼륨 그룹은 공유 스토리지에 액세스할 수 있는 모든 컴퓨터에 표시됩니다. 그러나 **local**인 볼륨 그룹을 생성할 수 있습니다. 이 볼륨 그룹은 클러스터의 하나의 노드에만 표시되는 볼륨 그룹을 생성할 수 있습니다.

다음 명령은 클러스터 환경에서 실행할 때 명령이 실행된 노드에 로컬인 볼륨 그룹을 생성합니다. 이 명령은 물리 볼륨 **/dev/sdd 1** 및 **/dev/sde1** 이 포함된 **local volume**이라는 로컬 볼륨을 생성합니다.

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

4.3.9절. “볼륨 그룹의 매개변수 변경” 에 설명된 **-c** 옵션을 사용하여 기존 볼륨 그룹이 로컬 또는 클러스터형지 여부를 변경할 수 있습니다.

기존 볼륨 그룹이 **volumes** 명령을 사용하여 클러스터형 볼륨 그룹인지 확인할 수 있습니다. 이 그룹은 볼륨이 클러스터형 경우 **c** 속성을 표시합니다. 다음 명령은 볼륨 그룹의 특성 **usx 00** 및 **testvg1** 을 표시합니다. 이 예에서는 **Attr** 제목 아래에 **c** 속성으로 표시된 대로 **testvg1** 은 클러스터된 반면, **testvg1**은 클러스터형 상태가 아닙니다.

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

Cryostats 명령에 대한 자세한 내용은 4.3.5절. “볼륨 그룹 표시”4.8절. “LVM에 대한 사용자 정의 보고” 및 **Cryostat** 도움말 페이지를 참조하십시오.

4.3.4. 볼륨 그룹에 물리 볼륨 추가

기존 볼륨 그룹에 물리 볼륨을 추가하려면 **Cryostatextend** 명령을 사용합니다. **Cryo stat extend** 명령은 하나 이상의 사용 가능한 물리 볼륨을 추가하여 볼륨 그룹의 용량을 늘립니다.

다음 명령은 물리 볼륨 **/dev/sdf1** 을 볼륨 그룹 **Cryostat 1**에 추가합니다.

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

4.3.5. 볼륨 그룹 표시

LVM 볼륨 그룹의 속성을 표시하는 데 사용할 수 있는 명령은 두 가지가 있습니다. **protects** 및 **Cryostat display .**

모든 디스크를 볼륨 그룹으로 스캔하고 **LVM** 캐시 파일을 다시 빌드하는 **Cryostatscan** 명령도 볼륨 그룹을 표시합니다. **Cryostatscan** 명령에 대한 자세한 내용은 **4.3.6**절. "볼륨 그룹용 디스크 스캔에서 캐시 파일 빌드" 을 참조하십시오.

Cryo stats 명령은 볼륨 그룹 정보를 구성 가능한 형식으로 제공하여 볼륨 그룹당 한 행을 표시합니다. **Cryostats** 명령은 많은 형식 제어를 제공하며 스크립팅에 유용합니다. **output**을 사용자 지정하는 방법에 대한 자세한 내용은 **4.8**절. "**LVM**에 대한 사용자 정의 보고" 을 참조하십시오.

Cryo stat display 명령은 볼륨 그룹 속성(예: 크기, 확장 영역, 물리 볼륨 수 등)을 고정된 형식으로 표시합니다. 다음 예제에서는 볼륨 그룹 **new_vg** 에 대한 **Cryostatdisplay** 명령의 출력을 보여줍니다. 볼륨 그룹을 지정하지 않으면 기존 볼륨 그룹이 모두 표시됩니다.

```
# vgdisplay new_vg
  --- Volume group ---
  VG Name               new_vg
  System ID
  Format                lvm2
  Metadata Areas        3
  Metadata Sequence No  11
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                1
  Open LV               0
  Max PV                0
  Cur PV                3
  Act PV                3
  VG Size               51.42 GB
  PE Size               4.00 MB
  Total PE              13164
  Alloc PE / Size       13 / 52.00 MB
  Free  PE / Size       13151 / 51.37 GB
  VG UUID               jxQJ0a-ZKk0-OpMO-0118-nlwO-wwqd-fD5D32
```

4.3.6. 볼륨 그룹용 디스크 스캔에서 캐시 파일 빌드

Cryo stats can 명령은 **LVM** 물리 볼륨 및 볼륨 그룹을 찾는 시스템에서 지원되는 모든 디스크 장치를 검사합니다. 이렇게 하면 현재 **LVM** 장치 목록을 유지 관리하는 **/etc/lvm/cache/.cache** 파일에 **LVM** 캐시 파일이 빌드됩니다.

LVM은 시스템 시작 시 및 **LVM** 작업 중 **LVM** 작업 중 또는 **LVM**이 불일치 를 감지하는 경우와 같이 **LVM**을 자동으로 실행합니다.

참고

하드웨어 구성을 변경하고 노드에서 장치를 추가하거나 삭제할 때 **Cryostatscan** 명령을 수동으로 실행하여 시스템 부팅 시 존재하지 않은 시스템에 새 장치를 표시해야 할 수 있습니다. 예를 들어 **SAN**의 시스템에 새 디스크를 추가하거나 물리 볼륨으로 레이블이 지정된 새 디스크를 핫플러그하는 경우 이 작업이 필요할 수 있습니다.

/etc/lvm/lvm.conf 파일에 필터를 정의하여 특정 장치를 방지하도록 검사를 제한할 수 있습니다. 필터를 사용하여 스캔할 장치를 제어하는 방법에 대한 자세한 내용은 **4.5**절. “필터를 사용하여 **LVM** 장치 스캔 제어”을 참조하십시오.

다음 예제에서는 **Cryostatscan** 명령 의 출력을 보여줍니다.

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

4.3.7. 볼륨 그룹에서 물리 볼륨 제거

볼륨 그룹에서 사용되지 않는 물리 볼륨을 제거하려면 **Cryostatreduce** 명령을 사용합니다. **Cryo stat reduce** 명령은 하나 이상의 빈 물리 볼륨을 제거하여 볼륨 그룹의 용량을 줄입니다. 이렇게 하면 해당 물리 볼륨을 다른 볼륨 그룹에서 사용하거나 시스템에서 제거할 수 있습니다.

볼륨 그룹에서 물리 볼륨을 제거하기 전에 **pvdisplay** 명령을 사용하여 논리 볼륨에서 물리 볼륨을 사용하지 않도록 할 수 있습니다.

```
# 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
```

물리 볼륨이 여전히 사용 중인 경우 **pvmove** 명령을 사용하여 데이터를 다른 물리 볼륨으로 마이그레이션해야 합니다. 그런 다음 **Cryo stat reduce** 명령을 사용하여 물리 볼륨을 제거합니다.

다음 명령은 볼륨 그룹 **my_volume_group** 에서 물리 볼륨 **/dev/hda1** 을 제거합니다.

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

논리 볼륨에 실패한 물리 볼륨이 포함된 경우 해당 논리 볼륨을 사용할 수 없습니다. 볼륨 그룹에서 누락된 물리 볼륨을 제거하려면 누락된 물리 볼륨에 할당된 논리 볼륨이 없는 경우 **volume group**에서 **--removemissing** 매개변수를 사용할 수 있습니다.

실패하는 물리 볼륨에 미러 세그먼트 유형의 논리 볼륨의 미러 이미지가 포함된 경우, **Cryostatreduce --removemissing -- mirror sonly --force** 명령을 사용하여 미러 에서 해당 이미지를 제거할 수 있습니다. 이렇게 하면 물리 볼륨에서 미러 이미지인 논리 볼륨만 제거됩니다.

LVM 미러 장애 복구에 대한 자세한 내용은 **6.2절. "LVM Mirror 실패에서 복구"** 을 참조하십시오. 볼륨 그룹에서 손실된 물리 볼륨을 제거하는 방법에 대한 자세한 내용은 참조하십시오. **6.5절. "볼륨 그룹에서 손실된 물리 볼륨 제거"**

4.3.8. 볼륨 그룹 활성화 및 비활성화

볼륨 그룹을 생성하면 기본적으로 활성화됩니다. 즉, 해당 그룹의 논리 볼륨에 액세스할 수 있으며 변경될 수 있습니다.

볼륨 그룹을 비활성 상태로 만들어야 하므로 커널에 알 수 없는 다양한 상황이 있습니다. 볼륨 그룹을 비활성화하거나 활성화하려면 **Cryostat change** 명령의 **-a (--available)** 인수 를 사용합니다.

다음 예제에서는 **my_volume_group** 볼륨 그룹을 비활성화합니다.

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

클러스터형 잠금이 활성화된 경우 한 노드 또는 **'l'**에서만 볼륨 그룹을 활성화 또는 비활성화하려면 **'e'**를 추가하여 로컬 노드에서만 볼륨 그룹을 활성화하거나/비활성화합니다. 단일 호스트 스냅샷이 있는 논리 볼륨은 한 노드에서만 사용할 수 있으므로 항상 독점적으로 활성화됩니다.

4.4.11절. "논리 볼륨 그룹의 매개 변수 변경" 에 설명된 대로 **lvchange** 명령을 사용하여 개별 논리 볼륨을 비활성화할 수 있습니다. 클러스터의 개별 노드에서 논리 볼륨을 활성화하는 방법에 대한 자세한 내

용은 4.7절. “클러스터의 개별 노드에서 논리 볼륨 활성화” 을 참조하십시오.

4.3.9. 볼륨 그룹의 매개변수 변경

Cryo statchange 명령은 4.3.8절. “볼륨 그룹 활성화 및 비활성화” 에 설명된 대로 볼륨 그룹을 비활성화 및 활성화하는 데 사용됩니다. 이 명령을 사용하여 기존 볼륨 그룹의 여러 볼륨 그룹 매개변수를 변경할 수도 있습니다.

다음 명령은 볼륨 그룹의 최대 논리 볼륨 수를 **128**으로 변경합니다.

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

Cryostatchange 명령을 사용하여 변경할 수 있는 볼륨 그룹 매개변수에 대한 설명은 **Cryostat change (8)** 도움말 페이지를 참조하십시오.

4.3.10. 볼륨 그룹 제거

논리 볼륨이 없는 볼륨 그룹을 제거하려면 **Cryostat remove** 명령을 사용합니다.

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

4.3.11. 볼륨 그룹 분할

볼륨 그룹의 물리 볼륨을 분할하고 새 볼륨 그룹을 만들려면 **Cryostat split** 명령을 사용합니다.

논리 볼륨은 볼륨 그룹 간에 분할할 수 없습니다. 기존의 각 논리 볼륨은 이전 또는 새 볼륨 그룹을 구성하는 물리 볼륨에 전적으로 있어야 합니다. 그러나 필요한 경우 **pvmove** 명령을 사용하여 강제로 분할할 수 있습니다.

다음 예제에서는 새 볼륨 그룹 **smallvg** 를 원래 볼륨 그룹 **bigvg** 에서 분할합니다.

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

4.3.12. 볼륨 그룹 결합

두 개의 볼륨 그룹을 단일 볼륨 그룹으로 결합하려면 **Cryostat merge** 명령을 사용합니다. 볼륨의 물리 확장 크기가 같고 두 볼륨 그룹의 물리 및 논리 볼륨 요약이 대상 볼륨 그룹 제한에 적합한 경우 비활성 "소스" 볼륨을 활성 또는 "대상" 볼륨과 병합할 수 있습니다.

다음 명령은 비활성 볼륨 그룹 **my_vg** 를 활성 또는 비활성 볼륨 그룹 데이터베이스에 병합하여 자세한 런타임 정보를 제공합니다.

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

4.3.13. 볼륨 그룹 메타데이터 백업

lvm.conf 파일에서 비활성화하지 않는 한 메타데이터 백업 및 아카이브는 볼륨 그룹 또는 논리 볼륨에 대한 모든 구성 변경 시 자동으로 생성됩니다. 기본적으로 메타데이터 백업은 **/etc/lvm/backup** 파일에 저장되고 메타데이터 아카이브는 **/etc/lvm/archive** 파일에 저장됩니다. **Cryostat cfg backup** 명령을 사용하여 **/etc/lvm/backup** 파일에 메타데이터를 수동으로 백업할 수 있습니다.

Cryo statcfgrestore 명령은 볼륨 그룹의 메타데이터를 아카이브에서 볼륨 그룹의 모든 물리 볼륨으로 복원합니다.

physical volume metadata를 복구하기 위해 **Cryostatcfgrestore** 명령을 사용하는 예는 6.3절. “물리 볼륨 메타데이터 복구” 를 참조하십시오.

4.3.14. 볼륨 그룹 이름 변경

Cryostat rename 명령을 사용하여 기존 볼륨 그룹의 이름을 바꿉니다.

다음 명령 중 하나는 기존 볼륨 그룹의 이름을 **my_volume_group**으로 변경합니다.

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

```
# vgrename vg02 my_volume_group
```

4.3.15. 다른 시스템으로 볼륨 그룹 이동

전체 **LVM** 볼륨 그룹을 다른 시스템으로 이동할 수 있습니다. 이 작업을 수행할 때 **Cryostat export** 및 **Cryo statimport** 명령을 사용하는 것이 좋습니다.

참고

Cryostat import 명령의 **--force** 인수를 사용할 수 있습니다. 이를 통해 누락된 물리 볼륨인 볼륨 그룹을 가져오고 나중에 **Cryostatreduce --removemissing** 명령을 실행할 수 있습니다.

Cryo statexport 명령을 사용하면 비활성 볼륨 그룹에 시스템에 액세스할 수 없으므로 물리 볼륨을 분리할 수 있습니다. **Cryo statimport** 명령을 사용하면 **Cryostat export** 명령으로 비활성화된 후 시스템에서 볼륨 그룹에 액세스할 수 있습니다.

한 시스템에서 다른 시스템으로 볼륨 그룹을 이동하려면 다음 단계를 수행합니다.

1. 볼륨 그룹의 활성 볼륨의 파일에 액세스하는 사용자가 없는지 확인한 다음 논리 볼륨을 분리합니다.

2. **volume group**의 **-a n** 인수를 사용하여 볼륨 그룹을 비활성으로 표시하여 볼륨 그룹의 추가 활동을 방지합니다.

3. **volumes export** 명령을 사용하여 볼륨 그룹을 내보냅니다. 이렇게 하면 제거할 시스템에서 액세스할 수 없습니다.

   볼륨 그룹을 내보내면 다음 예와 같이 **pvscan** 명령을 실행할 때 물리 볼륨이 내보낸 볼륨 그룹에 있는 것으로 표시됩니다.

   ```
   # 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]
     ...
   ```

   다음에 시스템이 종료되면 볼륨 그룹을 구성하는 디스크를 분리하고 새 시스템에 연결할 수 있습니다.

4. 디스크가 새 시스템에 연결되면 **Cryostat import** 명령을 사용하여 볼륨 그룹을 가져와 새 시스템에서 액세스할 수 있도록 합니다.

5.
Cryostat change 명령의 **-a y** 인수를 사용하여 볼륨 그룹을 활성화합니다.

6.
파일 시스템을 마운트하여 사용할 수 있도록 합니다.

4.3.16. 볼륨 그룹 디렉터리 다시 생성

볼륨 그룹 디렉터리 및 논리 볼륨 특수 파일을 다시 생성하려면 **Cryostat mknodes** 명령을 사용합니다. 이 명령은 활성 논리 볼륨에 필요한 **/dev** 디렉토리에서 **LVM2** 특수 파일을 확인합니다. 누락된 특수 파일을 생성하고 사용되지 않은 파일을 제거합니다.

Cryostat mknodes 명령을 **Cryostatscan** 명령에 **mk nodes** 인수를 지정하여 통합할 수 있습니다.

4.4. 논리 볼륨 관리

이 섹션에서는 논리 볼륨 관리의 다양한 측면을 수행하는 명령에 대해 설명합니다.

4.4.1. 선형 논리 볼륨 생성

논리 볼륨을 생성하려면 **lvcreate** 명령을 사용합니다. 논리 볼륨의 이름을 지정하지 않으면 기본 이름 **lvol#** 이 사용됩니다. 여기서 **#** 은 논리 볼륨의 내부 번호입니다.

논리 볼륨을 생성하면 논리 볼륨이 볼륨 그룹을 구성하는 물리 볼륨의 사용 가능한 확장 영역을 사용하여 볼륨 그룹에서 분할됩니다. 일반적으로 논리 볼륨은 기본 물리 볼륨에서 사용 가능한 공간을 차세대 방식으로 사용합니다. 논리 볼륨을 수정하면 물리 볼륨에서 공간을 확보 및 재할당합니다.

다음 명령은 볼륨 그룹 **Cryostat 1** 에 논리 볼륨 **10GB**를 생성합니다.

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

논리 볼륨 크기의 기본 단위는 메가바이트입니다. 다음 명령은 볼륨 그룹 **testvg** 에 **testlv** 라는 **1500** 메가바이트 선형 논리 볼륨을 생성하여 블록 장치 **/dev/testvg/testlv** 를 생성합니다.

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

다음 명령은 볼륨 그룹 **Cryostat 0** 의 사용 가능한 확장 영역에서 **gfslv** 라는 **50GB** 논리 볼륨을 생성

합니다.

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

lvcreate 명령의 **-l** 인수를 사용하여 **Extent**에서 논리 볼륨의 크기를 지정할 수 있습니다. 이 인수를 사용하여 관련 볼륨 그룹, 논리 볼륨 또는 물리 볼륨 세트의 크기 백분율을 지정할 수도 있습니다. **%VG** 접미사는 볼륨 그룹의 총 크기, **%FREE** 접미사인 **%FREE**는 볼륨 그룹의 나머지 여유 공간을 나타냅니다. **%PVS** 접미사는 지정된 물리 볼륨의 사용 가능한 공간을 나타냅니다. 스냅샷의 경우 크기는 **%ORIGIN** 접미사 **(100%ORIGIN)**를 사용하여 원본 논리 볼륨의 전체 크기의 백분율로 표시할 수 있습니다**(0%ORIGIN**은 전체 원본의 공간을 제공합니다). 백분율로 표시되는 경우 크기는 새 논리 볼륨의 논리 확장 영역 수에 대한 상한을 정의합니다. 새 **LV**의 정확한 논리 확장 영역 수는 명령이 완료될 때까지 결정되지 않습니다.

다음 명령은 볼륨 그룹 **testvg** 의 총 공간의 **60 %**를 사용하는 **mylv** 라는 논리 볼륨을 생성합니다.

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

다음 명령은 볼륨 그룹 **testvg** 의 할당되지 않은 공간을 모두 사용하는 **lv** 라는 논리 볼륨을 생성합니다.

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

lvcreate 명령의 **-l** 인수를 사용하여 전체 볼륨 그룹을 사용하는 논리 볼륨을 생성할 수 있습니다. 전체 볼륨 그룹을 사용하는 논리 볼륨을 생성하는 또 다른 방법은 **"Total PE"** 크기를 찾고 해당 결과를 **lvcreate** 명령에 대한 입력으로 사용하는 것입니다.

다음 명령은 **testvg** 라는 볼륨 그룹을 채우는 **mylv** 라는 논리 볼륨을 생성합니다.

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

물리 볼륨을 제거해야 하는 경우 논리 볼륨을 만드는 데 사용되는 기본 물리 볼륨이 중요할 수 있으므로 논리 볼륨을 생성할 때 이러한 가능성을 고려해야 할 수 있습니다. 볼륨 그룹에서 물리 볼륨을 제거하는 방법에 대한 자세한 내용은 4.3.7절. “볼륨 그룹에서 물리 볼륨 제거” 을 참조하십시오.

볼륨 그룹의 특정 물리 볼륨에서 할당할 논리 볼륨을 만들려면 **lvcreate** 명령줄의 끝에 물리 볼륨 또는 볼륨을 지정합니다. 다음 명령은 물리 볼륨 **/dev/sdg1** 에서 할당된 **testvg** 볼륨 그룹에 **testlv** 라는 논리 볼륨을 생성합니다.

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

논리 볼륨에 사용할 물리 볼륨의 확장 영역을 지정할 수 있습니다. 다음 예제는 볼륨 그룹 **testvg** 에서 **0~24**개의 물리 볼륨 **/dev/sda1** 의 물리 볼륨 **/dev/sdb1**의 물리 볼륨 **/dev/sdb1** 의 확장 영역 **0**에서 **24**까지의 선형 논리 볼륨을 생성합니다.

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

다음 예제에서는 **0**에서 **25**개의 물리 볼륨 **/dev/sda1** 중 **25**개까지의 선형 논리 볼륨을 생성한 다음 범위 **100**에서 논리 볼륨을 계속 실행합니다.

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

논리 볼륨의 확장 영역을 할당하는 방법에 대한 기본 정책은 볼륨 그룹과 동일한 정책을 적용하는 을 상속합니다. 이러한 정책은 **lvchange** 명령을 사용하여 변경할 수 있습니다. 할당 정책에 대한 자세한 내용은 **4.3.1**절. “볼륨 그룹 만들기” 을 참조하십시오.

4.4.2. 스트립된 볼륨 생성

순차 읽기 및 쓰기의 경우 줄인 논리 볼륨을 생성하면 데이터 **I/O**의 효율성을 향상시킬 수 있습니다. 스트라이핑된 볼륨에 대한 일반적인 내용은 **2.3.2**절. “제거된 논리 볼륨” 을 참조하십시오.

제거된 논리 볼륨을 생성할 때 **lvcreate** 명령의 **-i** 인수를 사용하여 스트라이프 수를 지정합니다. 이를 통해 논리 볼륨의 물리 볼륨 수가 결정됩니다. 스트라이프 수는 볼륨 그룹의 물리 볼륨 수보다 클 수 없습니다(**--alloc**을 임의의 인수가 사용되지 않는 한).

스트라이핑된 논리 볼륨을 구성하는 기본 물리 장치가 크기가 다른 경우 스트립된 볼륨의 최대 크기는 가장 작은 기본 장치에 의해 결정됩니다. 예를 들어 두 개의 분할된 스트라이프에서 최대 크기는 작은 장치의 두 배 크기입니다. **3**개의 분할된 스트라이프에서 최대 크기는 가장 작은 장치의 **3**배 크기입니다.

다음 명령은 **2**개의 물리 볼륨에 걸쳐 스트라이핑된 논리 볼륨을 **64**킬로바이트로 만듭니다. 논리 볼륨의 크기는 **50GB**이며 **gfslv** 라는 이름이 지정되며 볼륨 그룹 **Cryostat 0**에서 제거됩니다.

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

linear 볼륨과 마찬가지로 스트라이프에 사용하는 물리 볼륨의 확장 영역을 지정할 수 있습니다. 다음 명령은 두 개의 물리 볼륨에서 스트라이핑된 볼륨 **100** 확장 영역의 크기를 스트라이프 라고 하며 볼륨 그

룹 **testvg** 에 있습니다. 스트라이프는 **/dev/sda1** 의 섹터 **0-49**를 사용하고 **/dev/sdb1** 의 **50-99** 섹터를 사용합니다.

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

4.4.3. RAID 논리 볼륨

LVM은 **RAID0/1/4/5/6/10**을 지원합니다.

참고

RAID 논리 볼륨은 클러스터가 인식되지 않습니다. **RAID** 논리 볼륨은 한 컴퓨터에서만 생성하고 활성화할 수 있지만 두 개 이상의 컴퓨터에서 동시에 활성화할 수 없습니다. 비독점 미러링 볼륨이 필요한 경우 **4.4.4**절. "미러링된 볼륨 생성" 에 설명된 대로 미러 세그먼트 유형으로 볼륨을 생성해야 합니다.

RAID 논리 볼륨을 생성하려면 **raid** 유형을 **lvcreate** 명령의 **--type** 인수로 지정합니다. 표 4.1. "RAID segment 유형" 가능한 **RAID** 세그먼트 유형을 설명합니다.

표 4.1. RAID segment 유형

세그먼트 유형	설명
raid1	RAID1 미러링. 이는 **-m** 을 지정할 때 **lvcreate** 명령의 **--type** 인수의 기본값이지만 스트라이핑은 지정하지 않습니다.
raid4	RAID4 전용 패리티 디스크
raid5	raid5_ls와 동일합니다.
raid5_la	RAID5의 ASCII로 남아 있습니다. 데이터 연속으로 순환 패리티 0

세그먼트 유형	설명
raid5_ra	RAID5 오른쪽 symmetric. 데이터 연속으로 회전 패리티 N
raid5_ls	RAID5 왼쪽 대칭. 데이터 재시작을 통한 회전 패리티 0
raid5_rs	RAID5 오른쪽 대칭. 데이터 다시 시작을 사용하는 회전 패리티 N
raid6	raid6_zr와 동일
raid6_zr	RAID6 0 재시작 데이터 재시작을 통해 패리티 0(왼쪽-오른쪽) 회전

세그먼트 유형	설명
raid6_nr	RAID6 재시작 데이터 다시 시작을 통한 패리티 N(왼쪽-오른쪽)
raid6_nc	RAID6 N을 계속 데이터 연속으로 회전 패리티 N (left-to-right)
raid10	미러가 제거되었습니다. -m 을 지정하고 1보다 큰 스트라이프 수를 지정하는 경우 lvcreate 명령의 --type 인수의 기본값입니다. 미러 세트 제거
raid0/raid0_meta (Red Hat Enterprise Linux 7.3 이상)	스트라이핑. RAID0은 스트라이프 크기 단위로 여러 데이터 하위 볼륨에 논리 볼륨 데이터를 분산합니다. 이는 성능을 향상시키는 데 사용됩니다. 데이터 하위 볼륨에 오류가 발생하면 논리 볼륨 데이터가 손실됩니다. RAID0 볼륨 생성에 대한 자세한 내용은 4.4.3.1절. “RAID0 볼륨 생성 (Red Hat Enterprise Linux 7.3 및 later)” 을 참조하십시오.

대부분의 사용자에게 사용 가능한 5가지 기본 유형(raid1,raid4,raid5,raid6,raid10) 중 하나를 지정하면 됩니다.

RAID 논리 볼륨을 생성할 때 LVM은 배열의 모든 데이터 또는 패리티 하위 볼륨에 대해 하나의 크기인 메타데이터 하위 볼륨을 생성합니다. 예를 들어 2방향 RAID1 배열을 생성하면 두 개의 메타데이터 하위 볼륨(lv_rmeta_0 및 lv_rmeta_1)과 두 개의 데이터 하위 볼륨(lv_rimage_0 및 lv_rimage_1)이 생성됩니다. 마찬가지로 3방향 스트라이프(1 암시적 패리티 장치 추가) RAID4를 생성하면 4개의 메타데이터 하위 볼륨(lv_rmeta_0,lv_rmeta _2, lv_rmeta_3) 및 4 데이터 하위 볼륨(lv_rimage_0)이 생성됩니다. lv_rimage_1,lv_rimage_2, lv_rimage_3).

다음 명령은 볼륨 그룹 **my_vg** 에 **1**기가바이트인 **my_lv** 라는 **2**방향 **RAID1** 배열을 생성합니다.

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

-m 인수에 대해 지정하는 값에 따라 다른 사본 수를 사용하여 **RAID1** 배열을 생성할 수 있습니다. 마찬가지로 **-i** 인수를 사용하여 **RAID 4/5/6** 논리 볼륨의 스트라이프 수를 지정합니다. **-I** 인수를 사용하여 스트라이프 크기를 지정할 수도 있습니다.

다음 명령은 볼륨 그룹 **my_vg** 에서 이름이 **my_lv** 인 **RAID5** 배열(**+1** 암시적 패리티 드라이브)을 **1**기가바이트 단위로 생성합니다. **LVM** 스트라이핑 볼륨에 대해 수행하는 것처럼 스트라이프 수를 지정합니다. 올바른 패리티 드라이브 수가 자동으로 추가됩니다.

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

다음 명령은 **1** 기가바이트 크기의 볼륨 그룹 **my_vg** 에 **my_lv** 라는 **RAID6** 배열 (**3** 스트라이프 **+ 2** 암시적 패리티 드라이브)을 생성합니다.

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

LVM을 사용하여 **RAID** 논리 볼륨을 생성한 후 다른 **LVM** 논리 볼륨과 마찬가지로 볼륨을 활성화, 변경, 제거, 표시 및 사용할 수 있습니다.

RAID10 논리 볼륨을 생성할 때 동기화 작업으로 논리 볼륨을 초기화하는 데 필요한 배경 **I/O**는 특히 많은 **RAID** 논리 볼륨을 생성할 때 볼륨 그룹 메타데이터 업데이트와 같은 **LVM** 장치에 대한 다른 **I/O** 작업을 충돌할 수 있습니다. 이로 인해 다른 **LVM** 작업이 느려질 수 있습니다.

복구 제한을 구현하여 **RAID** 논리 볼륨이 초기화되는 속도를 제어할 수 있습니다. **lvcreate** 명령의 **--minrecoveryrate** 및 **--maxrecoveryrate** 옵션을 사용하여 해당 작업의 최소 및 최대 **I/O** 속도를 설정하여 동기화 작업이 수행되는 속도를 제어합니다. 이러한 옵션을 다음과 같이 지정합니다.

- **--maxrecoveryrate** *속도*[**bBsSkKmMgG**]

  RAID 논리 볼륨의 최대 복구 속도를 설정하여 **nominal I/O** 작업을 대규모로 설정하지 않도록 합니다. ***Rate*** 는 배열의 각 장치에 대한 초당 양으로 지정됩니다. 접미사가 제공되지 않으면 **kiB/sec/device**로 가정합니다. 복구 속도를 **0**으로 설정하면 바인딩되지 않음을 의미합니다.

- **--minrecoveryrate** 속도**[bBsSkKmMgG]**

  RAID 논리 볼륨의 최소 복구 속도를 설정하여 동기화 작업의 **I/O**가 무분별 **I/O**가 있는 경우에도 최소 처리량을 확보합니다. ***Rate*** 는 배열의 각 장치에 대한 초당 양으로 지정됩니다. 접미사가 제공되지 않으면 **kiB/sec/device**로 가정합니다.

다음 명령은 최대 **128 kiB/sec/device**의 최대 복구 속도로 크기가 **10GB**인 **3**개의 스트라이프를 사용하여 **2**방향 **RAID10** 배열을 생성합니다. 배열의 이름은 **my_lv** 이며 볼륨 그룹 **my_vg** 에 있습니다.

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

RAID 스크럽 작업에 대한 최소 및 최대 복구 속도를 지정할 수도 있습니다. **RAID** 스크럽링에 대한 자세한 내용은 **4.4.3.11**절. **"RAID** 논리 볼륨 삭제**"** 을 참조하십시오.

참고

LVM RAID Calculator 애플리케이션을 사용하여 **RAID** 스토리지에 논리 볼륨을 생성하는 명령을 생성할 수 있습니다. 이 애플리케이션은 현재 또는 계획된 스토리지에 대해 입력한 정보를 사용하여 이러한 명령을 생성합니다. **LVM RAID** 계산기 애플리케이션은 **https://access.redhat.com/labs/lvmraidcalculator/** 에서 확인할 수 있습니다.

다음 섹션에서는 **LVM RAID** 장치에서 수행할 수 있는 관리 작업에 대해 설명합니다.

- **4.4.3.1**절. **"RAID0** 볼륨 생성**(Red Hat Enterprise Linux 7.3** 및 **later)".**
- **4.4.3.2**절. **"**선형 장치를 **RAID** 장치로 변환**"**
- **4.4.3.3**절. **"LVM RAID1** 논리 볼륨을 **LVM** 선형 논리 볼륨으로 변환**"**
- **4.4.3.4**절. **"**미러링된 **LVM** 장치를 **RAID1** 장치로 변환**"**
- **4.4.3.5**절. **"RAID** 논리 볼륨 크기 조정**"**

4.4.3.1. RAID0 볼륨 생성(Red Hat Enterprise Linux 7.3 및 later)

RAID0 볼륨을 생성하는 명령의 형식은 다음과 같습니다.

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

표 4.2. RAID0 명령 생성 매개변수

매개변수	설명

매개변수	설명
--type raid0[_meta]	**raid0** 을 지정하면 메타데이터 볼륨 없이 **RAID0** 볼륨이 생성됩니다. **raid0_meta** 를 지정하면 메타데이터 볼륨이 포함된 **RAID0** 볼륨이 생성됩니다. **RAID0**은 비현실적이므로 미러링된 데이터 블록을 **RAID1/10**으로 저장하거나 패리티 블록을 **RAID4/5/6**으로 계산 및 저장할 필요가 없습니다. 따라서 미러링된 또는 패리티 블록의 재동기화 진행 상황에 대한 상태를 유지하기 위해 메타데이터 볼륨이 필요하지 않습니다. 그러나 **RAID0**에서 **RAID4/5/6/10**으로의 변환에 메타데이터 볼륨이 필요하며, **raid0_meta** 를 지정하면 각 할당 실패를 방지하기 위해 해당 메타데이터 볼륨을 사전 할당해야 합니다.
--stripes *Stripes*	논리 볼륨을 분산할 장치 수를 지정합니다.
--stripesize *StripeSize*	각 스트라이프의 크기를 킬로바이트로 지정합니다. 이는 다음 장치로 이동하기 전에 하나의 장치에 기록된 데이터 양입니다.
VolumeGroup	사용할 볼륨 그룹을 지정합니다.
***PhysicalVolumePath* ...**	사용할 장치를 지정합니다. 이 값을 지정하지 않으면 **LVM**에서 각 스트라이프에 하나씩 ***Stripes*** 옵션으로 지정된 장치 수를 선택합니다.

4.4.3.2. 선형 장치를 RAID 장치로 변환

lvconvert 명령의 **--type** 인수를 사용하여 기존 선형 논리 볼륨을 **RAID** 장치로 변환할 수 있습니다.

다음 명령은 볼륨 그룹 **my_vg** 의 선형 논리 볼륨 **my_lv** 를 **2**방향 **RAID1** 배열로 변환합니다.

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

RAID 논리 볼륨은 메타데이터 및 데이터 하위 볼륨 쌍으로 구성되기 때문에 선형 장치를 **RAID1** 배열로 변환할 때 새 메타데이터 하위 볼륨이 생성되고 선형 볼륨이 있는 동일한 물리 볼륨에 있는 원본 논리 볼륨과 연결됩니다. 추가 이미지는 **metadata**/**data** 하위 볼륨 쌍에 추가됩니다. 예를 들어 원본 장치가 다음과 같은 경우**For example, if the original device is as follows:**

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

2방향 **RAID1** 배열로 변환된 후 장치에는 다음 데이터 및 메타데이터 하위 볼륨 쌍이 포함됩니다.

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

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

원래 논리 볼륨과 쌍을 동일한 물리 볼륨에 배치할 수 없는 메타데이터 이미지를 동일한 물리 볼륨에 배치할 수 없는 경우 **lvconvert** 가 실패합니다.

4.4.3.3. LVM RAID1 논리 볼륨을 LVM 선형 논리 볼륨으로 변환

-m0 인수를 지정하여 **lvconvert** 명령을 사용하여 기존 **RAID1 LVM** 논리 볼륨을 **LVM** 선형 논리 볼륨으로 변환할 수 있습니다. 이렇게 하면 **RAID** 데이터 하위 볼륨과 **RAID** 배열을 구성하는 모든 **RAID** 메타데이터 하위 볼륨이 제거되고 최상위 **RAID1** 이미지를 선형 논리 볼륨으로 유지합니다.

다음 예에서는 기존 **LVM RAID1** 논리 볼륨을 표시합니다.

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

다음 명령은 **LVM RAID1** 논리 볼륨 **my_vg/my_lv** 를 **LVM** 선형 장치로 변환합니다.

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

LVM RAID1 논리 볼륨을 **LVM** 선형 볼륨으로 변환할 때 제거할 물리 볼륨을 지정할 수 있습니다. 다음 예제에서는 **/dev/sda1** 및 **/dev/sdb1** 의 두 이미지로 구성된 **LVM RAID1** 논리 볼륨의 레이아웃을 보여줍니다. 이 예에서 **lvconvert** 명령은 **/dev/sda1** 을 제거하고 **/dev/sdb1** 을 선형 장치를 구성하는 물리 볼륨으로 남겨 둡니다.

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

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

4.4.3.4. 미러링된 LVM 장치를 RAID1 장치로 변환

세그먼트 유형의 미러 를 사용하여 기존 미러링된 **LVM** 장치를 **--type raid1** 인수를 지정하여 **lvconvert** 명령을 사용하여 **RAID1 LVM** 장치로 변환할 수 있습니다. 미러 하위 볼륨(***_mimage_***)의 이름을 **RAID** 하위 볼륨(***_rimage_***)으로 변경합니다. 또한 미러 로그가 제거되고 해당 데이터 하위 볼륨과 동일한 물리 볼륨의 데이터 하위 볼륨에 대해 메타데이터 하위 볼륨(***_rmeta_***)이 생성됩니다.

다음 예제에서는 미러링된 논리 볼륨 **my_vg/my_lv** 의 레이아웃을 보여줍니다.

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

다음 명령은 미러링된 논리 볼륨 **my_vg/my_lv** 를 **RAID1** 논리 볼륨으로 변환합니다.

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

4.4.3.5. RAID 논리 볼륨 크기 조정

다음과 같은 방법으로 **RAID** 논리 볼륨의 크기를 조정할 수 있습니다.

- **lvresize** 또는 **lvextend** 명령을 사용하여 모든 유형의 **RAID** 논리 볼륨의 크기를 늘릴 수 있습니다. 이는 **RAID** 이미지 수를 변경하지 않습니다. 스트라이핑 **RAID** 논리 볼륨의 경우 스트라이프 **RAID** 논리 볼륨을 만들 때와 동일한 스트라이프 반올림 제약 조건이 적용됩니다. **RAID** 볼륨 확장에 대한 자세한 내용은 4.4.18절. “RAID 볼륨 확장” 을 참조하십시오.

- **lvresize** 또는 **lvreduce** 명령을 사용하여 모든 유형의 **RAID** 논리 볼륨의 크기를 줄일 수 있습니다. 이는 **RAID** 이미지 수를 변경하지 않습니다. **lvextend** 명령과 마찬가지로 스트라이프 라

운드링 제약 조건은 스트라이프 RAID 논리 볼륨을 생성할 때와 동일합니다. 논리 볼륨 크기를 줄이는 명령의 예는 4.4.16절. “논리 볼륨 축소” 를 참조하십시오.

- Red Hat Enterprise Linux 7.4에서는 lvconvert 명령의 --stripes N 매개변수를 사용하여 스트라이핑된 RAID 논리 볼륨(raid4/5/6/10)의 스트라이프 수를 변경할 수 있습니다. 이렇게 하면 스트라이프가 추가되거나 제거된 용량으로 RAID 논리 볼륨의 크기가 늘어나거나 줄어듭니다. raid10 볼륨은 스트라이프만 추가할 수 있습니다. 이 기능은 RAID 논리 볼륨의 속성을 변경하는 동시에 동일한 RAID 수준을 유지할 수 있는 RAID *reshaping* 기능의 일부입니다. lvconvert 명령을 사용하여 RAID 논리 볼륨을 재구성하는 RAID 리셰이핑 및 예제에 대한 자세한 내용은 lvmraid(7) 도움말 페이지를 참조하십시오.

4.4.3.6. 기존 RAID1 장치의 이미지 수 변경

기존 RAID1 어레이의 이미지 수를 이전의 LVM 미러링 구현의 이미지 수를 변경할 수 있는 것처럼 변경할 수 있습니다. lvconvert 명령을 사용하여 추가 또는 제거할 추가 메타데이터/데이터 하위 볼륨 쌍 수를 지정합니다. 이전 LVM 미러링 구현에서 볼륨 구성을 변경하는 방법에 대한 자세한 내용은 4.4.4.4절. “미러링된 볼륨 구성 변경” 을 참조하십시오.

lvconvert 명령을 사용하여 RAID1 장치에 이미지를 추가하는 경우 결과 장치의 총 이미지 수를 지정하거나 장치에 추가할 이미지 수를 지정할 수 있습니다. 선택적으로 새 메타데이터/데이터 이미지 쌍이 있는 물리 볼륨을 지정할 수도 있습니다.

메타데이터 하위 볼륨(*_rmeta_*)은 항상 데이터 하위 볼륨 *_rimage_*과 동일한 물리적 장치에 존재합니다. metadata/data 하위 볼륨 쌍은 RAID 배열의 다른 metadata/data 하위 볼륨 쌍과 동일한 물리 볼륨에 생성되지 않습니다(여기서 --alloc을 지정하지 않는 경우).

RAID1 볼륨에 이미지를 추가하는 명령의 형식은 다음과 같습니다.

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

예를 들어 다음 명령은 2방향 RAID1 배열인 my_vg/my_lv 장치를 표시합니다.

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

다음 명령은 **2**방향 **RAID1** 장치 **my_vg/my_lv** 를 **3**방향 **RAID1** 장치로 변환합니다.

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

RAID1 배열에 이미지를 추가할 때 이미지에 사용할 물리 볼륨을 지정할 수 있습니다. 다음 명령은 **2**방향 **RAID1** 장치 **my_vg/my_lv** 를 **3**방향 **RAID1** 장치로 변환하여 배열에 물리 볼륨 **/dev/sdd1** 을 사용하도록 지정합니다.

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

RAID1 배열에서 이미지를 제거하려면 다음 명령을 사용합니다. **lvconvert** 명령을 사용하여 **RAID1** 장치에서 이미지를 제거하는 경우 결과 장치의 총 이미지 수를 지정하거나 장치에서 제거할 이미지 수를 지정할 수 있습니다. 선택적으로 장치를 제거할 물리 볼륨을 지정할 수도 있습니다.

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

또한 이미지 및 관련 **metadata** 하위 볼륨이 제거되면 번호가 높은 이미지가 슬롯을 채우기 위해 축소됩니다. **lv_rimage_1 , lv_rimage_1** 및 **lv_rimage_ 2** 로 구성된 **3-way RAID1** 배열에서 **lv_rimage_1** 을 제거하면 **lv_rimage_0** 및 **lv_rimage_1** 로 구성된 **RAID1** 배열이 생성됩니다. 하위 볼륨 **lv_rimage_2** 는 이름이 변경되고 빈 슬롯을 대체하여 **lv_rimage_1** 이 됩니다.

다음 예제에서는 **3-way RAID1** 논리 볼륨 **my_vg/my_lv** 의 레이아웃을 보여줍니다.

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

다음 명령은 **3-way RAID1** 논리 볼륨을 **2way RAID1** 논리 볼륨으로 변환합니다.

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

다음 명령은 **3**방향 **RAID1** 논리 볼륨을 **2**방향 **RAID1** 논리 볼륨으로 변환하여 **/dev/sde1** 로 제거할 이미지가 포함된 물리 볼륨을 지정합니다.

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

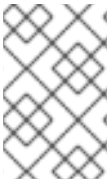
4.4.3.7. RAID 이미지로부터 논리 볼륨 분리

RAID 논리 볼륨의 이미지를 분리하여 새 논리 볼륨을 형성할 수 있습니다. **RAID** 이미지를 분할하는 절차는 **4.4.4.2절. “파티셔닝 Off aRedundant Image of a Mirrored Logical Volume”** 에 설명된 대로 미러링된 논리 볼륨의 중복 이미지를 분할하는 절차와 동일합니다.

RAID 이미지를 분할하는 명령 형식은 다음과 같습니다.

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

기존 **RAID1** 논리 볼륨(**4.4.3.6절. "기존 RAID1 장치의 이미지 수 변경"**에 설명된 대로)에서 **RAID** 이미지를 제거하는 경우와 마찬가지로 장치 중간에서 **RAID** 데이터 하위 볼륨(및 관련 **metadata** 하위 볼륨)을 제거하면 슬롯을 채우기 위해 더 높은 번호가 지정된 이미지가 축소됩니다. 따라서 **RAID** 배열을 구성하는 논리 볼륨의 인덱스 번호는 잘못된 정수 시퀀스입니다.



참고

RAID1 배열이 아직 동기화되지 않은 경우 **RAID** 이미지를 분할할 수 없습니다.

다음 예제에서는 **2**방향 **RAID1** 논리 볼륨인 **my_lv** 를 두 개의 선형 논리 볼륨인 **my_lv** 및 **new** 로 나눕니다.

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv             12.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]        /dev/sde1(1)
  [my_lv_rimage_1]        /dev/sdf1(1)
  [my_lv_rmeta_0]         /dev/sde1(0)
  [my_lv_rmeta_1]         /dev/sdf1(0)
# lvconvert --splitmirror 1 -n new my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
  LV      Copy%  Devices
  my_lv          /dev/sde1(1)
  new            /dev/sdf1(1)
```

다음 예제에서는 **3**방향 **RAID1** 논리 볼륨인 **my_lv** 를 **2**방향 **RAID1** 논리 볼륨, **my_lv** 및 선형 논리 볼륨, **new**로 분할합니다.

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

4.4.3.8. RAID 이미지 분할 및 병합

lvconvert 명령의 **--splitmirrors** 인수와 함께 **--trackchanges** 인수를 사용하여 변경 사항을 추적하는 동안 읽기 전용 사용을 위해 **RAID1** 배열의 이미지를 일시적으로 분할할 수 있습니다. 이렇게 하면 이미지를 나중에 배열에 다시 병합하고 이미지가 분할된 이후 변경된 배열의 부분만 다시 동기화할 수 있습니다.

RAID 이미지를 분할하는 **lvconvert** 명령의 형식은 다음과 같습니다.

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

--trackchanges 인수를 사용하여 **RAID** 이미지를 분할하면 분할할 이미지를 지정할 수 있지만 분할할 볼륨 이름은 변경할 수 없습니다. 또한 결과 볼륨에는 다음과 같은 제약 조건이 있습니다.

- 만든 새 볼륨은 읽기 전용입니다.
- 새 볼륨의 크기를 조정할 수 없습니다.
- 나머지 배열의 이름을 변경할 수 없습니다.
- 나머지 배열은 조정할 수 없습니다.
- 새 볼륨과 나머지 배열을 독립적으로 활성화할 수 있습니다.

--merge 인수와 함께 후속 **lvconvert** 명령을 실행하여 지정된 **--trackchanges** 인수와 함께 분할된 이미지를 병합할 수 있습니다. 이미지를 병합할 때 이미지가 분할된 이후 변경된 배열의 부분만 다시 동기화됩니다.

RAID 이미지를 병합하는 **lvconvert** 명령의 형식은 다음과 같습니다.

```
lvconvert --merge raid_image
```

다음 예제에서는 **RAID1** 논리 볼륨을 생성한 다음 나머지 배열로 변경 사항을 추적하면서 해당 볼륨에서 이미지를 분할합니다.

```
# lvcreate --type raid1 -m 2 -L 1G -n my_lv .vg
  Logical volume "my_lv" created
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]       /dev/sdb1(1)
  [my_lv_rimage_1]       /dev/sdc1(1)
  [my_lv_rimage_2]       /dev/sdd1(1)
  [my_lv_rmeta_0]        /dev/sdb1(0)
  [my_lv_rmeta_1]        /dev/sdc1(0)
  [my_lv_rmeta_2]        /dev/sdd1(0)
# lvconvert --splitmirrors 1 --trackchanges my_vg/my_lv
  my_lv_rimage_2 split from my_lv for read-only purposes.
  Use 'lvconvert --merge my_vg/my_lv_rimage_2' to merge back into my_lv
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]       /dev/sdb1(1)
  [my_lv_rimage_1]       /dev/sdc1(1)
  my_lv_rimage_2         /dev/sdd1(1)
  [my_lv_rmeta_0]        /dev/sdb1(0)
  [my_lv_rmeta_1]        /dev/sdc1(0)
  [my_lv_rmeta_2]        /dev/sdd1(0)
```

다음 예제에서는 RAID1 볼륨에서 이미지를 분할하고 나머지 배열에 변경 사항을 추적한 다음 볼륨을 다시 배열에 병합합니다.

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

RAID1 볼륨에서 이미지를 분할한 후에는 두 번째 lvconvert --splitmirrors 명령을 실행하여 --trackchanges 인수를 지정하지 않고 이미지를 분할하는 초기 lvconvert 명령을 반복하여 분할을 영구적으로 만들 수 있습니다. 이렇게 하면 --trackchanges 인수가 생성된 링크가 중단됩니다.

--trackchanges 인수로 이미지를 분할한 후에는 추적 중인 이미지를 영구적으로 분할하지 않으려면 해당 배열에서 후속 **lvconvert --splitmirrors** 명령을 실행할 수 없습니다.

다음 명령 시퀀스는 이미지를 분할하고 이미지를 추적한 다음 추적 중인 이미지를 영구적으로 분할합니다.

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

그러나 다음 명령 시퀀스는 실패합니다.

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

마찬가지로 분할 이미지가 추적 중인 이미지가 아니므로 다음 명령 순서도 실패합니다.

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

4.4.3.9. RAID 오류 정책 설정

LVM RAID는 **lvm.conf** 파일의 **raid_fault_policy** 필드에 정의된 기본 설정에 따라 장치 오류를 자동으로 처리합니다.

- **raid_fault_policy** 필드가 **allocate** 로 설정된 경우 시스템은 볼륨 그룹의 예비 장치로 실패

한 장치를 대체하려고 합니다. 사용 가능한 예비 장치가 없으면 시스템 로그에 보고됩니다.

- **raid_fault_policy** 필드가 **warn** 로 설정된 경우 시스템은 경고를 생성하고 로그는 장치가 실패했음을 나타냅니다. 이를 통해 사용자는 수행할 작업 과정을 결정할 수 있습니다.

사용성을 지원하기에 충분한 장치가 남아 있는 경우 **RAID** 논리 볼륨이 계속 작동합니다.

4.4.3.9.1. RAID Fault 정책 할당

다음 예에서 **raid_fault_policy** 필드가 **lvm.conf** 파일에 할당 하도록 설정되어 있습니다. **RAID** 논리 볼륨은 다음과 같이 표시됩니다.

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

/dev/sde 장치가 실패하면 시스템 로그에 오류 메시지가 표시됩니다.

```
# grep lvm /var/log/messages
Jan 17 15:57:18 bp-01 lvm[8599]: Device #0 of raid1 array, my_vg-my_lv, has failed.
Jan 17 15:57:18 bp-01 lvm[8599]: /dev/sde1: read failed after 0 of 2048 at
250994294784: Input/output error
Jan 17 15:57:18 bp-01 lvm[8599]: /dev/sde1: read failed after 0 of 2048 at
250994376704: Input/output error
Jan 17 15:57:18 bp-01 lvm[8599]: /dev/sde1: read failed after 0 of 2048 at 0:
Input/output error
Jan 17 15:57:18 bp-01 lvm[8599]: /dev/sde1: read failed after 0 of 2048 at
4096: Input/output error
Jan 17 15:57:19 bp-01 lvm[8599]: Couldn't find device with uuid
3lugiV-3eSP-AFAR-sdrP-H20O-wM2M-qdMANy.
Jan 17 15:57:27 bp-01 lvm[8599]: raid1 array, my_vg-my_lv, is not in-sync.
Jan 17 15:57:36 bp-01 lvm[8599]: raid1 array, my_vg-my_lv, is now in-sync.
```

raid_fault_policy 필드가 **allocate** 로 설정되었으므로 실패한 장치가 볼륨 그룹의 새 장치로 교체됩니다.

```
# lvs -a -o name,copy_percent,devices vg
  Couldn't find device with uuid 3lugiV-3eSP-AFAR-sdrP-H20O-wM2M-qdMANy.
```

```
  LV            Copy%  Devices
  lv            100.00 lv_rimage_0(0),lv_rimage_1(0),lv_rimage_2(0)
  [lv_rimage_0]        /dev/sdh1(1)
  [lv_rimage_1]        /dev/sdf1(1)
  [lv_rimage_2]        /dev/sdg1(1)
  [lv_rmeta_0]         /dev/sdh1(0)
  [lv_rmeta_1]         /dev/sdf1(0)
  [lv_rmeta_2]         /dev/sdg1(0)
```

실패한 장치를 교체했지만 디스플레이는 **LVM**에서 실패한 장치를 찾을 수 없음을 나타냅니다. 이는 실패한 장치가 **RAID** 논리 볼륨에서 제거되었지만 아직 실패한 장치가 볼륨 그룹에서 제거되지 않았기 때문입니다. 볼륨 그룹에서 실패한 장치를 제거하려면 **Cryostatreduce --removemissing *VG***를 실행할 수 있습니다.

raid_fault_policy 를 **allocate** 로 설정했지만 예비 장치가 없는 경우 할당이 실패하고 논리 볼륨을 그대로 둡니다. 할당이 실패하면 드라이브를 수정한 다음 논리 볼륨을 비활성화 및 활성화하는 옵션이 있습니다. 이는 4.4.3.9.2절. “warn RAID Fault Policy” 에 설명되어 있습니다. 또는 4.4.3.10절. “RAID 장치 교체” 에 설명된 대로 실패한 장치를 교체할 수 있습니다.

4.4.3.9.2. warn RAID Fault Policy

다음 예에서 **raid_fault_policy** 필드가 **lvm.conf** 파일에서 **warn** 로 설정되어 있습니다. **RAID** 논리 볼륨은 다음과 같이 표시됩니다.

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

/dev/sdh 장치가 실패하면 시스템 로그에 오류 메시지가 표시됩니다. 그러나 이 경우 **LVM**에서 이미지 중 하나를 교체하여 **RAID** 장치를 자동으로 복구하지 않습니다. 대신 장치가 실패한 경우 다음과 같이 장치를 **lvconvert** 명령의 **--repair** 인수로 교체할 수 있습니다.

```
# lvconvert --repair my_vg/my_lv
  /dev/sdh1: read failed after 0 of 2048 at 250994294784: Input/output error
  /dev/sdh1: read failed after 0 of 2048 at 250994376704: Input/output error
  /dev/sdh1: read failed after 0 of 2048 at 0: Input/output error
  /dev/sdh1: read failed after 0 of 2048 at 4096: Input/output error
  Couldn't find device with uuid fbI0YO-GX7x-firU-Vy5o-vzwx-vAKZ-feRxfF.
Attempt to replace failed RAID images (requires full device resync)? [y/n]: y

# lvs -a -o name,copy_percent,devices my_vg
```

```
  Couldn't find device with uuid fbI0YO-GX7x-firU-Vy5o-vzwx-vAKZ-feRxfF.
  LV            Copy%  Devices
  my_lv           64.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]       /dev/sde1(1)
  [my_lv_rimage_1]       /dev/sdf1(1)
  [my_lv_rimage_2]       /dev/sdg1(1)
  [my_lv_rmeta_0]        /dev/sde1(0)
  [my_lv_rmeta_1]        /dev/sdf1(0)
  [my_lv_rmeta_2]        /dev/sdg1(0)
```

실패한 장치를 교체했지만 디스플레이는 **LVM**에서 실패한 장치를 찾을 수 없음을 나타냅니다. 이는 실패한 장치가 **RAID** 논리 볼륨에서 제거되었지만 아직 실패한 장치가 볼륨 그룹에서 제거되지 않았기 때문입니다. 볼륨 그룹에서 실패한 장치를 제거하려면 **Cryostatreduce --removemissing *VG***를 실행할 수 있습니다.

장치 오류가 일시적인 오류이거나 실패한 장치를 복구할 수 있는 경우 **lvchange** 명령의 **--refresh** 옵션을 사용하여 실패한 장치의 복구를 시작할 수 있습니다. 이전에는 논리 볼륨을 비활성화한 다음 활성화해야 했습니다.

다음 명령은 논리 볼륨을 새로 고칩니다.

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

4.4.3.10. RAID 장치 교체

RAID는 기존 **LVM** 미러링과 같지 않습니다. **LVM** 미러링을 제거하려면 실패한 장치를 제거하거나 미러링된 논리 볼륨이 중단되었습니다. **RAID** 배열은 실패한 장치로 계속 실행될 수 있습니다. 실제로 **RAID1** 이외의 **RAID** 유형의 경우 장치를 제거하는 것은 더 낮은 수준의 **RAID(**예**: RAID6**에서 **RAID5**로 또는 **RAID4** 또는 **RAID5**에서 **RAID0)**로 변환하는 것을 의미합니다. 따라서 실패한 장치를 무조건 제거하고 대체를 잠재적으로 할당하는 대신 **LVM**을 사용하면 **lvconvert** 명령의 **--replace** 인수를 사용하여 **1**단계 솔루션의 **RAID** 볼륨의 장치를 교체할 수 있습니다.

lvconvert --replace 의 형식은 다음과 같습니다.

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

다음 예제에서는 **RAID1** 논리 볼륨을 생성한 다음 해당 볼륨에서 장치를 교체합니다.

```
# lvcreate --type raid1 -m 2 -L 1G -n my_lv my_vg
  Logical volume "my_lv" created
# lvs -a -o name,copy_percent,devices my_vg
  LV            Copy%  Devices
  my_lv          100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
```

```
  [my_lv_rimage_0]       /dev/sdb1(1)
  [my_lv_rimage_1]       /dev/sdb2(1)
  [my_lv_rimage_2]       /dev/sdc1(1)
  [my_lv_rmeta_0]        /dev/sdb1(0)
  [my_lv_rmeta_1]        /dev/sdb2(0)
  [my_lv_rmeta_2]        /dev/sdc1(0)
# lvconvert --replace /dev/sdb2 my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
  LV             Copy%  Devices
  my_lv           37.50 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]       /dev/sdb1(1)
  [my_lv_rimage_1]       /dev/sdc2(1)
  [my_lv_rimage_2]       /dev/sdc1(1)
  [my_lv_rmeta_0]        /dev/sdb1(0)
  [my_lv_rmeta_1]        /dev/sdc2(0)
  [my_lv_rmeta_2]        /dev/sdc1(0)
```

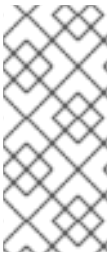
다음 예제에서는 **RAID1** 논리 볼륨을 생성한 다음 해당 볼륨에서 장치를 교체하는데 사용할 물리 볼륨을 지정합니다.

```
# lvcreate --type raid1 -m 1 -L 100 -n my_lv my_vg
  Logical volume "my_lv" created
# lvs -a -o name,copy_percent,devices my_vg
  LV             Copy%  Devices
  my_lv          100.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]       /dev/sda1(1)
  [my_lv_rimage_1]       /dev/sdb1(1)
  [my_lv_rmeta_0]        /dev/sda1(0)
  [my_lv_rmeta_1]        /dev/sdb1(0)
# pvs
  PV         VG       Fmt  Attr PSize    PFree
  /dev/sda1  my_vg    lvm2 a--  1020.00m  916.00m
  /dev/sdb1  my_vg    lvm2 a--  1020.00m  916.00m
  /dev/sdc1  my_vg    lvm2 a--  1020.00m 1020.00m
  /dev/sdd1  my_vg    lvm2 a--  1020.00m 1020.00m
# lvconvert --replace /dev/sdb1 my_vg/my_lv /dev/sdd1
# lvs -a -o name,copy_percent,devices my_vg
  LV             Copy%  Devices
  my_lv           28.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]       /dev/sda1(1)
  [my_lv_rimage_1]       /dev/sdd1(1)
  [my_lv_rmeta_0]        /dev/sda1(0)
  [my_lv_rmeta_1]        /dev/sdd1(0)
```

다음 예와 같이 여러 대체 인수를 지정하여 한 번에 두 개 이상의 **RAID** 장치를 교체할 수 있습니다.

```
# lvcreate --type raid1 -m 2 -L 100 -n my_lv my_vg
  Logical volume "my_lv" created
# lvs -a -o name,copy_percent,devices my_vg
  LV             Copy%  Devices
  my_lv          100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
```

```
  [my_lv_rimage_0]       /dev/sda1(1)
  [my_lv_rimage_1]       /dev/sdb1(1)
  [my_lv_rimage_2]       /dev/sdc1(1)
  [my_lv_rmeta_0]        /dev/sda1(0)
  [my_lv_rmeta_1]        /dev/sdb1(0)
  [my_lv_rmeta_2]        /dev/sdc1(0)
# lvconvert --replace /dev/sdb1 --replace /dev/sdc1 my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           60.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]       /dev/sda1(1)
  [my_lv_rimage_1]       /dev/sdd1(1)
  [my_lv_rimage_2]       /dev/sde1(1)
  [my_lv_rmeta_0]        /dev/sda1(0)
  [my_lv_rmeta_1]        /dev/sdd1(0)
  [my_lv_rmeta_2]        /dev/sde1(0)
```



참고

lvconvert --replace 명령을 사용하여 교체 드라이브를 지정하면 배열에 이미 사용된 드라이브의 추가 공간에서 교체 드라이브를 할당해서는 안 됩니다. 예를 들어 **lv_rimage_0** 및 **lv_rimage_1** 은 동일한 물리 볼륨에 있으면 안 됩니다.

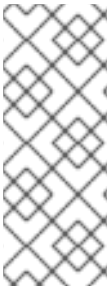
4.4.3.11. RAID 논리 볼륨 삭제

LVM은 **RAID** 논리 볼륨에 대한 스크럽 지원을 제공합니다. **RAID** 스크럽은 배열에서 모든 데이터 및 패리티 블록을 읽고 해당 블록이 일관된지 여부를 확인하는 프로세스입니다.

lvchange 명령의 **--syncaction** 옵션을 사용하여 **RAID** 스크럽 작업을 시작합니다. 검사 또는 복구 작업을 지정합니다. 검사 작업은 배열을 통과하여 배열의 불일치 수를 기록하지만 복구하지는 않습니다. 복구 작업은 찾기에 따라 불일치를 수정합니다.

RAID 논리 볼륨을 스크럽하는 명령 형식은 다음과 같습니다.

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



참고

lvchange --syncaction repair Cryostat */raid_lv 작업은* **lv convert --repair Cryostat** */raid_lv* 작업과 동일한 기능을 수행하지 않습니다. **lvchange --syncaction** 복구 작업은 배열에서 백그라운드 동기화 작업을 시작합니다. **lvconvert --repair** 작업은 미러 또는 **RAID** 논리 볼륨에서 실패한 장치를 복구/수정하도록 설계되었습니다.

새로운 **RAID scrubbing** 작업을 지원하기 위해 **lvs** 명령은 이제 **raid_sync_action** 및 **raid_mismatch_count** 라는 두 개의 새 출력 가능한 필드를 지원합니다. 이러한 필드는 기본적으로 인쇄되지 않습니다. 이러한 필드를 표시하려면 다음과 같이 **lv**의 **-o** 매개변수를 사용하여 지정합니다.

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

raid_sync_action 필드에는 **raid** 볼륨이 수행 중인 현재 동기화 작업이 표시됩니다. 다음 값 중 하나일 수 있습니다.

- **idle:** 모든 동기화 작업이 완료 (없음)
- **Resync:** 배열 초기화 또는 시스템 오류 후 복구
- **Replacing a device in the array**
- **check: array inconsistencies** 검색
- 복구**:** 불일치를 찾고 복구

raid_mismatch_count 필드에는 검사 작업 중 발견된 불일치 수가 표시됩니다.

이제 **lvs** 명령의 **Cpy%Sync** 필드가 검사 및 복구를 포함하여 **raid_sync_action** 작업 진행 상황을 출력합니다.

lvs 명령 출력의 **lv_attr** 필드에는 **RAID** 스크러빙 작업을 지원하기 위한 추가 표시기가 제공됩니다. 이 필드의 비트 **9**는 논리 볼륨의 상태를 표시하며 이제 다음 지표를 지원합니다.

- (*m*)**ismatches**는 **RAID** 논리 볼륨에 불일치가 있음을 나타냅니다. 이 문자는 스크럽 작업이 **RAID**의 일부가 일관되지 않은 부분을 감지한 후에 표시됩니다.
- **(R)efresh**는 **RAID** 배열의 장치가 오류가 발생했음을 나타내며, 커널은 **LVM**이 장치 레이블을 읽고 장치를 작동할 것으로 간주하더라도 실패한 것으로 간주합니다. *논리 볼륨은 장치를 현재*

사용할 수 있음을 알리는 (r)efreshed이거나 실패한 것으로 의심되는 경우 장치를 (r) 배치해야 합니다.

lvs 명령에 대한 자세한 내용은 4.8.2절. “오브젝트 표시 필드” 을 참조하십시오.

RAID 스크러블링 작업을 수행할 때 동기화 작업에 필요한 백그라운드 I/O는 볼륨 그룹 메타데이터 업데이트 등 LVM 장치에 대한 다른 I/O 작업을 크라우드할 수 있습니다. 이로 인해 다른 LVM 작업이 느려질 수 있습니다. 복구 제한 사항을 구현하여 RAID 논리 볼륨이 스크럽되는 속도를 제어할 수 있습니다.

lvchange 명령의 --minrecoveryrate 및 --maxrecoveryrate 옵션을 사용하여 해당 작업의 최소 및 최대 I/O 속도를 설정하여 동기화 작업이 수행되는 속도를 제어합니다. 이러한 옵션을 다음과 같이 지정합니다.

- --maxrecoveryrate 속도[bBsSkKmMgG]

  RAID 논리 볼륨의 최대 복구 속도를 설정하여 nominal I/O 작업을 대규모로 설정하지 않도록 합니다. Rate 는 배열의 각 장치에 대한 초당 양으로 지정됩니다. 접미사가 제공되지 않으면 kiB/sec/device로 가정합니다. 복구 속도를 0으로 설정하면 바인딩되지 않음을 의미합니다.

- --minrecoveryrate 속도[bBsSkKmMgG]

  RAID 논리 볼륨의 최소 복구 속도를 설정하여 동기화 작업의 I/O가 무분별 I/O가 있는 경우에도 최소 처리량을 확보합니다. Rate 는 배열의 각 장치에 대한 초당 양으로 지정됩니다. 접미사가 제공되지 않으면 kiB/sec/device로 가정합니다.

4.4.3.12. RAID takeover(Red Hat Enterprise Linux 7.4 및 later)

LVM은 Raid takeover 를 지원하므로 RAID 논리 볼륨을 한 RAID 수준에서 다른 단계로 변환합니다(예: RAID 5에서 RAID 6으로). RAID 수준 변경은 일반적으로 장치 오류 또는 나머지 논리 볼륨에 대한 복원성을 늘리거나 줄이기 위해 수행됩니다. RAID takeover에 lvconvert 를 사용합니다. RAID에 대한 정보와 lvconvert 를 사용하여 RAID 논리 볼륨을 변환하는 예제는 lvmraid(7) 매뉴얼 페이지를 참조하십시오.

4.4.3.13. RAID 논리 볼륨 조정(Red Hat Enterprise Linux 7.4 및 later)

RAID reshaping 은 동일한 RAID 수준을 유지하면서 RAID 논리 볼륨의 속성을 변경한다는 것을 의미합니다. 변경할 수 있는 일부 속성에는 RAID 레이아웃, 스트라이프 크기 및 스트라이프 수가 포함됩니

다. **lvconvert** 명령을 사용하여 **RAID** 논리 볼륨을 재구성하는 **RAID** 리셰이핑 및 예제에 대한 자세한 내용은 **lvmraid(7)** 도움말 페이지를 참조하십시오.

4.4.3.14. RAID1 논리 볼륨에서 I/O 작업 제어

lvchange 명령의 **--writemost** 및 **--write behind** 매개변수를 사용하여 **RAID1** 논리 볼륨에서 장치의 I/O 작업을 제어할 수 있습니다. 이러한 매개 변수를 사용하는 형식은 다음과 같습니다.

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

  RAID1 논리 볼륨의 장치를 대부분 쓰기로 표시합니다. 이러한 드라이브에 대한 모든 읽기는 필요하지 않은 한 피할 수 있습니다. 이 매개 변수를 설정하면 **I/O** 작업 수를 드라이브에 최소로 유지합니다. 기본적으로 **write-mostly** 속성은 논리 볼륨에서 지정된 물리 볼륨에 대해 **yes**로 설정됩니다. 물리 볼륨에 **:n** 을 추가하거나 **:t** 를 지정하여 값을 전환하여 **write-mostly** 플래그를 제거할 수 있습니다. **--writemostly** 인수는 단일 명령에서 두 번 이상 지정할 수 있으므로 논리 볼륨의 모든 물리 볼륨에 대해 가장 쓰기 특성을 한 번에 전환할 수 있습니다.

- **--[RAID]writebehind IOCount**

  가장 많이 쓰기로 표시된 **RAID1** 논리 볼륨의 장치에 허용되는 미해결 쓰기의 최대 수를 지정합니다. 이 값을 초과하면 쓰기가 동기가 되어 배열이 쓰기가 완료되기 전에 구성 장치에 대한 모든 쓰기가 완료됩니다. 값을 **0**으로 설정하면 기본 설정이 지워지고 시스템에서 임의로 값을 선택할 수 있습니다.

4.4.3.15. RAID 논리 볼륨(Red Hat Enterprise Linux 7.4 이상)에서 지역 크기 변경

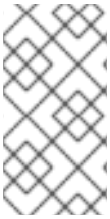
RAID 논리 볼륨을 생성할 때 논리 볼륨의 영역 크기는 **/etc/lvm/lvm.conf** 파일의 **raid_region_size** 매개변수 값이 됩니다. 이 기본값을 **lvcreate** 명령의 **-R** 옵션으로 덮어쓸 수 있습니다.

RAID 논리 볼륨을 생성한 후에는 **lvconvert** 명령의 **-R** 옵션을 사용하여 볼륨의 영역 크기를 변경할 수 있습니다. 다음 예제에서는 논리 볼륨 **Cryostat /raidlv** 의 지역 크기를 **4096K**로 변경합니다. 영역 크기를 변경하려면 **RAID** 볼륨을 동기화해야 합니다.

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

4.4.4. 미러링된 볼륨 생성

Red Hat Enterprise Linux 7.0 릴리스의 경우 LVM은 4.4.3절. "RAID 논리 볼륨" 에 설명된 대로 RAID 1/4/5/6/10을 지원합니다. RAID 논리 볼륨은 클러스터가 인식되지 않습니다. RAID 논리 볼륨은 한 컴퓨터에서만 생성하고 활성화할 수 있지만 두 개 이상의 컴퓨터에서 동시에 활성화할 수 없습니다. 비독점 미러링 볼륨이 필요한 경우 이 섹션에 설명된 대로 미러 세그먼트 유형으로 볼륨을 생성해야 합니다.



참고

미러 의 세그먼트 유형으로 기존 LVM 장치를 RAID1 LVM 장치로 변환하는 방법에 대한 자세한 내용은 4.4.3.4절. "미러링된 LVM 장치를 RAID1 장치로 변환" 을 참조하십시오.

참고

클러스터에 미러링된 LVM 논리 볼륨을 생성하려면 단일 노드에서 미러 세그먼트 유형으로 미러링된 LVM 논리 볼륨을 생성하는 것과 동일한 명령 및 절차가 필요합니다. 그러나 클러스터에 미러링된 LVM 볼륨을 생성하려면 클러스터 및 클러스터 미러 인프라를 실행해야 하며 클러스터가 정족수화되어야 하며 lvm.conf 파일의 잠금 유형을 클러스터 잠금을 활성화하려면 올바르게 설정해야 합니다. 클러스터에 미러링된 볼륨을 생성하는 예는 5.5절. "클러스터에서 미러링된 LVM 논리 볼륨 생성" 를 참조하십시오.

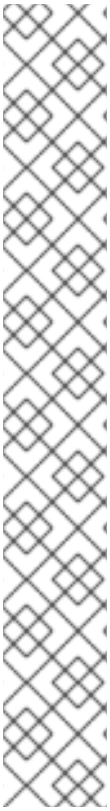
클러스터의 여러 노드에서 빠르게 연속으로 여러 LVM 미러 생성 및 변환 명령을 실행하려고 하면 이러한 명령이 백포트될 수 있습니다. 이로 인해 요청된 일부 작업이 시간 초과되고 이후에 실패할 수 있습니다. 이 문제를 방지하려면 클러스터의 한 노드에서 클러스터 미러 생성 명령을 실행하는 것이 좋습니다.

미러링된 볼륨을 생성할 때 lvcreate 명령의 -m 인수를 사용하여 만들 데이터의 사본 수를 지정합니다. m1 을 지정하면 하나의 미러가 생성되며 파일 시스템의 두 복사본(lineline 논리 볼륨과 하나의 복사)을 생성합니다. 마찬가지로 -m2 를 지정하면 미러 두 개가 생성되어 파일 시스템의 복사본 3개가 생성됩니다.

다음 명령은 단일 미러를 사용하여 미러링된 논리 볼륨을 생성합니다. 볼륨은 크기가 50 기가바이트이며 mirrorlv 이며 볼륨 그룹>.< 0:에서 제거됩니다.

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

LVM 미러는 복사 중인 장치를 기본적으로 512KB의 크기로 나눕니다. lvcreate 명령의 -R 인수를 사용하여 지역 크기를 메가바이트로 지정할 수 있습니다. lvm.conf 파일에서 mirror_region_size 설정을 편집하여 기본 영역 크기를 변경할 수도 있습니다.



참고

클러스터 인프라의 제한으로 인해 **1.5TB**보다 큰 클러스터 미러를 **512KB**의 기본 리전 크기로 생성할 수 없습니다. 더 큰 미러가 필요한 사용자는 지역 크기를 기본에서 더 큰 값으로 늘려야 합니다. 영역 크기를 늘리지 않으면 **LVM** 생성이 중단되고 다른 **LVM** 명령도 중단될 수 있습니다.

1.5TB보다 큰 미러의 영역 크기를 지정하기 위한 일반적인 지침으로 테라바이트에서 미러 크기를 가져와서 **lvcreate** 명령에 **-R** 인수로 사용하여 해당 수를 다음 **2**의 값으로 반올림할 수 있습니다. 예를 들어 미러 크기가 **1.5TB**인 경우 **-R 2** 를 지정할 수 있습니다. 미러 크기가 **3TB**인 경우 **-R 4** 를 지정할 수 있습니다. 미러 크기가 **5TB**인 경우 **-R 8** 을 지정할 수 있습니다.

다음 명령은 영역 크기가 **2MB**인 미러링된 논리 볼륨을 생성합니다.

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

미러가 생성되면 미러 지역이 동기화됩니다. 대규모 미러 구성 요소의 경우 동기화 프로세스에 시간이 오래 걸릴 수 있습니다. 다시 시작할 필요가 없는 새 미러를 만드는 경우 **--nosync** 인수를 지정하여 첫 번째 장치의 초기 동기화가 필요하지 않음을 나타낼 수 있습니다.

LVM은 미러 또는 미러와 동기화되는 영역을 추적하는 데 사용하는 작은 로그를 유지 관리합니다. 기본적으로 이 로그는 디스크에 유지되므로 재부팅 후에도 영구적으로 유지되며 시스템이 재부팅되거나 충돌할 때마다 미러를 다시 동기화할 필요가 없습니다. 대신 이 로그가 **--mirrorlog core** 인수를 사용하여 메모리에 보관하도록 지정할 수 있습니다. 따라서 추가 로그 장치에 대한 필요성을 제거하지만 전체 미러를 재부팅할 때마다 다시 동기화해야 합니다.

다음 명령은 볼륨 그룹 **bigvg** 에서 미러링된 논리 볼륨을 생성합니다. 논리 볼륨의 이름은 **ondiskmirvol** 이며 단일 미러가 있습니다. 볼륨은 크기가 **12MB**이며 미러 로그를 메모리에 유지합니다.

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

미러 로그는 미러 복사본이 생성되는 장치와 별도의 장치에 생성됩니다. 그러나 **--alloc**을 사용하여 **mirror leg** 중 하나와 동일한 장치에 미러 로그를 생성할 수 있습니다. 이로 인해 성능이 저하될 수 있지만 두 개의 기본 장치만 있어도 미러를 만들 수 있습니다.

다음 명령은 미러 로그가 미러 로그 중 하나와 동일한 장치에 미러 로그가 단일 미러인 미러링된 논리 볼륨을 생성합니다. 이 예에서 볼륨 그룹은 두 개의 장치로만 구성됩니다. 이 명령은 **Cryostat 0** 볼륨 그룹

*에 **mirrorlv** 라는 **500MB** 볼륨을 생성합니다.*

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

참고

*클러스터형 미러를 사용하는 경우 미러 로그 관리는 현재 클러스터 **ID**가 가장 낮은 클러스터 노드를 전적으로 담당합니다. 따라서 클러스터 미러 로그를 보유한 장치를 클러스터의 하위 집합에서 사용할 수 없게 되면 **ID**가 가장 낮은 클러스터 노드가 미러 로그에 액세스할 수 있는 경우 클러스터형 미러가 아무런 영향 없이 계속 작동할 수 있습니다. 미러는**disturbed**이므로 자동 수정 작업**(repair)**도 발행되지 않습니다. **lowest-ID** 클러스터 노드가 미러 로그에 대한 액세스 권한을 잃으면 자동 작업이 시작됩니다(다른 노드에서 로그에 대한 액세스 가능성은 아님).*

*미러링된 미러 로그를 생성하려면 **--mirrorlog** 미러링 된 인수를 지정할 수 있습니다. 다음 명령은 볼륨 그룹 **bigvg** 에서 미러링된 논리 볼륨을 생성합니다. 논리 볼륨의 이름은 **twologvol** 이며 단일 미러가 있습니다. 볼륨은 크기가 **12MB**이고 미러 로그는 각 로그가 별도의 장치에 보관되어 있습니다.*

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

*표준 미러 로그와 마찬가지로 **Cryostatcreate** 명령의 임의의 인수를 사용하여 **--alloc** 을 사용하여 미러 브릿지와 동일한 장치에서 중복 미러 로그를 생성할 수 있습니다. 이로 인해 성능이 저하될 수 있지만 각 로그가 미러 다리보다 별도의 장치에 보관될 수 있는 충분한 기본 장치가 없는 경우에도 중복 미러 로그를 만들 수 있습니다.*

*미러가 생성되면 미러 지역이 동기화됩니다. 대규모 미러 구성 요소의 경우 동기화 프로세스에 시간이 오래 걸릴 수 있습니다. 다시 시작할 필요가 없는 새 미러를 만드는 경우 **--nosync** 인수를 지정하여 첫 번째 장치의 초기 동기화가 필요하지 않음을 나타낼 수 있습니다.*

*미러 다리 및 로그에 사용할 장치와 사용할 장치 확장 영역을 지정할 수 있습니다. 로그를 특정 디스크에 강제 적용하려면 해당 디스크를 배치할 디스크에 정확히 하나의 범위를 지정합니다. **LVM**은 명령줄에 장치가 나열된 순서를 따를 필요가 없습니다. 할당이 적용되는 유일한 공간인 물리 볼륨이 나열된 경우. 이미 할당된 목록에 포함된 물리 확장 프로그램은 무시됩니다.*

*다음 명령은 미러링되지 않은 단일 로그와 단일 미러를 사용하여 미러링된 논리 볼륨을 생성합니다. 볼륨은 크기가 **500MB**이고 **mirrorlv**라는 이름은 **mirrorlv** 이며 볼륨 그룹 **Cryostat 0**에서 제거됩니다. 미러의 첫 번째 길이는 장치 **/dev/sda1** 이고, 미러의 두 번째는 **/dev/sdb1** 장치에 있으며 미러 로그는 **/dev/sdc1** 에 있습니다.*

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

다음 명령은 단일 미러를 사용하여 미러링된 논리 볼륨을 생성합니다. 볼륨은 크기가 **500MB**이고 **mirrorlv**라는 이름은 **mirrorlv** 이며 볼륨 그룹 **Cryostat 0**에서 제거됩니다. 미러의 첫 번째 길이는 장치 **/dev/sda1** 장치의 **0**에서 **499**까지 확장 영역 **0**부터 **499**까지입니다. 미러 로그는 장치 **/dev/sdb1** 의 범위 **0**에서 시작하여 미러 로그는 장치 **/dev/sdc1** 의 범위 **0**에서 시작됩니다. 이는 **1MB Extent**입니다. 지정된 확장 영역이 이미 할당된 경우 무시됩니다.

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

참고

제거 및 미러링을 단일 논리 볼륨으로 결합할 수 있습니다. 미러 수(**--mirrors X**)와 스트라이프 수(**-stripes Y**)를 동시에 지정하면 구성 장치가 제거된 미러 장치가 생성됩니다.

4.4.4.1. 미러링된 논리 볼륨 실패 정책

lvm.conf 파일의 **activation** 섹션에 **mirror_image_fault_policy** 및 **mirror_log_fault_policy** 매개변수를 사용하여 장치 오류 발생 시 미러링된 논리 볼륨이 작동하는 방식을 정의할 수 있습니다. 이러한 매개 변수 를 제거 하도록 설정하면 시스템에서 결함이 있는 장치를 제거하고 없이 실행하려고 합니다. 이러한 매개 변수가 할당 되도록 설정되면 시스템은 결함이 있는 장치를 제거하고 새 장치에 공간을 할당하여 실패한 장치를 대체하려고 합니다. 이 정책은 교체를 위해 적절한 장치와 공간을 할당할 수 없는 경우 제거 정책처럼 작동합니다.

기본적으로 **mirror_log_fault_policy** 매개변수는 **allocate** 로 설정됩니다. 로그에 이 정책을 사용하면 속도가 빠르며 충돌 및 재부팅을 통해 동기화 상태를 기억할 수 있습니다. 제거 하도록 이 정책을 설정하면 로그 장치가 메모리 내 로그를 사용하여 미러가 변환됩니다. 이 경우 미러는 충돌 간 동기화 상태를 기억하고 재부팅하지 않으며 전체 미러가 다시 동기화됩니다.

기본적으로 **mirror_image_fault_policy** 매개변수는 를 제거 하도록 설정됩니다. 이 정책을 사용하면 미러 이미지가 실패한 경우 나머지 복사본이 하나만 있으면 미러가 미러되지 않은 장치로 변환됩니다. 미러 장치에 할당 하도록 이 정책을 설정하려면 미러가 장치를 다시 동기화해야 합니다. 이는 느린 프로세스이지만 장치의 미러 특성을 유지합니다.

참고

LVM 미러에 장치 오류가 발생하면 2단계 복구가 수행됩니다. 첫 번째 단계는 실패한 장치를 제거하는 것입니다. 이로 인해 미러가 선형 장치로 줄어들 수 있습니다. 두 번째 단계는 **mirror_log_fault_policy** 매개변수가 **allocate** 로 설정된 경우 실패한 장치를 교체하는 것입니다. 그러나 다른 단계를 사용할 수 있는 경우 실패의 일부가 아닌 미러에서 두 번째 단계가 이전에 사용 중 사용 중인 장치를 선택한다는 보장은 없습니다.

LVM 미러 오류에서 수동으로 복구하는 방법에 대한 자세한 내용은 6.2절. “LVM Mirror 실패에서 복구” 을 참조하십시오.

4.4.4.2. 파티셔닝 Off aRedundant Image of a Mirrored Logical Volume

미러링된 논리 볼륨의 중복 이미지를 분리하여 새 논리 볼륨을 구성할 수 있습니다. 이미지를 분할하려면 **lvconvert** 명령의 **--splitmirrors** 인수를 사용하여 분할할 중복 이미지 수를 지정합니다. 명령의 **--name** 인수를 사용하여 **newly-split-off** 논리 볼륨의 이름을 지정해야 합니다.

다음 명령은 미러링된 논리 볼륨 **Cryostat /lv**에서 **copy** 라는 새 논리 볼륨 을 분할합니다. 새 논리 볼륨에는 두 개의 미러 다리가 있습니다. 이 예에서 **LVM**은 분할할 장치를 선택합니다.

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

분할할 장치를 지정할 수 있습니다. 다음 명령은 미러링된 논리 볼륨 **Cryostat /lv**에서 **copy** 라는 새 논리 볼륨 을 분할합니다. 새 논리 볼륨에는 **/dev/sdc1** 및 **/dev/sde1** 장치로 구성된 두 개의 미러 브릿지가 포함되어 있습니다.

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

4.4.4.3. 미러링된 논리 장치 복구

lvconvert --repair 명령을 사용하여 디스크 오류 후 미러를 복구할 수 있습니다. 그러면 미러가 일관된 상태로 되돌아갑니다. **lvconvert --repair** 명령은 시스템에서 실패한 장치를 교체할지 여부를 나타내는 대화형 명령입니다.

- 프롬프트를 건너뛰고 실패한 장치를 모두 교체하려면 명령줄에서 **-y** 옵션을 지정합니다.
- 프롬프트를 건너뛰고 실패한 장치를 교체하려면 명령줄에서 **-f** 옵션을 지정합니다.
-

프롬프트를 건너뛰고 미러 이미지와 미러 로그에 대한 다른 대체 정책을 표시하려면 **--use-policies** 인수를 지정하여 **lvm.conf** 파일의 **mirror_log_fault _policy** 매개변수로 지정된 장치 교체 정책을 사용할 수 있습니다.

4.4.4.4. 미러링된 볼륨 구성 변경

lvconvert 명령을 사용하여 논리 볼륨에 포함된 미러 수를 늘리거나 줄일 수 있습니다. 이를 통해 미러링된 볼륨에서 선형 볼륨 또는 선형 볼륨에서 미러링된 볼륨으로 논리 볼륨을 변환할 수 있습니다. 이 명령을 사용하여 **corelog** 와 같은 기존 논리 볼륨의 다른 미러 매개변수를 재구성할 수도 있습니다.

선형 볼륨을 미러링된 볼륨으로 변환할 때 기존 볼륨의 미러 저장소를 만듭니다. 즉, 볼륨 그룹에 미러 다리와 미러 로그의 장치 및 공간이 포함되어야 합니다.

미러 레코드가 손실된 경우 **LVM**은 볼륨을 선형 볼륨으로 변환하여 미러 중복 없이 볼륨에 계속 액세스할 수 있도록 합니다. **leg**를 교체한 후 **lvconvert** 명령을 사용하여 미러를 복원합니다. 이 절차는 **6.2절. "LVM Mirror 실패에서 복구"** 에서 제공됩니다.

다음 명령은 선형 논리 볼륨 **Cryostat 00/lvol1** 을 미러링된 논리 볼륨으로 변환합니다.

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

다음 명령은 미러링된 논리 볼륨 **Cryostat 00/lvol1** 을 선형 논리 볼륨으로 변환하여 미러 브릿지를 제거합니다.

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

다음 예제에서는 기존 논리 볼륨 **Cryostat 00/lvol1** 에 추가 미러 버전을 추가합니다. 이 예에서는 **lvconvert** 명령이 볼륨을 두 개의 미러 브릿지가 있는 볼륨으로 변경한 후 볼륨의 구성을 보여줍니다.

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

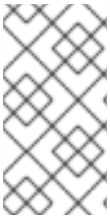
```
[lvol1_mimage_0]      /dev/sda1(0)
[lvol1_mimage_1]      /dev/sdb1(0)
[lvol1_mimage_2]      /dev/sdc1(0)
[lvol1_mlog]          /dev/sdd1(0)
```

4.4.5. 씬 프로비저닝된 논리 볼륨 생성

논리 볼륨은 씬 프로비저닝할 수 있습니다. 이를 통해 사용 가능한 확장 영역보다 큰 논리 볼륨을 만들 수 있습니다. 씬 프로비저닝을 사용하면 씬 풀이라는 여유 공간 스토리지 풀을 관리할 수 있으며, 애플리케이션에 필요할 때 임의의 수의 장치에 할당할 수 있습니다. 그런 다음 애플리케이션이 논리 볼륨에 실제로 쓸 때 나중에 할당하기 위해 씬 풀에 바인딩될 수 있는 장치를 생성할 수 있습니다. 스토리지 공간을 비용 효율적으로 할당하는 데 필요할 때 씬 풀을 동적으로 확장할 수 있습니다.

참고

이 섹션에서는 씬 프로비저닝된 논리 볼륨을 생성하고 확장하는 데 사용하는 기본 명령에 대해 설명합니다. **LVM** 씬 프로비저닝 및 씬 프로비저닝된 논리 볼륨이 있는 **LVM** 명령 및 유틸리티 사용에 대한 자세한 내용은 **lvmthin(7)** 도움말 페이지를 참조하십시오.



참고

클러스터의 노드에서는 씬 볼륨이 지원되지 않습니다. **thin** 풀과 모든 씬 볼륨은 하나의 클러스터 노드에서만 독점적으로 활성화되어야 합니다.

씬 볼륨을 생성하려면 다음 작업을 수행합니다.

1. **Cryostat create** 명령을 사용하여 볼륨 그룹을 생성합니다.

2. **lvcreate** 명령을 사용하여 **thin** 풀을 생성합니다.

3. **lvcreate** 명령을 사용하여 **thin** 풀에 **thin** 볼륨을 생성합니다.

lvcreate 명령의 **-T** (또는 **--thin**) 옵션을 사용하여 씬 풀 또는 씬 볼륨을 생성할 수 있습니다. 단일 명령을 사용하여 **lvcreate** 명령의 **-T** 옵션을 사용하여 해당 풀에 **thin** 풀과 **thin** 볼륨을 동시에 생성할 수도 있습니다.

다음 명령은 **lvcreate** 명령의 **-T** 옵션을 사용하여 볼륨 그룹 **Cryostat 001** 에 **mythinpool** 이라는 씬 풀을 생성합니다. 이는 크기가 **100M**입니다. 물리 공간 풀을 생성하므로 풀 크기를 지정해야 합니다.

lvcreate 명령의 **-T** 옵션은 인수를 사용하지 않습니다. 명령이 지정하는 다른 옵션에서 생성할 장치 유형을 유추합니다.

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

다음 명령은 **lvcreate** 명령의 **-T** 옵션을 사용하여 씬 풀 **Cryostat 001/mythinpool**에 **thinvolume** 이라는 씬 볼륨을 생성합니다. 이 경우 가상 크기를 지정하고 이를 포함하는 풀보다 큰 볼륨의 가상 크기를 지정합니다.

```
# lvcreate -V 1G -T vg001/mythinpool -n thinvolume
  Logical volume "thinvolume" created
# lvs
  LV         VG      Attr     LSize   Pool       Origin Data%  Move Log Copy%  Convert
  mythinpool  vg001    twi-a-tz 100.00m                  0.00
  thinvolume  vg001    Vwi-a-tz   1.00g mythinpool         0.00
```

다음 명령은 **lvcreate** 명령의 **-T** 옵션을 사용하여 **lvcreate** 명령에 크기와 가상 크기 인수를 모두 지정하여 해당 풀에 **thin** 풀과 **thin** 볼륨을 생성합니다. 이 명령은 볼륨 그룹에 **mythinpool** 이라는 씬 풀을 생성하고 해당 풀에 **thinvolume**이라는 씬 볼륨 도 생성합니다.

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

lvcreate 명령의 **--thinpool** 매개변수를 지정하여 **thin** 풀을 생성할 수도 있습니다. **T** 옵션과 달리 **--thinpool** 매개변수에는 생성 중인 **thin pool** 논리 볼륨의 이름인 인수가 필요합니다. 다음 예제에서는 **lvcreate** 명령의 **--thinpool** 매개변수를 지정하여 볼륨 그룹 **mythinpool**에 **mythinpool** 이라는 **thin** 풀을 생성합니다. 이는 크기가 **100M**입니다.

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

청크 크기를 사용하려면 다음 기준을 사용합니다.

- 청크 크기가 작은 경우 더 많은 메타데이터가 필요하며 성능이 저하되지만 스냅샷에 더 나은 공간 사용률을 제공합니다.

- 대규모 청크 크기는 메타데이터 조작이 덜 필요하지만 스냅샷을 덜 효율적으로 만듭니다.

LVM2는 청크 크기를 다음과 같은 방식으로 계산합니다.

기본적으로 **LVM**은 **64KiB** 청크 크기로 시작하고 씬 풀 메타데이터 장치의 결과 크기가 **128MiB**를 초과하면 값을 늘어남에 따라 메타데이터 크기가 압축됩니다. 이로 인해 일부 큰 청크 크기 값이 생성되어 스냅샷 사용에 효율성이 떨어질 수 있습니다. 이 경우 청크 크기 및 더 큰 메타데이터 크기가 더 나은 옵션입니다.

볼륨 데이터 크기가 **TiB** 범위인 경우 최대 지원되는 최대 크기인 **~15.8GiB** 메타데이터 크기를 사용하고 요구 사항에 따라 청크 크기를 사용합니다. 그러나 이 볼륨 데이터 크기를 확장해야 하고 청크 크기가 작은 경우 메타데이터 크기를 늘릴 수 없습니다.

주의

Red Hat은 기본 청크 크기를 사용하는 것이 좋습니다. 청크 크기가 너무 작고 메타데이터에 필요한 공간이 부족하면 볼륨에서 데이터를 만들 수 없습니다. 논리 볼륨을 모니터링하여 메타데이터 볼륨이 완전히 가득 차기 전에 생성되거나 생성된 스토리지가 확장되었는지 확인합니다. 씬 풀을 충분히 큰 청크 크기로 설정하여 메타데이터 공간이 부족하지 않도록 해야 합니다.

풀 생성에는 스트라이핑이 지원됩니다. 다음 명령은 두 개의 **64 kB** 스트라이프와 **256 kB** 크기의 청크 크기를 사용하여 볼륨 그룹 **Cryostat 001** 에 풀 이라는 **100M** 씬 풀을 생성합니다. 또한 **1T thin volume, Cryostat 00/thin_lv** 를 생성합니다.

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

lvextend 명령을 사용하여 **thin** 볼륨의 크기를 확장할 수 있습니다. 그러나 씬 풀의 크기를 줄일 수는 없습니다.

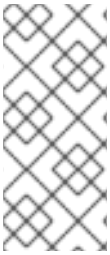
다음 명령은 다른 **100M**을 확장하여 크기가 **100M**인 기존 씬 풀의 크기를 조정합니다.

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

다른 유형의 논리 볼륨과 마찬가지로 볼륨의 이름을 **lvrename** 으로 변경하고, **lvremove** 로 볼륨을 제거할 수 있으며, 볼륨에 대한 정보를 **lvs** 및 **lvdisplay** 명령으로 표시할 수 있습니다.

기본적으로 **lvcreate** 명령은 공식(**Pool_LV_size / Pool_LV_chunk_size * 64**)에 따라 **thin pool**의 메타데이터 논리 볼륨의 크기를 설정합니다. 스냅샷 수가 많거나 씬 풀에 대한 청크 크기가 작으므로 나중에 씬 풀의 크기가 크게 증가할 것으로 예상되는 경우 **lvcreate** 명령의 **--poolmetadatasize** 매개변수를 사용하여 씬 풀의 기본값을 늘려야 할 수 있습니다. 씬 풀의 **metadata** 논리 볼륨에 지원되는 값은 **2MiB**에서 **16GiB** 사이의 범위에 있습니다.

lvconvert 명령의 **--thinpool** 매개변수를 사용하여 기존 논리 볼륨을 씬 풀 볼륨으로 변환할 수 있습니다. 기존 논리 볼륨을 씬 풀 볼륨으로 변환하는 경우 **lvconvert** 의 **--thinpool** 매개변수와 함께 **--poolmetadata** 매개변수를 사용하여 기존 논리 볼륨을 씬 풀 볼륨의 메타데이터 볼륨으로 변환해야 합니다.



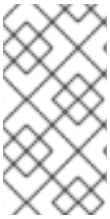
참고

논리 볼륨을 씬 풀 볼륨 또는 씬 풀 메타데이터 볼륨으로 변환하면 논리 볼륨의 콘텐츠가 제거됩니다. 이 경우 **lvconvert** 는 장치의 콘텐츠를 보존하지 않고 대신 콘텐츠를 덮어쓰기 때문입니다.

다음 예제에서는 볼륨 그룹 **lv1** 의 기존 논리 볼륨 **lv1**을 **thin** 풀 볼륨으로 변환하고 볼륨 그룹 **Cryostat 001** 의 기존 논리 볼륨 **lv2** 를 해당 씬 풀 볼륨의 **metadata** 볼륨으로 변환합니다.

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

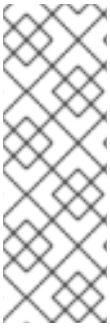
4.4.6. 스냅샷 볼륨 생성



참고

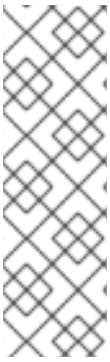
LVM은 씬 프로비저닝된 스냅샷을 지원합니다. 씬 프로비저닝된 스냅샷 볼륨 생성에 대한 자세한 내용은 4.4.7절. “씬 프로비저닝된 스냅샷 볼륨 생성” 을 참조하십시오.

lvcreate 명령의 -s 인수를 사용하여 스냅샷 볼륨을 생성합니다. 스냅샷 볼륨은 쓸 수 있습니다.



참고

LVM 스냅샷은 클러스터의 노드 전반에서 지원되지 않습니다. 클러스터형 볼륨 그룹에서 스냅샷 볼륨을 생성할 수 없습니다. 그러나 클러스터형 논리 볼륨에서 일관된 데이터 백업을 생성해야 하는 경우 볼륨을 독점적으로 활성화한 다음 스냅샷을 만들 수 있습니다. 하나의 노드에서만 논리 볼륨을 활성화하는 방법에 대한 자세한 내용은 4.7절. “클러스터의 개별 노드에서 논리 볼륨 활성화” 을 참조하십시오.



참고

미러링된 논리 볼륨에서 LVM 스냅샷이 지원됩니다.

스냅샷은 RAID 논리 볼륨에 대해 지원됩니다. RAID 논리 볼륨 생성에 대한 자세한 내용은 4.4.3절. “RAID 논리 볼륨” 을 참조하십시오.

LVM에서는 원본 볼륨의 크기와 볼륨에 필요한 메타데이터보다 큰 스냅샷 볼륨을 만들 수 없습니다. 이것보다 큰 스냅샷 볼륨을 지정하면 원본 크기에 필요한 만큼만 큰 스냅샷 볼륨을 생성합니다.

기본적으로 스냅샷 볼륨은 정상적인 활성화 명령 중에 건너뜁니다. 스냅샷 볼륨 활성화 제어에 대한 자세한 내용은 4.4.20절. “논리 볼륨 활성화 제어” 을 참조하십시오.

다음 명령은 /dev/vg00/snap 이라는 크기가 100MB인 스냅샷 논리 볼륨을 생성합니다. 그러면 /dev/vg00/lvol1 이라는 원본 논리 볼륨의 스냅샷이 생성됩니다. 원래 논리 볼륨에 파일 시스템이 포함된 경우 원래 파일 시스템이 계속 업데이트되는 동안 파일 시스템의 콘텐츠에 액세스하여 백업을 실행하기 위해 임의의 디렉터리에 snapshot 논리 볼륨을 마운트할 수 있습니다.

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

스냅샷 논리 볼륨을 생성한 후 lvdisplay 명령에서 origin 볼륨을 지정하면 모든 스냅샷 논리 볼륨 목록과 해당 상태(활성 또는 비활성)가 포함된 출력이 출력됩니다.

다음 예제에서는 /dev/new_vg/lvol0 논리 볼륨 /dev/new_vg/newvgsnap 이 생성된 논리 볼륨 /dev/new_vg/lvol0의 상태를 보여줍니다.

```
# 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
```

기본적으로 lvs 명령은 원본 볼륨과 사용 중인 스냅샷 볼륨의 현재 백분율을 표시합니다. 다음 예제에서는 스냅샷 볼륨 /dev/new_vg/newvgsnap 이 생성된 논리 볼륨 /dev/new_vg/lvol0 을 포함하는 시스템의 lvs 명령에 대한 기본 출력을 보여줍니다.

```
# 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
```

주의

스냅샷은 원본 볼륨이 변경됨에 따라 크기가 증가하므로 lvs 명령을 사용하여 스냅샷 볼륨의 백분율을 정기적으로 모니터링하여 채워지지 않도록 하는 것이 중요합니다. 원본의 변경되지 않은 부분에 대한 쓰기 작업은 스냅샷을 손상시키지 않고 성공하지 못하므로 100% 가득 차 있는 스냅샷이 완전히 손실됩니다.

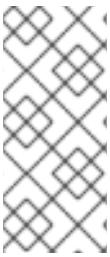
스냅샷은 가득 차 있을 때 무효화됩니다. 스냅샷 장치의 마운트된 파일 시스템은 강제로 마운트 해제되어 있어 마운트 지점에 액세스할 때 불필요한 파일 시스템 오류를 방지할 수 있습니다. 또한 lvm.conf 파일에 snapshot_autoextend_threshold 옵션을 지정할 수 있습니다. 이 옵션을 사용하면 나머지 스냅샷 공간이 설정한 임계값 미만으로 줄어들 때마다 스냅샷을 자동으로 확장할 수 있습니다. 이 기능을 사용하려면 볼륨 그룹에 할당되지 않은 공간이 있어야 합니다.

LVM에서는 원본 볼륨의 크기와 볼륨에 필요한 메타데이터보다 큰 스냅샷 볼륨을 만들 수 없습니다. 마찬가지로 스냅샷 자동 확장은 스냅샷에 필요한 최대 계산된 크기 이상으로 스냅샷 볼륨의 크기를 늘리지 않습니다. 스냅샷이 원본을 덮을 정도로 충분히 크게 증가하면 자동 확장을 위해 더 이상 모니터링되지 않습니다.

snapshot_autoextend_threshold 및 **snapshot_autoextend_percent** 설정에 대한 정보는 **lvm.conf** 파일 자체에 제공됩니다. **lvm.conf** 파일에 대한 자세한 내용은 *부록 B. LVM 구성 파일* 을 참조하십시오.

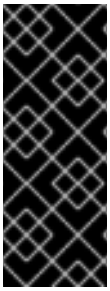
4.4.7. 씬 프로비저닝된 스냅샷 볼륨 생성

Red Hat Enterprise Linux는 씬 프로비저닝된 스냅샷 볼륨을 지원합니다. 씬 스냅샷 볼륨의 이점 및 제한 사항에 대한 자세한 내용은 *2.3.6절. "씬 프로비저닝된 스냅샷 볼륨"* 을 참조하십시오.



참고

이 섹션에서는 씬 프로비저닝된 스냅샷 볼륨을 생성하고 확장하는 데 사용하는 기본 명령에 대해 설명합니다. **LVM** 씬 프로비저닝 및 씬 프로비저닝된 논리 볼륨이 있는 **LVM** 명령 및 유틸리티 사용에 대한 자세한 내용은 **lvmthin(7)** 도움말 페이지를 참조하십시오.



중요

thin 스냅샷 볼륨을 생성할 때 볼륨 크기를 지정하지 않습니다. **size** 매개변수를 지정하면 생성되는 스냅샷은 씬 스냅샷 볼륨이 아니며 데이터를 저장하는 데 씬 풀을 사용하지 않습니다. 예를 들어 **lvcreate -s Cryostat/thinvolume -L10M** 명령은 원본 볼륨이 **thin** 볼륨인 경우에도 씬 스냅샷을 생성하지 않습니다.

씬 스냅샷은 씬 프로비저닝된 원본 볼륨 또는 씬 프로비저닝되지 않은 원본 볼륨에 대해 생성할 수 있습니다.

lvcreate 명령의 **--name** 옵션을 사용하여 스냅샷 볼륨의 이름을 지정할 수 있습니다. 다음 명령은 **mysnapshot1** 이라는 씬 프로비저닝된 논리 볼륨 **Cryostat 001/thinvolume** 의 씬 프로비저닝된 스냅샷 볼륨을 생성합니다.

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

참고

씬 프로비저닝을 사용할 때는 스토리지 관리자가 스토리지 풀을 모니터링하고 용량을 늘리기 시작하는 것이 중요합니다. **thin** 볼륨의 크기 확장에 대한 자세한 내용은 다음을 참조하십시오. **4.4.5절. "씬 프로비저닝된 논리 볼륨 생성"**

씬 스냅샷 볼륨은 다른 씬 볼륨과 동일한 특성을 갖습니다. 볼륨을 독립적으로 활성화하고, 볼륨 확장, 볼륨 이름 변경, 볼륨 제거, 볼륨 스냅샷도 수행할 수 있습니다.

기본적으로 스냅샷 볼륨은 정상적인 활성화 명령 중에 건너뜁니다. 스냅샷 볼륨 활성화 제어에 대한 자세한 내용은 **4.4.20절. "논리 볼륨 활성화 제어"** 을 참조하십시오.

프로비저닝되지 않은 논리 볼륨의 씬 프로비저닝된 스냅샷을 생성할 수도 있습니다. 프로비저닝되지 않은 논리 볼륨은 씬 풀 내에 포함되어 있지 않으므로 외부 원본 이라고 합니다. 외부 원본 볼륨은 다른 씬 풀에서도 씬 프로비저닝된 여러 스냅샷 볼륨에서 사용하고 공유할 수 있습니다. 씬 프로비저닝된 스냅샷이 생성될 때 외부 원본은 비활성이고 읽기 전용이어야 합니다.

외부 원본의 씬 프로비저닝된 스냅샷을 생성하려면 **--thinpool** 옵션을 지정해야 합니다. 다음 명령은 읽기 전용 비활성 볼륨 **origin_volume** 의 씬 스냅샷 볼륨을 생성합니다. **thin snapshot** 볼륨의 이름은 **mythinsnap** 입니다. 그런 다음 논리 볼륨 **origin_volume** 은 기존 **thin pool Cryostat 001 /pool** 을 사용할 볼륨 그룹 **mythinsnap** 에서 **thin snapshot** 볼륨 **mythinsnap**의 씬 외부 원본이 됩니다. 원본 볼륨이 스냅샷 볼륨과 동일한 볼륨 그룹에 있어야 하므로 원본 논리 볼륨을 지정할 때 볼륨 그룹을 지정할 필요가 없습니다.

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

다음 명령과 같이 첫 번째 스냅샷 볼륨의 두 번째 씬 프로비저닝 스냅샷 볼륨을 생성할 수 있습니다.

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

Red Hat Enterprise Linux 7.2에서는 **lvs** 명령의 **lv_ancestors** 및 **lv_descendants** 보고 필드를 지정하여 씬 스냅샷 논리 볼륨의 모든 상위 항목 및 하위 항목 목록을 표시할 수 있습니다.

다음 예제에서:

- **stack1** 은 볼륨 그룹 **Cryostat 001** 의 원본 볼륨입니다.

- *stack2 는 stack1의 스냅샷입니다.*
- *stack3 은 stack2의 스냅샷입니다.*
- *stack4 는 stack3의 스냅샷입니다.*

기타:

- *stack5 는 stack2의 스냅샷이기도 합니다.*
- *stack6 은 stack5의 스냅샷입니다.*

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

참고

lv_ancestors 및 lv_descendants 필드에는 기존 종속성이 표시되지만 체인 중간에 항목이 제거된 경우 제거된 항목이 추적되지 않습니다. 예를 들어 이 샘플 구성에서 논리 볼륨 stack3 을 제거하면 다음과 같습니다.

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

그러나 Red Hat Enterprise Linux 7.3에서는 제거된 논리 볼륨을 추적하고 표시하도록 시스템을 구성할 수 있으며 lv_ancestors_full 및 lv_descendants_full 필드를 지정하여 해당 볼륨을 포함하는 전체 종속성 체인을 표시할 수 있습니다. 기록 논리 볼륨 추적, 표시 및 제거에 대한 자세한 내용은 4.4.21절. “추적 및 표시 논리 볼륨 (Red Hat Enterprise Linux 7.3 및 later)” 을 참조하십시오.

4.4.8. LVM 캐시 논리 볼륨 생성

Red Hat Enterprise Linux 7.1 릴리스에서 LVM은 LVM 캐시 논리 볼륨을 완전 지원합니다. 캐시 논리 볼륨은 빠른 블록 장치(예: SSD 드라이브)로 구성된 작은 논리 볼륨을 사용하여 자주 사용되는 논리 볼륨에 자주 사용되는 블록을 저장하여 더 크고 느린 논리 볼륨의 성능을 향상시킵니다.

LVM 캐싱은 다음 LVM 논리 볼륨 유형을 사용합니다. 이러한 모든 관련 논리 볼륨은 동일한 볼륨 그룹에 있어야 합니다.

- origin 논리 볼륨 - 크고 느린 논리 볼륨
- 캐시 풀 논리 볼륨 - 캐시 데이터 논리 볼륨과 캐시 메타데이터 논리 볼륨이라는 두 개의 장치로 구성된 작고 빠른 논리 볼륨
- 캐시 데이터 논리 볼륨 - 캐시 풀 논리 볼륨의 데이터 블록을 포함하는 논리 볼륨
- 캐시 메타데이터 논리 볼륨 - 캐시 풀 논리 볼륨의 메타데이터가 포함된 논리 볼륨(예: 원본 논리 볼륨 또는 캐시 데이터 논리 볼륨)에 데이터 블록이 저장된 위치를 지정하는 계정 정보가 있습니다.

- **cache 논리 볼륨 - origin 논리 볼륨과 캐시 풀 논리 볼륨이 포함된 논리 볼륨입니다. 이는 다양한 캐시 볼륨 구성 요소를 캡슐화하는 결과 사용 가능한 장치입니다.**

다음 절차에서는 **LVM** 캐시 논리 볼륨을 생성합니다.

1. 느린 물리 볼륨과 빠른 물리 볼륨이 포함된 볼륨 그룹을 만듭니다. 이 예제에서. **/dev/sde1** 은 느린 장치이며 **/dev/sdf1** 은 빠른 장치이며 두 장치 모두 볼륨 그룹 **VG** 에 포함됩니다.

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

2. **origin** 볼륨을 생성합니다. 이 예에서는 크기가 **10GB**이고 느린 물리 볼륨인 **/dev/sde1** 로 구성된 **lv** 라는 원본 볼륨을 생성합니다.

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

3. 캐시 풀 논리 볼륨을 생성합니다. 이 예제에서는 볼륨 그룹 **VG** 의 일부인 빠른 장치 **/dev/sdf1** 에 **cpool** 이라는 캐시 풀 논리 볼륨을 생성합니다. 이 명령이 생성하는 캐시 풀 논리 볼륨은 숨겨진 캐시 데이터 **cpool_cdata** 및 숨겨진 캐시 메타데이터 **cpool_cmeta** 로 구성됩니다.

```
# lvcreate --type cache-pool -L 5G -n cpool VG /dev/sdf1
  Using default stripesize 64.00 KiB.
  Logical volume "cpool" created.
# lvs -a -o name,size,attr,devices VG
  LV              LSize  Attr       Devices
  [cpool]         5.00g Cwi---C--- cpool_cdata(0)
  [cpool_cdata]   5.00g Cwi-ao---- /dev/sdf1(4)
  [cpool_cmeta]   8.00m ewi-ao---- /dev/sdf1(2)
```

복잡한 구성의 경우 캐시 데이터와 캐시 메타데이터 논리 볼륨을 개별적으로 생성한 다음 볼륨을 캐시 풀 논리 볼륨으로 통합해야 할 수 있습니다. 이 절차에 대한 자세한 내용은 **lvmcache(7)** 도움말 페이지를 참조하십시오.

4. 캐시 풀 논리 볼륨을 원본 논리 볼륨에 연결하여 캐시 논리 볼륨을 만듭니다. 생성된 사용자 액세스 캐시 논리 볼륨은 원본 논리 볼륨의 이름을 사용합니다. **origin** 논리 볼륨은 원래 이름에 **_corig** 가 추가된 숨겨진 논리 볼륨이 됩니다. 이 변환은 라이브로 수행할 수 있지만 먼저 백업을 수행했는지 확인해야 합니다.

```
# lvconvert --type cache --cachepool cpool VG/lv
  Logical volume cpool is now cached.
```

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

5.

선택적으로 **Red Hat Enterprise Linux** 릴리스 **7.2**부터 캐시된 논리 볼륨을 씬 풀 논리 볼륨으로 변환할 수 있습니다. 풀에서 생성된 씬 논리 볼륨이 캐시를 공유합니다.

다음 명령은 씬 풀 메타데이터(**lv_tmeta**)를 할당하기 위해 빠른 장치 **/dev/sdf1** 을 사용합니다. 이는 캐시 풀 볼륨에서 사용하는 장치와 같습니다. 즉, **thin** 풀 메타데이터 볼륨은 캐시 데이터 논리 볼륨 **cpool_cdata** 및 캐시 메타데이터 논리 볼륨 **cpool_cmeta** 와 둘 다 해당 장치를 공유합니다.

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

추가 관리 예제를 포함한 **LVM** 캐시 볼륨에 대한 자세한 내용은 **lvmcache(7)** 매뉴얼 페이지를 참조하십시오.

씬 프로비저닝된 논리 볼륨을 생성하는 방법에 대한 자세한 내용은 **4.4.5**절. “씬 프로비저닝된 논리 볼륨 생성” 을 참조하십시오.

4.4.9. 스냅샷 볼륨 병합

lvconvert 명령의 **--merge** 옵션을 사용하여 스냅샷을 원본 볼륨에 병합할 수 있습니다. 원본 볼륨과 스냅샷 볼륨이 모두 열려 있지 않으면 병합이 즉시 시작됩니다. 그렇지 않으면 병합은 **origin** 또는 **snapshot**이 처음 활성화되어 있고 둘 다 닫을 때 시작됩니다. 스냅샷을 닫을 수 없는 원본(예: 루트 파일

시스템)에 병합하면 다음에 원본 볼륨이 활성화될 때까지 지연됩니다. 병합이 시작되면 결과 논리 볼륨의 이름, 마이너 번호 및 UUID가 표시됩니다. 병합이 진행 중인 동안 원본에 대한 읽기 또는 쓰기는 병합되는 스냅샷으로 지시된 대로 나타납니다. 병합이 완료되면 병합된 스냅샷이 제거됩니다.

다음 명령은 스냅샷 볼륨 192.0.2. 00/lvol1_snap 을 원본과 병합합니다.

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

명령줄에서 스냅샷을 여러 개 지정하거나 LVM 오브젝트 태그를 사용하여 각 원본에 여러 스냅샷을 병합하도록 지정할 수 있습니다. 다음 예에서 논리 volumes 00/lvol1, Cryostat00/lvol2, Cryostat 00/lvol3 은 모두 @some_tag 태그가 지정됩니다. 다음 명령은 세 개의 모든 볼륨에 대해 직렬로 스냅샷 논리 볼륨을 병합합니다. Cryostat 00/lvol1, then Cryostat 00/lvol2, then Cryostat 00/lvol3. --background 옵션을 사용하면 모든 스냅샷 논리 볼륨 병합이 병렬로 시작됩니다.

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

LVM 오브젝트 태그 지정에 대한 자세한 내용은 부록 D. LVM 오브젝트 태그 을 참조하십시오. lvconvert --merge 명령에 대한 자세한 내용은 lvconvert(8) 도움말 페이지를 참조하십시오.

4.4.10. 영구 장치 번호

메이저 및 마이너 장치 번호는 모듈 로드 시 동적으로 할당됩니다. 일부 애플리케이션은 블록 장치가 항상 동일한 장치 (major 및 minor) 번호로 활성화되는 경우 가장 잘 작동합니다. 다음 인수를 사용하여 lvcreate 및 lvchange 명령을 사용하여 이를 지정할 수 있습니다.

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

큰 마이너 번호를 사용하여 다른 장치에 동적으로 할당되지 않았는지 확인하십시오.

NFS를 사용하여 파일 시스템을 내보내는 경우 export 파일에서 fsid 매개변수를 지정하면 LVM 내에서 영구 장치 번호를 설정하지 않아도 됩니다.

4.4.11. 논리 볼륨 그룹의 매개 변수 변경

논리 볼륨의 매개 변수를 변경하려면 lvchange 명령을 사용합니다. 변경할 수 있는 매개변수 목록은 lvchange(8) 도움말 페이지를 참조하십시오.

lvchange 명령을 사용하여 논리 볼륨을 활성화하고 비활성화할 수 있습니다. 볼륨 그룹의 모든 논리 볼륨을 동시에 활성화하고 비활성화하려면 4.3.9절. "볼륨 그룹의 매개변수 변경" 에 설명된 대로

Cryostatchange 명령을 사용합니다.

다음 명령은 볼륨 그룹 **Cryostat 00** 의 볼륨 **lvol1** 에 대한 권한을 읽기 전용으로 변경합니다.

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

4.4.12. 논리 볼륨 이름 변경

기존 논리 볼륨의 이름을 변경하려면 **lvrename** 명령을 사용합니다.

다음 명령 중 하나는 볼륨 그룹 **lvold**의 논리 볼륨 **lvold** 의 이름을 **lv new** 로 바꿉니다.

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

```
# lvrename vg02 lvold lvnew
```

루트 논리 볼륨 이름을 변경하려면 추가 구성이 필요합니다. 루트 볼륨 이름 변경에 대한 자세한 내용은 **Red Hat Enterprise Linux**에서 루트 볼륨 그룹 또는 논리 볼륨 이름을 변경하는 방법을 참조하십시오.

클러스터의 개별 노드에서 논리 볼륨 활성화에 대한 자세한 내용은 **4.7**절. “클러스터의 개별 노드에서 논리 볼륨 활성화” 을 참조하십시오.

4.4.13. 논리 볼륨 제거

비활성 논리 볼륨을 제거하려면 **lvremove** 명령을 사용합니다. 논리 볼륨이 현재 마운트되어 있으면 제거하기 전에 볼륨을 마운트 해제합니다. 또한 클러스터형 환경에서는 논리 볼륨을 제거하기 전에 비활성화해야 합니다.

다음 명령은 볼륨 그룹 **testvg** 에서 논리 볼륨 **/dev/testvg/testlv** 를 제거합니다. 이 경우 논리 볼륨이 비활성화되지 않았습니다.

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

lvchange -an 명령으로 제거하기 전에 논리 볼륨을 명시적으로 비활성화할 수 있습니다. 이 경우 활성 논리 볼륨을 제거할지 여부를 확인하는 프롬프트가 표시되지 않습니다.

4.4.14. 논리 볼륨 표시

LVM 논리 볼륨의 속성을 표시하는 데 사용할 수 있는 명령은 **lvs,lvdisplay, lvscan** 입니다.

lvs 명령은 논리 볼륨 정보를 구성 가능한 형식으로 제공하여 논리 볼륨당 한 행을 표시합니다. **lvs** 명령은 많은 형식 제어를 제공하며 스크립팅에 유용합니다. **lvs** 명령을 사용하여 출력을 사용자 지정하는 방법에 대한 자세한 내용은 4.8절. “LVM에 대한 사용자 정의 보고” 을 참조하십시오.

lvdisplay 명령은 고정된 형식으로 논리 볼륨 속성(예: 크기, 레이아웃, 매핑)을 표시합니다.

다음 명령은 **Cryostat 00** 의 **lvol2** 속성을 보여줍니다. 이 원래 논리 볼륨에 대한 스냅샷 논리 볼륨이 생성된 경우 이 명령은 모든 스냅샷 논리 볼륨 목록과 해당 상태(활성 또는 비활성)도 표시합니다.

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

lvscan 명령은 시스템의 모든 논리 볼륨을 스캔하여 다음 예와 같이 나열합니다.

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

4.4.15. 논리 볼륨 증가

논리 볼륨의 크기를 늘리려면 **lvextend** 명령을 사용합니다.

논리 볼륨을 확장할 때 볼륨을 확장할 양 또는 확장 후 원하는 크기를 표시할 수 있습니다.

다음 명령은 논리 볼륨 **/dev/myvg/homevol** 을 **12GB**로 확장합니다.

```
# 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
```

다음 명령은 다른 기가바이트를 논리 볼륨 **/dev/myvg/homevol** 에 추가합니다.

```
# 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 명령과 마찬가지로 lvextend 명령의 -l 인수를 사용하여 논리 볼륨의 크기를 늘릴 확장 영역 수를 지정할 수 있습니다. 이 인수를 사용하여 볼륨 그룹의 백분율 또는 볼륨 그룹에서 남은 사용 가능한 공간의 백분율을 지정할 수도 있습니다. 다음 명령은 testlv 라는 논리 볼륨을 확장하여 볼륨 그룹 myvg 의 할당되지 않은 공간을 모두 채웁니다.

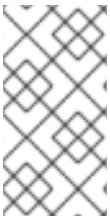
```
# lvextend -l +100%FREE /dev/myvg/testlv
  Extending logical volume testlv to 68.59 GB
  Logical volume testlv successfully resized
```

논리 볼륨을 확장한 후 파일 시스템 크기를 높여야 합니다.

기본적으로 대부분의 파일 시스템 크기 조정 도구는 파일 시스템의 크기가 기본 논리 볼륨의 크기로 증가하므로 두 명령 각각에 대해 동일한 크기를 지정할 필요가 없습니다.

4.4.16. 논리 볼륨 축소

lvreduce 명령을 사용하여 논리 볼륨의 크기를 줄일 수 있습니다.



참고

GFS2 또는 XFS 파일 시스템에서 축소는 지원되지 않으므로 GFS2 또는 XFS 파일 시스템이 포함된 논리 볼륨의 크기를 줄일 수 없습니다.

감소 중인 논리 볼륨에 파일 시스템이 포함되어 있는 경우 데이터 손실을 방지하기 위해 파일 시스템이 축소되는 논리 볼륨의 공간을 사용하지 않도록 해야 합니다. 따라서 논리 볼륨에 파일 시스템이 포함된 경우 lvreduce 명령의 --resizefs 옵션을 사용하는 것이 좋습니다. 이 옵션을 사용하면 lvreduce 명령은 논리 볼륨을 축소하기 전에 파일 시스템을 줄입니다. 파일 시스템이 가득 차거나 파일 시스템이 축소를 지원하지 않는 경우와 같이 파일 시스템을 축소하는 데 실패하는 경우 lvreduce 명령이 실패하고 논리 볼륨을 축소하지 않습니다.

주의

대부분의 경우 **lvreduce** 명령은 가능한 데이터 손실에 대해 경고하고 확인을 요청합니다. 그러나 경우에 따라 논리 볼륨이 비활성 상태이거나 **--resizefs** 옵션을 사용하지 않는 경우와 같이 이러한 프롬프트가 표시되지 않는 경우 데이터 손실을 방지하기 위해 이러한 확인 프롬프트를 사용하지 않아야 합니다.

lvreduce 명령의 **--test** 옵션을 사용하면 작업이 안전한 위치를 표시하지 않습니다. 이 옵션은 파일 시스템을 확인하거나 파일 시스템 크기 조정을 테스트하지 않습니다.

다음 명령은 볼륨 그룹 **lvol1** 의 논리 볼륨 **lvol 1**을 **64MB**로 줄입니다. 이 예에서 **lvol1** 에는 논리 볼륨과 함께 크기를 조정하는 파일 시스템이 포함되어 있습니다. 이 예에서는 명령에 대한 출력을 보여줍니다.

```
# lvreduce --resizefs -L 64M vg00/lvol1
fsck from util-linux 2.23.2
/dev/mapper/vg00-lvol1: clean, 11/25688 files, 8896/102400 blocks
resize2fs 1.42.9 (28-Dec-2013)
Resizing the filesystem on /dev/mapper/vg00-lvol1 to 65536 (1k) blocks.
The filesystem on /dev/mapper/vg00-lvol1 is now 65536 blocks long.

  Size of logical volume vg00/lvol1 changed from 100.00 MiB (25 extents) to 64.00 MiB (16 extents).
  Logical volume vg00/lvol1 successfully resized.
```

크기 조정 값 앞에 **-** 기호를 지정하면 논리 볼륨의 실제 크기에서 값이 뺀 것을 나타냅니다. 다음 예제에서는 **64** 메가바이트의 절대 크기로 논리 볼륨을 축소하는 대신 값 **64** 메가바이트로 볼륨을 축소하려는 경우 사용하는 명령을 보여줍니다.

```
# lvreduce --resizefs -L -64M vg00/lvol1
```

4.4.17. 스트립 볼륨 확장

스트라이핑된 논리 볼륨의 크기를 늘리려면 기본 물리 볼륨에 스트라이프를 지원하기 위해 볼륨 그룹을 구성하는 충분한 여유 공간이 있어야 합니다. 예를 들어 전체 볼륨 그룹을 사용하는 양방향 스트라이프가 있는 경우 볼륨 그룹에 단일 물리 볼륨을 추가하면 스트라이프를 확장할 수 없습니다. 대신 볼륨 그룹에 두 개 이상의 물리 볼륨을 추가해야 합니다.

예를 들어 다음 **Cryostats** 명령으로 표시된 대로 두 개의 기본 물리 볼륨으로 구성된 볼륨 그룹

***Cryostat**를 고려해 보십시오.*

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

볼륨 그룹의 전체 공간을 사용하여 스트라이프를 만들 수 있습니다.

```
# 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)
```

이제 볼륨 그룹에 사용 가능한 공간이 없습니다.

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

*다음 명령은 볼륨 그룹에 또 다른 물리 볼륨을 추가합니다. 그러면 **135GB**의 추가 공간이 있습니다.*

```
# 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
```

이 시점에서 데이터를 스트라이프하기 위해 두 개의 기본 장치가 필요하므로 스트립 논리 볼륨을 볼륨 그룹의 전체 크기로 확장할 수 없습니다.

```
# 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
```

스트라이핑된 논리 볼륨을 확장하려면 다른 물리 볼륨을 추가한 다음 논리 볼륨을 확장합니다. 이 예에서는 볼륨 그룹에 두 개의 물리 볼륨을 추가하여 논리 볼륨을 볼륨 그룹의 전체 크기로 확장할 수 있습니다.

```
# 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
```

스트라이핑된 논리 볼륨을 확장하는 데 기본 물리 장치가 충분하지 않은 경우 확장 기능을 사용하지 않는 경우 볼륨을 확장할 수 있습니다. 이로 인해 성능이 저하되지 않을 수 있습니다. 논리 볼륨에 공간을 추가할 때 기본 작업은 기존 논리 볼륨의 마지막 세그먼트와 동일한 스트립 매개 변수를 사용하지만 이러한 매개 변수를 재정의할 수 있습니다. 다음 예제에서는 **lvextend** 초기 명령이 실패한 후 남은 여유 공간을 사용하도록 기존의 스트라이핑 논리 볼륨을 확장합니다.

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

4.4.18. RAID 볼륨 확장

새 **RAID** 영역의 동기화를 수행하지 않고 **lvextend** 명령을 사용하여 **RAID** 논리 볼륨을 확장할 수 있습니다.

lvcreate 명령을 사용하여 **RAID** 논리 볼륨을 생성할 때 **--nosync** 옵션을 지정하면 논리 볼륨이 생성될 때 **RAID** 영역이 동기화되지 않습니다. 나중에 **--nosync** 옵션을 사용하여 생성한 **RAID** 논리 볼륨을 확장하면 **RAID** 확장이 해당 시점에 동기화되지 않습니다.

lvs 명령을 사용하여 볼륨 속성을 표시하여 기존 논리 볼륨이 **--nosync** 옵션으로 생성되었는지 여부를 확인할 수 있습니다. 논리 볼륨은 초기 동기화없이 생성된 **RAID** 볼륨인 경우 특성 필드의 첫 번째 문자로 **"R"**을 표시하고 초기 동기화로 생성된 경우 **"r"**이 표시됩니다.

다음 명령은 초기 동기화 없이 생성된 **lv** 라는 **RAID** 논리 볼륨의 속성을 표시하여 **"R"**을 특성 필드에서 첫 번째 문자로 표시합니다. 특성 필드의 일곱 번째 문자는 대상 **RAID** 유형을 나타내는 **"r"**입니다. **attribute** 필드의 의미에 대한 자세한 내용은 표 4.5. “LVs 표시 필드” 을 참조하십시오.

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

lvextend 명령으로 이 논리 볼륨을 늘리면 RAID 확장이 다시 동기화되지 않습니다.

lvcreate 명령의 --nosync 옵션을 지정하지 않고 RAID 논리 볼륨을 생성한 경우 lvextend 명령의 --nosync 옵션을 지정하여 미러를 다시 동기화하지 않고 논리 볼륨을 확장할 수 있습니다.

다음 예제에서는 --nosync 옵션 없이 생성된 RAID 논리 볼륨을 확장하여 RAID 볼륨이 생성될 때 동기화되었음을 나타냅니다. 그러나 이 예에서는 볼륨이 확장될 때 볼륨이 동기화되지 않도록 지정합니다. 볼륨에는 "r" 속성이 있지만 --nosync 옵션을 사용하여 lvextend 명령을 실행한 후 볼륨에는 "R" 속성이 있습니다.

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

RAID 볼륨이 비활성화된 경우 --nosync 옵션을 사용하여 볼륨을 생성하는 경우에도 볼륨을 확장할 때 동기화를 자동으로 생략하지 않습니다. 대신 논리 볼륨의 확장된 부분을 완전히 다시 동기화할지 여부를 묻는 메시지가 표시됩니다.

참고

RAID 볼륨에서 복구 작업을 수행하는 경우 --nosync 옵션을 사용하여 볼륨을 생성하거나 확장하면 논리 볼륨을 확장할 수 없습니다. 그러나 --nosync 옵션을 지정하지 않은 경우 복구 중 RAID 볼륨을 확장할 수 있습니다.

4.4.19. 클링 할당 정책을 사용하여 논리 볼륨 확장

LVM 볼륨을 확장할 때 lvextend 명령의 --alloc cling 옵션을 사용하여 클링 할당 정책을 지정할 수 있습니다. 이 정책은 기존 논리 볼륨의 마지막 부분과 동일한 물리 볼륨에서 공간을 선택합니다. 물리 볼륨에 공간이 충분하지 않고 lvm.conf 파일에 태그 목록이 정의되어 있는 경우 LVM은 물리적 볼륨에 태그가 연결되어 있는지 여부를 확인하고 기존 Extent와 새 Extent 간의 물리 볼륨 태그와 일치시킵니다.

예를 들어 단일 볼륨 그룹 내의 두 사이트 간에 미러링된 논리 볼륨이 있는 경우 @site1 및 @site2 태그로 물리 볼륨을 태그하여 배치되는 위치에 따라 물리 볼륨을 태그할 수 있습니다. 그런 다음 lvm.conf 파일에 다음 행을 지정할 수 있습니다.

-

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

물리 볼륨 태그에 대한 자세한 내용은 부록 D. LVM 오브젝트 태그 을 참조하십시오.

다음 예제에서는 다음 행을 포함하도록 **lvm.conf** 파일이 수정되었습니다.

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

또한 이 예에서 물리 볼륨 **/dev/sdb1,/dev/sdc1,/dev/sdd1,/dev/sde1, /dev/sdf1 ,/dev /sdg1** 로 구성된 볼륨 그룹 **taft** 가 생성되었습니다. 이러한 물리 볼륨에는 **A,B** 및 **C** 태그가 지정되었습니다. 이 예제에서는 **C** 태그를 사용하지 않지만 **LVM**에서 태그를 사용하여 미러 브릿지에 사용할 물리 볼륨을 선택함을 보여줍니다.

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

다음 명령은 볼륨 그룹 **taft** 에서 **10GB** 미러링된 볼륨을 생성합니다.

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

다음 명령은 미러 다리 및 **RAID** 메타데이터 하위 볼륨에 사용되는 장치를 보여줍니다.

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

다음 명령은 클링 할당 정책을 사용하여 동일한 태그가 있는 물리 볼륨을 사용하여 미러 브리지를 확장해야 함을 나타내는 미러링된 볼륨의 크기를 확장합니다.

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

다음 **display** 명령은 다리와 동일한 태그가 있는 물리 볼륨을 사용하여 미러가 확장되었음을 보여줍니다. **C** 태그가 있는 물리 볼륨은 무시되었습니다.

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

4.4.20. 논리 볼륨 활성화 제어

lvcreate 또는 **lvchange** 명령의 **-k** 또는 **--setactivationskip {y|n}** 옵션을 사용하여 논리 볼륨을 건너뛰도록 지정할 수 있습니다. 이 플래그는 비활성화 중에 적용되지 않습니다.

다음 예제와 같이 **k** 속성을 표시하는 **lvs** 명령을 사용하여 이 플래그가 논리 볼륨에 설정되어 있는지 여부를 확인할 수 있습니다.

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

기본적으로 씬 스냅샷 볼륨은 활성화 건너뛰기 위해 플래그가 지정됩니다. **standard -ay** 또는 **--activate y** 옵션 외에도 **-K** 또는 **--ignoreactivationskip** 옵션을 사용하여 **k** 속성이 설정된 논리 볼륨을 활성화할 수 있습니다.

다음 명령은 **thin snapshot** 논리 볼륨을 활성화합니다.

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

lvcreate 명령의 **-kn** 또는 **--setactivationskip n** 옵션을 지정하여 논리 볼륨을 생성할 때 영구 "활성화 건너뛰기" 플래그를 해제할 수 있습니다. **lvchange** 명령의 **-kn** 또는 **--setactivationskip n** 옵션을 지정하여 기존 논리 볼륨에 대한 플래그를 해제할 수 있습니다. **-ky** 또는 **--setactivationskip y** 옵션을 사용하여 플래그를 다시 설정할 수 있습니다.

다음 명령은 활성화 건너뛰기 플래그 없이 스냅샷 논리 볼륨을 생성합니다.

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

다음 명령은 스냅샷 논리 볼륨에서 활성화 건너뛰기 플래그를 제거합니다.

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

/etc/lvm/lvm.conf 파일의 **auto_set_activation_skip** 설정을 사용하여 기본 활성화 건너뛰기 설정을 제어할 수 있습니다.

4.4.21. 추적 및 표시 논리 볼륨 (Red Hat Enterprise Linux 7.3 및 later)

Red Hat Enterprise Linux 7.3부터 **lvm.conf** 구성 파일에서 **record_lvs_history** 메타데이터 옵션을 활성화하여 제거된 **thin snapshot** 및 **thin** 논리 볼륨을 추적하도록 시스템을 구성할 수 있습니다. 이를 통해 원래 종속성 체인에서 제거되어 과거 논리 볼륨이 된 논리 볼륨이 포함된 전체 씬 스냅샷 종속성 체인을 표시할 수 있습니다.

lvm.conf 구성 파일의 **lvs_history_retention_time** 메타데이터 옵션을 사용하여 보존 시간(초)을 지정하여 정의된 기간 동안 기록 볼륨을 유지하도록 시스템을 구성할 수 있습니다.

이전 논리 볼륨은 볼륨의 다음 보고 필드를 포함하여 제거된 논리 볼륨을 간략하게 표시합니다.

- **lv_time_removed:** 논리 볼륨의 제거 시간
- **lv_time:** 논리 볼륨의 생성 시간
- **lv_name:** 논리 볼륨의 이름
- **lv_uuid:** 논리 볼륨의 **UUID**
- **VG_ NAME:** 논리 볼륨이 포함된 볼륨 그룹입니다.

*볼륨이 제거되면 기록 논리 볼륨 이름이 접두사로 **hypen**을 가져옵니다. 예를 들어 논리 볼륨 **lvol1** 을 제거하면 기록 볼륨의 이름은 **-lvol1** 입니다. 기록 논리 볼륨을 다시 활성화할 수 없습니다.*

***record_lvs_history** 메타데이터 옵션을 활성화하면 **lvremove** 명령의 **--nohistory** 옵션을 지정하여 논리 볼륨을 제거할 때 개별적으로 기록 논리 볼륨을 보존할 수 있습니다.*

*기록 논리 볼륨을 볼륨 디스플레이에 포함하려면 **LVM display** 명령의 **-H|-history** 옵션을 지정합니다. **-H** 옵션과 함께 **lv_full_ancestors** 및 **lv_full_descendants** 보고 필드를 지정하여 기록 볼륨을 포함하는 전체 씬 스냅샷 종속성 체인을 표시할 수 있습니다.*

다음 일련의 명령은 기록 논리 볼륨을 표시하고 관리하는 방법에 대한 예를 제공합니다.

1. *__lvm.conf__ 파일에서 **record_lvs_history=1** 을 설정하여 기록 논리 볼륨이 유지되는지 확인합니다. 이 메타데이터 옵션은 기본적으로 활성화되어 있지 않습니다.*

2. *다음 명령을 입력하여 씬 프로비저닝된 스냅샷 체인을 표시합니다.*

   이 예제에서는 다음을 수행합니다.

   - ***lvol1** 은 체인의 첫 번째 볼륨인 원본 볼륨입니다.*
   - ***lvol2** 는 **lvol1** 의 스냅샷입니다.*
   - ***lvol3** 은 **lvol2** 의 스냅샷입니다.*
   - ***lvol4** 는 **lvol3** 의 스냅샷입니다.*
   - ***lvol5** 는 **lvol3** 의 스냅샷이기도 합니다.*

   ***lvs display** 명령에 **-H** 옵션이 포함되어 있지만 씬 스냅샷 볼륨이 아직 제거되지 않았으며 표시할 기록 논리 볼륨이 없습니다.*

```
# lvs -H -o name,full_ancestors,full_descendants
  LV    FAncestors       FDescendants
  lvol1                  lvol2,lvol3,lvol4,lvol5
  lvol2 lvol1            lvol3,lvol4,lvol5
  lvol3 lvol2,lvol1      lvol4,lvol5
  lvol4 lvol3,lvol2,lvol1
  lvol5 lvol3,lvol2,lvol1
  pool
```

3.

스냅샷 체인에서 논리 볼륨 **lvol3** 을 제거한 다음 다음 **lvs** 명령을 다시 실행하여 **ancestors** 및 하위 항목과 함께 기록 논리 볼륨이 표시되는 방법을 확인합니다.

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

4.

lv_time_removed reporting 필드를 사용하여 기록 볼륨이 제거된 시간을 표시할 수 있습니다.

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

5.

다음 예와 같이 **vgname/lvname** 형식을 지정하여 **display** 명령으로 기록 논리 볼륨을 개별적으로 참조할 수 있습니다. **lv_attr** 필드의 **5**번째 비트는 **h** 로 설정되어 볼륨이 기록 볼륨임을 나타냅니다.

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

6.

볼륨에 라이브 하위 항목이 없는 경우 **LVM**에서 이전 논리 볼륨을 유지하지 않습니다. 즉, 스냅샷 체인 끝에 논리 볼륨을 제거하면 논리 볼륨이 기록 논리 볼륨으로 유지되지 않습니다.

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

7.

*다음 명령을 실행하여 **lvol1** 및 **lvol2** 볼륨을 제거하고 **lvs** 명령이 제거되면 볼륨을 표시하는 방법을 확인합니다.*

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

8.

*기록 논리 볼륨을 완전히 제거하려면 다음 예제와 같이 현재 하이픈을 포함하는 기록 볼륨의 이름을 지정하여 **lvremove** 명령을 다시 실행할 수 있습니다.*

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

9.

하위 항목에 라이브 볼륨을 포함하는 체인이 있는 한 이전 논리 볼륨이 유지됩니다. 즉, 이전 논리 볼륨을 제거하면 다음 예제와 같이 체인의 기존 하위 항목이 연결되지 않은 경우 체인의 모든 논리 볼륨도 제거됩니다.

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

4.5. 필터를 사용하여 LVM 장치 스캔 제어

시작 시 192.0.2. scan 명령을 실행하여 LVM 레이블을 찾고, 해당 중 어느 것이 물리 볼륨인지 확인하고 메타데이터를 읽고 볼륨 그룹 목록을 구축하기 위해 시스템의 블록 장치를 스캔합니다. 물리 볼륨의 이름은 시스템에 있는 각 노드의 LVM 캐시 파일 /etc/lvm/cache/.cache 에 저장됩니다. 후속 명령은 다시 스캔을 피하기 위해 해당 파일을 읽을 수 있습니다.

lvm.conf 구성 파일에서 필터를 설정하여 LVM 스캔 장치를 제어할 수 있습니다. lvm.conf 파일의 필터는 /dev 디렉토리의 장치 이름에 적용되는 일련의 간단한 정규식으로 구성되어 있으며, 각 블록 장치를 수락하거나 거부할지 여부를 결정합니다.

다음 예제에서는 필터를 사용하여 LVM 스캔 장치를 제어하는 방법을 보여줍니다. 정규 표현식이 전체 경로 이름에 대해 자유롭게 일치하므로 이러한 예 중 일부는 권장되는 관행을 나타내는 것은 아닙니다. 예를 들어 a/loop/ 는 a/.*loop.*/ 와 동일하며 /dev/solooperation/lvol1 과 일치합니다.

다음 필터는 검색된 모든 장치를 추가합니다. 이는 구성 파일에 필터가 구성되어 있지 않기 때문에 기본 동작입니다.

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

다음 필터는 드라이브에 미디어가 없는 경우 지연을 방지하기 위해 cdrom 장치를 제거합니다.

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

다음 필터는 모든 루프를 추가하고 다른 모든 블록 장치를 제거합니다.

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

다음 필터는 모든 루프 및 IDE를 추가하고 다른 모든 블록 장치를 제거합니다.

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

다음 필터는 첫 번째 IDE 드라이브에 파티션 8만 추가하고 다른 모든 블록 장치를 제거합니다.

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

참고

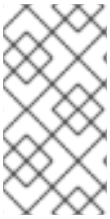
lvmetad 데몬이 실행 중인 경우 pvscan --cache device 명령을 실행할 때 /etc/lvm/lvm.conf 파일의 filter = 설정이 적용되지 않습니다. 장치를 필터링하려면 global_filter = 설정을 사용해야 합니다. 글로벌 필터를 장애가 발생한 장치는 LVM에서 열지 않으며 스캔하지 않습니다. 예를 들어 VM에서 LVM 장치를 사용하는 경우 글로벌 필터를 사용해야 할 수 있으며 VM의 장치 내용이 물리적 호스트에서 스캔되지 않도록 해야 합니다.

lvm.conf 파일에 대한 자세한 내용은 부록 B. LVM 구성 파일 및 lvm.conf(5) 도움말 페이지를 참조하십시오.

4.6. 온라인 데이터 재할당

pvmove 명령과 함께 시스템을 사용하는 동안 데이터를 이동할 수 있습니다.

pvmove 명령은 섹션으로 이동할 데이터를 분할하고 각 섹션을 이동하는 임시 미러를 생성합니다. pvmove 명령 작업에 대한 자세한 내용은 pvmove(8) 도움말 페이지를 참조하십시오.



참고

클러스터에서 pvmove 작업을 수행하려면 cmirror 패키지가 설치되고 cmirrord 서비스가 실행 중인지 확인해야 합니다.

다음 명령은 할당된 모든 공간을 물리 볼륨 /dev/sdc1 에서 볼륨 그룹의 기타 사용 가능한 물리 볼륨으로 이동합니다.

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

다음 명령은 논리 볼륨 MyLV 의 확장 영역만 이동합니다.

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

pvmove 명령을 실행하는 데 시간이 오래 걸릴 수 있으므로 백그라운드에서 명령을 실행하여 진행 상황 업데이트를 표시하지 않도록 할 수 있습니다. 다음 명령은 할당된 모든 Extent를 물리 볼륨 /dev/sdc1 로 이동하여 백그라운드에서 /dev/sdf1 로 이동합니다.

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

다음 명령은 **pvmove** 명령의 진행 상황을 **5**초 간격으로 백분율로 보고합니다.

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

4.7. 클러스터의 개별 노드에서 논리 볼륨 활성화

클러스터 환경에 **LVM**을 설치한 경우 한 노드에서 독점적으로 논리 볼륨을 활성화해야 할 수 있습니다.

하나의 노드에서만 논리 볼륨을 활성화하려면 **lvchange -aey** 명령을 사용합니다. 또는 **lvchange -aly** 명령을 사용하여 로컬 노드에서만 논리 볼륨을 활성화할 수 있지만 독점적으로는 활성화할 수 없습니다. 나중에 추가 노드에서 동시에 활성화할 수 있습니다.

부록 **D. LVM** *오브젝트 태그* 에 설명된 **LVM** 태그를 사용하여 개별 노드에서 논리 볼륨을 활성화할 수도 있습니다. *부록* **B. LVM** *구성 파일* 에 설명된 구성 파일에서 노드 활성화를 지정할 수도 있습니다.

4.8. LVM에 대한 사용자 정의 보고

LVM은 사용자 지정 보고서를 생성하고 보고서의 출력을 필터링하는 다양한 구성 및 명령줄 옵션을 제공합니다. **LVM** 보고 기능 및 기능에 대한 자세한 내용은 **lvmreport(7)** 매뉴얼 페이지를 참조하십시오.

pvs,lvs 및 **Cryostats** 명령을 사용하여 **LVM** 오브젝트에 대한 간결하고 사용자 지정 가능한 보고서를 생성할 수 있습니다. 이러한 명령이 생성되는 보고서에는 각 오브젝트에 대한 출력 한 줄이 포함됩니다. 각 줄에는 오브젝트와 관련된 속성의 정렬된 필드 목록이 포함되어 있습니다. 보고할 오브젝트를 선택하는 방법은 물리 볼륨, 볼륨 그룹, 논리 볼륨, 물리 볼륨 세그먼트 및 논리 볼륨 세그먼트에서 선택할 수 있습니다.

Red Hat Enterprise Linux 7.3 릴리스에서는 **lvm fullreport** 명령을 사용하여 물리 볼륨, 볼륨 그룹, 논리 볼륨 세그먼트, 물리 볼륨 세그먼트 및 논리 볼륨 세그먼트에 대한 정보를 한 번에 보고할 수 있습니다. 이 명령 및 해당 기능에 대한 자세한 내용은 **lvm-fullreport(8)** 도움말 페이지를 참조하십시오.

Red Hat Enterprise Linux 7.3 릴리스부터 **LVM**에서는 작업, 메시지, 개체별 상태 로그와 **LVM** 명령 실행 중에 수집된 전체 오브젝트 식별이 포함된 로그 보고서를 지원합니다. **LVM** 로그 보고서의 예는 *4.8.6 절. "명령 로그 보고(Red Hat Enterprise Linux 7.3 이상)"* 를 참조하십시오. **LVM** 로그 보고서에 대한 자세한 내용은 **lvmreport(7)** 도움말 페이지를 참조하십시오.

다음 섹션에서는 **pvs,lvs** 및 **Cryostats** 명령을 사용하여 보고서를 사용자 지정하는 방법에 대한 요약 정보를 제공합니다.

- 4.8.1절. “형식 제어”..., 보고서의 형식을 제어하는 데 사용할 수 있는 명령 인수의 요약을 제공합니다.
- 4.8.2절. “오브젝트 표시 필드”.. - 각 **LVM** 오브젝트에 대해 표시할 수 있는 필드 목록을 제공합니다.
- 4.8.3절. “LVM 보고서 정렬”생성된 보고서를 정렬하는 데 사용할 수 있는 명령 인수의 요약을 제공합니다.**Provides a summary of command arguments you can use to sort the generated report.**
- 4.8.4절. “단위 지정”보고서 출력 단위를 지정하기 위한 지침을 제공하는 .
- 4.8.5절. “JSON 형식 출력(Red Hat Enterprise Linux 7.3 이상)”.. **JSON** 형식 출력(**Red Hat Enterprise Linux 7.3** 이상)을 지정하는 예제를 제공합니다.
- 4.8.6절. “명령 로그 보고(Red Hat Enterprise Linux 7.3 이상)”명령 로그의 예를 제공합니다.

4.8.1. 형식 제어

pvs,lvs 또는 **Cryostats** 명령을 사용할지 여부에 따라 표시된 기본 필드 세트와 정렬 순서가 결정됩니다. 다음 인수를 사용하여 이러한 명령의 출력을 제어할 수 있습니다.

- **-o** 인수를 사용하여 기본값 이외의 필드에 표시되는 필드를 변경할 수 있습니다. 예를 들어 다음 출력은 **pvs** 명령의 기본 표시입니다(물리 볼륨에 대한 정보를 표시함).

```
# 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
```

다음 명령은 물리 볼륨 이름과 크기만 표시합니다.

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

- **-o** 인수와 함께 사용되는 더하기 기호(**+**)를 사용하여 출력에 필드를 추가할 수 있습니다.

  다음 예제는 기본 필드 외에 물리 볼륨의 **UUID**를 표시합니다.

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

- 명령에 **-v** 인수를 추가하면 몇 가지 추가 필드가 포함됩니다. 예를 들어 **pvs -v** 명령은 기본 필드 외에도 **DevSize** 및 **PV UUID** 필드를 표시합니다.

```
# pvs -v
    Scanning for physical volume names
  PV         VG     Fmt  Attr PSize  PFree  DevSize PV UUID
  /dev/sdb1  new_vg lvm2 a-   17.14G 17.14G  17.14G onFF2w-1fLC-ughJ-D9eB-M7iv-
6XqA-dqGeXY
  /dev/sdc1  new_vg lvm2 a-   17.14G 17.09G  17.14G Joqlch-yWSj-kuEn-IdwM-01S9-
XO8M-mcpsVe
  /dev/sdd1  new_vg lvm2 a-   17.14G 17.14G  17.14G yvfvZK-Cf31-j75k-dECm-0RZ3-
0dGW-tUqkCS
```

- **--noheadings** 인수는 제목 행을 표시하지 않습니다. 이 명령은 스크립트를 작성하는 데 유용할 수 있습니다.

  다음 예제에서는 모든 물리 볼륨 목록을 생성하는 **pv_name** 인수와 함께 **--noheadings** 인수를 사용합니다.

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

```
/dev/sdc1
/dev/sdd1
```

- **--separator separator** 인수는 구분 자를 사용하여 각 필드를 구분합니다.

  다음 예제에서는 **equals** 기호(=)를 사용하여 **pvs** 명령의 기본 출력 필드를 구분합니다.

```
# 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
```

  구분 기호 인수를 사용할 때 필드를 정렬하려면 **--aligned** 인수와 함께 **separator** 인수를 사용합니다.

```
# 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
```

표시 인수의 전체 목록은 **pvs(8), Cryostats(8)** 및 **lvs(8)** 도움말 페이지를 참조하십시오.

볼륨 그룹 필드는 물리 볼륨(및 물리 볼륨 세그먼트) 필드 또는 논리 볼륨(및 논리 볼륨 세그먼트) 필드와 혼합할 수 있지만 물리 볼륨 및 논리 볼륨 필드는 혼합할 수 없습니다. 예를 들어 다음 명령은 각 물리 볼륨에 대해 하나의 출력 행을 표시합니다.

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

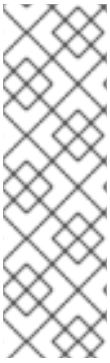
4.8.2. 오브젝트 표시 필드

이 섹션에서는 **pvs, Cryostats** 및 **lvs** 명령을 사용하여 **LVM** 오브젝트에 대해 표시할 수 있는 정보를 나열하는 일련의 테이블을 제공합니다.

편의를 위해 명령에 대한 기본값과 일치하는 경우 필드 이름 접두사를 삭제할 수 있습니다. 예를 들어 **pvs** 명령을 사용하면 **name**은 **pv_ name** 을 의미하지만, **Cryostats** 명령을 사용하면 **name** 이 **Cryostat _name**으로 해석됩니다.

다음 명령을 실행하는 것은 **pvs -o pv_free** 를 실행하는 것과 동일합니다.

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



참고

pvs, Cryostats , lvs 출력의 특성 필드에 있는 문자 수가 이후 릴리스에서 증가할 수 있습니다. 기존 문자 필드는 위치가 변경되지 않지만 새 필드가 끝에 추가될 수 있습니다. 이를 위해서는 특정 특성 문자를 검색하는 스크립트를 작성할 때 필드 시작 부분에 대한 상대적 위치에 따라 문자를 검색하지만 필드의 끝 부분에 대한 상대적 위치는 고려하지 않아야 합니다. 예를 들어 **lv_attr** 필드의 **ninth** 비트에서 문자 **p** 를 검색하려면 문자열 **"^/.....p/"**을 검색할 수 있지만 문자열 **"/*p$/"**를 검색해서는 안 됩니다.

4.8.2.1. pvs 명령

표 4.3. "pvs 명령 표시 필드" **pvs** 명령의 표시 인수와 함께 헤더 표시 및 필드에 표시된 대로 필드 이름을 나열합니다.

표 4.3. pvs 명령 표시 필드

인수	header	설명
dev_size	DevSize	물리 볼륨이 생성된 기본 장치의 크기
pe_start	첫 번째 PE	기본 장치에서 첫 번째 물리 확장 영역의 시작 부분까지 오프셋
pv_attr	attr	물리 볼륨의 상태: (a)llocatable 또는 e(x)ported.
pv_fmt	FMT	물리 볼륨의 메타데이터 형식(lvm2 또는 lvm1)
pv_free	PFree	물리 볼륨에 남은 여유 공간
pv_name	PV	물리 볼륨 이름

인수	header	설명
pv_pe_alloc_count	alloc	사용된 물리 확장 영역 수
pv_pe_count	PE	물리 확장 영역 수
pvseg_size	SSize	물리 볼륨의 세그먼트 크기입니다.
pvseg_start	시작	물리 볼륨 세그먼트의 시작 물리 확장
pv_size	PSize	물리 볼륨의 크기
pv_tags	PV 태그	물리 볼륨에 연결된 LVM 태그
pv_used	사용됨	물리 볼륨에서 현재 사용된 공간의 크기
pv_uuid	PV UUID	물리 볼륨의 UUID

pvs 명령은 기본적으로 pv_name , pv_name,pv_fmt,pv_attr,pv_size,pv_free 라는 필드를 표시합니다. 디스플레이는 pv_name 에 따라 정렬됩니다.

```
# 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 명령과 함께 -v 인수를 사용하면 기본 디스플레이인 dev_size,pv_uuid 에 다음 필드가 추가됩니다.

```
# pvs -v
    Scanning for physical volume names
  PV         VG     Fmt  Attr PSize  PFree  DevSize PV UUID
  /dev/sdb1  new_vg lvm2 a-   17.14G 17.14G  17.14G onFF2w-1fLC-ughJ-D9eB-M7iv-6XqA-dqGeXY
  /dev/sdc1  new_vg lvm2 a-   17.14G 17.09G  17.14G Joqlch-yWSj-kuEn-IdwM-01S9-XO8M-mcpsVe
  /dev/sdd1  new_vg lvm2 a-   17.14G 17.13G  17.14G yvfvZK-Cf31-j75k-dECm-0RZ3-0dGW-tUqkCS
```

pvs 명령의 --segments 인수를 사용하여 각 물리 볼륨 세그먼트에 대한 정보를 표시할 수 있습니다. 세그먼트는 확장 영역 그룹입니다. 세그먼트 보기는 논리 볼륨이 조각화되는지 여부를 확인하려는 경우

유용할 수 있습니다.

***pvs --segments** 명령은 기본적으로 **pv_name, Cryostat_name ,pv_ fmt,pv_attr,pv_size,pv_free,pvseg_start,pvseg_size.** 디스플레이는 물리 볼륨 내에서 **pv_name** 및 **pvseg_size** 에 따라 정렬됩니다.*

```
# 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** 명령을 사용하여 **LVM** 물리 볼륨으로 초기화되지 않은 **LVM**에서 감지한 장치를 확인할 수 있습니다.*

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

4.8.2.2. vgs 명령

표 4.4. "VGs 필드 표시" 헤더 표시에 표시되는 필드 이름 및 필드에 대한 설명과 함께 field name의 표시 인수를 나열합니다.

표 4.4. VGs 필드 표시

인수	header	설명
lv_count	#LV	볼륨 그룹에 포함된 논리 볼륨 수
max_lv	MaxLV	볼륨 그룹에 허용되는 최대 논리 볼륨 수(0 무제한)
max_pv	MaxPV	볼륨 그룹에 허용되는 최대 물리 볼륨 수(0 무제한)
pv_count	#PV	볼륨 그룹을 정의하는 물리 볼륨 수
snap_count	#SN	볼륨 그룹에 포함된 스냅샷 수
vg_attr	attr	볼륨 그룹의 상태: (w)r(r)eadonly, resi(z)eported, (p)artial 및 (c)lustered.
vg_extent_count	#ext	볼륨 그룹의 물리 확장 영역 수
vg_extent_size	ext	볼륨 그룹의 물리 확장 영역의 크기
vg_fmt	FMT	볼륨 그룹의 메타데이터 형식(lvm2 또는 lvm1)
vg_free	VFree	볼륨 그룹에 남아 있는 여유 공간의 크기
vg_free_count	무료	볼륨 그룹에서 사용 가능한 물리 확장 영역 수
vg_name	VG	볼륨 그룹 이름
vg_seqno	seq	볼륨 그룹의 리비전을 나타내는 숫자
vg_size	VSize	볼륨 그룹의 크기
vg_sysid	SYS ID	LVM1 시스템 ID
vg_tags	VG 태그	볼륨 그룹에 연결된 LVM 태그
vg_uuid	VG UUID	볼륨 그룹의 UUID

Cryostats 명령은 기본적으로 다음 필드를 표시합니다. **Cryostat _name,pv_count,lv_count,snap_count, Cryostat_attr, Cryostat_ size.** 디스플레이는 **vg_name** 에 따라 정렬됩니다.

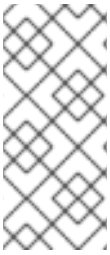
```
# vgs
  VG     #PV #LV #SN Attr   VSize  VFree
  new_vg   3   1   1 wz--n- 51.42G 51.36G
```

Cryostats 명령과 함께 **-v** 인수를 사용하면 기본 디스플레이에 다음 필드가 추가됩니다. **Cryostat _extent_size, Cryostat_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
```

4.8.2.3. lvs 명령

표 4.5. "LVs 표시 필드" **lvs** 명령의 표시 인수와 함께 헤더 표시에 표시되는 필드 이름 및 필드에 대한 설명을 나열합니다.



참고

Red Hat Enterprise Linux의 최신 릴리스에서는 출력에 추가 필드가 추가되면서 **lvs** 명령의 출력이 다를 수 있습니다. 그러나 필드의 순서는 동일하게 유지되며 추가 필드는 디스플레이 끝에 표시됩니다.

표 4.5. LVs 표시 필드

인수	header	설명
chunksize **chunk_size**	**chunk**	스냅샷 볼륨의 단위 크기
copy_percent	**Copy%**	**pv_move** 명령을 사용하여 물리 확장 영역을 이동할 때에도 미러링된 논리 볼륨의 동기화 백분율

인수	header	설명
장치	장치	논리 볼륨을 구성하는 기본 장치: 물리 볼륨, 논리 볼륨, 물리 확장 영역 및 논리 확장 영역을 시작합니다.
lv_ancestors	상위	(Red Hat Enterprise Linux 7.2 이상) 씬 풀 스냅샷의 경우 논리 볼륨의 상위
lv_descendants	하위 항목	(Red Hat Enterprise Linux 7.2 이상) 씬 풀 스냅샷의 경우 논리 볼륨의 하위 항목입니다.
lv_attr	attr	논리 볼륨의 상태. 논리 볼륨 특성 비트는 다음과 같습니다. 비트 1: 볼륨 유형: (m)irrored, (m)irrored, (o)rigin, (o)rigin과 병합 스냅샷 (r)aid, (r)aid, (s)napshot, 병합 (S)napshot, (p)vrating, (v)irtual, (v)irtual, mirror or raid (i)mage, mirror or raid (l)mage out-of-sync, mirror (l)og device, under (c)onversion, thin (V)olume, (t)hin pool data, raid or thin pool m(e)tadata 또는 pool metadata space. 비트 2: 권한: (w)r(r)ead 전용, (R) 비 읽기 전용 볼륨의 활성화 비트 3: 할당 정책: (a)nywhere, (c)ontiguous, (i)nherited, c(l)ing, (n)ormal. 예를 들어 pvmove 명령을 실행하는 동안 볼륨이 현재 할당 변경에 대해 잠겨 있으면 대문자로 표시됩니다. 비트 4: 고정된 (m)inor 비트 5: 상태: (a)ctive, (s)uspended 스냅샷, (I)nvalid 스냅샷, 유효하지 않은 (S)uspended 스냅샷, 스냅샷 (m)erge failed, 일시 정지 스냅샷 (M)erge failed, mapped (d)evice without tables, mapped device present with (i)active table 비트 6: 장치 (o)pen

인수	header	설명
		비트 7: 대상 유형: (m)irror, (r)aid, (s)napshot, (t)hin, (u)nknown, (v)irtual. 이렇게 하면 동일한 커널 대상과 관련된 논리 볼륨이 함께 그룹화됩니다. 따라서 예를 들어 미러 이미지, 미러 로그는 원본 장치 매퍼 미러 커널 드라이버를 사용하는 경우 (m)로 표시되는 반면 md raid 커널 드라이버를 사용하는 것과 동일한 방식으로 모두 (r) 표시됩니다. 원래 장치 매퍼 드라이버를 사용하는 스냅샷은 (s)로 표시되는 반면 thin 프로비저닝 드라이버를 사용하는 스냅샷은 (t)로 표시됩니다. 비트 8: 새로 할당된 데이터 블록을 사용하기 전에 (z)ero 블록으로 덮어씁니다. 비트 9: (p)artial, (r)efresh 필요, (m)ismatches (w)ismatches가 거의 존재하지 않음을 나타냅니다. (p) 이 논리 볼륨이 시스템에서 사용되는 물리 볼륨 중 하나 이상이 누락되었음을 나타냅니다. (r)efresh는 이 RAID 논리 볼륨에서 사용하는 물리 볼륨 중 하나 이상이 누락되었음을 나타냅니다. (r)efresh는 쓰기 오류가 발생했음을 나타냅니다. 쓰기 오류는 해당 물리 볼륨의 일시적인 오류 또는 실패한 표시로 인해 발생할 수 있습니다. 장치를 새로 고치거나 교체해야 합니다. (m)ismatches는 RAID 논리 볼륨에 일관성이 없는 배열의 일부가 있음을 나타냅니다. 불일치는 RAID 논리 볼륨에서 검사 작업을 시작하여 검색됩니다. (검사 작업, 점검 및 복구 는 lvchange 명령을 통해 RAID 논리 볼륨에서 수행할 수 있습니다.) (w)rite는 대부분 쓰기로 표시된 RAID 1 논리 볼륨의 장치를 나타냅니다. 비트 10: s(k)ip 활성화: 이 볼륨은 활성화 중에 건너뛰도록 플래그가 지정됩니다.
lv_kernel_major	KMaj	논리 볼륨의 실제 주요 장치 번호(비활성인 경우-1)
lv_kernel_minor	KMIN	논리 볼륨의 실제 마이너 장치 번호(비활성인 경우-1)
lv_major	Maj	논리 볼륨의 영구 주요 장치 번호(수정되지 않은 경우-1)
lv_minor	분	논리 볼륨의 영구 마이너 장치 번호(수정되지 않은 경우-1)
lv_name	LV	논리 볼륨의 이름
lv_size	LSize	논리 볼륨의 크기
lv_tags	포르 태그	논리 볼륨에 연결된 LVM 태그
lv_uuid	LV UUID	논리 볼륨의 UUID입니다.

인수	header	설명
mirror_log	log	미러 로그가 있는 장치
modules	모듈	이 논리 볼륨을 사용하는 데 필요한 커널 장치 매퍼 대상
move_pv	이동	pvmove 명령을 사용하여 생성된 임시 논리 볼륨의 소스 물리 볼륨
origin	출처	스냅샷 볼륨의 원본 장치
regionsize region_size	리전	미러링된 논리 볼륨의 단위 크기
seg_count	#Seg	논리 볼륨의 세그먼트 수
seg_size	SSize	논리 볼륨의 세그먼트 크기입니다.
seg_start	시작	논리 볼륨의 세그먼트 오프셋
seg_tags	Seg 태그	논리 볼륨의 세그먼트에 연결된 LVM 태그
segtype	유형	논리 볼륨의 세그먼트 유형(예: mirror, striped, linear)
snap_percent	snap%	사용 중인 스냅샷 볼륨의 현재 백분율
스트라이프	#Str	논리 볼륨의 스트라이프 수 또는 미러 수
stripesize stripe_size	스트라이프	스트라이핑된 논리 볼륨의 단위 크기

lvs 명령은 기본적으로 **lv_name, Cryostat_name ,lv_ attr,lv_size,origin,snap_percent,move_pv,mirror_log,copy_percent,convert_lv.** 기본 디스플레이는 볼륨 그룹 내의 **vg_name** 및 **lv_name** 에 따라 정렬됩니다.

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

lvs 명령과 함께 **-v** 인수를 사용하면 기본 디스플레이에 다음 필드가 추가됩니다. **seg_count,lv_major,lv_ kernel_major,lv_kernel_minor,lv_uuid.**

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

lvs 명령의 **--segments** 인수를 사용하여 세그먼트 정보를 강조하는 기본 열이 있는 정보를 표시할 수 있습니다. **segments** 인수를 사용하면 **seg** 접두사가 선택 사항입니다. **lvs --segments** 명령은 기본적으로 **lv_name, Cryostat_name ,lv_ at trs** ,스트라이프 유형 **,segtype,seg_size** 필드를 표시합니다. 기본 디스플레이는 **vg_name,** 볼륨 그룹 내의**lv_name,** 논리 볼륨 내에서 **seg_start** 에 따라 정렬됩니다. 논리 볼륨이 조각화된 경우 이 명령의 출력에 다음이 표시됩니다.

```
# 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 명령과 함께 **-v** 인수를 사용하면 기본 디스플레이(**seg_start,stripesize,chunksize**) 에 다음 필드가 추가됩니다.

```
# 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
```

다음 예제에서는 하나의 논리 볼륨이 구성된 시스템의 **lvs** 명령의 기본 출력 다음에 지정된 세그먼트

*인수와 함께 **lvs** 명령의 기본 출력을 보여줍니다.*

```
# lvs
  LV    VG     Attr   LSize  Origin Snap%  Move Log Copy%
  lvol0 new_vg -wi-a- 52.00M
# lvs --segments
  LV    VG     Attr   #Str Type   SSize
  lvol0 new_vg -wi-a-    1 linear 52.00M
```

4.8.3. LVM 보고서 정렬

*일반적으로 **lvs, Cryostats** 또는 **pvs** 명령의 전체 출력은 올바르게 정렬되고 정렬되기 전에 내부적으로 생성되어 저장해야 합니다. **--unbuffered** 인수를 지정하여 불필요한 출력을 생성하는 즉시 표시할 수 있습니다.*

*정렬할 열 대체 정렬된 목록을 지정하려면 보고 명령의 **-O** 인수를 사용합니다. 이러한 필드를 출력 자체에 포함할 필요는 없습니다.*

*다음 예제에서는 물리 볼륨 이름, 크기, 사용 가능한 공간을 표시하는 **pvs** 명령의 출력을 보여줍니다.*

```
# 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
```

다음 예제에서는 사용 가능한 공간 필드를 기준으로 정렬한 동일한 출력을 보여줍니다.

```
# 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
```

다음 예제에서는 정렬할 필드를 표시할 필요가 없음을 보여줍니다.

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

역방향 정렬을 표시하려면 **-** 문자를 사용하여 **-O** 인수 뒤에 지정하는 필드 앞에 표시됩니다.

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

4.8.4. 단위 지정

LVM 보고서 디스플레이의 장치를 지정하려면 **report** 명령의 **--units** 인수를 사용합니다. **(b) ilottes, (k)egabytes, (g)igabytes, (t)erabytes, (e)xabytes, (p)etabytes, (h)uman-readable-readable-readable-readable**를 지정할 수 있습니다. 기본 디스플레이는 사람이 읽을 수 있습니다. **lvm.conf** 파일의 **global** 섹션에서 **units** 매개변수를 설정하여 기본값을 덮어쓸 수 있습니다.

다음 예제에서는 기본 기가바이트 대신 **pvs** 명령의 출력을 메가바이트로 지정합니다.

```
# 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
```

기본적으로 단위는 **2**의 거듭제곱**(24**개**)**으로 표시됩니다. 단위 사양 **(B, K, M, G, T, H)**을 활용하여 **1000**의 여러 단위에 장치를 표시하도록 지정할 수 있습니다.

다음 명령은 기본 동작인 **1024**의 수로 출력을 표시합니다.

```
# 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
```

다음 명령은 **1000**개의 다중으로 출력을 표시합니다.

```
#  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
```

또한 **(s)ectors(512**바이트로 정의됨) 또는 사용자 지정 단위를 지정할 수도 있습니다.

다음 예제에서는 **pvs** 명령의 출력을 여러 섹터로 표시합니다.

```
# 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
```

다음 예제는 **PVC** 명령 의 출력을 **4MB** 단위로 표시합니다.

```
# pvs --units 4m
  PV         VG     Fmt  Attr PSize    PFree
  /dev/sdb1  new_vg lvm2 a-   4388.00U 4388.00U
  /dev/sdc1  new_vg lvm2 a-   4388.00U 4375.00U
  /dev/sdd1  new_vg lvm2 a-   4388.00U 4388.00U
```

4.8.5. JSON 형식 출력(Red Hat Enterprise Linux 7.3 이상)

Red Hat Enterprise Linux 7.3부터 **LVM** 디스플레이 명령의 **--reportformat** 옵션을 사용하여 출력을 **JSON** 형식으로 표시할 수 있습니다.

다음 예제에서는 **lv**의 출력을 표준 기본 형식으로 보여줍니다.

```
# lvs
  LV      VG          Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  my_raid my_vg       Rwi-a-r---  12.00m                                    100.00
  root    rhel_host-075 -wi-ao----   6.67g
  swap    rhel_host-075 -wi-ao---- 820.00m
```

다음 명령은 **JSON** 형식을 지정할 때 동일한 **LVM** 구성의 출력을 보여줍니다.

```
# lvs --reportformat json
  {
      "report": [
          {
              "lv": [
                  {"lv_name":"my_raid", "vg_name":"my_vg", "lv_attr":"Rwi-a-r---", "lv_size":"12.00m",
"pool_lv":"", "origin":"", "data_percent":"", "metadata_percent":"", "move_pv":"", "mirror_log":"",
"copy_percent":"100.00", "convert_lv":""},
                  {"lv_name":"root", "vg_name":"rhel_host-075", "lv_attr":"-wi-ao----", "lv_size":"6.67g",
```

```
"pool_lv":"", "origin":"", "data_percent":"", "metadata_percent":"", "move_pv":"", "mirror_log":"",
"copy_percent":"", "convert_lv":""},
                {"lv_name":"swap", "vg_name":"rhel_host-075", "lv_attr":"-wi-ao----", "lv_size":"820.00m",
"pool_lv":"", "origin":"", "data_percent":"", "metadata_percent":"", "move_pv":"", "mirror_log":"",
"copy_percent":"", "convert_lv":""}
            ]
        }
    ]
}
```

output_format 설정을 사용하여 **/etc/lvm/lvm.conf** 파일에서 보고서 형식을 구성 옵션으로 설정할 수도 있습니다. 그러나 명령줄의 **--reportformat** 설정은 이 설정보다 우선합니다.

4.8.6. 명령 로그 보고(Red Hat Enterprise Linux 7.3 이상)

Red Hat Enterprise Linux 7.3부터 보고서 중심 및 처리 지향 **LVM** 명령 모두 **log/report_command_log** 구성 설정으로 활성화된 경우 명령 로그를 보고할 수 있습니다. 표시할 필드 집합을 확인하고 이 보고서에 따라 정렬할 수 있습니다.**You can determine the set of fields to display and to sort by for this report.**

다음 예제에서는 **LVM** 명령에 대한 전체 로그 보고서를 생성하도록 **LVM**을 구성합니다. 이 예에서는 볼륨이 포함된 볼륨 그룹 **VG** 와 마찬가지로 **lvol0** 및 **lvol1** 모두 성공적으로 처리되었음을 확인할 수 있습니다.

```
# lvmconfig --type full log/command_log_selection
command_log_selection="all"

# lvs
  Logical Volume
  ==============
  LV    LSize Cpy%Sync
  lvol1 4.00m 100.00
  lvol0 4.00m

  Command Log
  ===========
  Seq LogType Context    ObjType ObjName ObjGrp  Msg     Errno RetCode
    1 status  processing lv      lvol0   vg      success     0       1
    2 status  processing lv      lvol1   vg      success     0       1
    3 status  processing vg      vg              success     0       1

# lvchange -an vg/lvol1
  Command Log
  ===========
  Seq LogType Context    ObjType ObjName ObjGrp  Msg     Errno RetCode
    1 status  processing lv      lvol1   vg      success     0       1
    2 status  processing vg      vg              success     0       1
```

*LVM 보고서 및 명령 로그를 구성하는 방법에 대한 자세한 내용은 **lvmreport** 도움말 페이지를 참조하십시오.*

5장. LVM 설정 예

이 장에서는 몇 가지 기본 LVM 구성 예제를 설명합니다.

5.1. 3개의 디스크에서 LVM 논리 볼륨 만들기

이 예제 절차에서는 /dev/sda1,/dev/sdb1, /dev/sdc1 의 디스크로 구성된 new_logic_volume이라는 LVM 논리 볼륨을 생성합니다.

1.
볼륨 그룹의 디스크를 사용하려면 pvcreate 명령을 사용하여 LVM 물리 볼륨으로 레이블을 지정합니다.

주의

이 명령은 /dev/sda1,/dev/sdb1, /dev/sdc1 의 모든 데이터를 제거합니다.

```
# 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.
생성한 LVM 물리 볼륨으로 구성된 볼륨 그룹을 만듭니다. 다음 명령은 new_vol_group 볼륨 그룹을 생성합니다.

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

Cryostats 명령을 사용하여 새 볼륨 그룹의 속성을 표시할 수 있습니다.

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

3.
생성한 볼륨 그룹에서 논리 볼륨을 만듭니다. 다음 명령은 볼륨 그룹 new_vol_group 에서 new_logic_volume 논리 볼륨을 생성합니다. 이 예에서는 볼륨 그룹의 2GB를 사용하는 논리 볼

륨을 생성합니다.

```
# lvcreate -L 2G -n new_logical_volume new_vol_group
  Logical volume "new_logical_volume" created
```

4.

*논리 볼륨에 파일 시스템을 생성합니다. 다음 명령은 논리 볼륨에 **GFS2** 파일 시스템을 생성합니다.*

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

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

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

Syncing...
All Done
```

다음 명령은 논리 볼륨을 마운트하고 파일 시스템 디스크 공간 사용을 보고합니다.

```
# mount /dev/new_vol_group/new_logical_volume /mnt
# df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/new_vol_group/new_logical_volume
                       1965840        20   1965820   1% /mnt
```

5.2. 스트립 논리 볼륨 생성

*이 예제 절차에서는 **/dev/sda1,/dev/sdb1, /dev/sdc1** 의 디스크에서 데이터를 스트라이핑하는 **LVM** 스트라이핑된 논리 볼륨 을 생성합니다.*

1.

*볼륨 그룹에서 사용할 디스크에는 **pvcreate** 명령을 사용하여 **LVM** 물리 볼륨으로 레이블을 지정합니다.*

주의

*이 명령은 **/dev/sda1,/dev/sdb1, /dev/sdc1** 의 모든 데이터를 제거합니다.*

```
# 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.

*볼륨 그룹 **volgroup01** 을 만듭니다. 다음 명령은 볼륨 그룹 **volgroup01** 을 생성합니다.*

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

***Cryostats** 명령을 사용하여 새 볼륨 그룹의 속성을 표시할 수 있습니다.*

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

3.

*생성한 볼륨 그룹에서 스트라이핑된 논리 볼륨을 만듭니다. 다음 명령은 볼륨 그룹 **volgroup01** 에서 제거된 논리 볼륨 스트라이핑_ **logic_volume**을 생성합니다. 이 예에서는 크기가 **2GB**인 논리 볼륨을 생성합니다. **3**개의 스트라이프와 스트라이프 크기가 **4**킬로바이트입니다.*

```
# lvcreate -i 3 -I 4 -L 2G -n striped_logical_volume volgroup01
  Rounding size (512 extents) up to stripe boundary size (513 extents)
  Logical volume "striped_logical_volume" created
```

4.

*스트라이핑된 논리 볼륨에 파일 시스템을 만듭니다. 다음 명령은 논리 볼륨에 **GFS2** 파일 시스템을 생성합니다.*

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

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

Device:                    /dev/volgroup01/striped_logical_volume
Blocksize:                 4096
```

```
Filesystem Size:          492484
Journals:                 1
Resource Groups:          8
Locking Protocol:         lock_nolock
Lock Table:

Syncing...
All Done
```

다음 명령은 논리 볼륨을 마운트하고 파일 시스템 디스크 공간 사용을 보고합니다.

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

5.3. 볼륨 그룹 분할

이 예제 절차에서는 기존 볼륨 그룹이 세 개의 물리 볼륨으로 구성됩니다. 물리 볼륨에 사용되지 않은 공간이 충분한 경우 새 디스크를 추가하지 않고 새 볼륨 그룹을 만들 수 있습니다.

초기 설정에서는 논리 볼륨 **mylv** 가 볼륨 그룹 **myvol**에서 수정되며, 볼륨 그룹 **myvol** 은 세 개의 물리 볼륨, **/dev/sda1**,**/dev/sdb1**, **/dev/sdc1** 로 구성됩니다.

이 절차를 완료하면 **myvg** 볼륨 그룹이 **/dev/sda1** 및 **/dev/sdb1** 로 구성됩니다. 두 번째 볼륨 그룹 **yourvg** 는 **/dev/sdc1** 로 구성됩니다.

1. **pvscan** 명령을 사용하여 볼륨 그룹에서 현재 사용 가능한 공간 크기를 결정합니다.

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

2. **pvmove** 명령을 사용하여 **/dev/sdc1** 에서 사용된 모든 물리 확장 영역을 **/dev/sdb1** 로 이동합니다. **pvmove** 명령을 실행하는 데 시간이 오래 걸릴 수 있습니다.

```
# 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%
```

데이터를 이동한 후 **/dev/sdc1** 의 모든 공간이 사용 가능한 것을 확인할 수 있습니다.

```
# 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.

새 볼륨 그룹 **yourvg** 를 생성하려면 **Cryostat split** 명령을 사용하여 볼륨 그룹 **myvg** 를 분할합니다.

볼륨 그룹을 분할하려면 논리 볼륨이 비활성 상태여야 합니다. 파일 시스템이 마운트된 경우 논리 볼륨을 비활성화하기 전에 파일 시스템을 마운트 해제해야 합니다.

lvchange 명령 또는 **Cryostatchange** 명령을 사용하여 논리 볼륨을 비활성화합니다. 다음 명령은 논리 볼륨 **mylv** 를 비활성화한 다음 **myvg** 볼륨 그룹 **myvg**에서 볼륨 그룹 **myvg** 를 분할하여 물리 볼륨 **/dev/sdc1** 을 새 볼륨 그룹인 **yourvg** 로 이동합니다.

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

Cryostats 명령을 사용하여 두 볼륨 그룹의 속성을 확인할 수 있습니다.

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

4.

새 볼륨 그룹을 만든 후 새 논리 볼륨 **yourlv** 를 만듭니다.

```
# lvcreate -L 5G -n yourlv yourvg
  Logical volume "yourlv" created
```

5.

새 논리 볼륨에 파일 시스템을 생성하고 마운트합니다.

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

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

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

Syncing...
All Done

# mount /dev/yourvg/yourlv /mnt
```

6.

논리 볼륨 **mylv** 를 비활성화해야 하므로 마운트하기 전에 다시 활성화해야 합니다.

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

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

5.4. 논리 볼륨에서 디스크 제거

다음 예제 절차에서는 기존 논리 볼륨에서 디스크를 제거하거나 디스크를 다른 볼륨의 일부로 사용하는 방법을 보여줍니다. 디스크를 제거하려면 먼저 **LVM** 물리 볼륨의 확장 영역을 다른 디스크 또는 디스크 세트로 이동해야 합니다.

5.4.1. 기존 물리 볼륨으로 이동

이 예에서 논리 볼륨은 볼륨 그룹 **myvg** 의 **4** 개의 물리 볼륨에 배포됩니다.

```
# 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
```

이 예에서는 볼륨 그룹에서 제거할 수 있도록 확장 영역을 **/dev/sdb1** 에서 이동합니다.

1. 볼륨 그룹의 다른 물리 볼륨에 사용 가능한 확장 영역이 충분한 경우 다른 옵션 없이 제거하려는 장치에서 **pvmove** 명령을 실행할 수 있으며 **Extent**는 다른 장치에 배포됩니다.

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

pvmove 명령 실행이 완료되면 **Extent** 배포는 다음과 같습니다.

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

2. **Cryostat reduce** 명령을 사용하여 볼륨 그룹에서 물리 볼륨 **/dev/sdb1** 을 제거합니다.

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

이제 디스크를 물리적으로 제거하거나 다른 사용자에게 할당할 수 있습니다.

5.4.2. 확장 영역을 새 디스크로 이동

이 예에서 논리 볼륨은 볼륨 그룹 **myvg** 의 세 개의 물리 볼륨에 다음과 같이 배포됩니다.

```
# 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
```

이 예제 절차에서는 **/dev/sdb1** 의 **Extent**를 새 장치인 **/dev/sdd1** 로 이동합니다.

1. **/dev/sdd1** 에서 새 물리 볼륨을 만듭니다.

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

2. 새 물리 볼륨 **/dev/sdd1** 을 기존 볼륨 그룹 **myvg** 에 추가합니다.

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

3. **pvmove** 명령을 사용하여 데이터를 **/dev/sdb1** 에서 **/dev/sdd1** 로 이동합니다.

```
# pvmove /dev/sdb1 /dev/sdd1
  /dev/sdb1: Moved: 10.0%
...
  /dev/sdb1: Moved: 79.7%
...
  /dev/sdb1: Moved: 100.0%

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

4. 데이터를 **/dev/sdb1** 에서 이동한 후 볼륨 그룹에서 제거할 수 있습니다.

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

이제 디스크를 다른 볼륨 그룹에 재배치하거나 시스템에서 디스크를 제거할 수 있습니다.

5.5. 클러스터에서 미러링된 LVM 논리 볼륨 생성

클러스터에 미러링된 LVM 논리 볼륨을 생성하려면 세그먼트 유형의 미러 가 있는 단일 노드에 미러링된 LVM 논리 볼륨을 생성하는 것과 동일한 명령 및 절차가 필요합니다. 그러나 클러스터에 미러링된 LVM 볼륨을 생성하려면 다음을 수행합니다.

- 클러스터 및 클러스터 미러 인프라가 실행 중이어야 합니다.
- 클러스터가 정족수여야 합니다.
- 클러스터 잠금을 활성화하려면 lvm.conf 파일의 잠금 유형을 올바르게 설정해야 하며 use_lvmetad 설정은 0이어야 합니다. 그러나 Red Hat Enterprise Linux 7에서는 시작 절차의 일부로 ocf:heartbeat:clvm Pacemaker 리소스 에이전트 자체에서 이러한 작업을 수행합니다.

Red Hat Enterprise Linux 7에서 클러스터는 Pacemaker를 통해 관리됩니다. 클러스터형 LVM 논리 볼륨은 Pacemaker 클러스터와 함께만 지원되며 클러스터 리소스로 구성해야 합니다.

다음 절차에서는 클러스터에 미러링된 LVM 볼륨을 생성합니다.

1. 클러스터 소프트웨어 및 LVM 패키지를 설치하고 클러스터 소프트웨어를 시작하고 클러스터를 생성합니다. 클러스터의 펜싱을 구성해야 합니다. 문서 고가용성 애드온 관리 에서는 클러스터를 생성하고 클러스터의 노드의 펜싱을 구성하는 샘플 프로세스를 제공합니다. 문서 고가용성 애드온 참조 는 클러스터 구성의 구성 요소에 대한 자세한 정보를 제공합니다.
2. 클러스터의 모든 노드에서 공유하는 미러링된 논리 볼륨을 생성하려면 클러스터의 모든 노드의 lvm.conf 파일에서 잠금 유형을 올바르게 설정해야 합니다. 기본적으로 잠금 유형은 local로 설정됩니다. 이를 변경하려면 클러스터의 각 노드에서 다음 명령을 실행하여 클러스터형 잠금을 활성화합니다.

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

3. 클러스터의 dlm 리소스를 설정합니다. 클러스터의 모든 노드에서 리소스를 실행하도록 복제된 리소스로 리소스를 생성합니다.

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

4.

clvmd 를 클러스터 리소스로 구성합니다. **dlm** 리소스와 마찬가지로, 클러스터의 모든 노드에서 실행되도록 복제된 리소스로 리소스를 생성합니다. **clvmd** 가 실행되는 모든 노드에서 **cmirrord** 데몬을 활성화하려면 **with_cmirrord=true** 매개변수를 설정해야 합니다.

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

clvmd 리소스를 이미 구성했지만 **with_cmirrord=true** 매개변수를 지정하지 않은 경우 다음 명령을 사용하여 매개 변수를 포함하도록 리소스를 업데이트할 수 있습니다.

```
# pcs resource update clvmd with_cmirrord=true
```

5.

clvmd 및 **dlm** 종속성을 설정하고 순서를 시작합니다. **clvmd** 는 **dlm** 후에 시작해야 하며 **dlm** 과 동일한 노드에서 실행해야 합니다.

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

6.

미러를 생성합니다. 첫 번째 단계는 물리 볼륨을 만드는 것입니다. 다음 명령은 세 개의 물리 볼륨을 생성합니다. 미러 레코드에는 두 개의 물리 볼륨이 사용되며 세 번째 물리 볼륨에는 미러 로그가 포함됩니다.

```
# pvcreate /dev/sdb1
  Physical volume "/dev/sdb1" successfully created
# pvcreate /dev/sdc1
  Physical volume "/dev/sdc1" successfully created
# pvcreate /dev/sdd1
  Physical volume "/dev/sdd1" successfully created
```

7.

볼륨 그룹을 만듭니다. 이 예제에서는 이전 단계에서 만든 세 개의 물리 볼륨으로 구성된 볼륨 그룹 **Cryostat 001** 을 생성합니다.

```
# vgcreate vg001 /dev/sdb1 /dev/sdc1 /dev/sdd1
  Clustered volume group "vg001" successfully created
```

Cryostat create 명령의 출력은 볼륨 그룹이 클러스터됨을 나타냅니다. 볼륨 그룹의 속성을 표시하는 **Cryostat** 명령을 사용하여 볼륨 그룹이 클러스터링되었는지 확인할 수 있습니다. 볼륨 그룹이 클러스터형 경우 **c** 속성이 표시됩니다.

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

8.

*미러링된 논리 볼륨을 생성합니다. 이 예에서는 볼륨 그룹 **Cryostat 001** 에서 논리 볼륨 **mirrorlv** 를 생성합니다. 이 볼륨에는 하나의 미러 다리가 있습니다. 이 예에서는 논리 볼륨에 사용할 물리 볼륨의 확장 영역을 지정합니다.*

```
# lvcreate --type mirror -l 1000 -m 1 vg001 -n mirrorlv /dev/sdb1:1-1000 /dev/sdc1:1-1000
/dev/sdd1:0
  Logical volume "mirrorlv" created
```

***lvs** 명령을 사용하여 미러 생성의 진행 상황을 표시할 수 있습니다. 다음 예제에서는 미러가 **47%** 동기화된 다음 **91%** 동기화된 다음 미러가 완료되면 **100%** 동기화되었음을 보여줍니다.*

```
# lvs vg001/mirrorlv
  LV       VG       Attr   LSize Origin Snap%  Move Log         Copy%  Convert
  mirrorlv vg001    mwi-a- 3.91G                vg001_mlog     47.00
# lvs vg001/mirrorlv
  LV       VG       Attr   LSize Origin Snap%  Move Log         Copy%  Convert
  mirrorlv vg001    mwi-a- 3.91G                vg001_mlog     91.00
#  lvs vg001/mirrorlv
  LV       VG       Attr   LSize Origin Snap%  Move Log         Copy%  Convert
  mirrorlv vg001    mwi-a- 3.91G                vg001_mlog    100.00
```

미러의 완료는 시스템 로그에 표시됩니다.

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

9.

***lvs** 명령을 **-o +devices** 옵션과 함께 사용하여 미러 브릿지를 구성하는 장치를 포함하여 미러의 구성을 표시할 수 있습니다. 이 예제의 논리 볼륨이 두 개의 선형 이미지와 하나의 로그로 구성되어 있음을 알 수 있습니다.*

```
# lvs -a -o +devices
  LV               VG        Attr   LSize  Origin Snap%  Move Log          Copy%  Convert Devices
  mirrorlv         vg001     mwi-a-  3.91G                mirrorlv_mlog 100.00
mirrorlv_mimage_0(0),mirrorlv_mimage_1(0)
  [mirrorlv_mimage_0] vg001      iwi-ao  3.91G                                      /dev/sdb1(1)
  [mirrorlv_mimage_1] vg001      iwi-ao  3.91G                                      /dev/sdc1(1)
  [mirrorlv_mlog]     vg001      lwi-ao  4.00M                                    /dev/sdd1(0)
```

lvs** 의 **seg_pe_ranges** 옵션을 사용하여 데이터 레이아웃을 표시할 수 있습니다. 이 옵션을 사용하여 레이아웃이 제대로 중복되었는지 확인할 수 있습니다. 이 명령의 출력에서는 **lvcreate

*및 **lvresize** 명령이 입력으로 사용하는 것과 동일한 형식으로 **PE** 범위를 표시합니다.*

```
# lvs -a -o +seg_pe_ranges --segments
  PE Ranges
  mirrorlv_mimage_0:0-999 mirrorlv_mimage_1:0-999
  /dev/sdb1:1-1000
  /dev/sdc1:1-1000
  /dev/sdd1:0-0
```

참고

***LVM** 미러링된 볼륨 안전 중 하나의 장애 복구에 대한 자세한 내용은 **6.2**절. “**LVM Mirror** 실패에서 복구”을 참조하십시오.*

6장. LVM 문제 해결

이 장에서는 다양한 LVM 문제 해결 방법을 설명합니다.

6.1. 진단 문제 해결

명령이 예상대로 작동하지 않는 경우 다음과 같은 방법으로 진단을 수집할 수 있습니다.

- 점점 더 자세한 출력 수준을얻으려면 모든 명령의 **-vvv** , **-vvv** , **-vvv** 인수를 사용합니다.
- 문제가 논리 볼륨 활성화와 관련된 경우 구성 파일의 로그 섹션에서 **activation = 1** 을 설정하고 **-vvvv** 인수로 명령을 실행합니다. 이 출력을 검사한 후 메모리가 부족한 상태에서 시스템 잠금 문제가 발생하지 않도록 이 매개 변수를 **0**으로 재설정해야 합니다.
- 진단 목적으로 정보 덤프를 제공하는 **lvmdump** 명령을 실행합니다. 자세한 내용은 **lvmdump(8)** 매뉴얼 페이지를 참조하십시오.
- 추가 시스템 정보는 **lvs -v,pvs -a** 또는 **dmsetup info -c** 명령을 실행합니다.
- **/etc/lvm/backup** 파일에서 메타데이터의 마지막 백업과 **/etc/lvm/archive** 파일의 아카이브 버전을 검사합니다.
- **lvmconfig** 명령을 실행하여 현재 구성 정보를 확인합니다.
- **/etc/lvm** 디렉터리의 **.cache** 파일에서 물리적 볼륨이 있는 장치에 대한 레코드가 있는지 확인합니다.

6.2. LVM MIRROR 실패에서 복구

이 섹션에서는 물리 볼륨의 기본 장치가 중단되고 **mirror_log_fault_policy** 매개변수가 제거 되도록 설정되어 있기 때문에 LVM 미러링된 볼륨의 한 부분이 실패하는 상황에서 복구하는 예를 제공합니다. 이를 위해서는 미러를 수동으로 다시 빌드해야 합니다. **mirror_log_fault_policy** 매개변수 설정에 대한 자세한 내용은 4.4.4.1절. “미러링된 논리 볼륨 실패 정책” 을 참조하십시오.

미러 다리가 실패하면 LVM은 미러링된 볼륨을 선형 볼륨으로 변환하여 미러링된 중복이 없는 이전과

마찬가지로 계속 작동합니다. 이 시점에서 대체 물리적 장치로 사용할 새 디스크 장치를 시스템에 추가하고 미러를 다시 빌드할 수 있습니다.

다음 명령은 미러에 사용할 물리 볼륨을 생성합니다.

```
# 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
```

다음 명령은 볼륨 그룹 **Cryostat** 및 미러링 된 볼륨 **groupfs** 를 생성합니다.

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

lvs 명령을 사용하여 미러 브릿지 및 미러 로그의 미러링된 볼륨 및 기본 장치의 레이아웃을 확인할 수 있습니다. 첫 번째 예에서 미러는 아직 완전히 동기화되지 않았습니다. 계속하기 전에 **Copy%** 필드가 **100.00**으로 표시될 때까지 기다려야 합니다.

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

# lvs -a -o +devices
  LV              VG   Attr   LSize   Origin Snap%  Move Log         Copy%  Devices
  groupfs         vg   mwi-a- 752.00M                groupfs_mlog 100.00
groupfs_mimage_0(0),groupfs_mimage_1(0)
```

```
  [groupfs_mimage_0] vg   iwi-ao 752.00M                    /dev/sda1(0)
  [groupfs_mimage_1] vg   iwi-ao 752.00M                    /dev/sdb1(0)
  [groupfs_mlog]     vg   lwi-ao   4.00M     i              /dev/sdc1(0)
```

이 예에서는 미러 **/dev/sda1** 의 기본 브릿지가 실패합니다. 미러링된 볼륨에 대한 쓰기 활동으로 인해 **LVM**에서 실패한 미러를 탐지합니다. 이 경우 **LVM**은 미러를 단일 선형 볼륨으로 변환합니다. 이 경우 변환을 트리거하려면 **dd** 명령을 실행합니다.

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

lvs 명령을 사용하여 장치가 이제 선형 장치인지 확인할 수 있습니다. 실패한 디스크로 인해 **I/O** 오류가 발생했습니다.

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

이 시점에서 논리 볼륨을 계속 사용할 수 있어야 하지만 미러 중복은 없습니다.

미러링된 볼륨을 다시 빌드하려면 손상된 드라이브를 교체하고 물리 볼륨을 다시 생성합니다. **pvcreate** 명령을 실행할 때 새 디스크를 교체하는 대신 동일한 디스크를 사용하는 경우 "상위" 경고가 표시됩니다. **Cryostatreduce --removemissing** 명령을 실행하여 경고가 표시되지 않도록 할 수 있습니다.

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

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

```
  PV /dev/sdi1          lvm2 [603.94 GB]
  PV /dev/sdi2          lvm2 [603.94 GB]
  Total: 16 [2.11 TB] / in use: 14 [949.65 GB] / in no VG: 2 [1.18 TB]
```

다음으로 새 물리 볼륨을 사용하여 원래 볼륨 그룹을 확장합니다.

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

# pvscan
  PV /dev/sdb1   VG vg   lvm2 [67.83 GB / 67.10 GB free]
  PV /dev/sdb2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdc1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdc2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdd1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdd2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sde1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sde2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdf1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdf2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdg1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdg2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdh1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdh2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdi1   VG vg   lvm2 [603.93 GB / 603.93 GB free]
  PV /dev/sdi2   VG vg   lvm2 [603.93 GB / 603.93 GB free]
  Total: 16 [2.11 TB] / in use: 16 [2.11 TB] / in no VG: 0 [0   ]
```

선형 볼륨을 원래 미러링된 상태로 다시 변환합니다.

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

lvs 명령을 사용하여 미러가 복원되었는지 확인할 수 있습니다.

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

6.3. 물리 볼륨 메타데이터 복구

물리 볼륨의 볼륨 그룹 메타데이터 영역을 실수로 덮어쓰거나 삭제한 경우 메타데이터 영역이 잘못되었거나 시스템이 특정 **UUID**가 있는 물리 볼륨을 찾을 수 없음을 나타내는 오류 메시지가 표시됩니다. 물

리적 볼륨에 메타데이터가 손실된 메타데이터와 동일한 **UUID**를 지정하여 물리 볼륨에서 데이터를 복구할 수 있습니다.

주의

작동 중인 **LVM** 논리 볼륨을 사용하여 이 절차를 시도해서는 안 됩니다. 잘못된 **UUID**를 지정하면 데이터가 손실됩니다.

다음 예제에서는 메타데이터 영역이 누락되거나 손상되었는지 여부를 확인할 수 있는 출력 종류를 보여줍니다.

```
# 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.
  ...
```

/etc/lvm/archive 디렉토리를 확인하여 덮어쓰는 물리 볼륨의 **UUID**를 찾을 수 있습니다. 해당 볼륨 그룹에 대해 마지막으로 알려진 유효한 아카이브 **LVM** 메타데이터는 **VolumeGroupName_xxxx.vg** 파일을 찾습니다.

또는 볼륨을 비활성화하고 부분 **(-P)** 인수를 설정하면 손상된 물리적 볼륨의 **UUID**를 찾을 수 있습니다.

```
# 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 명령의 **--uuid** 및 **--restorefile** 인수를 사용하여 물리 볼륨을 복원합니다. 다음 예제에서는 위에 표시된 **UUID**인 **FmGRh3-zhok-iVI8-7qTD-S5BI-NYM5Sk** 를 사용하여 **/dev/sdh1** 장치를 물리 볼륨으로 레이블을 지정합니다. 이 명령은 볼륨 그룹의 최신 아카이브 메타데이터 메타데이터인 **VG_00050.vg** 에 포함된 메타데이터 정보를 사용하여 물리 볼륨 레이블을 복원합니다. **restorefile** 인수는 **pvcreate** 명령에 볼륨 그룹의 이전 물리 볼륨과 호환되는 새 물리 볼륨을 사용하도록 하여 이전 물리 볼륨에 포함된 데이터를 배치하지 않도록 합니다(예: 원래 **pvcreate** 명령이 메타데이터 배치를 제어하는 명령줄 인수를 사용하거나 다른 기본값을 사용하여 물리적 볼륨이 원래 생성된 경우). **pvcreate** 명령은 **LVM** 메타데이터 영역만 덮어쓰고 기존 데이터 영역에는 영향을 미치지 않습니다.

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

그런 다음 **Cryostat cfgrestore** 명령을 사용하여 볼륨 그룹의 메타데이터를 복원할 수 있습니다.

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

이제 논리 볼륨을 표시할 수 있습니다.

```
# 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)
```

다음 명령은 볼륨을 활성화하고 활성 볼륨을 표시합니다.

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

디스크상의 **LVM** 메타데이터가 가능한 한 많은 공간을 차지하면 이 명령은 물리 볼륨을 복구할 수 있습니다. 메타데이터 영역을 과장한 경우 볼륨의 데이터가 영향을 받을 수 있습니다. **fsck** 명령을 사용하여 해당 데이터를 복구할 수 있습니다.

6.4. MISSING 물리 볼륨 교체

물리 볼륨이 실패하거나 교체해야 하는 경우 **6.3절. “물리 볼륨 메타데이터 복구”** 에 설명된 물리 볼륨 메타데이터를 복구하기 위한 것과 동일한 절차를 수행하여 기존 볼륨 그룹에 손실된 항목을 교체하도록 새 물리 볼륨에 레이블을 지정할 수 있습니다. **Cryostatdisplay** 명령의 **--partial** 및 **--verbose** 인수를 사용하여 더 이상 존재하지 않는 물리 볼륨의 **UUID**와 크기를 표시할 수 있습니다. 동일한 크기의 다른 물리 볼륨을 대체하려면 **pvcreate** 명령을 **--restorefile** 및 **--uuid** 인수와 함께 사용하여 누락된 물리 볼륨과 동일한 **UUID**로 새 장치를 초기화할 수 있습니다. 그런 다음 **Cryostat cfgrestore** 명령을 사용하여 볼륨 그룹의 메타데이터를 복원할 수 있습니다.

6.5. 볼륨 그룹에서 손실된 물리 볼륨 제거

물리 볼륨이 손실되면 **Cryostat change** 명령의 **--partial** 인수를 사용하여 볼륨 그룹의 나머지 물리 볼륨을 활성화할 수 있습니다. **Cryostatreduce** 명령의 **--removemissing** 인수를 사용하여 볼륨 그룹에서

해당 물리 볼륨을 사용한 모든 논리 볼륨을 제거할 수 있습니다.

--test 인수와 함께 **Cryostatreduce** 명령을 실행하여 제거할 항목을 확인해야 합니다.

대부분의 **LVM** 작업과 마찬가지로, 즉시 **Cryostat cfgrestore** 명령을 사용하여 볼륨 그룹 메타데이터를 이전 상태로 복원하면 대부분의 **LVM** 작업도 되돌릴 수 있습니다. 예를 들어 **--test** 인수 없이 **--removemissing** 인수의 **--removemissing** 인수를 사용한 후 유지하려는 논리 볼륨을 제거한 경우 여전히 물리 볼륨을 교체하고 다른 **Cryostat cfgrestore** 명령을 사용하여 볼륨 그룹을 이전 상태로 되돌릴 수 있습니다.

6.6. 논리 볼륨에 여유 범위가 충분하지 않음

Cryostatdisplay 또는 **Cryostats** 명령의 출력에 따라 충분한 **Extent**가 있다고 생각할 때 논리 볼륨을 생성할 때 **"Insufficient free extents"** 오류 메시지가 표시될 수 있습니다. 이는 이러한 명령이 사람이 읽을 수 있는 출력을 제공하기 위해 **2**개의 **10**진수 위치로 반올림하기 때문입니다. 정확한 크기를 지정하려면 여러 바이트 대신 사용 가능한 물리 확장 영역 수를 사용하여 논리 볼륨의 크기를 결정합니다.

기본적으로 **Cryo statdisplay** 명령에는 사용 가능한 물리 확장 영역을 나타내는 이 출력 행이 포함됩니다.

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

또는 **Cryostat** 명령의 **Cryostat _free_count** 및 **Cryostat _extent_count** 인수를 사용하여 사용 가능한 확장 영역과 총 확장 영역 수를 표시할 수 있습니다.

```
# 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 무료 물리 확장 영역을 사용하면 소문자 **l** 인수를 사용하여 바이트 대신 **Extent**를 사용하여 다음 명령을 입력할 수 있습니다.

```
# lvcreate -l 8780 -n testlv testvg
```

이는 볼륨 그룹의 사용 가능한 모든 확장 영역을 사용합니다.

```
# 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
```

또는 **lvcreate** 명령의 **-l** 인수를 사용하여 볼륨 그룹에 남아 있는 여유 공간의 백분율을 사용하도록 논리 볼륨을 확장할 수 있습니다. 자세한 내용은 **4.4.1절. "선형 논리 볼륨 생성"** 에서 참조하십시오.

6.7. 다중 경로 장치에 대한 PV 경고 중복

다중 경로 스토리지가 있는 **LVM**을 사용하는 경우 일부 **LVM** 명령(예: **Cryostat** 또는 **lvchange**)은 볼륨 그룹 또는 논리 볼륨을 나열할 때 다음과 같은 메시지를 표시할 수 있습니다.

```
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/dm-5 not /dev/sdd
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/emcpowerb not /dev/sde
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/sddlmab not /dev/sdf
```

이러한 경고의 근본 원인에 대한 정보를 제공하면 이 섹션에서는 다음 두 가지 경우에 이 문제를 해결하는 방법을 설명합니다.

- 출력에 표시되는 두 장치가 모두 동일한 장치에 대한 단일 경로입니다.
- 출력에 표시되는 두 장치가 모두 다중 경로 맵입니다.

6.7.1. PV 경고 중복의 근본 원인

기본 구성을 사용하면 **LVM** 명령이 **/dev** 의 장치를 스캔하고 결과 모든 장치에 **LVM** 메타데이터가 있는지 확인합니다. 이는 다음과 같은 **/etc/lvm/lvm.conf** 의 기본 필터로 인해 발생합니다.

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

장치 매퍼 멀티패스 또는 **EMC PowerPath** 또는 **Cryostat Dynamic Link Manager(HDLM)**와 같은 기타 다중 경로 소프트웨어를 사용하는 경우 특정 논리 단위 번호(**LUN**)로의 각 경로는 **/dev/sdb** 또는 **/dev/sdc** 와 같은 다른 **SCSI** 장치로 등록됩니다. 그런 다음 다중 경로 소프트웨어는 **/dev/mapper/mpath1** 또는 장치 맵퍼 **Multipath**의 경우 **/dev/mapper/mpatha, EMC PowerPath**를 위한 **/dev/emcpowera** 또는 **CryostatHDLM**의 경우 **/dev/sddlmab** 과 같은 개별 경로에 매핑되는 새 장치를 생성합니다. 각 **LUN**에는 **/dev** 에 동일한 기본 데이터를 가리키는 여러 장치 노드가 있으므로 모두 동일한 **LVM** 메타데이터를 포함하므로 **LVM** 명령은 동일한 메타데이터를 여러 번 검색하고 중복으로 보고합니다.

이러한 중복 메시지는 경고일 뿐이며 **LVM** 작업이 실패했음을 의미하지 않습니다. 대신 사용자 중 하나

만 물리 볼륨으로 사용되었으며 다른 장치도 무시됩니다. 메시지가 잘못된 장치가 선택되었거나 경고가 사용자에게 중단되는 경우 필터를 적용하여 물리 볼륨에 필요한 장치만 검색하고 다중 경로 장치에 대한 기본 경로를 종료할 수 있습니다.

6.7.2. 단일 경로에 대한 중복 경고

다음 예제에서는 표시된 중복 장치가 동일한 장치에 대한 단일 경로 둘 다인 중복된 **PV** 경고를 보여줍니다. 이 경우 **/dev/sdd** 및 **/dev/sdf** 둘 다 **multipath -ll** 명령에 대한 출력의 동일한 다중 경로 맵 아래에 있습니다.

```
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using **/dev/sdd** not **/dev/sdf**
```

이 경고가 표시되지 않도록 **/etc/lvm/lvm.conf** 파일에서 필터를 구성하여 **LVM**에서 메타데이터를 검색하는 장치를 제한할 수 있습니다. 필터는 **/dev** 검사(또는 **/etc/lvm/lvm.conf** 파일의 **dir** 키워드로 지정됨)로 지정된 각 장치에 적용할 패턴 목록입니다. 패턴은 임의의 문자로 구분된 정규식이며 (허용을 위해) 또는 **r** (거부용)으로 지정됩니다. 목록은 순서대로 트래버스되며 장치와 일치하는 첫 번째 **regex**는 장치를 수락하거나 거부(**ignored**)할지 여부를 결정합니다. 일치하는 패턴과 일치하지 않는 장치가 허용됩니다. **LVM** 필터에 대한 일반적인 정보는 **4.5절. "필터를 사용하여 LVM 장치 스캔 제어"** 에서 참조하십시오.

구성하는 필터는 루트 볼륨 그룹과 함께 로컬 하드 드라이브 및 다중 경로 장치 등 **LVM** 메타데이터에 대해 확인해야 하는 모든 장치를 포함해야 합니다. 다중 경로 장치(예: **/dev/sdb**,**/dev/sdd** 등)에 대한 기본 경로를 거부하면 이러한 중복 **PV** 경고를 방지할 수 있습니다. 각 고유 메타데이터 영역은 다중 경로 장치 자체에서 한 번만 발견되기 때문입니다.

다음 예제에서는 사용 가능한 여러 스토리지 경로로 인해 중복된 **PV** 경고를 사용하지 않는 필터를 보여줍니다.

- 이 필터는 첫 번째 하드 드라이브(**/dev/sda** 및 모든 **device-mapper-multipath** 장치)에서 두 번째 파티션을 허용하는 동시에 다른 모든 항목을 거부합니다.

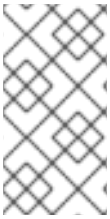
```
filter = [ "a|/dev/sda2$|", "a|/dev/mapper/mpath.*|", "r|.*|" ]
```

- 이 필터는 모든 **HP SmartArray** 컨트롤러와 모든 **EMC PowerPath** 장치를 허용합니다.

```
filter = [ "a|/dev/cciss/.*|", "a|/dev/emcpower.*|", "r|.*|" ]
```

- 이 필터는 첫 번째 **IDE** 드라이브 및 다중 경로 장치의 모든 파티션을 허용합니다.

```
filter = [ "a|/dev/hda.*|", "a|/dev/mapper/mpath.*|", "r|.*|" ]
```



참고

/etc/lvm/lvm.conf 파일에 새 필터를 추가할 때 # 또는 을 사용하여 원래 필터를 주석 처리했는지 확인합니다.

필터가 구성되고 /etc/lvm/lvm.conf 파일이 저장되면 해당 명령의 출력을 확인하여 물리 볼륨 또는 볼륨 그룹이 누락되어 있지 않은지 확인합니다.

```
# pvscan
# vgscan
```

다음 예제와 같이 --config 인수를 LVM 명령에 추가하여 /etc/lvm/lvm.conf 파일을 수정하지 않고 즉시 필터를 테스트할 수도 있습니다.

```
# lvs --config 'devices{ filter = [ "a|/dev/emcpower.*|", "r|.*|" ] }'
```

참고

--config 인수를 사용하여 필터를 테스트하면 서버 구성을 영구적으로 변경하지 않습니다. 테스트 후 /etc/lvm/lvm.conf 파일에 working 필터를 포함해야 합니다.

LVM 필터를 구성한 후에는 재부팅 시 필요한 장치만 검사되도록 dracut 명령을 사용하여 initrd 장치를 다시 빌드하는 것이 좋습니다.

6.7.3. Multipath Maps에 대한 경고 중복

다음 예제에서는 두 개의 다중 경로 맵인 두 장치에 대해 중복된 PV 경고를 보여줍니다. 이 예제에서는 두 개의 다른 경로를 보는 것이 아니라 두 개의 다른 장치를 살펴보겠습니다.

```
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using **/dev/mapper/mpatha** not
**/dev/mapper/mpathc**
```

```
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using **/dev/emcpowera** not
**/dev/emcpowerh**
```

이 상황은 동일한 장치에 대한 단일 경로 모두 단일 경로인 장치에 대한 중복 경고보다 심각합니다. 이러한 경고는 종종 시스템에 표시되지 않아야 하는 장치를 표시했음을 의미하기 때문입니다(예: LUN 복제 또는 미러). 이 경우 시스템에서 어떤 장치를 제거해야 하는지 명확하게 파악하지 못하는 경우 상황을 복구할 수 없습니다. 이 문제를 해결하려면 Red Hat 기술 지원에 문의하는 것이 좋습니다.

부록 A. 장치 매퍼

장치 매퍼는 볼륨 관리를 위한 프레임워크를 제공하는 커널 드라이버입니다. 논리 볼륨으로 사용할 수 있는 매핑된 장치를 생성하는 일반적인 방법을 제공합니다. 볼륨 그룹 또는 메타데이터 형식에 대해 구체적으로는 알 수 없습니다.

장치 매퍼는 여러 상위 수준의 기술을 위한 기반을 제공합니다. **LVM** 외에도 **Device-Mapper** 다중 경로 및 **dmraid** 명령은 장치 매퍼(**Device Mapper**)를 사용합니다. 장치 매퍼에 대한 애플리케이션 인터페이스는 **ioctl** 시스템 호출입니다. 사용자 인터페이스는 **dmsetup** 명령입니다.

LVM 논리 볼륨은 장치 매퍼를 사용하여 활성화됩니다. 각 논리 볼륨은 매핑된 장치로 변환됩니다. 각 세그먼트는 장치를 설명하는 매핑 테이블의 한 행으로 변환됩니다. 장치 매퍼는 선형 매핑, 스트라이핑 매핑 및 오류 매핑을 포함하여 다양한 매핑 대상을 지원합니다. 예를 들어 두 개의 디스크를 하나의 논리 볼륨으로 각 디스크에 하나씩 하나의 선형 매핑 쌍과 연결할 수 있습니다. **LVM**에서 볼륨을 생성할 때 **dmsetup** 명령으로 쿼리할 수 있는 기본 장치-매퍼 장치를 생성합니다. 매핑 테이블에 있는 장치 형식에 대한 자세한 내용은 **A.1**절. “장치 테이블 매핑” 을 참조하십시오. **dmsetup** 명령을 사용하여 장치를 쿼리하는 방법에 대한 자세한 내용은 **A.2**절. “**dmsetup** 명령” 을 참조하십시오.

A.1. 장치 테이블 매핑

매핑된 장치는 지원되는 장치 테이블 매핑을 사용하여 장치의 각 논리 섹터를 매핑하는 방법을 지정하는 테이블에 의해 정의됩니다. 매핑된 장치용 테이블은 양식의 줄 목록으로 구성됩니다.

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

장치 매퍼 테이블의 첫 번째 줄에서는 **start** 매개변수가 **0**과 같아야 합니다. 한 줄에 있는 **start + length** 매개변수는 다음 행의 **start** 과 같아야 합니다. 매핑 테이블의 행에 지정된 매핑 매개변수는 행에 지정된 **mapping** 유형에 따라 다릅니다.

장치 매퍼의 크기는 항상 섹터(**512** 바이트)에 지정됩니다.

장치가 장치 매퍼의 매핑 매개변수로 지정되면 파일 시스템의 장치 이름(예: **/dev/hda**) 또는 **major:minor** 형식의 주요 및 마이너 번호로 참조할 수 있습니다. **major:minor** 형식이 경로 이름 조회를 피할 수 없기 때문에 선호됩니다.

다음은 장치의 샘플 매핑 테이블을 보여줍니다. 이 표에는 네 개의 선형 대상이 있습니다.

```
0 35258368 linear 8:48 65920
```

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

각 행의 처음 **2** 매개변수는 세그먼트 시작 블록과 세그먼트 길이입니다. 다음 키워드는 매핑 대상이며 이 예제의 모든 경우에는 선형 입니다. 행의 나머지 부분은 선형 타겟의 매개 변수로 구성됩니다.

다음 하위 섹션에서는 다음 매핑 형식을 설명합니다.

- **Linear**
- 제거됨
- **mirror**
- **snapshot** 및 **snapshot-origin**
- 오류
- **zero**
- **multipath**
- **Crypt**

A.1.1. linear 매핑 대상

선형 매핑 대상은 블록 범위를 다른 블록 장치에 매핑합니다. 선형 대상의 형식은 다음과 같습니다.

```
start length linear device offset
```

start

가상 장치에서 블록 시작

length

이 세그먼트의 길이

device

*파일 시스템의 장치 이름 또는 **major** 형식의 메이저 및 부 번호에서 참조하는 블록 장치**:minor***

offset

디바이스에서 매핑의 오프셋 시작

*다음 예는 가상 장치 **0**에서 시작 블록이 있는 선형 대상을 보여줍니다. **1638400**의 세그먼트 길이, **8:2**의 주요**:**미네터 번호 쌍, **41146992** 장치에 대한 시작 오프셋입니다.*

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

*다음 예제에서는 **/dev/hda** 장치로 지정된 장치 매개 변수가 있는 선형 대상을 보여줍니다.*

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

A.1.2. 스트리트 매핑 대상

스트라이핑된 매핑 대상은 물리적 장치를 제거할 수 있도록 지원합니다. 이는 스트라이프 수와 스트립 청크 크기 뒤에 장치 이름과 섹터 쌍의 쌍으로 이루어진 인수로 사용됩니다. 스트라이핑된 대상의 형식은 다음과 같습니다.

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

*각 스트라이프마다 하나의 **device** 및 **offset** 매개변수 세트가 있습니다.*

start

가상 장치에서 블록 시작

length

이 세그먼트의 길이

#stripes

가상 장치에 대한 스트라이프 수

chunk_size

*다음 단계로 전환하기 전에 각 스트라이프에 작성된 섹터 수입니다. 커널 페이지 크기만큼 큰 **2**의 전원이어야 합니다.*

device

*파일 시스템의 장치 이름 또는 **major:minor** 형식의 메이저 및 마이너 번호로 참조하는 블록 장치입니다.*

offset

디바이스에서 매핑의 오프셋 시작

*다음 예제에서는 **3**개의 스트라이프와 청크 크기가 **128**인 스트라이핑된 대상을 보여줍니다.*

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

0

가상 장치에서 블록 시작

73728

이 세그먼트의 길이

striped 3 128

*청크 크기가 **128** 블록인 세 개의 장치에 스트라이프*

8:9

major:minor 첫 번째 장치의 번호

384

상기 제1 디바이스 상의 매핑의 오프셋 시작 - starting offset of the mapping on the first device

8:8

major:minor 두 번째 장치 번호

384

상기 제2 장치에 대한 매핑의 오프셋 시작 - starting offset of the mapping on the second device

8:7

major:minor 세 번째 장치 번호

9789824

세 번째 장치에서 매핑의 오프셋 시작

다음 예제에서는 주 및 부 숫자가 아닌 파일 시스템의 장치 이름으로 지정된 장치 매개 변수와 함께 256KiB 청크가 있는 2 스트라이프의 스트라이핑된 대상을 보여줍니다.

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

A.1.3. 미러 매핑 대상

미러 매핑 대상은 미러링된 논리적 장치의 매핑을 지원합니다. 미러링된 대상의 형식은 다음과 같습니다.

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

start

가상 장치에서 블록 시작

length

이 세그먼트의 길이

log_type

가능한 로그 유형과 해당 인수는 다음과 같습니다.

코어

*미러는 로컬이며 미러 로그는 코어 메모리에 유지됩니다. 이 로그 유형은 **1~3**개의 인수를 사용합니다.*

regionsize [[no]sync] [block_on_error]

disk

*미러는 로컬이며 미러 로그는 디스크에 유지됩니다. 이 로그 유형은 **2 ~ 4** 개의 인수를 사용합니다.*

logdevice regionsize [[no]sync] [block_on_error]

clustered_core

*미러는 클러스터되며 미러 로그는 코어 메모리에 유지됩니다. 이 로그 유형은 **2 ~ 4** 개의 인수를 사용합니다.*

regionsize UUID [[no]sync] [block_on_error]

clustered_disk

*미러가 클러스터링되고 미러 로그가 디스크에 유지됩니다. 이 로그 유형은 **3~5**개의 인수를 사용합니다.*

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

*LVM은 미러 또는 미러와 동기화되는 영역을 추적하는 데 사용하는 작은 로그를 유지 관리합니다. **regionsize** 인수는 이러한 지역의 크기를 지정합니다.*

*클러스터형 환경에서 **UUID** 인수는 미러 로그 장치와 연결된 고유 식별자이므로 클러스터 전체에서 로그 상태를 유지할 수 있습니다.*

*선택적 **[no]sync** 인수를 사용하여 미러를 **"in-sync"** 또는 **"out-of-sync"**로 지정할 수 있습니다. **block_on_error** 인수는 미러에 오류를 무시하지 않고 오류에 응답하도록 지시하는 데 사용됩니다.*

#log_args

매핑에 지정될 로그 인수 수

logargs

*미러의 로그 인수입니다. 제공되는 로그 인수 수는 **#log-args** 매개변수에 의해 지정되고 유효한 로그 인수는 **log_type** 매개변수에 따라 결정됩니다.*

#devs

미러의 다리 수; 장치와 오프셋은 각 다리에 대해 지정됨

device

*파일 시스템의 장치 이름 또는 **major:minor** 형식의 주요 및 마이너 번호로 참조되는 각 미러의 블록 장치입니다. **#devs** 매개변수에 표시된 대로 각 미러 브릿지에 대해 블록 장치 및 오프셋이 지정됩니다.*

offset

*장치에서 매핑의 오프셋 시작. **#devs** 매개변수에 표시된 대로 각 미러 브릿지에 대해 블록 장치 및 오프셋이 지정됩니다.*

다음 예제에서는 디스크에 보관된 미러 로그가 있는 클러스터형 미러의 미러 매핑 대상을 보여줍니다.

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

0

가상 장치에서 블록 시작

52428800

이 세그먼트의 길이

mirror clustered_disk

미러가 클러스터링되고 미러 로그가 디스크에서 유지됨을 지정하는 로그 유형이 있는 미러 대상

4

***4** 미러 로그 인수가 따릅니다.*

253:2

*로그 장치의 **major:minor** 번호*

1024

미러 로그에서 동기화 중인 항목을 추적하는 데 사용하는 영역 크기

UUID

*클러스터 전체에서 로그 정보를 유지 관리하는 미러 로그 장치의 **UUID***

block_on_error

***mirror**는 오류에 응답해야 합니다.*

3

미러에 있는 다리 수

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

장치의 주요:미네티터 번호 및 오프셋

A.1.4. 스냅샷 및 snapshot-origin Mapping Targets

볼륨의 첫 번째 LVM 스냅샷을 생성하면 장치 매퍼 장치 4개가 사용됩니다.

1. 소스 볼륨의 원래 매핑 테이블을 포함하는 선형 매핑이 있는 장치.

2. 소스 볼륨에 대해 COW(Copy-On-Write) 장치로 사용되는 선형 매핑이 있는 장치입니다. 각 쓰기의 경우 원본 데이터는 표시되지 않은 콘텐츠를 변경되지 않은 상태로 유지하기 위해 각 스냅샷의 COW 장치에 저장됩니다(COW 장치가 채워지기).

3. 표시되는 스냅샷 볼륨인 #1과 #2를 결합하는 스냅샷 매핑이 있는 장치입니다.

4. 원본 소스 볼륨에서 사용하는 "original" 볼륨(원래 소스 볼륨에서 사용하는 장치 번호)이며 장치 #1의 "snapshot-origin" 매핑으로 교체됩니다.

고정 이름 지정 체계는 이러한 장치를 만드는 데 사용됩니다. 예를 들어 다음 명령을 사용하여 base 라는 LVM 볼륨과 해당 볼륨을 기반으로 snap 이라는 스냅샷 볼륨을 생성할 수 있습니다.

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

이렇게 하면 다음 명령으로 볼 수 있는 4개의 장치가 생성됩니다.

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

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

snapshot-origin 대상의 형식은 다음과 같습니다.

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

start

가상 장치에서 블록 시작

length

이 세그먼트의 길이

origin

스냅샷의 기본 볼륨

***snapshot-origin** 에는 일반적으로 이를 기반으로 하나 이상의 스냅샷이 있습니다. 읽기는 백업 장치에 직접 매핑됩니다. 각 쓰기마다 원본 데이터가 **COW** 장치가 채워질 때까지 볼 수 있는 콘텐츠를 변경하지 않고 유지하기 위해 각 스냅샷의 **COW** 장치에 저장됩니다.*

스냅샷 대상의 형식은 다음과 같습니다.

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

start

가상 장치에서 블록 시작

length

이 세그먼트의 길이

origin

스냅샷의 기본 볼륨

COW-device

데이터의 변경된 청크가 저장되는 장치

P|N

***P(Persistent)** 또는 **N(영구 없음) -** 재부팅 후 스냅샷이 유지되는지 여부를 나타냅니다. 임시 스냅*

샷(N)의 경우 메타데이터를 디스크에 저장해야 하며 커널을 통해 메모리에 보관할 수 있습니다.

chunksize

COW 장치에 저장될 데이터의 변경된 청크 크기

*다음 예제에서는 **254:11**의 원본 장치가 있는 **snapshot-origin** 대상을 보여줍니다.*

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

*다음 예제에서는 원본 장치가 **254:11**이고 **COW** 장치가 **254:12**인 스냅샷 대상을 보여줍니다. 이 스냅샷 장치는 재부팅 시 지속되며 **COW** 장치에 저장된 데이터의 청크 크기는 **16**개 섹터입니다.*

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

A.1.5. 대상 매핑 오류 발생

*오류 매핑 대상을 사용하면 매핑된 섹터에 대한 모든 **I/O** 작업이 실패합니다.*

오류 매핑 대상을 테스트하는 데 사용할 수 있습니다. 장치가 실패하는 방식을 테스트하기 위해 장치 중간에서 나쁜 섹터로 장치 매핑을 만들거나 미러의 다리를 교체하고 다리를 오류 대상으로 교체 할 수 있습니다.

*오류 대상은 실패한 장치 대신 사용할 수 있습니다. 이는 시간 제한을 방지하고 실제 장치에서 재시도 할 수 있습니다. 실패 중에 **LVM** 메타데이터를 다시 정렬하는 동안 중간 대상으로 사용될 수 있습니다.*

*오류 매핑 대상에는 **start** 및 **length** 매개변수 외에 추가 매개변수가 필요하지 않습니다.*

다음 예제에서는 오류 대상을 보여줍니다.

```
0 65536 error
```

A.1.6. 제로 매핑 대상

***0** 매핑 대상은 **/dev/zero** 와 동등한 블록 장치입니다. 이 매핑에 대한 읽기 작업은 **0** 블록을 반환합니*

다. 이 매핑에 작성된 데이터는 삭제되지만 쓰기는 성공합니다. 제로 매핑 대상은 **start** 및 **length** 매개변수 외에 추가 매개변수가 필요하지 않습니다.

다음 예제에서는 **16Tb** 장치의 **0** 대상을 보여줍니다.

```
0 65536 zero
```

A.1.7. 다중 경로 매핑 대상

다중 경로 매핑 대상은 다중 경로 장치의 매핑을 지원합니다. 다중 경로 대상의 형식은 다음과 같습니다.

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

각 경로 그룹에 대해 하나의 **pathgroupargs** 매개변수 세트가 있습니다.

start

가상 장치에서 블록 시작

length

이 세그먼트의 길이

#features

다중 경로 기능 수와 해당 기능은 다음과 같습니다. 이 매개변수가 **0**이면 **feature** 매개변수가 없고 다음 장치 매핑 매개변수는 **#handlerargs** 입니다. 현재 **multipath.conf** 파일의 **features** 속성인 **queue_if_no_path** 를 사용하여 설정할 수 있는 기능이 하나 있습니다. 이는 사용 가능한 경로가 없는 경우 이 다중 경로 장치가 현재 대기열 **I/O** 작업으로 설정되어 있음을 나타냅니다.

다음 예에서 **multipath.conf** 파일의 **no_path_retry** 특성은 경로를 사용하기 위해 설정된 일련의 시도 후 모든 경로가 실패로 표시될 때까지만 대기열 **I/O** 작업으로 설정되어 있습니다. 이 경우 모든 경로 검사기가 지정된 수의 검사에 실패할 때까지 매핑은 다음과 같습니다.

```
0 71014400 multipath 1 queue_if_no_path 0 2 1 round-robin 0 2 1 66:128 \
1000 65:64 1000 round-robin 0 2 1 8:0 1000 67:192 1000
```

지정된 수의 검사에 실패한 모든 경로 검사기가 다음과 같이 표시되면 매핑이 다음과 같이 표시됩

니다.

```
0 71014400 multipath 0 0 2 1 round-robin 0 2 1 66:128 1000 65:64 1000 \
round-robin 0 2 1 8:0 1000 67:192 1000
```

#handlerargs

하드웨어 처리기 인수의 수와 해당 인수입니다. 하드웨어 처리기는 경로 그룹을 전환하거나 I/O 오류를 처리할 때 하드웨어 관련 작업을 수행하는 데 사용할 모듈을 지정합니다. 이 매개변수가 **0**으로 설정되면 다음 매개변수는 **#pathgroups** 입니다.

#pathgroups

경로 그룹의 수입니다. 경로 그룹은 다중 경로 장치가 부하를 분산할 경로 집합입니다. 각 경로 그룹에 대해 하나의 **pathgroupargs** 매개변수 세트가 있습니다.

pathgroup

시도할 다음 경로 그룹입니다.

pathgroupsargs

각 경로 그룹은 다음 인수로 구성됩니다.

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

경로 그룹의 각 경로에는 경로 인수 한 세트가 있습니다.

pathselector

다음 I/O 작업에 사용할 이 경로 그룹의 경로를 결정하는 데 사용되는 알고리즘을 지정합니다.

#selectorargs

다중 경로 매핑에서 이 인수를 따르는 경로 선택기 인수의 수입니다. 현재 이 인수의 값은 항상 **0**입니다.

#paths

이 경로 그룹의 경로 수입니다.

#pathargs

이 그룹의 각 경로에 지정된 경로 인수의 수입니다. 현재 이 수는 항상 1입니다. ioreqs 인수입니다.

device

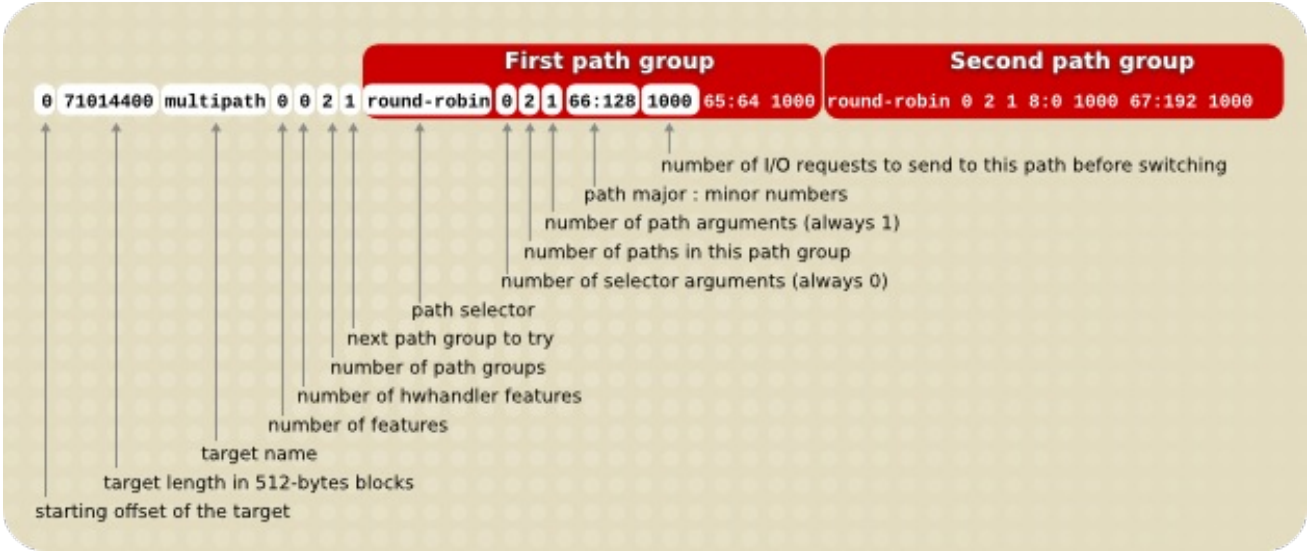
major 형식의 메이저 및 마이너 숫자에서 참조하는 경로의 블록 장치 번호입니다.minor

ioreqs

현재 그룹의 다음 경로로 전환하기 전에 이 경로로 라우팅할 I/O 요청 수입니다.

그림 A.1. "다중 경로 매핑 대상" 2개의 경로 그룹이 있는 다중 경로 대상의 형식을 표시합니다.

그림 A.1. 다중 경로 매핑 대상



다음 예제에서는 동일한 다중 경로 장치에 대한 순수한 장애 조치 대상 정의를 보여줍니다. 이 대상에는 경로 그룹당 하나의 열린 경로만 있는 4개의 경로 그룹이 있으므로 다중 경로 장치는 한 번에 하나의 경로만 사용합니다.

```
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
```

다음 예제에서는 동일한 다중 경로 장치에 대한 전체 분배(multibus) 대상 정의를 보여줍니다. 이 대상에는 모든 경로가 포함된 경로 그룹이 하나만 있습니다. 이 설정에서 다중 경로는 부하를 모든 경로에 균

등하게 분배합니다.

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

*다중 경로에 대한 자세한 내용은 **DM Multipath** 설명서를 참조하십시오.*

A.1.8. 암호 매핑 대상

***crypt** 대상은 지정된 장치를 통과하는 데이터를 암호화합니다. 커널 **Crypto API**를 사용합니다.*

***crypt** 대상의 형식은 다음과 같습니다.*

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

start

가상 장치에서 블록 시작

length

이 세그먼트의 길이

cipher

*암호는 **cipher[-chainmode]-ivmode[:iv options]** 으로 구성됩니다.*

cipher

*사용 가능한 암호는 **/proc/crypto (예: aes)**에 나열됩니다.*

chainmode

*항상 **cbc** 를 사용하십시오. **ebc** 를 사용하지 마십시오. 초기 벡터(**IV**)를 사용하지 마십시오.*

ivmode[:iv options]

IV**는 암호화가 다양하기 위해 사용되는 초기 벡터입니다. **IV** 모드는 **plain** 또는 **essiv:hash** 입니다. **ivmode of -plain** 은 **IV**로 섹터 번호(및 **IV** 오프셋 추가)를 사용합니다. **ivmode of -essiv

는 워터마크 취약점을 방지할 수 있는 개선 사항입니다.

key

hex에 제공된 암호화 키

IV-offset

초기 벡터(**IV**) 오프셋

device

파일 시스템의 장치 이름 또는 **major** 형식의 메이저 및 부 번호에서 참조하는 블록 장치**:minor**

offset

디바이스에서 매핑의 오프셋 시작

다음은 **crypt** 대상의 예입니다.

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

A.2. DMSETUP 명령

dmsetup 명령은 장치 매퍼와 통신하기 위한 명령행 래퍼입니다. **LVM** 장치에 대한 일반적인 시스템 정보는 다음 하위 섹션에 설명된 대로 **dmsetup** 명령의 **info,ls,status** 및 **deps** 옵션을 찾을 수 있습니다.

dmsetup 명령의 추가 옵션 및 기능에 대한 자세한 내용은 **dmsetup(8)** 매뉴얼 페이지를 참조하십시오.

A.2.1. dmsetup info 명령

dmsetup info device 명령은 장치 매퍼 장치에 대한 요약 정보를 제공합니다. 장치 이름을 지정하지 않으면 출력은 현재 구성된 모든 장치 매퍼 장치에 대한 정보입니다. 장치를 지정하는 경우 이 명령은 해당 장치에 대한 정보만 제공합니다.

dmsetup info 명령은 다음 카테고리에 정보를 제공합니다.

이름

장치 이름입니다. **LVM** 장치는 볼륨 그룹 이름과 하이픈으로 구분된 논리 볼륨 이름으로 표시됩니다. 원래 이름의 하이픈은 두 개의 하이픈으로 변환됩니다. 표준 **LVM** 작업 중에 **LVM** 장치의 이름을 이 형식으로 사용해서는 안 됩니다. **LVM** 장치를 직접 지정해서는 안 됩니다. 대신 **vg/lv** 대안을 사용해야 합니다.

상태

가능한 장치 상태는 **SUSPENDED,ACTIVE, READ-ONLY** 입니다. **dmsetup suspend** 명령은 장치 상태를 **SUSPENDED** 로 설정합니다. 장치가 일시 중지되면 해당 장치에 대한 모든 **I/O** 작업이 중지됩니다. **dmsetup resume** 명령은 장치 상태를 **ACTIVE** 로 복원합니다.

읽기 헤드

읽기 작업이 진행 중인 열려 있는 모든 파일에 대해 시스템이 미리 읽는 데이터 블록 수입니다. 기본적으로 커널은 적절한 값을 자동으로 선택합니다. **dmsetup** 명령의 **--readahead** 옵션을 사용하여 이 값을 변경할 수 있습니다.

존재하는 테이블

이 카테고리에 대한 가능한 상태는 **LIVE** 및 **INACTIVE** 입니다. **INACTIVE** 상태는 **dmsetup resume** 명령이 장치 상태를 **ACTIVE** 로 복원할 때 테이블이 로드되었음을 나타냅니다. 이 시점에서 테이블의 상태가 **LIVE** 임을 나타냅니다. 자세한 내용은 **dmsetup** 도움말 페이지를 참조하십시오.

열려 있는 수

열린 참조 수는 장치가 열려 있는 횟수를 나타냅니다. 마운트 명령은 장치를 엽니다.

이벤트 번호

현재 수신된 이벤트 수입니다. **dmsetup wait n** 명령을 실행하면 **n'th** 이벤트를 기다린 후 호출이 수신될 때까지 차단할 수 있습니다.

메이저, 마이너

메이저 및 마이너 장치 번호입니다.

대상 수

*장치를 구성하는 세그먼트 수입니다. 예를 들어 **3**개의 디스크에 걸쳐 있는 선형 장치는 **3**개의 대상을 갖습니다. 디스크의 시작 및 끝으로 구성된 선형 장치는 있지만 중간 장치에는 두 개의 대상이 없었습니다.*

UUID

*장치의 **UUID**입니다.*

*다음 예제에서는 **dmsetup info** 명령에 대한 부분적인 출력을 보여줍니다.*

```
# dmsetup info
Name:              testgfsvg-testgfslv1
State:             ACTIVE
Read Ahead:        256
Tables present:    LIVE
Open count:        0
Event number:      0
Major, minor:      253, 2
Number of targets: 2
UUID: LVM-K528WUGQgPadNXYcFrrf9LnPlUMswgkCkpgPlgYzSvigM7SfeWCypddNSWtNzc2N
...
Name:              VolGroup00-LogVol00
State:             ACTIVE
Read Ahead:        256
Tables present:    LIVE
Open count:        1
Event number:      0
Major, minor:      253, 0
Number of targets: 1
UUID: LVM-tOcS1kqFV9drb0X1Vr8sxeYP0tqcrpdegyqj5lZxe45JMGlmvtqLmbLpBcenh2L3
```

A.2.2. dmsetup ls 명령

***dmsetup ls** 명령을 사용하여 매핑된 장치의 장치 이름을 나열할 수 있습니다. **dmsetup ls --target target_type** 명령을 사용하여 지정된 유형의 대상이 하나 이상 있는 장치를 나열할 수 있습니다. **dmsetup ls** 명령의 다른 옵션은 **dmsetup** 도움말 페이지를 참조하십시오.*

다음 예는 현재 구성된 매핑된 장치의 장치 이름을 나열하는 명령을 보여줍니다.

```
# dmsetup ls
testgfsvg-testgfslv3    (253:4)
testgfsvg-testgfslv2    (253:3)
testgfsvg-testgfslv1    (253:2)
VolGroup00-LogVol01     (253:1)
VolGroup00-LogVol00     (253:0)
```

다음 예제에서는 현재 구성된 미러 매핑의 장치 이름을 나열하는 명령을 보여줍니다.

```
# 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)
```

다중 경로 또는 다른 장치 매퍼 장치에 스택되는 **LVM** 구성은 정렬하기가 복잡할 수 있습니다. **dmsetup ls** 명령은 다음 예제와 같이 장치 간 종속성을 트리로 표시하는 **--tree** 옵션을 제공합니다.

```
# dmsetup ls --tree
vgtest-lvmir (253:13)
 ├─vgtest-lvmir_mimage_1 (253:12)
 │  └─mpathep1 (253:8)
 │    └─mpathe (253:5)
 │      ├─ (8:112)
 │      └─ (8:64)
 ├─vgtest-lvmir_mimage_0 (253:11)
 │  └─mpathcp1 (253:3)
 │    └─mpathc (253:2)
 │      ├─ (8:32)
 │      └─ (8:16)
 └─vgtest-lvmir_mlog (253:4)
   └─mpathfp1 (253:10)
     └─mpathf (253:6)
       ├─ (8:128)
       └─ (8:80)
```

A.2.3. dmsetup status 명령

dmsetup status device 명령은 지정된 장치의 각 대상에 대한 상태 정보를 제공합니다. 장치 이름을 지정하지 않으면 출력은 현재 구성된 모든 장치 매퍼 장치에 대한 정보입니다. **dmsetup status --target target_type** 명령을 사용하여 지정된 유형의 대상이 하나 이상 있는 장치의 상태만 나열할 수 있습니다.

다음 예제에서는 현재 구성된 모든 매핑된 장치의 대상 상태를 나열하는 명령을 보여줍니다.

```
# dmsetup status
testgfsvg-testgfslv3: 0 312352768 linear
testgfsvg-testgfslv2: 0 312352768 linear
testgfsvg-testgfslv1: 0 312352768 linear
```

```
testgfsvg-testgfslv1: 312352768 50331648 linear
VolGroup00-LogVol01: 0 4063232 linear
VolGroup00-LogVol00: 0 151912448 linear
```

A.2.4. dmsetup deps 명령

dmsetup deps device 명령은 지정된 장치의 매핑 테이블에서 참조하는 장치에 대해 (마이저, 마이너) 쌍 목록을 제공합니다. 장치 이름을 지정하지 않으면 출력은 현재 구성된 모든 장치 매퍼 장치에 대한 정보입니다.

다음 예제에서는 현재 구성된 모든 장치에 대한 종속성을 나열하는 명령을 보여줍니다.

```
# 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)
```

다음 예제에서는 장치 **lock_stress-grant-grant-02.1722** 의 종속 항목만 나열하는 명령을 보여줍니다.

```
# dmsetup deps lock_stress-grant--02.1722
3 dependencies  : (253, 33) (253, 32) (253, 31)
```

A.3. UDEV 장치 관리자에 대한 장치 매퍼 지원

udev 장치 관리자의 기본 역할은 **/dev** 디렉터리에서 노드를 설정하는 동적 방법을 제공하는 것입니다. 이러한 노드 생성은 사용자 공간의 **udev** 규칙 적용에 의해 결정됩니다. 이러한 규칙은 특정 장치를 추가, 제거 또는 변경으로 인해 커널에서 직접 전송된 **udev** 이벤트에서 처리됩니다. 이는 지원 핫플러그를 위한 편리한 중앙 메커니즘을 제공합니다.

udev 장치 관리자는 실제 노드를 만드는 것 외에도 이름을 지정할 수 있는 심볼릭 링크를 만들 수 있습니다. 이렇게 하면 필요한 경우**/dev** 디렉토리에서 사용자 지정 이름 지정 및 디렉터리 구조를 자유롭게 선택할 수 있습니다.

각 **udev** 이벤트에는 처리 중인 장치에 대한 기본 정보(예: 이름, 해당 하위 시스템, 장치의 유형, 사용된 메이저 및 마이너 번호, 이벤트 유형)가 포함되어 있습니다. **udev** 규칙 내에서도 액세스할 수 있는 **/sys** 디렉토리에 있는 **/sys** 디렉토리에 있는 모든 정보에 액세스할 수 있으며 이 정보를 기반으로 간단한 필터를 활용하고 이 정보를 기반으로 규칙을 실행할 수 있습니다.

udev 장치 관리자는 또한 노드의 권한을 중앙 집중식으로 설정하는 방법을 제공합니다. 사용자 지정 규

칙 세트를 쉽게 추가하여 이벤트를 처리하는 동안 사용 가능한 모든 장치에 대한 권한을 정의할 수 있습니다.

udev 규칙에 직접 프로그램 후크를 추가할 수도 있습니다. udev 장치 관리자는 이러한 프로그램을 호출하여 이벤트를 처리하는 데 필요한 추가 처리를 제공할 수 있습니다. 또한 프로그램은 이 처리의 결과로 환경 변수를 내보낼 수 있습니다. 제공된 모든 결과는 규칙에서 정보의 보조 소스로 사용할 수 있습니다.

udev 라이브러리를 사용하는 모든 소프트웨어는 사용 가능한 모든 정보로 udev 이벤트를 수신하고 처리할 수 있으므로 처리는 udev 데몬에만 바인딩되지 않습니다.

A.3.1. 장치 매퍼와 udev 통합

장치 매퍼는 udev 통합을 직접 지원합니다. 이렇게 하면 LVM 장치를 포함하여 장치 매퍼 장치와 관련된 모든 udev 처리와 장치 매퍼가 동기화됩니다. udev 데몬의 규칙 애플리케이션은 장치의 변경 사항 소스(예: dmsetup 및 LVM)와의 병렬 처리 형식이므로 동기화가 필요합니다. 이러한 지원이 없으면 사용자가 이전 변경 이벤트의 결과로 udev 규칙에 의해 열려 있고 처리된 장치를 제거하려고 시도하는 것이 일반적이었습니다. 해당 장치의 변경 사이에 매우 짧은 시간이 있을 때 이는 특히 일반적이었습니다.

Red Hat Enterprise Linux는 일반 및 LVM에도 장치 매퍼 장치에 대해 공식적으로 지원되는 udev 규칙을 제공합니다. 표 A.1. "장치 맵 장치의 udev 규칙" /lib/udev/rules.d 에 설치된 이러한 규칙을 요약합니다.

표 A.1. 장치 맵 장치의 udev 규칙

파일 이름	설명
10-DM.rules	일반 장치 매퍼 규칙을 포함하고 /dev/dm-N 대상을 사용하여 /dev/mapper 에 심볼릭 링크를 만듭니다. 여기서 N 은 커널에 의해 장치에 동적으로 할당된 번호입니다 (/dev/dm-N 은 노드임). 참고: /dev/dm-N 노드는 N 번호가 동적으로 할당되고 장치 활성화 방식에 따라 변경되기 때문에 스크립트에 사용하지 않아야 합니다. 따라서 /dev/mapper 디렉터리의 실제 이름을 사용해야 합니다. 이 레이아웃은 노드/symlink 를 생성해야 하는 방법에 대한 udev 요구 사항을 지원하기 위한 것입니다.

파일 이름	설명
11-DM-lvm.rules	**LVM** 장치에 적용되는 규칙을 포함하고 볼륨 그룹의 논리 볼륨에 대한 심볼릭 링크를 만듭니다. 심볼릭 링크는 **/dev/ dm-N** 대상이 있는 **/dev/ Cryostatname** 디렉터리에 생성됩니다. 참고: 장치 매퍼 하위 시스템에 대한 향후 모든 규칙의 이름을 지정하기 위한 표준과 일치하려면 **udev** 규칙은 **11-dm-subsystem_name.rules** 형식을 따라야 합니다. **udev** 규칙을 제공하는 모든 **libdevmapper** 사용자는 이 표준을 따라야 합니다.
13-dm-disk.rules	일반적으로 모든 장치 매퍼 장치에 적용할 규칙이 포함되어 있으며 **/dev/ disk/by-id** 및 **/dev /disk/by-uuid** 디렉터리에 심볼릭 링크를 생성합니다.
95-dm-notify.rules	**libdevmapper** 를 사용하여 대기 프로세스에 알리는 규칙이 포함되어 있습니다(예: **LVM** 및 **dmsetup**). 모든 **udev** 처리가 완료되었는지 확인하기 위해 이전 규칙이 적용된 후 알림이 수행됩니다. 그런 다음 알림을 받은 프로세스가 재개됩니다.
69-dm-lvm-metad.rules	시스템에 새로 표시된 블록 장치에서 **LVM** 스캔을 트리거하고 가능한 경우 **LVM** 자동 활성화를 수행하는 후크가 포함되어 있습니다. **lvm.conf** 파일에서 **use_lvmetad=1** 으로 설정된 **lvmetad** 데몬을 지원합니다. 클러스터형 환경에서는 **lvmetad** 데몬 및 자동 활성화가 지원되지 않습니다.

12-dm-permissions.rules 파일을 통해 사용자 지정 권한 규칙을 추가할 수 있습니다. 이 파일은 **/lib/udev/rules** 디렉터리에 설치되지 않습니다. **/ usr/share/doc/device-mapper-**버전 디렉터리에 있습니다. **12-dm-permissions.rules** 파일은 예로 지정된 일부 일치하는 규칙에 따라 권한을 설정하는 방법에 대한 힌트를 포함하는 템플릿입니다. 파일에는 몇 가지 일반적인 상황에 대한 예제가 포함되어 있습니다. 이 파일을 편집하여 **/etc/udev/rules.d** 디렉토리에 수동으로 배치하면 업데이트가 유지되므로 설정이 유지됩니다.

이러한 규칙은 이벤트를 처리하는 동안 다른 규칙에서 사용할 수 있는 모든 기본 변수를 설정합니다.

다음 변수는 **10-dm.rules** 로 설정됩니다.

- *DM_NAME: 장치 매퍼 장치 이름*
- *DM_UUID: 장치 매퍼 장치 UUID*
- *DM_SUSPENDED: 장치 매퍼 장치의 일시 중단 상태*
- *DM_UDEV_RULES_VSN:udev 규칙 버전 (이전에 언급된 변수가 공식 장치 맵퍼 규칙에 의해 직접 설정되어 있는지 확인하는 다른 모든 규칙)*

다음 변수는 11-dm-lvm.rules:에서 설정됩니다.

- *DM_LV_NAME: 논리 볼륨 이름*
- *DM_VG_NAME: 볼륨 그룹 이름*
- *DM_LV_LAYER: LVM 계층 이름*

이러한 모든 변수는 12-dm-permissions.rules 파일에서 사용하여 12-dm-permissions.rules 파일에 설명된 대로 특정 장치 매퍼 장치에 대한 권한을 정의할 수 있습니다.

A.3.2. udev를 지원하는 명령 및 인터페이스

표 A.2. "udev를 지원하기 위한 dmsetup 명령" udev 통합을 지원하는 dmsetup 명령이 요약되어 있습니다.

표 A.2. udev를 지원하기 위한 dmsetup 명령

명령	설명
dmsetup udevcomplete	udev 가 규칙 처리를 완료하고 대기 중인 프로세스 잠금 해제를 완료했음을 알리는 데 사용됩니다(95-dm-notify.rules의 udev 규칙 내에서 호출됨).

명령	설명
dmsetup udevcomplete_all	디버깅 목적으로 모든 대기 중인 프로세스의 잠금을 수동으로 해제하는 데 사용됩니다.
dmsetup udevcookies	디버깅 목적으로 모든 기존 쿠키(시스템 전체 세마포)를 표시하는 데 사용됩니다.
dmsetup udevcreatecookie	쿠키 생성(**semaphore**)을 수동으로 만드는 데 사용됩니다. 이는 하나의 동기화 리소스에서 더 많은 프로세스를 실행하는 데 유용합니다.
dmsetup udevreleasecookie	하나의 동기화 쿠키 아래에 배치된 모든 프로세스와 관련된 모든 **udev** 처리를 기다리는 데 사용됩니다.

udev 통합을 지원하는 **dmsetup** 옵션은 다음과 같습니다.

--udevcookie

udev 트랜잭션에 추가하려는 모든 **dmsetup** 프로세스에 대해 정의해야 합니다. **udevcreatecookie** 및 **udevreleasecookie** 와 함께 사용됩니다.

```
COOKIE=$(dmsetup udevcreatecookie)
  dmsetup command --udevcookie $COOKIE ....
  dmsetup command --udevcookie $COOKIE ....
  ....
  dmsetup command --udevcookie $COOKIE ....
dmsetup udevreleasecookie --udevcookie $COOKIE
```

--udevcookie 옵션을 사용하는 것 외에도 변수를 프로세스 환경으로 내보낼 수 있습니다.

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

--noudevrules

udev 규칙을 비활성화합니다. 노드/**symlinks**는 **libdevmapper** 자체(이전 방식)에 의해 생성됩니다. **udev** 가 제대로 작동하지 않는 경우 이 옵션은 디버깅을 위한 것입니다.

--noudevsync

***udev** 동기화를 비활성화합니다. 이는 디버깅 목적으로도 사용됩니다.*

***dmsetup** 명령 및 해당 옵션에 대한 자세한 내용은 **dmsetup(8)** 도움말 페이지를 참조하십시오.*

***LVM** 명령은 **udev** 통합을 지원하는 다음 옵션을 지원합니다.*

- ***--noudevrules: dmsetup** 명령과 마찬가지로 **udev** 규칙을 비활성화합니다.*
- ***--noudevsync: dmsetup** 명령과 마찬가지로 **udev** 동기화를 비활성화합니다.*

***lvm.conf** 파일에는 **udev** 통합을 지원하는 다음 옵션이 포함되어 있습니다.*

- ***udev_rules:** 전역적으로 모든 **LVM2** 명령에 대해 **udev_rules** 를 활성화/비활성화합니다.*
- ***udev_sync:** 전역적으로 모든 **LVM** 명령에 대해 **udev** 동기화를 활성화/비활성화합니다.*

***lvm.conf** 파일 옵션에 대한 자세한 내용은 **lvm.conf** 파일의 인라인 주석을 참조하십시오.*

부록 B. LVM 구성 파일

LVM은 여러 구성 파일을 지원합니다. 시스템을 시작할 때 lvm.conf 구성 파일은 기본적으로 /etc/lvm 으로 설정된 환경 변수 LVM_SYSTEM_DIR 에 의해 지정된 디렉터리에서 로드됩니다.

lvm.conf 파일은 로드할 추가 구성 파일을 지정할 수 있습니다. 이후 파일의 설정은 이전 파일의 설정을 재정의합니다. 모든 구성 파일을 로드한 후 사용 중인 설정을 표시하려면 lvmconfig 명령을 실행합니다.

추가 구성 파일을 로드하는 방법에 대한 자세한 내용은 D.2절. “호스트 태그” 을 참조하십시오.

B.1. LVM 구성 파일

다음 파일은 LVM 구성에 사용됩니다.

/etc/lvm/lvm.conf

도구에서 읽은 중앙 구성 파일입니다.

etc/lvm/lvm_hosttag.conf

각 호스트 태그마다 존재하는 경우 추가 구성 파일을 읽습니다. lvm_hosttag.conf. 해당 파일이 새 태그를 정의하면 추가 설정 파일이 읽을 파일 목록에 추가됩니다. 호스트 태그에 대한 자세한 내용은 D.2절. “호스트 태그” 을 참조하십시오.

LVM 구성 파일 외에도 LVM을 실행하는 시스템에는 LVM 시스템 설정에 영향을 주는 다음과 같은 파일이 포함되어 있습니다.

/etc/lvm/cache/.cache

장치 이름 필터 캐시 파일(구성 가능).

/etc/lvm/backup/

자동 볼륨 그룹 메타데이터 백업용 디렉터리(구성 가능).

/etc/lvm/archive/

자동 볼륨 그룹 메타데이터 아카이브(디렉터리 및 아카이브 기록 깊이와 관련하여 구성 가능)를

위한 디렉터리입니다.

/var/lock/lvm/

단일 호스트 구성에서 병렬 툴 실행이 메타데이터를 손상시키지 않도록 잠금 파일을 사용합니다. 클러스터 전체의 DLM이 사용됩니다.

B.2. LVMCONFIG 명령

현재 LVM 구성을 표시하거나 lvmconfig 명령을 사용하여 파일을 파일에 저장할 수 있습니다. lvmconfig 명령에서 제공하는 다양한 기능은 다음과 같습니다.

- 모든 태그 구성 파일과 병합된 현재 lvm 구성을 덤프할 수 있습니다.
- 값이 기본값과 다른 모든 현재 구성 설정을 덤프할 수 있습니다.
- 현재 LVM 버전에 도입된 새 구성 설정을 특정 LVM 버전에 덤프할 수 있습니다.
- 전체 프로필에 사용자 지정할 수 있는 모든 구성 설정을 명령 및 메타데이터 프로필에 대해 별도로 덤프할 수 있습니다. LVM 프로필에 대한 자세한 내용은 B.3절. “LVM 프로필” 에서 참조하십시오.
- 특정 버전의 LVM에 대한 구성 설정만 덤프할 수 있습니다.
- 현재 구성을 검증할 수 있습니다.

지원되는 기능의 전체 목록과 lvmconfig 옵션 지정에 대한 정보는 lvmconfig 도움말 페이지를 참조하십시오.

B.3. LVM 프로필

LVM 프로필은 다양한 환경 또는 사용의 특정 특성을 달성하는 데 사용할 수 있는 선택된 사용자 지정 가능한 구성 설정 집합입니다. 일반적으로 프로필 이름은 해당 환경을 반영하거나 사용해야 합니다. LVM 프로필은 기존 구성을 덮어씁니다.

LVM에서 인식하는 LVM 프로필 그룹에는 명령 프로필 및 메타데이터 프로필이라는 두 가지 그룹이 있습니다.

- *명령 프로필은 글로벌 LVM 명령 수준에서 선택한 구성 설정을 재정의하는 데 사용됩니다. 프로필은 LVM 명령 실행 시작 시 적용되며 LVM 명령 실행 시 사용됩니다. LVM 명령을 실행할 때 --commandprofile ProfileName 옵션을 지정하여 명령 프로필을 적용합니다.*

- *메타데이터 프로필은 볼륨 그룹/로그 논리 볼륨 수준에서 선택한 구성 설정을 재정의하는 데 사용됩니다. 처리 중인 각 볼륨 그룹/논리적 볼륨에 독립적으로 적용됩니다. 따라서 각 볼륨 그룹/로그 논리 볼륨은 메타데이터에 사용된 프로필 이름을 저장할 수 있으므로 다음에 볼륨 그룹/로그 논리 볼륨이 처리되면 프로필이 자동으로 적용됩니다. 볼륨 그룹과 해당 논리 볼륨에 다른 프로필이 정의된 경우 논리 볼륨에 대해 정의된 프로필이 선호됩니다.*

  - *Cryostatcreate 또는 lvcreate 명령을 사용하여 볼륨 그룹 또는 논리 볼륨을 생성할 때 --metadataprofile ProfileName 옵션을 지정하여 볼륨 그룹 또는 논리 볼륨에 메타데이터 프로필을 연결할 수 있습니다.*

  - *lvchange 또는 Cryostat change 명령의 --metadataprofile ProfileName 또는 --detachprofile 옵션을 지정하여 기존 볼륨 그룹 또는 논리 볼륨에 메타데이터 프로필을 연결하거나 분리할 수 있습니다.*

  - *현재 볼륨 그룹 또는 논리 볼륨에 연결된 메타데이터 프로필 을 표시하려면 -o Cryostat_profile 및 -o lv _profile 출력 옵션을 지정할 수 있습니다.*

명령 프로필에 허용되는 옵션 세트와 메타데이터 프로필에 허용된 옵션 세트는 함께 사용할 수 없습니다. 이 두 세트 중 하나에 속하는 설정은 함께 혼합할 수 없으며 LVM 툴에서 해당 프로필을 거부합니다.

LVM은 몇 가지 사전 정의된 구성 프로필을 제공합니다. LVM 프로필은 기본적으로 /etc/lvm/profile 디렉터리에 저장됩니다. 이 위치는 /etc/lvm/lvm.conf 파일의 profile_dir 설정을 사용하여 변경할 수 있습니다. 각 프로필 구성은 프로필 디렉터리의 ProfileName. profile 파일에 저장됩니다. LVM 명령에서 프로필을 참조하는 경우 .profile 접미사가 생략됩니다.

다른 값을 사용하여 추가 프로필을 생성할 수 있습니다. 이를 위해 LVM은 각 유형의 프로필 로 사용자 지정할 수 있는 모든 설정이 포함된 metadata_profile_template.profile 파일(명령 프로필용) 및 metadata_profile_template.profile 파일을 제공합니다. 이러한 템플릿 프로필을 복사하고 필요에 따라 편집할 수 있습니다.

또는 lvmconfig 명령을 사용하여 두 프로필 유형에 대한 프로필 파일의 지정된 섹션에 대한 새 프로필

*을 생성할 수 있습니다. 다음 명령은 섹션 의 설정으로 구성된 **ProfileName.profile**이라는 새 명령 프로필을 생성합니다.*

```
lvmconfig --file ProfileName.profile --type profilable-command section
```

*다음 명령은 섹션 의 설정으로 구성된 **ProfileName.profile**이라는 새 메타데이터 프로필을 생성합니다.*

```
lvmconfig --file ProfileName.profile --type profilable-metadata section
```

섹션을 지정하지 않으면 프로필에서 사용자 정의할 수 있는 모든 설정이 보고됩니다.

B.4. 샘플 LVM.CONF 파일

*다음은 샘플 **lvm.conf** 구성 파일입니다. 구성 파일은 이 파일과 약간 다를 수 있습니다.*

참고

*다음 명령을 실행하여 주석이 포함된 모든 기본값을 설정하고 주석을 포함하여 **lvm.conf** 파일을 생성할 수 있습니다.*

```
lvmconfig --type default --withcomments
```

```
# This is an example configuration file for the LVM2 system.
# It contains the default settings that would be used if there was no
# /etc/lvm/lvm.conf file.
#
# Refer to 'man lvm.conf' for further information including the file layout.
#
# Refer to 'man lvm.conf' for information about how settings configured in
# this file are combined with built-in values and command line options to
# arrive at the final values used by LVM.
#
# Refer to 'man lvmconfig' for information about displaying the built-in
# and configured values used by LVM.
#
# If a default value is set in this file (not commented out), then a
# new version of LVM using this file will continue using that value,
# even if the new version of LVM changes the built-in default value.
#
# To put this file in a different directory and override /etc/lvm set
# the environment variable LVM_SYSTEM_DIR before running the tools.
#
# N.B. Take care that each setting only appears once if uncommenting
```

```
# example settings in this file.


# Configuration section config.
# How LVM configuration settings are handled.
config {

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

 # Configuration option config/abort_on_errors.
 # Abort the LVM process if a configuration mismatch is found.
 abort_on_errors = 0

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

# Configuration section devices.
# How LVM uses block devices.
devices {

 # Configuration option devices/dir.
 # Directory in which to create volume group device nodes.
 # Commands also accept this as a prefix on volume group names.
 # This configuration option is advanced.
 dir = "/dev"

 # Configuration option devices/scan.
 # Directories containing device nodes to use with LVM.
 # This configuration option is advanced.
 scan = [ "/dev" ]

 # Configuration option devices/obtain_device_list_from_udev.
 # Obtain the list of available devices from udev.
 # This avoids opening or using any inapplicable non-block devices or
 # subdirectories found in the udev directory. Any device node or
 # symlink not managed by udev in the udev directory is ignored. This
 # setting applies only to the udev-managed device directory; other
 # directories will be scanned fully. LVM needs to be compiled with
 # udev support for this setting to apply.
 obtain_device_list_from_udev = 1

 # Configuration option devices/external_device_info_source.
 # Select an external device information source.
 # Some information may already be available in the system and LVM can
 # use this information to determine the exact type or use of devices it
 # processes. Using an existing external device information source can
 # speed up device processing as LVM does not need to run its own native
```

```
# routines to acquire this information. For example, this information
# is used to drive LVM filtering like MD component detection, multipath
# component detection, partition detection and others.
#
# Accepted values:
#   none
#     No external device information source is used.
#   udev
#     Reuse existing udev database records. Applicable only if LVM is
#     compiled with udev support.
#
external_device_info_source = "none"

# Configuration option devices/preferred_names.
# Select which path name to display for a block device.
# If multiple path names exist for a block device, and LVM needs to
# display a name for the device, the path names are matched against
# each item in this list of regular expressions. The first match is
# used. Try to avoid using undescriptive /dev/dm-N names, if present.
# If no preferred name matches, or if preferred_names are not defined,
# the following built-in preferences are applied in order until one
# produces a preferred name:
# Prefer names with path prefixes in the order of:
# /dev/mapper, /dev/disk, /dev/dm-*, /dev/block.
# Prefer the name with the least number of slashes.
# Prefer a name that is a symlink.
# Prefer the path with least value in lexicographical order.
#
# Example
# preferred_names = [ "^/dev/mpath/", "^/dev/mapper/mpath", "^/dev/[hs]d" ]
#
preferred_names = [ "^/dev/mpath/", "^/dev/mapper/mpath", "^/dev/[hs]d" ]

# Configuration option devices/filter.
# Limit the block devices that are used by LVM commands.
# This is a list of regular expressions used to accept or reject block
# device path names. Each regex is delimited by a vertical bar '|'
# (or any character) and is preceded by 'a' to accept the path, or
# by 'r' to reject the path. The first regex in the list to match the
# path is used, producing the 'a' or 'r' result for the device.
# When multiple path names exist for a block device, if any path name
# matches an 'a' pattern before an 'r' pattern, then the device is
# accepted. If all the path names match an 'r' pattern first, then the
# device is rejected. Unmatching path names do not affect the accept
# or reject decision. If no path names for a device match a pattern,
# then the device is accepted. Be careful mixing 'a' and 'r' patterns,
# as the combination might produce unexpected results (test changes.)
# Run vgscan after changing the filter to regenerate the cache.
# See the use_lvmetad comment for a special case regarding filters.
#
# Example
# Accept every block device:
# filter = [ "a|.*/|" ]
# Reject the cdrom drive:
# filter = [ "r|/dev/cdrom|" ]
# Work with just loopback devices, e.g. for testing:
```

```
# filter = [ "a|loop|", "r|.*|" ]
# Accept all loop devices and ide drives except hdc:
# filter = [ "a|loop|", "r|/dev/hdc|", "a|/dev/ide|", "r|.*|" ]
# Use anchors to be very specific:
# filter = [ "a|^/dev/hda8$|", "r|.*/|" ]
#
# This configuration option has an automatic default value.
# filter = [ "a|.*/|" ]

# Configuration option devices/global_filter.
# Limit the block devices that are used by LVM system components.
# Because devices/filter may be overridden from the command line, it is
# not suitable for system-wide device filtering, e.g. udev and lvmetad.
# Use global_filter to hide devices from these LVM system components.
# The syntax is the same as devices/filter. Devices rejected by
# global_filter are not opened by LVM.
# This configuration option has an automatic default value.
# global_filter = [ "a|.*/|" ]

# Configuration option devices/cache_dir.
# Directory in which to store the device cache file.
# The results of filtering are cached on disk to avoid rescanning dud
# devices (which can take a very long time). By default this cache is
# stored in a file named .cache. It is safe to delete this file; the
# tools regenerate it. If obtain_device_list_from_udev is enabled, the
# list of devices is obtained from udev and any existing .cache file
# is removed.
cache_dir = "/etc/lvm/cache"

# Configuration option devices/cache_file_prefix.
# A prefix used before the .cache file name. See devices/cache_dir.
cache_file_prefix = ""

# Configuration option devices/write_cache_state.
# Enable/disable writing the cache file. See devices/cache_dir.
write_cache_state = 1

# Configuration option devices/types.
# List of additional acceptable block device types.
# These are of device type names from /proc/devices, followed by the
# maximum number of partitions.
#
# Example
# types = [ "fd", 16 ]
#
# This configuration option is advanced.
# This configuration option does not have a default value defined.

# Configuration option devices/sysfs_scan.
# Restrict device scanning to block devices appearing in sysfs.
# This is a quick way of filtering out block devices that are not
# present on the system. sysfs must be part of the kernel and mounted.)
sysfs_scan = 1

# Configuration option devices/multipath_component_detection.
# Ignore devices that are components of DM multipath devices.
```

```
    multipath_component_detection = 1

    # Configuration option devices/md_component_detection.
    # Ignore devices that are components of software RAID (md) devices.
    md_component_detection = 1

    # Configuration option devices/fw_raid_component_detection.
    # Ignore devices that are components of firmware RAID devices.
    # LVM must use an external_device_info_source other than none for this
    # detection to execute.
    fw_raid_component_detection = 0

    # Configuration option devices/md_chunk_alignment.
    # Align PV data blocks with md device's stripe-width.
    # This applies if a PV is placed directly on an md device.
    md_chunk_alignment = 1

    # Configuration option devices/default_data_alignment.
    # Default alignment of the start of a PV data area in MB.
    # If set to 0, a value of 64KiB will be used.
    # Set to 1 for 1MiB, 2 for 2MiB, etc.
    # This configuration option has an automatic default value.
    # default_data_alignment = 1

    # Configuration option devices/data_alignment_detection.
    # Detect PV data alignment based on sysfs device information.
    # The start of a PV data area will be a multiple of minimum_io_size or
    # optimal_io_size exposed in sysfs. minimum_io_size is the smallest
    # request the device can perform without incurring a read-modify-write
    # penalty, e.g. MD chunk size. optimal_io_size is the device's
    # preferred unit of receiving I/O, e.g. MD stripe width.
    # minimum_io_size is used if optimal_io_size is undefined (0).
    # If md_chunk_alignment is enabled, that detects the optimal_io_size.
    # This setting takes precedence over md_chunk_alignment.
    data_alignment_detection = 1

    # Configuration option devices/data_alignment.
    # Alignment of the start of a PV data area in KiB.
    # If a PV is placed directly on an md device and md_chunk_alignment or
    # data_alignment_detection are enabled, then this setting is ignored.
    # Otherwise, md_chunk_alignment and data_alignment_detection are
    # disabled if this is set. Set to 0 to use the default alignment or the
    # page size, if larger.
    data_alignment = 0

    # Configuration option devices/data_alignment_offset_detection.
    # Detect PV data alignment offset based on sysfs device information.
    # The start of a PV aligned data area will be shifted by the
    # alignment_offset exposed in sysfs. This offset is often 0, but may
    # be non-zero. Certain 4KiB sector drives that compensate for windows
    # partitioning will have an alignment_offset of 3584 bytes (sector 7
    # is the lowest aligned logical block, the 4KiB sectors start at
    # LBA -1, and consequently sector 63 is aligned on a 4KiB boundary).
    # pvcreate --dataalignmentoffset will skip this detection.
    data_alignment_offset_detection = 1
```

```
# Configuration option devices/ignore_suspended_devices.
# Ignore DM devices that have I/O suspended while scanning devices.
# Otherwise, LVM waits for a suspended device to become accessible.
# This should only be needed in recovery situations.
ignore_suspended_devices = 0

# Configuration option devices/ignore_lvm_mirrors.
# Do not scan 'mirror' LVs to avoid possible deadlocks.
# This avoids possible deadlocks when using the 'mirror' segment type.
# This setting determines whether LVs using the 'mirror' segment type
# are scanned for LVM labels. This affects the ability of mirrors to
# be used as physical volumes. If this setting is enabled, it is
# impossible to create VGs on top of mirror LVs, i.e. to stack VGs on
# mirror LVs. If this setting is disabled, allowing mirror LVs to be
# scanned, it may cause LVM processes and I/O to the mirror to become
# blocked. This is due to the way that the mirror segment type handles
# failures. In order for the hang to occur, an LVM command must be run
# just after a failure and before the automatic LVM repair process
# takes place, or there must be failures in multiple mirrors in the
# same VG at the same time with write failures occurring moments before
# a scan of the mirror's labels. The 'mirror' scanning problems do not
# apply to LVM RAID types like 'raid1' which handle failures in a
# different way, making them a better choice for VG stacking.
ignore_lvm_mirrors = 1

# Configuration option devices/disable_after_error_count.
# Number of I/O errors after which a device is skipped.
# During each LVM operation, errors received from each device are
# counted. If the counter of a device exceeds the limit set here,
# no further I/O is sent to that device for the remainder of the
# operation. Setting this to 0 disables the counters altogether.
disable_after_error_count = 0

# Configuration option devices/require_restorefile_with_uuid.
# Allow use of pvcreate --uuid without requiring --restorefile.
require_restorefile_with_uuid = 1

# Configuration option devices/pv_min_size.
# Minimum size in KiB of block devices which can be used as PVs.
# In a clustered environment all nodes must use the same value.
# Any value smaller than 512KiB is ignored. The previous built-in
# value was 512.
pv_min_size = 2048

# Configuration option devices/issue_discards.
# Issue discards to PVs that are no longer used by an LV.
# Discards are sent to an LV's underlying physical volumes when the LV
# is no longer using the physical volumes' space, e.g. lvremove,
# lvreduce. Discards inform the storage that a region is no longer
# used. Storage that supports discards advertise the protocol-specific
# way discards should be issued by the kernel (TRIM, UNMAP, or
# WRITE SAME with UNMAP bit set). Not all storage will support or
# benefit from discards, but SSDs and thinly provisioned LUNs
# generally do. If enabled, discards will only be issued if both the
# storage and kernel provide support.
issue_discards = 0
```

```
 # Configuration option devices/allow_changes_with_duplicate_pvs.
 # Allow VG modification while a PV appears on multiple devices.
 # When a PV appears on multiple devices, LVM attempts to choose the
 # best device to use for the PV. If the devices represent the same
 # underlying storage, the choice has minimal consequence. If the
 # devices represent different underlying storage, the wrong choice
 # can result in data loss if the VG is modified. Disabling this
 # setting is the safest option because it prevents modifying a VG
 # or activating LVs in it while a PV appears on multiple devices.
 # Enabling this setting allows the VG to be used as usual even with
 # uncertain devices.
 allow_changes_with_duplicate_pvs = 0
}

# Configuration section allocation.
# How LVM selects space and applies properties to LVs.
allocation {

 # Configuration option allocation/cling_tag_list.
 # Advise LVM which PVs to use when searching for new space.
 # When searching for free space to extend an LV, the 'cling' allocation
 # policy will choose space on the same PVs as the last segment of the
 # existing LV. If there is insufficient space and a list of tags is
 # defined here, it will check whether any of them are attached to the
 # PVs concerned and then seek to match those PV tags between existing
 # extents and new extents.
 #
 # Example
 # Use the special tag "@*" as a wildcard to match any PV tag:
 # cling_tag_list = [ "@*" ]
 # LVs are mirrored between two sites within a single VG, and
 # PVs are tagged with either @site1 or @site2 to indicate where
 # they are situated:
 # cling_tag_list = [ "@site1", "@site2" ]
 #
 # This configuration option does not have a default value defined.

 # Configuration option allocation/maximise_cling.
 # Use a previous allocation algorithm.
 # Changes made in version 2.02.85 extended the reach of the 'cling'
 # policies to detect more situations where data can be grouped onto
 # the same disks. This setting can be used to disable the changes
 # and revert to the previous algorithm.
 maximise_cling = 1

 # Configuration option allocation/use_blkid_wiping.
 # Use blkid to detect existing signatures on new PVs and LVs.
 # The blkid library can detect more signatures than the native LVM
 # detection code, but may take longer. LVM needs to be compiled with
 # blkid wiping support for this setting to apply. LVM native detection
 # code is currently able to recognize: MD device signatures,
 # swap signature, and LUKS signatures. To see the list of signatures
 # recognized by blkid, check the output of the 'blkid -k' command.
 use_blkid_wiping = 1
```

```
# Configuration option allocation/wipe_signatures_when_zeroing_new_lvs.
# Look for and erase any signatures while zeroing a new LV.
# The --wipesignatures option overrides this setting.
# Zeroing is controlled by the -Z/--zero option, and if not specified,
# zeroing is used by default if possible. Zeroing simply overwrites the
# first 4KiB of a new LV with zeroes and does no signature detection or
# wiping. Signature wiping goes beyond zeroing and detects exact types
# and positions of signatures within the whole LV. It provides a
# cleaner LV after creation as all known signatures are wiped. The LV
# is not claimed incorrectly by other tools because of old signatures
# from previous use. The number of signatures that LVM can detect
# depends on the detection code that is selected (see
# use_blkid_wiping.) Wiping each detected signature must be confirmed.
# When this setting is disabled, signatures on new LVs are not detected
# or erased unless the --wipesignatures option is used directly.
wipe_signatures_when_zeroing_new_lvs = 1

# Configuration option allocation/mirror_logs_require_separate_pvs.
# Mirror logs and images will always use different PVs.
# The default setting changed in version 2.02.85.
mirror_logs_require_separate_pvs = 0

# Configuration option allocation/raid_stripe_all_devices.
# Stripe across all PVs when RAID stripes are not specified.
# If enabled, all PVs in the VG or on the command line are used for raid0/4/5/6/10
# when the command does not specify the number of stripes to use.
# This was the default behaviour until release 2.02.162.
# This configuration option has an automatic default value.
# raid_stripe_all_devices = 0

# Configuration option allocation/cache_pool_metadata_require_separate_pvs.
# Cache pool metadata and data will always use different PVs.
cache_pool_metadata_require_separate_pvs = 0

# Configuration option allocation/cache_mode.
# The default cache mode used for new cache.
#
# Accepted values:
#   writethrough
#     Data blocks are immediately written from the cache to disk.
#   writeback
#     Data blocks are written from the cache back to disk after some
#     delay to improve performance.
#
# This setting replaces allocation/cache_pool_cachemode.
# This configuration option has an automatic default value.
# cache_mode = "writethrough"

# Configuration option allocation/cache_policy.
# The default cache policy used for new cache volume.
# Since kernel 4.2 the default policy is smq (Stochastic multique),
# otherwise the older mq (Multiqueue) policy is selected.
# This configuration option does not have a default value defined.

# Configuration section allocation/cache_settings.
# Settings for the cache policy.
```

```
# See documentation for individual cache policies for more info.
# This configuration section has an automatic default value.
# cache_settings {
# }

# Configuration option allocation/cache_pool_chunk_size.
# The minimal chunk size in KiB for cache pool volumes.
# Using a chunk_size that is too large can result in wasteful use of
# the cache, where small reads and writes can cause large sections of
# an LV to be mapped into the cache. However, choosing a chunk_size
# that is too small can result in more overhead trying to manage the
# numerous chunks that become mapped into the cache. The former is
# more of a problem than the latter in most cases, so the default is
# on the smaller end of the spectrum. Supported values range from
# 32KiB to 1GiB in multiples of 32.
# This configuration option does not have a default value defined.

# Configuration option allocation/thin_pool_metadata_require_separate_pvs.
# Thin pool metdata and data will always use different PVs.
thin_pool_metadata_require_separate_pvs = 0

# Configuration option allocation/thin_pool_zero.
# Thin pool data chunks are zeroed before they are first used.
# Zeroing with a larger thin pool chunk size reduces performance.
# This configuration option has an automatic default value.
# thin_pool_zero = 1

# Configuration option allocation/thin_pool_discards.
# The discards behaviour of thin pool volumes.
#
# Accepted values:
#   ignore
#   nopassdown
#   passdown
#
# This configuration option has an automatic default value.
# thin_pool_discards = "passdown"

# Configuration option allocation/thin_pool_chunk_size_policy.
# The chunk size calculation policy for thin pool volumes.
#
# Accepted values:
#   generic
#     If thin_pool_chunk_size is defined, use it. Otherwise, calculate
#     the chunk size based on estimation and device hints exposed in
#     sysfs - the minimum_io_size. The chunk size is always at least
#     64KiB.
#   performance
#     If thin_pool_chunk_size is defined, use it. Otherwise, calculate
#     the chunk size for performance based on device hints exposed in
#     sysfs - the optimal_io_size. The chunk size is always at least
#     512KiB.
#
# This configuration option has an automatic default value.
# thin_pool_chunk_size_policy = "generic"
```

```
 # Configuration option allocation/thin_pool_chunk_size.
 # The minimal chunk size in KiB for thin pool volumes.
 # Larger chunk sizes may improve performance for plain thin volumes,
 # however using them for snapshot volumes is less efficient, as it
 # consumes more space and takes extra time for copying. When unset,
 # lvm tries to estimate chunk size starting from 64KiB. Supported
 # values are in the range 64KiB to 1GiB.
 # This configuration option does not have a default value defined.

 # Configuration option allocation/physical_extent_size.
 # Default physical extent size in KiB to use for new VGs.
 # This configuration option has an automatic default value.
 # physical_extent_size = 4096
}

# Configuration section log.
# How LVM log information is reported.
log {

 # Configuration option log/report_command_log.
 # Enable or disable LVM log reporting.
 # If enabled, LVM will collect a log of operations, messages,
 # per-object return codes with object identification and associated
 # error numbers (errnos) during LVM command processing. Then the
 # log is either reported solely or in addition to any existing
 # reports, depending on LVM command used. If it is a reporting command
 # (e.g. pvs, vgs, lvs, lvm fullreport), then the log is reported in
 # addition to any existing reports. Otherwise, there's only log report
 # on output. For all applicable LVM commands, you can request that
 # the output has only log report by using --logonly command line
 # option. Use log/command_log_cols and log/command_log_sort settings
 # to define fields to display and sort fields for the log report.
 # You can also use log/command_log_selection to define selection
 # criteria used each time the log is reported.
 # This configuration option has an automatic default value.
 # report_command_log = 0

 # Configuration option log/command_log_sort.
 # List of columns to sort by when reporting command log.
 # See <lvm command> --logonly --configreport log -o help
 # for the list of possible fields.
 # This configuration option has an automatic default value.
 # command_log_sort = "log_seq_num"

 # Configuration option log/command_log_cols.
 # List of columns to report when reporting command log.
 # See <lvm command> --logonly --configreport log -o help
 # for the list of possible fields.
 # This configuration option has an automatic default value.
 # command_log_cols =
"log_seq_num,log_type,log_context,log_object_type,log_object_name,log_object_id,log_object_group,lo
g_object_group_id,log_message,log_errno,log_ret_code"

 # Configuration option log/command_log_selection.
 # Selection criteria used when reporting command log.
 # You can define selection criteria that are applied each
```

```
# time log is reported. This way, it is possible to control the
# amount of log that is displayed on output and you can select
# only parts of the log that are important for you. To define
# selection criteria, use fields from log report. See also
# <lvm command> --logonly --configreport log -S help for the
# list of possible fields and selection operators. You can also
# define selection criteria for log report on command line directly
# using <lvm command> --configreport log -S <selection criteria>
# which has precedence over log/command_log_selection setting.
# For more information about selection criteria in general, see
# lvm(8) man page.
# This configuration option has an automatic default value.
# command_log_selection = "!(log_type=status && message=success)"

# Configuration option log/verbose.
# Controls the messages sent to stdout or stderr.
verbose = 0

# Configuration option log/silent.
# Suppress all non-essential messages from stdout.
# This has the same effect as -qq. When enabled, the following commands
# still produce output: dumpconfig, lvdisplay, lvmdiskscan, lvs, pvck,
# pvdisplay, pvs, version, vgcfgrestore -l, vgdisplay, vgs.
# Non-essential messages are shifted from log level 4 to log level 5
# for syslog and lvm2_log_fn purposes.
# Any 'yes' or 'no' questions not overridden by other arguments are
# suppressed and default to 'no'.
silent = 0

# Configuration option log/syslog.
# Send log messages through syslog.
syslog = 1

# Configuration option log/file.
# Write error and debug log messages to a file specified here.
# This configuration option does not have a default value defined.

# Configuration option log/overwrite.
# Overwrite the log file each time the program is run.
overwrite = 0

# Configuration option log/level.
# The level of log messages that are sent to the log file or syslog.
# There are 6 syslog-like log levels currently in use: 2 to 7 inclusive.
# 7 is the most verbose (LOG_DEBUG).
level = 0

# Configuration option log/indent.
# Indent messages according to their severity.
indent = 1

# Configuration option log/command_names.
# Display the command name on each line of output.
command_names = 0

# Configuration option log/prefix.
```

```
    # A prefix to use before the log message text.
    # (After the command name, if selected).
    # Two spaces allows you to see/grep the severity of each message.
    # To make the messages look similar to the original LVM tools use:
    # indent = 0, command_names = 1, prefix = " -- "
    prefix = "  "

    # Configuration option log/activation.
    # Log messages during activation.
    # Don't use this in low memory situations (can deadlock).
    activation = 0

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

# Configuration section backup.
# How LVM metadata is backed up and archived.
# In LVM, a 'backup' is a copy of the metadata for the current system,
# and an 'archive' contains old metadata configurations. They are
# stored in a human readable text format.
backup {

    # Configuration option backup/backup.
    # Maintain a backup of the current metadata configuration.
    # Think very hard before turning this off!
    backup = 1

    # Configuration option backup/backup_dir.
    # Location of the metadata backup files.
    # Remember to back up this directory regularly!
    backup_dir = "/etc/lvm/backup"

    # Configuration option backup/archive.
    # Maintain an archive of old metadata configurations.
    # Think very hard before turning this off.
    archive = 1

    # Configuration option backup/archive_dir.
    # Location of the metdata archive files.
    # Remember to back up this directory regularly!
    archive_dir = "/etc/lvm/archive"

    # Configuration option backup/retain_min.
    # Minimum number of archives to keep.
    retain_min = 10

    # Configuration option backup/retain_days.
    # Minimum number of days to keep archive files.
    retain_days = 30
```

```
}

# Configuration section shell.
# Settings for running LVM in shell (readline) mode.
shell {

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

# Configuration section global.
# Miscellaneous global LVM settings.
global {

 # Configuration option global/umask.
 # The file creation mask for any files and directories created.
 # Interpreted as octal if the first digit is zero.
 umask = 077

 # Configuration option global/test.
 # No on-disk metadata changes will be made in test mode.
 # Equivalent to having the -t option on every command.
 test = 0

 # Configuration option global/units.
 # Default value for --units argument.
 units = "h"

 # Configuration option global/si_unit_consistency.
 # Distinguish between powers of 1024 and 1000 bytes.
 # The LVM commands distinguish between powers of 1024 bytes,
 # e.g. KiB, MiB, GiB, and powers of 1000 bytes, e.g. KB, MB, GB.
 # If scripts depend on the old behaviour, disable this setting
 # temporarily until they are updated.
 si_unit_consistency = 1

 # Configuration option global/suffix.
 # Display unit suffix for sizes.
 # This setting has no effect if the units are in human-readable form
 # (global/units = "h") in which case the suffix is always displayed.
 suffix = 1

 # Configuration option global/activation.
 # Enable/disable communication with the kernel device-mapper.
 # Disable to use the tools to manipulate LVM metadata without
 # activating any logical volumes. If the device-mapper driver
 # is not present in the kernel, disabling this should suppress
 # the error messages.
 activation = 1

 # Configuration option global/fallback_to_lvm1.
 # Try running LVM1 tools if LVM cannot communicate with DM.
 # This option only applies to 2.4 kernels and is provided to help
 # switch between device-mapper kernels and LVM1 kernels. The LVM1
 # tools need to be installed with .lvm1 suffices, e.g. vgscan.lvm1.
```

```
# They will stop working once the lvm2 on-disk metadata format is used.
# This configuration option has an automatic default value.
# fallback_to_lvm1 = 1

# Configuration option global/format.
# The default metadata format that commands should use.
# The -M 1|2 option overrides this setting.
#
# Accepted values:
#   lvm1
#   lvm2
#
# This configuration option has an automatic default value.
# format = "lvm2"

# Configuration option global/format_libraries.
# Shared libraries that process different metadata formats.
# If support for LVM1 metadata was compiled as a shared library use
# format_libraries = "liblvm2format1.so"
# This configuration option does not have a default value defined.

# Configuration option global/segment_libraries.
# This configuration option does not have a default value defined.

# Configuration option global/proc.
# Location of proc filesystem.
# This configuration option is advanced.
proc = "/proc"

# Configuration option global/etc.
# Location of /etc system configuration directory.
etc = "/etc"

# Configuration option global/locking_type.
# Type of locking to use.
#
# Accepted values:
#   0
#     Turns off locking. Warning: this risks metadata corruption if
#     commands run concurrently.
#   1
#     LVM uses local file-based locking, the standard mode.
#   2
#     LVM uses the external shared library locking_library.
#   3
#     LVM uses built-in clustered locking with clvmd.
#     This is incompatible with lvmetad. If use_lvmetad is enabled,
#     LVM prints a warning and disables lvmetad use.
#   4
#     LVM uses read-only locking which forbids any operations that
#     might change metadata.
#   5
#     Offers dummy locking for tools that do not need any locks.
#     You should not need to set this directly; the tools will select
#     when to use it instead of the configured locking_type.
#     Do not use lvmetad or the kernel device-mapper driver with this
```

```
#     locking type. It is used by the --readonly option that offers
#     read-only access to Volume Group metadata that cannot be locked
#     safely because it belongs to an inaccessible domain and might be
#     in use, for example a virtual machine image or a disk that is
#     shared by a clustered machine.
#
    locking_type = 3

# Configuration option global/wait_for_locks.
# When disabled, fail if a lock request would block.
wait_for_locks = 1

# Configuration option global/fallback_to_clustered_locking.
# Attempt to use built-in cluster locking if locking_type 2 fails.
# If using external locking (type 2) and initialisation fails, with
# this enabled, an attempt will be made to use the built-in clustered
# locking. Disable this if using a customised locking_library.
fallback_to_clustered_locking = 1

# Configuration option global/fallback_to_local_locking.
# Use locking_type 1 (local) if locking_type 2 or 3 fail.
# If an attempt to initialise type 2 or type 3 locking failed, perhaps
# because cluster components such as clvmd are not running, with this
# enabled, an attempt will be made to use local file-based locking
# (type 1). If this succeeds, only commands against local VGs will
# proceed. VGs marked as clustered will be ignored.
fallback_to_local_locking = 1

# Configuration option global/locking_dir.
# Directory to use for LVM command file locks.
# Local non-LV directory that holds file-based locks while commands are
# in progress. A directory like /tmp that may get wiped on reboot is OK.
locking_dir = "/run/lock/lvm"

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

# Configuration option global/library_dir.
# Search this directory first for shared libraries.
# This configuration option does not have a default value defined.

# Configuration option global/locking_library.
# The external locking library to use for locking_type 2.
# This configuration option has an automatic default value.
# locking_library = "liblvm2clusterlock.so"

# Configuration option global/abort_on_internal_errors.
# Abort a command that encounters an internal error.
# Treat any internal errors as fatal errors, aborting the process that
```

```
# encountered the internal error. Please only enable for debugging.
abort_on_internal_errors = 0

# Configuration option global/detect_internal_vg_cache_corruption.
# Internal verification of VG structures.
# Check if CRC matches when a parsed VG is used multiple times. This
# is useful to catch unexpected changes to cached VG structures.
# Please only enable for debugging.
detect_internal_vg_cache_corruption = 0

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

# Configuration option global/mirror_segtype_default.
# The segment type used by the short mirroring option -m.
# The --type mirror|raid1 option overrides this setting.
#
# Accepted values:
#   mirror
#     The original RAID1 implementation from LVM/DM. It is
#     characterized by a flexible log solution (core, disk, mirrored),
#     and by the necessity to block I/O while handling a failure.
#     There is an inherent race in the dmeventd failure handling logic
#     with snapshots of devices using this type of RAID1 that in the
#     worst case could cause a deadlock. (Also see
#     devices/ignore_lvm_mirrors.)
#   raid1
#     This is a newer RAID1 implementation using the MD RAID1
#     personality through device-mapper. It is characterized by a
#     lack of log options. (A log is always allocated for every
#     device and they are placed on the same device as the image,
#     so no separate devices are required.) This mirror
#     implementation does not require I/O to be blocked while
#     handling a failure. This mirror implementation is not
#     cluster-aware and cannot be used in a shared (active/active)
#     fashion in a cluster.
#
mirror_segtype_default = "raid1"

# Configuration option global/raid10_segtype_default.
# The segment type used by the -i -m combination.
# The --type raid10|mirror option overrides this setting.
# The --stripes/-i and --mirrors/-m options can both be specified
# during the creation of a logical volume to use both striping and
# mirroring for the LV. There are two different implementations.
#
# Accepted values:
#   raid10
#     LVM uses MD's RAID10 personality through DM. This is the
#     preferred option.
#   mirror
```

```
#     LVM layers the 'mirror' and 'stripe' segment types. The layering
#     is done by creating a mirror LV on top of striped sub-LVs,
#     effectively creating a RAID 0+1 array. The layering is suboptimal
#     in terms of providing redundancy and performance.
#
raid10_segtype_default = "raid10"

# Configuration option global/sparse_segtype_default.
# The segment type used by the -V -L combination.
# The --type snapshot|thin option overrides this setting.
# The combination of -V and -L options creates a sparse LV. There are
# two different implementations.
#
# Accepted values:
#   snapshot
#     The original snapshot implementation from LVM/DM. It uses an old
#     snapshot that mixes data and metadata within a single COW
#     storage volume and performs poorly when the size of stored data
#     passes hundreds of MB.
#   thin
#     A newer implementation that uses thin provisioning. It has a
#     bigger minimal chunk size (64KiB) and uses a separate volume for
#     metadata. It has better performance, especially when more data
#     is used. It also supports full snapshots.
#
sparse_segtype_default = "thin"

# Configuration option global/lvdisplay_shows_full_device_path.
# Enable this to reinstate the previous lvdisplay name format.
# The default format for displaying LV names in lvdisplay was changed
# in version 2.02.89 to show the LV name and path separately.
# Previously this was always shown as /dev/vgname/lvname even when that
# was never a valid path in the /dev filesystem.
# This configuration option has an automatic default value.
# lvdisplay_shows_full_device_path = 0

# Configuration option global/use_lvmetad.
# Use lvmetad to cache metadata and reduce disk scanning.
# When enabled (and running), lvmetad provides LVM commands with VG
# metadata and PV state. LVM commands then avoid reading this
# information from disks which can be slow. When disabled (or not
# running), LVM commands fall back to scanning disks to obtain VG
# metadata. lvmetad is kept updated via udev rules which must be set
# up for LVM to work correctly. (The udev rules should be installed
# by default.) Without a proper udev setup, changes in the system's
# block device configuration will be unknown to LVM, and ignored
# until a manual 'pvscan --cache' is run. If lvmetad was running
# while use_lvmetad was disabled, it must be stopped, use_lvmetad
# enabled, and then started. When using lvmetad, LV activation is
# switched to an automatic, event-based mode. In this mode, LVs are
# activated based on incoming udev events that inform lvmetad when
# PVs appear on the system. When a VG is complete (all PVs present),
# it is auto-activated. The auto_activation_volume_list setting
# controls which LVs are auto-activated (all by default.)
# When lvmetad is updated (automatically by udev events, or directly
# by pvscan --cache), devices/filter is ignored and all devices are
```

```
# scanned by default. lvmetad always keeps unfiltered information
# which is provided to LVM commands. Each LVM command then filters
# based on devices/filter. This does not apply to other, non-regexp,
# filtering settings: component filters such as multipath and MD
# are checked during pvscan --cache. To filter a device and prevent
# scanning from the LVM system entirely, including lvmetad, use
# devices/global_filter.
   use_lvmetad = 0

# Configuration option global/lvmetad_update_wait_time.
# The number of seconds a command will wait for lvmetad update to finish.
# After waiting for this period, a command will not use lvmetad, and
# will revert to disk scanning.
# This configuration option has an automatic default value.
# lvmetad_update_wait_time = 10

# Configuration option global/use_lvmlockd.
# Use lvmlockd for locking among hosts using LVM on shared storage.
# Applicable only if LVM is compiled with lockd support in which
# case there is also lvmlockd(8) man page available for more
# information.
use_lvmlockd = 0

# Configuration option global/lvmlockd_lock_retries.
# Retry lvmlockd lock requests this many times.
# Applicable only if LVM is compiled with lockd support
# This configuration option has an automatic default value.
# lvmlockd_lock_retries = 3

# Configuration option global/sanlock_lv_extend.
# Size in MiB to extend the internal LV holding sanlock locks.
# The internal LV holds locks for each LV in the VG, and after enough
# LVs have been created, the internal LV needs to be extended. lvcreate
# will automatically extend the internal LV when needed by the amount
# specified here. Setting this to 0 disables the automatic extension
# and can cause lvcreate to fail. Applicable only if LVM is compiled
# with lockd support
# This configuration option has an automatic default value.
# sanlock_lv_extend = 256

# Configuration option global/thin_check_executable.
# The full path to the thin_check command.
# LVM uses this command to check that a thin metadata device is in a
# usable state. When a thin pool is activated and after it is
# deactivated, this command is run. Activation will only proceed if
# the command has an exit status of 0. Set to "" to skip this check.
# (Not recommended.) Also see thin_check_options.
# (See package device-mapper-persistent-data or thin-provisioning-tools)
# This configuration option has an automatic default value.
# thin_check_executable = "/usr/sbin/thin_check"

# Configuration option global/thin_dump_executable.
# The full path to the thin_dump command.
# LVM uses this command to dump thin pool metadata.
# (See package device-mapper-persistent-data or thin-provisioning-tools)
# This configuration option has an automatic default value.
```

```
# thin_dump_executable = "/usr/sbin/thin_dump"

# Configuration option global/thin_repair_executable.
# The full path to the thin_repair command.
# LVM uses this command to repair a thin metadata device if it is in
# an unusable state. Also see thin_repair_options.
# (See package device-mapper-persistent-data or thin-provisioning-tools)
# This configuration option has an automatic default value.
# thin_repair_executable = "/usr/sbin/thin_repair"

# Configuration option global/thin_check_options.
# List of options passed to the thin_check command.
# With thin_check version 2.1 or newer you can add the option
# --ignore-non-fatal-errors to let it pass through ignorable errors
# and fix them later. With thin_check version 3.2 or newer you should
# include the option --clear-needs-check-flag.
# This configuration option has an automatic default value.
# thin_check_options = [ "-q", "--clear-needs-check-flag" ]

# Configuration option global/thin_repair_options.
# List of options passed to the thin_repair command.
# This configuration option has an automatic default value.
# thin_repair_options = [ "" ]

# Configuration option global/thin_disabled_features.
# Features to not use in the thin driver.
# This can be helpful for testing, or to avoid using a feature that is
# causing problems. Features include: block_size, discards,
# discards_non_power_2, external_origin, metadata_resize,
# external_origin_extend, error_if_no_space.
#
# Example
# thin_disabled_features = [ "discards", "block_size" ]
#
# This configuration option does not have a default value defined.

# Configuration option global/cache_disabled_features.
# Features to not use in the cache driver.
# This can be helpful for testing, or to avoid using a feature that is
# causing problems. Features include: policy_mq, policy_smq.
#
# Example
# cache_disabled_features = [ "policy_smq" ]
#
# This configuration option does not have a default value defined.

# Configuration option global/cache_check_executable.
# The full path to the cache_check command.
# LVM uses this command to check that a cache metadata device is in a
# usable state. When a cached LV is activated and after it is
# deactivated, this command is run. Activation will only proceed if the
# command has an exit status of 0. Set to "" to skip this check.
# (Not recommended.) Also see cache_check_options.
# (See package device-mapper-persistent-data or thin-provisioning-tools)
# This configuration option has an automatic default value.
# cache_check_executable = "/usr/sbin/cache_check"
```

```
# Configuration option global/cache_dump_executable.
# The full path to the cache_dump command.
# LVM uses this command to dump cache pool metadata.
# (See package device-mapper-persistent-data or thin-provisioning-tools)
# This configuration option has an automatic default value.
# cache_dump_executable = "/usr/sbin/cache_dump"

# Configuration option global/cache_repair_executable.
# The full path to the cache_repair command.
# LVM uses this command to repair a cache metadata device if it is in
# an unusable state. Also see cache_repair_options.
# (See package device-mapper-persistent-data or thin-provisioning-tools)
# This configuration option has an automatic default value.
# cache_repair_executable = "/usr/sbin/cache_repair"

# Configuration option global/cache_check_options.
# List of options passed to the cache_check command.
# With cache_check version 5.0 or newer you should include the option
# --clear-needs-check-flag.
# This configuration option has an automatic default value.
# cache_check_options = [ "-q", "--clear-needs-check-flag" ]

# Configuration option global/cache_repair_options.
# List of options passed to the cache_repair command.
# This configuration option has an automatic default value.
# cache_repair_options = [ "" ]

# Configuration option global/system_id_source.
# The method LVM uses to set the local system ID.
# Volume Groups can also be given a system ID (by vgcreate, vgchange,
# or vgimport.) A VG on shared storage devices is accessible only to
# the host with a matching system ID. See 'man lvmsystemid' for
# information on limitations and correct usage.
#
# Accepted values:
#   none
#     The host has no system ID.
#   lvmlocal
#     Obtain the system ID from the system_id setting in the 'local'
#     section of an lvm configuration file, e.g. lvmlocal.conf.
#   uname
#     Set the system ID from the hostname (uname) of the system.
#     System IDs beginning localhost are not permitted.
#   machineid
#     Use the contents of the machine-id file to set the system ID.
#     Some systems create this file at installation time.
#     See 'man machine-id' and global/etc.
#   file
#     Use the contents of another file (system_id_file) to set the
#     system ID.
#
system_id_source = "none"

# Configuration option global/system_id_file.
# The full path to the file containing a system ID.
```

```
    # This is used when system_id_source is set to 'file'.
    # Comments starting with the character # are ignored.
    # This configuration option does not have a default value defined.

    # Configuration option global/use_lvmpolld.
    # Use lvmpolld to supervise long running LVM commands.
    # When enabled, control of long running LVM commands is transferred
    # from the original LVM command to the lvmpolld daemon. This allows
    # the operation to continue independent of the original LVM command.
    # After lvmpolld takes over, the LVM command displays the progress
    # of the ongoing operation. lvmpolld itself runs LVM commands to
    # manage the progress of ongoing operations. lvmpolld can be used as
    # a native systemd service, which allows it to be started on demand,
    # and to use its own control group. When this option is disabled, LVM
    # commands will supervise long running operations by forking themselves.
    # Applicable only if LVM is compiled with lvmpolld support.
    use_lvmpolld = 1

    # Configuration option global/notify_dbus.
    # Enable D-Bus notification from LVM commands.
    # When enabled, an LVM command that changes PVs, changes VG metadata,
    # or changes the activation state of an LV will send a notification.
    notify_dbus = 1
}

# Configuration section activation.
activation {

    # Configuration option activation/checks.
    # Perform internal checks of libdevmapper operations.
    # Useful for debugging problems with activation. Some of the checks may
    # be expensive, so it's best to use this only when there seems to be a
    # problem.
    checks = 0

    # Configuration option activation/udev_sync.
    # Use udev notifications to synchronize udev and LVM.
    # The --nodevsync option overrides this setting.
    # When disabled, LVM commands will not wait for notifications from
    # udev, but continue irrespective of any possible udev processing in
    # the background. Only use this if udev is not running or has rules
    # that ignore the devices LVM creates. If enabled when udev is not
    # running, and LVM processes are waiting for udev, run the command
    # 'dmsetup udevcomplete_all' to wake them up.
    udev_sync = 1

    # Configuration option activation/udev_rules.
    # Use udev rules to manage LV device nodes and symlinks.
    # When disabled, LVM will manage the device nodes and symlinks for
    # active LVs itself. Manual intervention may be required if this
    # setting is changed while LVs are active.
    udev_rules = 1

    # Configuration option activation/verify_udev_operations.
    # Use extra checks in LVM to verify udev operations.
    # This enables additional checks (and if necessary, repairs) on entries
```

```
# in the device directory after udev has completed processing its
# events. Useful for diagnosing problems with LVM/udev interactions.
verify_udev_operations = 0

# Configuration option activation/retry_deactivation.
# Retry failed LV deactivation.
# If LV deactivation fails, LVM will retry for a few seconds before
# failing. This may happen because a process run from a quick udev rule
# temporarily opened the device.
retry_deactivation = 1

# Configuration option activation/missing_stripe_filler.
# Method to fill missing stripes when activating an incomplete LV.
# Using 'error' will make inaccessible parts of the device return I/O
# errors on access. You can instead use a device path, in which case,
# that device will be used in place of missing stripes. Using anything
# other than 'error' with mirrored or snapshotted volumes is likely to
# result in data corruption.
# This configuration option is advanced.
missing_stripe_filler = "error"

# Configuration option activation/use_linear_target.
# Use the linear target to optimize single stripe LVs.
# When disabled, the striped target is used. The linear target is an
# optimised version of the striped target that only handles a single
# stripe.
use_linear_target = 1

# Configuration option activation/reserved_stack.
# Stack size in KiB to reserve for use while devices are suspended.
# Insufficent reserve risks I/O deadlock during device suspension.
reserved_stack = 64

# Configuration option activation/reserved_memory.
# Memory size in KiB to reserve for use while devices are suspended.
# Insufficent reserve risks I/O deadlock during device suspension.
reserved_memory = 8192

# Configuration option activation/process_priority.
# Nice value used while devices are suspended.
# Use a high priority so that LVs are suspended
# for the shortest possible time.
process_priority = -18

# Configuration option activation/volume_list.
# Only LVs selected by this list are activated.
# If this list is defined, an LV is only activated if it matches an
# entry in this list. If this list is undefined, it imposes no limits
# on LV activation (all are allowed).
#
# Accepted values:
#   vgname
#     The VG name is matched exactly and selects all LVs in the VG.
#   vgname/lvname
#     The VG name and LV name are matched exactly and selects the LV.
#   @tag
```

```
#     Selects an LV if the specified tag matches a tag set on the LV
#     or VG.
#   @*
#     Selects an LV if a tag defined on the host is also set on the LV
#     or VG. See tags/hosttags. If any host tags exist but volume_list
#     is not defined, a default single-entry list containing '@*'
#     is assumed.
#
# Example
# volume_list = [ "vg1", "vg2/lvol1", "@tag1", "@*" ]
#
# This configuration option does not have a default value defined.

# Configuration option activation/auto_activation_volume_list.
# Only LVs selected by this list are auto-activated.
# This list works like volume_list, but it is used only by
# auto-activation commands. It does not apply to direct activation
# commands. If this list is defined, an LV is only auto-activated
# if it matches an entry in this list. If this list is undefined, it
# imposes no limits on LV auto-activation (all are allowed.) If this
# list is defined and empty, i.e. "[]", then no LVs are selected for
# auto-activation. An LV that is selected by this list for
# auto-activation, must also be selected by volume_list (if defined)
# before it is activated. Auto-activation is an activation command that
# includes the 'a' argument: --activate ay or -a ay. The 'a' (auto)
# argument for auto-activation is meant to be used by activation
# commands that are run automatically by the system, as opposed to LVM
# commands run directly by a user. A user may also use the 'a' flag
# directly to perform auto-activation. Also see pvscan(8) for more
# information about auto-activation.
#
# Accepted values:
#   vgname
#     The VG name is matched exactly and selects all LVs in the VG.
#   vgname/lvname
#     The VG name and LV name are matched exactly and selects the LV.
#   @tag
#     Selects an LV if the specified tag matches a tag set on the LV
#     or VG.
#   @*
#     Selects an LV if a tag defined on the host is also set on the LV
#     or VG. See tags/hosttags. If any host tags exist but volume_list
#     is not defined, a default single-entry list containing '@*'
#     is assumed.
#
# Example
# auto_activation_volume_list = [ "vg1", "vg2/lvol1", "@tag1", "@*" ]
#
# This configuration option does not have a default value defined.

# Configuration option activation/read_only_volume_list.
# LVs in this list are activated in read-only mode.
# If this list is defined, each LV that is to be activated is checked
# against this list, and if it matches, it is activated in read-only
# mode. This overrides the permission setting stored in the metadata,
# e.g. from --permission rw.
```

```
#
# Accepted values:
#   vgname
#     The VG name is matched exactly and selects all LVs in the VG.
#   vgname/lvname
#     The VG name and LV name are matched exactly and selects the LV.
#   @tag
#     Selects an LV if the specified tag matches a tag set on the LV
#     or VG.
#   @*
#     Selects an LV if a tag defined on the host is also set on the LV
#     or VG. See tags/hosttags. If any host tags exist but volume_list
#     is not defined, a default single-entry list containing '@*'
#     is assumed.
#
# Example
# read_only_volume_list = [ "vg1", "vg2/lvol1", "@tag1", "@*" ]
#
# This configuration option does not have a default value defined.

# Configuration option activation/raid_region_size.
# Size in KiB of each raid or mirror synchronization region.
# For raid or mirror segment types, this is the amount of data that is
# copied at once when initializing, or moved at once by pvmove.
raid_region_size = 512

# Configuration option activation/error_when_full.
# Return errors if a thin pool runs out of space.
# The --errorwhenfull option overrides this setting.
# When enabled, writes to thin LVs immediately return an error if the
# thin pool is out of data space. When disabled, writes to thin LVs
# are queued if the thin pool is out of space, and processed when the
# thin pool data space is extended. New thin pools are assigned the
# behavior defined here.
# This configuration option has an automatic default value.
# error_when_full = 0

# Configuration option activation/readahead.
# Setting to use when there is no readahead setting in metadata.
#
# Accepted values:
#   none
#     Disable readahead.
#   auto
#     Use default value chosen by kernel.
#
readahead = "auto"

# Configuration option activation/raid_fault_policy.
# Defines how a device failure in a RAID LV is handled.
# This includes LVs that have the following segment types:
# raid1, raid4, raid5*, and raid6*.
# If a device in the LV fails, the policy determines the steps
# performed by dmeventd automatically, and the steps perfomed by the
# manual command lvconvert --repair --use-policies.
# Automatic handling requires dmeventd to be monitoring the LV.
```

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

# Configuration option activation/mirror_image_fault_policy.
# Defines how a device failure in a 'mirror' LV is handled.
# An LV with the 'mirror' segment type is composed of mirror images
# (copies) and a mirror log. A disk log ensures that a mirror LV does
# not need to be re-synced (all copies made the same) every time a
# machine reboots or crashes. If a device in the LV fails, this policy
# determines the steps perfomed by dmeventd automatically, and the steps
# performed by the manual command lvconvert --repair --use-policies.
# Automatic handling requires dmeventd to be monitoring the LV.
#
# Accepted values:
#   remove
#     Simply remove the faulty device and run without it. If the log
#     device fails, the mirror would convert to using an in-memory log.
#     This means the mirror will not remember its sync status across
#     crashes/reboots and the entire mirror will be re-synced. If a
#     mirror image fails, the mirror will convert to a non-mirrored
#     device if there is only one remaining good copy.
#   allocate
#     Remove the faulty device and try to allocate space on a new
#     device to be a replacement for the failed device. Using this
#     policy for the log is fast and maintains the ability to remember
#     sync state through crashes/reboots. Using this policy for a
#     mirror device is slow, as it requires the mirror to resynchronize
#     the devices, but it will preserve the mirror characteristic of
#     the device. This policy acts like 'remove' if no suitable device
#     and space can be allocated for the replacement.
#   allocate_anywhere
#     Not yet implemented. Useful to place the log device temporarily
#     on the same physical volume as one of the mirror images. This
#     policy is not recommended for mirror devices since it would break
#     the redundant nature of the mirror. This policy acts like
#     'remove' if no suitable device and space can be allocated for the
#     replacement.
#
mirror_image_fault_policy = "remove"

# Configuration option activation/mirror_log_fault_policy.
# Defines how a device failure in a 'mirror' log LV is handled.
# The mirror_image_fault_policy description for mirrored LVs also
# applies to mirrored log LVs.
mirror_log_fault_policy = "allocate"
```

```
# Configuration option activation/snapshot_autoextend_threshold.
# Auto-extend a snapshot when its usage exceeds this percent.
# Setting this to 100 disables automatic extension.
# The minimum value is 50 (a smaller value is treated as 50.)
# Also see snapshot_autoextend_percent.
# Automatic extension requires dmeventd to be monitoring the LV.
#
# Example
# Using 70% autoextend threshold and 20% autoextend size, when a 1G
# snapshot exceeds 700M, it is extended to 1.2G, and when it exceeds
# 840M, it is extended to 1.44G:
# snapshot_autoextend_threshold = 70
#
snapshot_autoextend_threshold = 100

# Configuration option activation/snapshot_autoextend_percent.
# Auto-extending a snapshot adds this percent extra space.
# The amount of additional space added to a snapshot is this
# percent of its current size.
#
# Example
# Using 70% autoextend threshold and 20% autoextend size, when a 1G
# snapshot exceeds 700M, it is extended to 1.2G, and when it exceeds
# 840M, it is extended to 1.44G:
# snapshot_autoextend_percent = 20
#
snapshot_autoextend_percent = 20

# Configuration option activation/thin_pool_autoextend_threshold.
# Auto-extend a thin pool when its usage exceeds this percent.
# Setting this to 100 disables automatic extension.
# The minimum value is 50 (a smaller value is treated as 50.)
# Also see thin_pool_autoextend_percent.
# Automatic extension requires dmeventd to be monitoring the LV.
#
# Example
# Using 70% autoextend threshold and 20% autoextend size, when a 1G
# thin pool exceeds 700M, it is extended to 1.2G, and when it exceeds
# 840M, it is extended to 1.44G:
# thin_pool_autoextend_threshold = 70
#
thin_pool_autoextend_threshold = 100

# Configuration option activation/thin_pool_autoextend_percent.
# Auto-extending a thin pool adds this percent extra space.
# The amount of additional space added to a thin pool is this
# percent of its current size.
#
# Example
# Using 70% autoextend threshold and 20% autoextend size, when a 1G
# thin pool exceeds 700M, it is extended to 1.2G, and when it exceeds
# 840M, it is extended to 1.44G:
# thin_pool_autoextend_percent = 20
#
thin_pool_autoextend_percent = 20
```

```
# Configuration option activation/mlock_filter.
# Do not mlock these memory areas.
# While activating devices, I/O to devices being (re)configured is
# suspended. As a precaution against deadlocks, LVM pins memory it is
# using so it is not paged out, and will not require I/O to reread.
# Groups of pages that are known not to be accessed during activation
# do not need to be pinned into memory. Each string listed in this
# setting is compared against each line in /proc/self/maps, and the
# pages corresponding to lines that match are not pinned. On some
# systems, locale-archive was found to make up over 80% of the memory
# used by the process.
#
# Example
# mlock_filter = [ "locale/locale-archive", "gconv/gconv-modules.cache" ]
#
# This configuration option is advanced.
# This configuration option does not have a default value defined.

# Configuration option activation/use_mlockall.
# Use the old behavior of mlockall to pin all memory.
# Prior to version 2.02.62, LVM used mlockall() to pin the whole
# process's memory while activating devices.
use_mlockall = 0

# Configuration option activation/monitoring.
# Monitor LVs that are activated.
# The --ignoremonitoring option overrides this setting.
# When enabled, LVM will ask dmeventd to monitor activated LVs.
monitoring = 1

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

# Configuration option activation/auto_set_activation_skip.
# Set the activation skip flag on new thin snapshot LVs.
# The --setactivationskip option overrides this setting.
# An LV can have a persistent 'activation skip' flag. The flag causes
# the LV to be skipped during normal activation. The lvchange/vgchange
# -K option is required to activate LVs that have the activation skip
# flag set. When this setting is enabled, the activation skip flag is
# set on new thin snapshot LVs.
# This configuration option has an automatic default value.
# auto_set_activation_skip = 1

# Configuration option activation/activation_mode.
# How LVs with missing devices are activated.
# The --activationmode option overrides this setting.
#
# Accepted values:
```

```
#   complete
#     Only allow activation of an LV if all of the Physical Volumes it
#     uses are present. Other PVs in the Volume Group may be missing.
#   degraded
#     Like complete, but additionally RAID LVs of segment type raid1,
#     raid4, raid5, radid6 and raid10 will be activated if there is no
#     data loss, i.e. they have sufficient redundancy to present the
#     entire addressable range of the Logical Volume.
#   partial
#     Allows the activation of any LV even if a missing or failed PV
#     could cause data loss with a portion of the LV inaccessible.
#     This setting should not normally be used, but may sometimes
#     assist with data recovery.
#
activation_mode = "degraded"

 # Configuration option activation/lock_start_list.
 # Locking is started only for VGs selected by this list.
 # The rules are the same as those for volume_list.
 # This configuration option does not have a default value defined.

 # Configuration option activation/auto_lock_start_list.
 # Locking is auto-started only for VGs selected by this list.
 # The rules are the same as those for auto_activation_volume_list.
 # This configuration option does not have a default value defined.
}

# Configuration section metadata.
# This configuration section has an automatic default value.
# metadata {

 # Configuration option metadata/check_pv_device_sizes.
 # Check device sizes are not smaller than corresponding PV sizes.
 # If device size is less than corresponding PV size found in metadata,
 # there is always a risk of data loss. If this option is set, then LVM
 # issues a warning message each time it finds that the device size is
 # less than corresponding PV size. You should not disable this unless
 # you are absolutely sure about what you are doing!
 # This configuration option is advanced.
 # This configuration option has an automatic default value.
 # check_pv_device_sizes = 1

 # Configuration option metadata/record_lvs_history.
 # When enabled, LVM keeps history records about removed LVs in
 # metadata. The information that is recorded in metadata for
 # historical LVs is reduced when compared to original
 # information kept in metadata for live LVs. Currently, this
 # feature is supported for thin and thin snapshot LVs only.
 # This configuration option has an automatic default value.
 # record_lvs_history = 0

 # Configuration option metadata/lvs_history_retention_time.
 # Retention time in seconds after which a record about individual
 # historical logical volume is automatically destroyed.
 # A value of 0 disables this feature.
 # This configuration option has an automatic default value.
```

```
# lvs_history_retention_time = 0

# Configuration option metadata/pvmetadatacopies.
# Number of copies of metadata to store on each PV.
# The --pvmetadatacopies option overrides this setting.
#
# Accepted values:
#   2
#     Two copies of the VG metadata are stored on the PV, one at the
#     front of the PV, and one at the end.
#   1
#     One copy of VG metadata is stored at the front of the PV.
#   0
#     No copies of VG metadata are stored on the PV. This may be
#     useful for VGs containing large numbers of PVs.
#
# This configuration option is advanced.
# This configuration option has an automatic default value.
# pvmetadatacopies = 1

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

# Configuration option metadata/pvmetadatasize.
# Approximate number of sectors to use for each metadata copy.
# VGs with large numbers of PVs or LVs, or VGs containing complex LV
# structures, may need additional space for VG metadata. The metadata
# areas are treated as circular buffers, so unused space becomes filled
# with an archive of the most recent previous versions of the metadata.
# This configuration option has an automatic default value.
# pvmetadatasize = 255

# Configuration option metadata/pvmetadataignore.
# Ignore metadata areas on a new PV.
# The --metadataignore option overrides this setting.
# If metadata areas on a PV are ignored, LVM will not store metadata
# in them.
# This configuration option is advanced.
# This configuration option has an automatic default value.
# pvmetadataignore = 0

# Configuration option metadata/stripesize.
# This configuration option is advanced.
# This configuration option has an automatic default value.
# stripesize = 64
```

```
    # Configuration option metadata/dirs.
    # Directories holding live copies of text format metadata.
    # These directories must not be on logical volumes!
    # It's possible to use LVM with a couple of directories here,
    # preferably on different (non-LV) filesystems, and with no other
    # on-disk metadata (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 are absolutely sure you know what you are doing!
    # Use the supplied toolset to make changes (e.g. vgcfgrestore).
    #
    # Example
    # dirs = [ "/etc/lvm/metadata", "/mnt/disk2/lvm/metadata2" ]
    #
    # This configuration option is advanced.
    # This configuration option does not have a default value defined.
    # }

    # Configuration section report.
    # LVM report command output formatting.
    # This configuration section has an automatic default value.
    # report {

    # Configuration option report/output_format.
    # Format of LVM command's report output.
    # If there is more than one report per command, then the format
    # is applied for all reports. You can also change output format
    # directly on command line using --reportformat option which
    # has precedence over log/output_format setting.
    # Accepted values:
    #   basic
    #     Original format with columns and rows. If there is more than
    #     one report per command, each report is prefixed with report's
    #     name for identification.
    #   json
    #     JSON format.
    # This configuration option has an automatic default value.
    # output_format = "basic"

    # Configuration option report/compact_output.
    # Do not print empty values for all report fields.
    # If enabled, all fields that don't have a value set for any of the
    # rows reported are skipped and not printed. Compact output is
    # applicable only if report/buffered is enabled. If you need to
    # compact only specified fields, use compact_output=0 and define
    # report/compact_output_cols configuration setting instead.
    # This configuration option has an automatic default value.
    # compact_output = 0

    # Configuration option report/compact_output_cols.
    # Do not print empty values for specified report fields.
    # If defined, specified fields that don't have a value set for any
    # of the rows reported are skipped and not printed. Compact output
    # is applicable only if report/buffered is enabled. If you need to
    # compact all fields, use compact_output=1 instead in which case
```

```
# the compact_output_cols setting is then ignored.
# This configuration option has an automatic default value.
# compact_output_cols = ""

# Configuration option report/aligned.
# Align columns in report output.
# This configuration option has an automatic default value.
# aligned = 1

# Configuration option report/buffered.
# Buffer report output.
# When buffered reporting is used, the report's content is appended
# incrementally to include each object being reported until the report
# is flushed to output which normally happens at the end of command
# execution. Otherwise, if buffering is not used, each object is
# reported as soon as its processing is finished.
# This configuration option has an automatic default value.
# buffered = 1

# Configuration option report/headings.
# Show headings for columns on report.
# This configuration option has an automatic default value.
# headings = 1

# Configuration option report/separator.
# A separator to use on report after each field.
# This configuration option has an automatic default value.
# separator = " "

# Configuration option report/list_item_separator.
# A separator to use for list items when reported.
# This configuration option has an automatic default value.
# list_item_separator = ","

# Configuration option report/prefixes.
# Use a field name prefix for each field reported.
# This configuration option has an automatic default value.
# prefixes = 0

# Configuration option report/quoted.
# Quote field values when using field name prefixes.
# This configuration option has an automatic default value.
# quoted = 1

# Configuration option report/colums_as_rows.
# Output each column as a row.
# If set, this also implies report/prefixes=1.
# This configuration option has an automatic default value.
# colums_as_rows = 0

# Configuration option report/binary_values_as_numeric.
# Use binary values 0 or 1 instead of descriptive literal values.
# For columns that have exactly two valid values to report
# (not counting the 'unknown' value which denotes that the
# value could not be determined).
# This configuration option has an automatic default value.
```

```
# binary_values_as_numeric = 0

# Configuration option report/time_format.
# Set time format for fields reporting time values.
# Format specification is a string which may contain special character
# sequences and ordinary character sequences. Ordinary character
# sequences are copied verbatim. Each special character sequence is
# introduced by the '%' character and such sequence is then
# substituted with a value as described below.
#
# Accepted values:
#   %a
#     The abbreviated name of the day of the week according to the
#     current locale.
#   %A
#     The full name of the day of the week according to the current
#     locale.
#   %b
#     The abbreviated month name according to the current locale.
#   %B
#     The full month name according to the current locale.
#   %c
#     The preferred date and time representation for the current
#     locale (alt E)
#   %C
#     The century number (year/100) as a 2-digit integer. (alt E)
#   %d
#     The day of the month as a decimal number (range 01 to 31).
#     (alt O)
#   %D
#     Equivalent to %m/%d/%y. (For Americans only. Americans should
#     note that in other countries%d/%m/%y is rather common. This
#     means that in international context this format is ambiguous and
#     should not be used.
#   %e
#     Like %d, the day of the month as a decimal number, but a leading
#     zero is replaced by a space. (alt O)
#   %E
#     Modifier: use alternative local-dependent representation if
#     available.
#   %F
#     Equivalent to %Y-%m-%d (the ISO 8601 date format).
#   %G
#     The ISO 8601 week-based year with century as adecimal number.
#     The 4-digit year corresponding to the ISO week number (see %V).
#     This has the same format and value as %Y, except that if the
#     ISO week number belongs to the previous or next year, that year
#     is used instead.
#   %g
#     Like %G, but without century, that is, with a 2-digit year
#     (00-99).
#   %h
#     Equivalent to %b.
#   %H
#     The hour as a decimal number using a 24-hour clock
#     (range 00 to 23). (alt O)
```

```
#  %I
#    The hour as a decimal number using a 12-hour clock
#    (range 01 to 12). (alt O)
#  %j
#    The day of the year as a decimal number (range 001 to 366).
#  %k
#    The hour (24-hour clock) as a decimal number (range 0 to 23);
#    single digits are preceded by a blank. (See also %H.)
#  %l
#    The hour (12-hour clock) as a decimal number (range 1 to 12);
#    single digits are preceded by a blank. (See also %I.)
#  %m
#    The month as a decimal number (range 01 to 12). (alt O)
#  %M
#    The minute as a decimal number (range 00 to 59). (alt O)
#  %O
#    Modifier: use alternative numeric symbols.
#  %p
#    Either "AM" or "PM" according to the given time value,
#    or the corresponding strings for the current locale. Noon is
#    treated as "PM" and midnight as "AM".
#  %P
#    Like %p but in lowercase: "am" or "pm" or a corresponding
#    string for the current locale.
#  %r
#    The time in a.m. or p.m. notation. In the POSIX locale this is
#    equivalent to %I:%M:%S %p.
#  %R
#    The time in 24-hour notation (%H:%M). For a version including
#    the seconds, see %T below.
#  %s
#    The number of seconds since the Epoch,
#    1970-01-01 00:00:00 +0000 (UTC)
#  %S
#    The second as a decimal number (range 00 to 60). (The range is
#    up to 60 to allow for occasional leap seconds.) (alt O)
#  %t
#    A tab character.
#  %T
#    The time in 24-hour notation (%H:%M:%S).
#  %u
#    The day of the week as a decimal, range 1 to 7, Monday being 1.
#    See also %w. (alt O)
#  %U
#    The week number of the current year as a decimal number,
#    range 00 to 53, starting with the first Sunday as the first
#    day of week 01. See also %V and %W. (alt O)
#  %V
#    The ISO 8601 week number of the current year as a decimal number,
#    range 01 to 53, where week 1 is the first week that has at least
#    4 days in the new year. See also %U and %W. (alt O)
#  %w
#    The day of the week as a decimal, range 0 to 6, Sunday being 0.
#    See also %u. (alt O)
#  %W
#    The week number of the current year as a decimal number,
```

```
#     range 00 to 53, starting with the first Monday as the first day
#     of week 01. (alt O)
#   %x
#     The preferred date representation for the current locale without
#     the time. (alt E)
#   %X
#     The preferred time representation for the current locale without
#     the date. (alt E)
#   %y
#     The year as a decimal number without a century (range 00 to 99).
#     (alt E, alt O)
#   %Y
#     The year as a decimal number including the century. (alt E)
#   %z
#     The +hhmm or -hhmm numeric timezone (that is, the hour and minute
#     offset from UTC).
#   %Z
#     The timezone name or abbreviation.
#   %%
#     A literal '%' character.
#
# This configuration option has an automatic default value.
# time_format = "%Y-%m-%d %T %z"

# Configuration option report/devtypes_sort.
# List of columns to sort by when reporting 'lvm devtypes' command.
# See 'lvm devtypes -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# devtypes_sort = "devtype_name"

# Configuration option report/devtypes_cols.
# List of columns to report for 'lvm devtypes' command.
# See 'lvm devtypes -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# devtypes_cols = "devtype_name,devtype_max_partitions,devtype_description"

# Configuration option report/devtypes_cols_verbose.
# List of columns to report for 'lvm devtypes' command in verbose mode.
# See 'lvm devtypes -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# devtypes_cols_verbose = "devtype_name,devtype_max_partitions,devtype_description"

# Configuration option report/lvs_sort.
# List of columns to sort by when reporting 'lvs' command.
# See 'lvs -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# lvs_sort = "vg_name,lv_name"

# Configuration option report/lvs_cols.
# List of columns to report for 'lvs' command.
# See 'lvs -o help' for the list of possible fields.
# This configuration option has an automatic default value.
# lvs_cols =
"lv_name,vg_name,lv_attr,lv_size,pool_lv,origin,data_percent,metadata_percent,move_pv,mirror_log,co
py_percent,convert_lv"
```

```
    # Configuration option report/lvs_cols_verbose.
    # List of columns to report for 'lvs' command in verbose mode.
    # See 'lvs -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # lvs_cols_verbose =
    "lv_name,vg_name,seg_count,lv_attr,lv_size,lv_major,lv_minor,lv_kernel_major,lv_kernel_minor,pool_lv
    origin,data_percent,metadata_percent,move_pv,copy_percent,mirror_log,convert_lv,lv_uuid,lv_profile"


    # Configuration option report/vgs_sort.
    # List of columns to sort by when reporting 'vgs' command.
    # See 'vgs -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # vgs_sort = "vg_name"

    # Configuration option report/vgs_cols.
    # List of columns to report for 'vgs' command.
    # See 'vgs -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # vgs_cols = "vg_name,pv_count,lv_count,snap_count,vg_attr,vg_size,vg_free"

    # Configuration option report/vgs_cols_verbose.
    # List of columns to report for 'vgs' command in verbose mode.
    # See 'vgs -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # vgs_cols_verbose =
    "vg_name,vg_attr,vg_extent_size,pv_count,lv_count,snap_count,vg_size,vg_free,vg_uuid,vg_profile"

    # Configuration option report/pvs_sort.
    # List of columns to sort by when reporting 'pvs' command.
    # See 'pvs -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # pvs_sort = "pv_name"

    # Configuration option report/pvs_cols.
    # List of columns to report for 'pvs' command.
    # See 'pvs -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # pvs_cols = "pv_name,vg_name,pv_fmt,pv_attr,pv_size,pv_free"

    # Configuration option report/pvs_cols_verbose.
    # List of columns to report for 'pvs' command in verbose mode.
    # See 'pvs -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # pvs_cols_verbose = "pv_name,vg_name,pv_fmt,pv_attr,pv_size,pv_free,dev_size,pv_uuid"

    # Configuration option report/segs_sort.
    # List of columns to sort by when reporting 'lvs --segments' command.
    # See 'lvs --segments -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # segs_sort = "vg_name,lv_name,seg_start"

    # Configuration option report/segs_cols.
    # List of columns to report for 'lvs --segments' command.
    # See 'lvs --segments -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
```

```
    # segs_cols = "lv_name,vg_name,lv_attr,stripes,segtype,seg_size"

    # Configuration option report/segs_cols_verbose.
    # List of columns to report for 'lvs --segments' command in verbose mode.
    # See 'lvs --segments -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # segs_cols_verbose =
    "lv_name,vg_name,lv_attr,seg_start,seg_size,stripes,segtype,stripesize,chunksize"

    # Configuration option report/pvsegs_sort.
    # List of columns to sort by when reporting 'pvs --segments' command.
    # See 'pvs --segments -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # pvsegs_sort = "pv_name,pvseg_start"

    # Configuration option report/pvsegs_cols.
    # List of columns to sort by when reporting 'pvs --segments' command.
    # See 'pvs --segments -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # pvsegs_cols = "pv_name,vg_name,pv_fmt,pv_attr,pv_size,pv_free,pvseg_start,pvseg_size"

    # Configuration option report/pvsegs_cols_verbose.
    # List of columns to sort by when reporting 'pvs --segments' command in verbose mode.
    # See 'pvs --segments -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # pvsegs_cols_verbose =
    "pv_name,vg_name,pv_fmt,pv_attr,pv_size,pv_free,pvseg_start,pvseg_size,lv_name,seg_start_pe,segty
    pe,seg_pe_ranges"

    # Configuration option report/vgs_cols_full.
    # List of columns to report for lvm fullreport's 'vgs' subreport.
    # See 'vgs -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # vgs_cols_full = "vg_all"

    # Configuration option report/pvs_cols_full.
    # List of columns to report for lvm fullreport's 'vgs' subreport.
    # See 'pvs -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # pvs_cols_full = "pv_all"

    # Configuration option report/lvs_cols_full.
    # List of columns to report for lvm fullreport's 'lvs' subreport.
    # See 'lvs -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # lvs_cols_full = "lv_all"

    # Configuration option report/pvsegs_cols_full.
    # List of columns to report for lvm fullreport's 'pvseg' subreport.
    # See 'pvs --segments -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # pvsegs_cols_full = "pvseg_all,pv_uuid,lv_uuid"

    # Configuration option report/segs_cols_full.
    # List of columns to report for lvm fullreport's 'seg' subreport.
    # See 'lvs --segments -o help' for the list of possible fields.
```

```
    # This configuration option has an automatic default value.
    # segs_cols_full = "seg_all,lv_uuid"

    # Configuration option report/vgs_sort_full.
    # List of columns to sort by when reporting lvm fullreport's 'vgs' subreport.
    # See 'vgs -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # vgs_sort_full = "vg_name"

    # Configuration option report/pvs_sort_full.
    # List of columns to sort by when reporting lvm fullreport's 'vgs' subreport.
    # See 'pvs -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # pvs_sort_full = "pv_name"

    # Configuration option report/lvs_sort_full.
    # List of columns to sort by when reporting lvm fullreport's 'lvs' subreport.
    # See 'lvs -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # lvs_sort_full = "vg_name,lv_name"

    # Configuration option report/pvsegs_sort_full.
    # List of columns to sort by when reporting for lvm fullreport's 'pvseg' subreport.
    # See 'pvs --segments -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # pvsegs_sort_full = "pv_uuid,pvseg_start"

    # Configuration option report/segs_sort_full.
    # List of columns to sort by when reporting lvm fullreport's 'seg' subreport.
    # See 'lvs --segments -o help' for the list of possible fields.
    # This configuration option has an automatic default value.
    # segs_sort_full = "lv_uuid,seg_start"

    # Configuration option report/mark_hidden_devices.
    # Use brackets [] to mark hidden devices.
    # This configuration option has an automatic default value.
    # mark_hidden_devices = 1

    # Configuration option report/two_word_unknown_device.
    # Use the two words 'unknown device' in place of '[unknown]'.
    # This is displayed when the device for a PV is not known.
    # This configuration option has an automatic default value.
    # two_word_unknown_device = 0
    # }

    # Configuration section dmeventd.
    # Settings for the LVM event daemon.
    dmeventd {

    # Configuration option dmeventd/mirror_library.
    # The library dmeventd uses when monitoring a mirror device.
    # libdevmapper-event-lvm2mirror.so attempts to recover from
    # failures. It removes failed devices from a volume group and
    # reconfigures a mirror as necessary. If no mirror library is
    # provided, mirrors are not monitored through dmeventd.
    mirror_library = "libdevmapper-event-lvm2mirror.so"
```

```
    # Configuration option dmeventd/raid_library.
    # This configuration option has an automatic default value.
    # raid_library = "libdevmapper-event-lvm2raid.so"

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

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

    # Configuration option dmeventd/executable.
    # The full path to the dmeventd binary.
    # This configuration option has an automatic default value.
    # executable = "/usr/sbin/dmeventd"
}

# Configuration section tags.
# Host tag settings.
# This configuration section has an automatic default value.
# tags {

    # Configuration option tags/hosttags.
    # Create a host tag using the machine name.
    # The machine name is nodename returned by uname(2).
    # This configuration option has an automatic default value.
    # hosttags = 0

    # Configuration section tags/<tag>.
    # Replace this subsection name with a custom tag name.
    # Multiple subsections like this can be created. The '@' prefix for
    # tags is optional. This subsection can contain host_list, which is a
    # list of machine names. If the name of the local machine is found in
    # host_list, then the name of this subsection is used as a tag and is
    # applied to the local machine as a 'host tag'. If this subsection is
    # empty (has no host_list), then the subsection name is always applied
    # as a 'host tag'.
    #
    # Example
    # The host tag foo is given to all hosts, and the host tag
    # bar is given to the hosts named machine1 and machine2.
    # tags { foo { } bar { host_list = [ "machine1", "machine2" ] } }
    #
    # This configuration section has variable name.
    # This configuration section has an automatic default value.
    # tag {

        # Configuration option tags/<tag>/host_list.
```

```
 # A list of machine names.
 # These machine names are compared to the nodename returned
 # by uname(2). If the local machine name matches an entry in
 # this list, the name of the subsection is applied to the
 # machine as a 'host tag'.
 # This configuration option does not have a default value defined.
 # }
# }
```

부록 C. LVM 선택 기준

Red Hat Enterprise Linux 릴리스 7.1부터 많은 LVM 보고 명령에서는 -S 또는 --select 옵션을 사용하여 해당 명령에 대한 선택 기준을 정의합니다. Red Hat Enterprise Linux 릴리스 7.2부터 많은 처리 명령은 선택 기준을 지원합니다. 선택 기준을 정의할 수 있는 다음 두 가지 범주의 명령은 다음과 같습니다.

- 보고 명령 - 선택 기준을 충족하는 행만 표시합니다. 선택 기준을 정의할 수 있는 보고 명령의 예로는 pvs, Cryostats ,lvs, pvdisplay ,pvdisplay, lvdisplay ,lvdisplay,lvm devtypes, dmsetup info -c 가 있습니다.

  -S 옵션 외에 -o 선택한 옵션을 지정하면 모든 행이 표시되고 행이 선택 기준과 일치하면 1이 표시되는 "선택" 열이 추가됩니다.

- 명령 처리 - 선택 기준을 충족하는 항목만 처리합니다. 선택 기준을 정의할 수 있는 처리 명령의 예로는 pvchange, Cryostatchange ,lvchange , Cryostatimport, Cryostatexport, Cryostatremove, lvremove 가 있습니다.

선택 기준은 비교 연산자를 사용하여 표시하거나 처리할 특정 필드의 유효한 값을 정의하는 일련의 문입니다. 선택한 필드는 차례로 논리 및 그룹화 연산자에 의해 결합됩니다.

선택 기준을 사용하여 표시할 필드를 지정할 때 선택 기준에 해당하는 필드에 대한 요구 사항이 없습니다. 선택 기준은 하나의 필드 세트를 포함할 수 있으며 출력에는 다른 필드 집합이 포함될 수 있습니다.

- 다양한 LVM 구성 요소에 사용할 수 있는 필드 목록은 C.3절. “선택 기준 필드” 을 참조하십시오.

- 허용되는 Operator 목록은 C.2절. “선택 기준 Operator” 에서 참조하십시오. Operator는 lvm(8) 도움말 페이지에도 제공됩니다.

- 보고 명령의 -S/-select 옵션에 대한 도움말 (또는 ?) 키워드를 지정하여 전체 필드 및 가능한 연산자 세트를 볼 수도 있습니다. 예를 들어 다음 명령은 lvs 명령에 대한 필드 및 가능한 연산자를 표시합니다.

```
# lvs -S help
```

Red Hat Enterprise Linux 7.2 릴리스의 경우 시간 값을 필드 유형의 시간에 대한 선택 기준으로 지정할 수 있습니다. 시간 값을 지정하는 방법에 대한 자세한 내용은 C.4절. “시간 값 지정” 을 참조하십시오.

C.1. 선택 기준 필드 유형

선택 기준에 대해 지정하는 필드는 특정 유형입니다. 각 필드의 도움말 출력에는 대괄호로 묶은 필드 유형이 표시됩니다. 다음 도움말 출력 예제에서는 필드 유형 문자열 , **string _list,숫자,백분율,크기 및 시간**을 나타내는 출력을 보여줍니다.

```
lv_name            - Name. LVs created for internal use are enclosed in brackets.[string]
lv_role            - LV role. [string list]
raid_mismatch_count - For RAID, number of mismatches found or repaired. [number]
copy_percent       - For RAID, mirrors and pvmove, current percentage in-sync. [percent]
lv_size            - Size of LV in current units. [size]
lv_time            - Creation time of the LV, if known [time]
```

표 C.1. "선택 기준 필드 유형" 선택 기준 필드 유형 설명

표 C.1. 선택 기준 필드 유형

필드 유형	설명
숫자	음수가 아닌 정수 값입니다.
크기	단위가 있는 부동 소수점 값, 'm' 단위가 지정되지 않은 경우 기본적으로 사용됩니다.
백분율	% 접미사가 있거나 없는 음수가 아닌 정수입니다.
string	' 또는 " 또는 unquoted"로 인용된 문자입니다.
문자열 목록	[ ] 또는 { }로 묶은 문자열과 "모든 항목이 "모든 항목과 일치해야 합니다" 또는 "하나 이상의 항목이 일치해야 합니다"로 구분된 문자열.

필드에 지정하는 값은 다음과 같습니다.

- 필드 유형의 구체적인 값
- 문자열 필드 유형의 모든 필드를 포함하는 정규식(예: "+~" 연산자).
- 예약된 값(예: **-1, unknown, undefined, undef**)은 정의되지 않은 숫자 값을 나타내는 모든 키워드입니다.

- *필드 값에 대해 정의된 동의어로, 원래 값과 마찬가지로 값에 대해 선택 기준에 사용할 수 있습니다. 필드 값에 대한 정의된 동의어 목록은 표 C.14. "선택 기준 구문" 을 참조하십시오.*

C.2. 선택 기준 OPERATOR

표 C.2. "선택 기준 그룹화 Operator" 선택 기준 그룹화 연산자에 대해 설명합니다.Describes the selection criteria grouping operators.

표 C.2. 선택 기준 그룹화 Operator

Operator 그룹화	설명
( )	문을 그룹화하는 데 사용
[ ]	문자열을 문자열 목록으로 그룹화하는 데 사용됩니다(예: 일치)
{ }	문자열을 문자열 목록으로 그룹화하는 데 사용됩니다(subset match)

표 C.3. "선택 기준 비교 Operator" 선택 기준 비교 연산자와 해당 연산자를 사용할 수 있는 필드 유형에 대해 설명합니다.Describes the selection criteria comparison operators and the field types with which they can be used.

표 C.3. 선택 기준 비교 Operator

비교 Operator	설명	필드 유형
=~	정규 표현식 일치	regex
!~	정규 표현식과 일치하지 않습니다.	regex
=	Equal to	숫자, 크기, 백분율, 문자열, 문자열 목록, 시간
!=	같지 않음	숫자, 크기, 백분율, 문자열, 문자열 목록, 시간
>=	크거나 같음	숫자, 크기, 백분율, 시간
>	보다 큼	숫자, 크기, 백분율, 시간
<=	작거나 같음Less than or equal to	숫자, 크기, 백분율, 시간
<	보다 적음	숫자, 크기, 백분율, 시간

비교 Operator	설명	필드 유형
이후	지정된 시간(>=과 동일) 이후	time
후	지정된 시간 후 (>와 동일)	time
until	지정된 시간 (Garly as <=) 까지	time
before	지정 시간 전 (<와 동일)	time

표 C.4. "선택 기준 논리 및 그룹화 Operator" 선택 기준 논리 및 그룹화 연산자에 대해 설명합니다.

표 C.4. 선택 기준 논리 및 그룹화 Operator

논리 및 그룹화 Operator	설명
&&	모든 필드가 일치해야 합니다.
,	모든 필드는 (&와 동일) 일치해야 합니다.
||	하나 이상의 필드가 일치해야 합니다.
#	하나 이상의 필드가 일치해야 합니다(|와 동일)
!	논리적 부정
(	left parenthesis (그룹 연산자)
)	right parenthesis (그룹 연산자)
[	list start (grouping operator)
]	목록 끝(그룹 작성 운영자)
{	목록 하위 집합 시작(그룹링 연산자)
}	목록 하위 집합 (그룹 연산자)

C.3. 선택 기준 필드

이 섹션에서는 지정할 수 있는 논리 및 물리 볼륨 선택 기준 필드에 대해 설명합니다.

표 C.5. "논리 볼륨 필드" 논리 볼륨 필드 및 해당 필드 유형을 설명합니다.

표 C.5. 논리 볼륨 필드

논리 볼륨 필드	설명	필드 유형
lv_uuid	고유 식별자	string
lv_name	이름(내부 사용을 위해 생성된 논리 볼륨은 대괄호로 묶)	string
lv_full_name	볼륨 그룹, 즉 VG/LV를 포함한 논리 볼륨의 전체 이름	string
lv_path	논리 볼륨의 전체 경로 이름(내부 논리 볼륨의 경우 비어 있음)	string
lv_dm_path	논리 볼륨의 내부 장치 매퍼 경로 이름(**/dev/mapper** 디렉터리)	string
lv_parent	다른 논리 볼륨의 구성 요소인 상위 논리 볼륨의 경우	string
lv_layout	논리 볼륨 레이아웃	문자열 목록
lv_role	논리 볼륨 역할	문자열 목록
lv_initial_image_sync	mirror/RAID 이미지가 변경된 초기 재동기화 중인 경우 설정	숫자
lv_image_synced	mirror/RAID 이미지가 동기화된 경우 설정	숫자
lv_merging	스냅샷 논리 볼륨이 원본으로 병합되는 경우 설정	숫자
lv_converting	논리 볼륨이 변환되는 경우 설정	숫자
lv_allocation_policy	논리 볼륨 할당 정책	string
lv_allocation_locked	논리 볼륨이 할당 변경에 대해 잠길 경우 설정	숫자
lv_fixed_minor	논리 볼륨이 고정된 마이너 번호가 할당되어 있는지 여부 설정	숫자
lv_merge_failed	스냅샷 병합에 실패한 경우 설정	숫자
lv_snapshot_invalid	snapshot 논리 볼륨이 유효하지 않은 경우 설정	숫자
lv_skip_activation	활성화 시 논리 볼륨을 건너뛰는 경우 설정	숫자

논리 볼륨 필드	설명	필드 유형
lv_when_full	씬 풀의 경우, 전체 동작	string
lv_active	논리 볼륨의 활성 상태	string
lv_active_locally	논리 볼륨이 로컬로 활성화되어 있는지 설정	숫자
lv_active_remotely	논리 볼륨이 원격으로 활성 상태인지 여부 설정	숫자
lv_active_exclusively	논리 볼륨이 독점적으로 활성화되어 있는지 여부 설정	숫자
lv_major	영구 메이저 수 또는 -1(지속되지 않는 경우)	숫자
lv_minor	영구 마이너 번호 또는 -1(영구지 않는 경우)	숫자
lv_read_ahead	현재 단위로 사전 설정 읽기	크기
lv_size	현재 단위로 논리 볼륨 크기	크기
lv_metadata_size	씬 및 캐시 풀의 경우 메타데이터를 보유한 논리 볼륨의 크기입니다.	크기
seg_count	논리 볼륨의 세그먼트 수	숫자
origin	스냅샷의 경우 이 논리 볼륨의 원본 장치	string
origin_size	스냅샷의 경우 이 논리 볼륨의 원본 장치의 크기입니다.	크기
data_percent	스냅샷 및 씬 풀 및 볼륨의 경우 논리 볼륨이 활성화된 경우 백분율이 꽉 납니다.	백분율
snap_percent	스냅샷의 경우 논리 볼륨이 활성 상태일 경우 백분율이 꽉 납니다.	백분율
metadata_percent	씬 풀의 경우 논리 볼륨이 활성 상태일 경우 메타데이터의 백분율이 가득합니다.	백분율
copy_percent	RAID, 미러 및 pvPolicies의 경우 현재 백분율 in-sync	백분율
sync_percent	RAID, 미러 및 pvPolicies의 경우 현재 백분율 in-sync	백분율
raid_mismatch_count	RAID의 경우 불일치가 발견되거나 복구되는 경우	숫자
raid_sync_action	RAID의 경우 현재 수행되는 동기화 작업	string

논리 볼륨 필드	설명	필드 유형
raid_write_behind	RAID1의 경우 가장 많은 장치를 쓸 수 있는 뛰어난 쓰기 수	숫자
raid_min_recovery_rate	RAID1의 경우 kiB/sec/disk의 최소 복구 I/O 로드	숫자
raid_max_recovery_rate	RAID1의 경우 kiB/sec/disk의 최대 복구 I/O 로드	숫자
move_pv	pvmove의 경우 pv passes에서 생성된 임시 논리 볼륨의 물리 볼륨을 가져옵니다.	string
convert_lv	lvconvert의 경우 lvconvert에 의해 생성된 임시 논리 볼륨의 이름	string
mirror_log	미러의 경우 동기화 로그를 포함하는 논리 볼륨	string
data_lv	씬 및 캐시 풀의 경우 관련 데이터가 포함된 논리 볼륨	string
metadata_lv	씬 및 캐시 풀의 경우 관련 메타데이터를 보유한 논리 볼륨	string
pool_lv	thin 볼륨의 경우 thin 풀 이 볼륨의 논리 볼륨	string
lv_tags	태그(있는 경우)	문자열 목록
lv_profile	이 논리 볼륨에 연결된 구성 프로필	string
lv_time	알려진 경우 논리 볼륨 생성 시간	time
lv_host	알려진 경우 논리 볼륨의 호스트 생성	string
lv_modules	이 논리 볼륨에 필요한 커널 장치 매퍼 모듈	문자열 목록

표 C.6. "논리 볼륨 장치 결합 정보 및 상태 필드" 논리 장치 정보와 논리적 장치 상태를 결합하는 논리 볼륨 장치 필드를 설명합니다.

표 C.6. 논리 볼륨 장치 결합 정보 및 상태 필드

논리 볼륨 필드	설명	필드 유형
lv_attr	논리 볼륨 장치 정보와 논리 볼륨 상태에 따라 선택합니다.	string

표 C.7. "논리 볼륨 장치 정보 필드" 논리 볼륨 장치 정보 필드 및 해당 필드 유형에 대해 설명합니다.

표 C.7. 논리 볼륨 장치 정보 필드

논리 볼륨 필드	설명	필드 유형
lv_kernel_major	논리 볼륨이 활성 상태가 아닌 경우 현재 주요 번호 또는 -1이 할당되어 있습니다.	숫자
lv_kernel_minor	논리 볼륨이 활성 상태가 아닌 경우 현재 마이너 번호 또는 -1이 할당되어 있습니다.	숫자
lv_kernel_read_ahead	현재 단위의 현재 읽기 앞부분 설정	크기
lv_permissions	논리 볼륨 권한	string
lv_suspended	논리 볼륨이 일시 중단된 경우 설정	숫자
lv_live_table	논리 볼륨에 실시간 테이블이 있는 경우 설정	숫자
lv_inactive_table	논리 볼륨에 비활성 테이블이 있는 경우 설정	숫자
lv_device_open	논리 볼륨 장치가 열려 있는지 여부 설정	숫자

표 C.8. "논리 볼륨 장치 상태 필드" 논리 볼륨 장치 상태 필드와 해당 필드 유형을 설명합니다.

표 C.8. 논리 볼륨 장치 상태 필드

논리 볼륨 필드	설명	필드 유형
cache_total_blocks	총 캐시 블록	숫자
cache_used_blocks	사용된 캐시 블록	숫자
cache_dirty_blocks	더티 캐시 블록	숫자
cache_read_hits	캐시 읽기 적중	숫자
cache_read_misses	캐시 읽기 누락	숫자
cache_write_hits	캐시 쓰기 적중	숫자
cache_write_misses	캐시 쓰기 누락	숫자

논리 볼륨 필드	설명	필드 유형
lv_health_status	논리 볼륨 상태	string

표 C.9. "물리 볼륨 레이블 필드" 물리 볼륨 레이블 필드와 해당 필드 유형을 설명합니다.

표 C.9. 물리 볼륨 레이블 필드

물리 볼륨 필드	설명	필드 유형
pv_fmt	메타데이터 유형	string
pv_uuid	고유 식별자	string
dev_size	현재 단위의 기본 장치 크기	크기
pv_name	이름	string
pv_mda_free	현재 단위로 이 장치의 메타데이터 영역 공간 여유	크기
pv_mda_size	현재 단위로 이 장치의 최소 메타데이터 영역 크기	크기

표 C.5. "논리 볼륨 필드" 물리 볼륨 필드와 해당 필드 유형을 설명합니다.

표 C.10. 물리 볼륨 필드

물리 볼륨 필드	설명	필드 유형
pe_start	기본 장치의 데이터 시작으로 오프셋	숫자
pv_size	현재 단위의 물리 볼륨 크기	크기
pv_free	현재 단위의 할당되지 않은 총 공간	크기
pv_used	현재 단위에 할당된 총 공간	크기
pv_attr	다양한 속성	string
pv_allocatable	이 장치를 할당에 사용할 수 있는지 설정	숫자
pv_exported	이 장치를 내보낼 경우 설정	숫자
pv_missing	시스템에 이 장치가 누락된 경우 설정	숫자

물리 볼륨 필드	설명	필드 유형
pv_pe_count	총 물리 확장 영역 수	숫자
pv_pe_alloc_count	할당된 총 물리 확장 영역 수	숫자
pv_tags	태그(있는 경우)	문자열 목록
pv_mda_count	이 장치의 메타데이터 영역 수	숫자
pv_mda_used_count	이 장치에서 사용되는 메타데이터 영역 수	숫자
pv_ba_start	현재 단위의 기본 장치의 PV Bootloader Area 시작 오프셋	크기
pv_ba_size	현재 단위의 PV Bootloader Area 크기	크기

표 C.11. "볼륨 그룹 필드" 볼륨 그룹 필드 및 해당 필드 유형을 설명합니다.

표 C.11. 볼륨 그룹 필드

볼륨 그룹 필드	설명	필드 유형
vg_fmt	메타데이터 유형	string
vg_uuid	고유 식별자	string
vg_name	이름	string
vg_attr	다양한 속성	string
vg_permissions	볼륨 그룹 권한	string
vg_extendable	볼륨 그룹을 확장할 수 있는 경우 설정	숫자
vg_exported	볼륨 그룹을 내보낼 경우 설정	숫자
vg_partial	볼륨 그룹이 부분적인 경우 설정	숫자
vg_allocation_policy	볼륨 그룹 할당 정책	string
vg_clustered	볼륨 그룹이 클러스터형 경우 설정	숫자
vg_size	현재 단위의 볼륨 그룹의 총 크기	크기

볼륨 그룹 필드	설명	필드 유형
vg_free	현재 단위의 여유 공간 총 용량	크기
vg_sysid	볼륨을 소유한 호스트를 나타내는 볼륨 그룹의 시스템 ID	string
vg_systemid	볼륨을 소유한 호스트를 나타내는 볼륨 그룹의 시스템 ID	string
vg_extent_size	현재 단위의 물리 확장 영역 크기	크기
vg_extent_count	총 물리 확장 영역 수	숫자
vg_free_count	할당되지 않은 물리 확장 영역의 총 수	숫자
max_lv	볼륨 그룹에 허용되는 최대 논리 볼륨 수 또는 무제한인 경우 0	숫자
max_pv	볼륨 그룹에 허용되는 최대 물리 볼륨 수 또는 무제한인 경우 0	숫자
pv_count	물리 볼륨 수	숫자
lv_count	논리 볼륨 수	숫자
snap_count	스냅샷 수	숫자
vg_seqno	내부 메타데이터의 버전 수 - 변경될 때마다 증가	숫자
vg_tags	태그(있는 경우)	문자열 목록
vg_profile	이 볼륨 그룹에 연결된 구성 프로필	string
vg_mda_count	이 볼륨 그룹의 메타데이터 영역 수	숫자
vg_mda_used_count	이 볼륨 그룹에서 사용되는 메타데이터 영역 수	숫자
vg_mda_free	현재 단위로 이 볼륨 그룹의 메타데이터 영역 공간 사용	크기
vg_mda_size	현재 단위에서 이 볼륨 그룹의 최소 메타데이터 영역 크기	크기
vg_mda_copies	볼륨 그룹의 사용 메타데이터 영역의 대상 수	숫자

표 C.12. "논리 볼륨 세그먼트 필드" 논리 볼륨 세그먼트 필드 및 해당 필드 유형을 설명합니다.

표 C.12. 논리 볼륨 세그먼트 필드

논리 볼륨 세그먼트 필드	설명	필드 유형
segtype	논리 볼륨 세그먼트 유형	string
스트라이프	스트라이프 또는 미러 다리 수	숫자
stripesize	스트라이프의 경우 다음 장치로 전환하기 전에 한 장치에 저장된 데이터 양	크기
stripe_size	스트라이프의 경우 다음 장치로 전환하기 전에 한 장치에 저장된 데이터 양	크기
regionsize	미러의 경우 장치를 동기화할 때 복사된 데이터 단위	크기
region_size	미러의 경우 장치를 동기화할 때 복사된 데이터 단위	크기
CHUNKSIZE	스냅샷의 경우 변경 사항을 추적할 때 사용되는 데이터 단위	크기
chunk_size	스냅샷의 경우 변경 사항을 추적할 때 사용되는 데이터 단위	크기
thin_count	씬 풀의 경우 이 풀의 씬 볼륨 수	숫자
삭제	씬 풀의 경우, 삭제 처리 방법	string
CacheMode	캐시 풀의 경우 쓰기 캐시 방법	string
zero	씬 풀의 경우 제로이 활성화된 경우	숫자
transaction_id	씬 풀의 경우 트랜잭션 ID	숫자
thin_id	thin 볼륨의 경우 thin 장치 ID	숫자
seg_start	현재 단위로 세그먼트 시작까지 논리 볼륨 내의 오프셋	크기
seg_start_pe	물리 확장 영역의 세그먼트 시작까지 논리 볼륨 내의 오프셋입니다.	숫자
seg_size	현재 단위의 세그먼트 크기	크기
seg_size_pe	물리 확장 영역의 세그먼트 크기	크기
seg_tags	태그(있는 경우)	문자열 목록
seg_pe_ranges	명령줄 형식의 기본 장치의 물리 확장 영역 범위	string

논리 볼륨 세그먼트 필드	설명	필드 유형
devices	시작 범위 번호와 함께 사용되는 기본 장치	string
seg_monitor	세그먼트의 DMeventd 모니터링 상태	string
cache_policy	캐시 정책(캐쉬된 세그먼트만 해당)	string
cache_settings	캐시 설정/매개 변수(캐쉬된 세그먼트만 해당)	문자열 목록

표 C.13. "물리 볼륨 세그먼트 필드" 물리 볼륨 세그먼트 필드 및 해당 필드 유형을 설명합니다.

표 C.13. 물리 볼륨 세그먼트 필드

물리 볼륨 세그먼트 필드	설명	필드 유형
pvseg_start	세그먼트 시작 물리적 확장 수	숫자
pvseg_size	세그먼트의 확장 영역 수	숫자

표 C.14. "선택 기준 구문" 필드 값에 사용할 수 있는 동의어를 나열합니다.Lists the synonyms you can use for field values. 이러한 동의어는 선택 기준과 원래 값과 마찬가지로 값에 사용할 수 있습니다. 이 표에서 필드 값 ""은 -S 'field_name=""'을 지정하여 일치시킬 수 있는 빈 문자열을 나타냅니다.

이 표에서 0 또는 1로 표시된 필드는 바이너리 값을 나타냅니다. 이 표에 표시된 내용 대신 0 또는 1이 아닌 "some text" 또는 ""로 표시되는 툴을 보고하기 위해 --binary 옵션을 지정할 수 있습니다.

표 C.14. 선택 기준 구문

필드	필드 값	Synonyms
pv_allocatable	allocatable	1
pv_allocatable	""	0
pv_exported	exported	1
pv_exported	""	0
pv_missing	누락됨	1
pv_missing	""	0

필드	필드 값	Synonyms
vg_extendable	extendable	1
vg_extendable	""	0
vg_exported	exported	1
vg_exported	""	0
vg_partial	부분적인	1
vg_partial	""	0
vg_clustered	클러스터형	1
vg_clustered	""	0
vg_permissions	writable	RW, 읽기-쓰기
vg_permissions	read-only	R, ro
vg_mda_copies	비관리	unknown, undefined, undef, -1
lv_initial_image_sync	초기 이미지 동기화	sync, 1
lv_initial_image_sync	""	0
lv_image_synced	이미지가 동기화됨	동기화됨, 1
lv_image_synce	""	0
lv_merging	병합	1
lv_merging	""	0
lv_converting	변환	1
lv_converting	""	0
lv_allocation_locked	할당 잠금됨	locked, 1
lv_allocation_locked	""	0
lv_fixed_minor	수정된 마이너	fixed, 1
lv_fixed_minor	""	0

필드	필드 값	Synonyms
lv_active_locally	로컬에서 활성화	활성, 로컬, 1
lv_active_locally	""	0
lv_active_remotely	원격으로 활성	활성, 원격으로, 1
lv_active_remotely	""	0
lv_active_exclusively	독점적으로 활성화	활성, 전용, 1
lv_active_exclusively	""	0
lv_merge_failed	병합 실패	실패, 1
lv_merge_failed	""	0
lv_snapshot_invalid	스냅샷이 유효하지 않음	잘못된 값, 1
lv_snapshot_invalid	""	0
lv_suspended	일시 중지됨	1
lv_suspended	""	0
lv_live_table	실시간 테이블 현재	라이브 테이블, 라이브, 1
lv_live_table	""	0
lv_inactive_table	비활성 테이블이 있음	비활성 테이블, 비활성, 1
lv_inactive_table	""	0
lv_device_open	open	1
lv_device_open	""	0
lv_skip_activation	활성화 건너뛰기	건너뛰기 1
lv_skip_activation	""	0
zero	zero	1
zero	""	0
lv_permissions	writable	RW, 읽기-쓰기

필드	필드 값	Synonyms
lv_permissions	read-only	R, ro
lv_permissions	read-only-override	ro-override, r-override, R
lv_when_full	오류	공간이 없는 경우 오류 발생
lv_when_full	queue	공간이 없는 경우 큐(Queue)
lv_when_full	""	정의되지 않음
cache_policy	""	정의되지 않음
seg_monitor	""	정의되지 않음
lv_health_status	""	정의되지 않음

C.4. 시간 값 지정

LVM 선택 시간 값을 지정할 때 C.4.1절. "표준 시간 선택 형식" 및 C.4.2절. "Freeform 시간 선택 형식" 에 설명된 대로 표준화된 시간 사양 또는 보다 무료 양식 사양을 사용할 수 있습니다.

/etc/lvm/lvm.conf 구성 파일의 보고서/시간 형식 구성 옵션을 사용하여 시간 값을 표시하는 방법을 지정할 수 있습니다. 이 옵션을 지정하는 방법에 대한 정보는 lvm.conf 파일에 제공됩니다.

시간 값을 지정할 때 표 C.3. "선택 기준 비교 Operator" 에 설명된 대로 , 이후 , 이후 ,이전까지 의 비교 연산자 별칭을 사용할 수 있습니다.

C.4.1. 표준 시간 선택 형식

다음 형식으로 LVM 선택 시간 값을 지정할 수 있습니다.

```
date time timezone
```

표 C.15. "시간 사양 형식" 이러한 시간 값을 지정할 때 사용할 수 있는 형식을 요약합니다.

표 C.15. 시간 사양 형식

필드	필드 값
date	YYYY-MM-DD
	YYYY-MM, auto DD=1
	YYYY, auto MM=01 및 DD=01
time	hh:mm:ss
	hh:mm, auto ss=0
	HH, auto mm=0, auto ss=0
timezone (+ 또는 - 기호 사용)	+HH:mm 또는 -hhh:mm
	+H 또는 -hh

*전체 날짜/시간 사양은 **YYYY-MM-DD hh:mm:sss**입니다. 사용자는 날짜/시간 부분을 오른쪽에서 왼쪽으로 둘 수 있습니다. 이러한 부분이 나올 때마다 범위는 두 번째 단위로 자동으로 가정됩니다. 예를 들어 다음과 같습니다.*

- ***"2015-07-07 9:51"**은 **"2015-07-07 9:51:00" - "2015-07-07 9:51:59"**를 의미합니다.*
- ***"2015-07"**은 **"2015-07-01 0:00:00" - "2015-07-31 23:59"**를 의미합니다.*
- ***"2015"**는 **"2015-01-01 0:00:00" - "2015-12-31 23:59"**을 의미합니다.*

다음 예제에서는 선택 기준에 사용되는 날짜/시간 사양을 보여줍니다.

```
lvs -S 'time since "2015-07-07 9:51"'
lvs -S 'time = "2015-07"'
lvs -S 'time = "2015"'
```

C.4.2. Freeform 시간 선택 형식

다음 권한을 사용하여 **LVM** 선택 기준에 날짜/시간 사양을 지정할 수 있습니다.

- 평일 이름("일요일") - "공요일" 또는 축약 - "Sun" - "Sat"
- 시간 레이블(오요일', "마이요일")
- 현재 날짜를 기준으로 한 날의 라벨("오일", "yesterday")
- 오늘 (N은 숫자)의 상대 오프셋을 사용하여 시간을 반환합니다.
- ("N" "seconds"/"minutes"/"hours"/"days"/"weeks"/"years" "ago")
- ("N" "secs"/"mins"/"hrs" ... "ago")
- ("N" "s"/"m"/"h" ... "ago")
- hh:mm:ss 형식 또는 AM/PM 접미사로 시간 사양
- 월 이름("January" - "December" 또는 약어 "Jan" - "Dec")

다음 예제에서는 선택 기준에서 사용되는 자유형 날짜/시간 사양을 보여줍니다.

```
lvs -S 'time since "yesterday 9AM"'
lvs -S 'time since "Feb 3 years 2 months ago"'
lvs -S 'time = "February 2015"'
lvs -S 'time since "Jan 15 2015" &&  time until yesterday'
lvs -S 'time since "today 6AM"'
```

C.5. 선택 기준 표시 예

이 섹션에서는 **LVM** 표시 명령에 선택 기준을 사용하는 방법을 보여주는 일련의 예제를 제공합니다. 이 섹션의 예제에서는 선택 기준을 사용하지 않을 때 다음 출력을 생성하는 **LVM** 볼륨으로 구성된 시스템을

사용합니다.

```
# lvs -a -o+layout,role
  LV              VG Attr       LSize   Pool Origin Data%  Meta%  Layout      Role
  root            f1 -wi-ao----   9.01g                             linear      public
  swap             f1 -wi-ao---- 512.00m                              linear      public
  [lvol0_pmspare] vg ewi-------   4.00m                               linear      private,   \
                                                                     pool,spare
  lvol1           vg Vwi-a-tz--   1.00g pool        0.00              thin,sparse public
  lvol2           vg Vwi-a-tz--   1.00g pool        0.00              thin,sparse public,    \
                                                                     origin,    \
                                                                     thinorigin
  lvol3           vg Vwi---tz-k   1.00g pool lvol2                    thin,sparse public,    \
                                                                     snapshot,  \
                                                                     thinsnapshot
  pool            vg twi-aotz-- 100.00m             0.00   1.07    thin,pool   private
  [pool_tdata]    vg Twi-ao---- 100.00m                               linear      private,   \
                                                                     thin,pool, \
                                                                     data
  [pool_tmeta]    vg ewi-ao----   4.00m                               linear      private,   \
                                                                     thin,pool, \
                                                                     metadata
```

*다음 명령은 정규식을 사용하여 이름에 **"lvol[13]"**이 있는 모든 논리 볼륨을 표시합니다.*

```
# lvs -a -o+layout,role -S 'lv_name=~lvol[13]'
  LV    VG   Attr       LSize Pool Origin Data%  Layout      Role
  lvol1 vg   Vwi-a-tz-- 1.00g pool        0.00   thin,sparse public
  lvol3 vg   Vwi---tz-k 1.00g pool lvol2         thin,sparse public,snapshot,thinsnapshot
```

*다음 명령은 **500MB**보다 큰 모든 논리 볼륨을 표시합니다.*

```
# lvs -a -o+layout,role -S 'lv_size>500m'
  LV    VG Attr       LSize   Pool Origin Data%  Layout      Role
  root  f1 -wi-ao----   9.01g                    linear      public
  swap  f1 -wi-ao---- 512.00m                     linear      public
  lvol1 vg Vwi-a-tz--   1.00g pool        0.00   thin,sparse public
  lvol2 vg Vwi-a-tz--   1.00g pool        0.00   thin,sparse public,origin,thinorigin
  lvol3 vg Vwi---tz-k   1.00g pool lvol2         thin,sparse public,snapshot,          \
                                                 thinsnapshot
```

*다음 명령은 논리 볼륨이 씬 풀을 구성하는 데 사용됨을 나타내는 **thin** 을 논리 볼륨 역할로 포함하는 모든 논리 볼륨을 표시합니다. 이 예에서는 중괄호(**{}**)를 사용하여 디스플레이의 하위 집합을 나타냅니다.*

```
# lvs -a -o+layout,role -S 'lv_role={thin}'
  LV          VG   Attr       LSize   Layout     Role
  [pool_tdata] vg   Twi-ao---- 100.00m linear     private,thin,pool,data
  [pool_tmeta] vg   ewi-ao----   4.00m linear     private,thin,pool,metadata
```

-

*다음 명령은 **"public"** 역할이 있는 논리 볼륨인 사용 가능한 모든 최상위 논리 볼륨을 표시합니다. 하위 집합을 표시하기 위해 문자열 목록에 중괄호({})를 지정하지 않으면 기본적으로 가정합니다. **lv_role= {public }** 을 지정하는 것과 동일합니다.*

```
# lvs -a -o+layout,role -S 'lv_role=public'
  LV    VG Attr       LSize   Pool Origin Data%  Layout      Role
  root  f1 -wi-ao----   9.01g                    linear      public
  swap  f1 -wi-ao---- 512.00m                    linear      public
  lvol1 vg Vwi-a-tz--   1.00g pool        0.00   thin,sparse public
  lvol2 vg Vwi-a-tz--   1.00g pool        0.00   thin,sparse public,origin,thinorigin
  lvol3 vg Vwi---tz-k   1.00g pool lvol2         thin,sparse public,snapshot,thinsnapshot
```

다음 명령은 씬 레이아웃을 사용하여 모든 논리 볼륨을 표시합니다.

```
# lvs -a -o+layout,role -S 'lv_layout={thin}'
  LV    VG Attr       LSize   Pool Origin Data% Meta% Layout      Role
  lvol1 vg Vwi-a-tz--   1.00g pool        0.00         thin,sparse public
  lvol2 vg Vwi-a-tz--   1.00g pool        0.00         thin,sparse public,origin,          \
                                                      thinorigin
  lvol3 vg Vwi---tz-k   1.00g pool lvol2               thin,sparse public,snapshot,        \
                                                      thinsnapshot
  pool  vg twi-aotz-- 100.00m              0.00  1.07  thin,pool   private
```

*다음 명령은 **"sparse,thin"**과 일치하는 레이아웃 필드가 있는 모든 논리 볼륨을 표시합니다. 일치 항목의 문자열 목록 멤버를 양수로 지정할 필요는 없습니다.*

```
# lvs -a -o+layout,role -S 'lv_layout=[sparse,thin]'
  LV    VG   Attr       LSize Pool Origin Data%  Layout      Role
  lvol1 vg   Vwi-a-tz-- 1.00g pool        0.00   thin,sparse public
  lvol2 vg   Vwi-a-tz-- 1.00g pool        0.00   thin,sparse public,origin,thinorigin
  lvol3 vg   Vwi---tz-k 1.00g pool lvol2         thin,sparse public,snapshot,thinsnapshot
```

다음 명령은 씬인 스파스 논리 볼륨의 논리 볼륨 이름을 표시합니다. 선택 기준에 사용되는 필드 목록은 표시할 필드 목록과 같을 필요가 없습니다.

```
# lvs -a -o lv_name -S 'lv_layout=[sparse,thin]'
  LV
  lvol1
  lvol2
  lvol3
```

C.6. 선택 기준 처리 예

*이 섹션에서는 **LVM** 논리 볼륨을 처리하는 명령에서 선택 기준을 사용하는 방법을 보여주는 일련의 예제를 제공합니다.*

*이 예에서는 씬 스냅샷을 포함한 논리 볼륨 그룹의 초기 구성을 보여줍니다. 씬 스냅샷에는 기본적으로 **"skip activation"** 플래그가 설정되어 있습니다. 이 예제에는 **"skip activation"** 플래그가 설정된 논리 볼륨 **lvol4** 도 포함되어 있습니다.*

```
# lvs -o name,skip_activation,layout,role
  LV    SkipAct         Layout      Role
  root                  linear      public
  swap                    linear      public
  lvol1                 thin,sparse public
  lvol2                 thin,sparse public,origin,thinorigin
  lvol3 skip activation thin,sparse public,snapshot,thinsnapshot
  lvol4 skip activation linear      public
  pool                   thin,pool   private
```

다음 명령은 씬 스냅샷인 모든 논리 볼륨에서 건너뛰기 활성화 플래그를 제거합니다.

```
# lvchange --setactivationskip n -S 'role=thinsnapshot'
  Logical volume "lvol3" changed.
```

*다음 명령은 **lvchange** 명령을 실행한 후 논리 볼륨의 구성을 보여줍니다. **"skip activation"** 플래그는 씬 스냅샷이 아닌 논리 볼륨에서 설정되지 않았습니다.*

```
# lvs -o name,active,skip_activation,layout,role
  LV    Active SkipAct         Layout      Role
  root  active                 linear      public
  swap  active                   linear      public
  lvol1 active                 thin,sparse public
  lvol2 active                 thin,sparse public,origin,thinorigin
  lvol3                      thin,sparse public,snapshot,thinsnapshot
  lvol4 active skip activation linear      public
  pool  active                  thin,pool   private
```

*다음 명령은 추가 **thin origin/snapshot** 볼륨이 생성된 후 논리 볼륨의 구성을 보여줍니다.*

```
# lvs -o name,active,skip_activation,origin,layout,role
  LV    Active SkipAct         Origin Layout      Role
  root  active                   linear      public
  swap  active                     linear      public
  lvol1 active                   thin,sparse public
  lvol2 active                   thin,sparse public,origin,thinorigin
  lvol3                    lvol2  thin,sparse public,snapshot,thinsnapshot
  lvol4 active skip activation       linear      public
```

```
  lvol5 active                    thin,sparse public,origin,thinorigin
  lvol6                    lvol5  thin,sparse public,snapshot,thinsnapshot
  pool  active                    thin,pool   private
```

*다음 명령은 **thin snapshot** 볼륨과 **lvol2** 의 원본 볼륨이 모두 있는 논리 볼륨을 활성화합니다.*

```
# lvchange -ay -S 'lv_role=thinsnapshot && origin=lvol2'

# lvs -o name,active,skip_activation,origin,layout,role
  LV    Active SkipAct         Origin Layout      Role
  root  active                   linear      public
  swap  active                     linear      public
  lvol1 active                   thin,sparse public
  lvol2 active                   thin,sparse public,origin,thinorigin
  lvol3 active                lvol2  thin,sparse public,snapshot,thinsnapshot
  lvol4 active skip activation       linear      public
  lvol5 active                   thin,sparse public,origin,thinorigin
  lvol6                    lvol5  thin,sparse public,snapshot,thinsnapshot
  pool  active                   thin,pool   private
```

*전체 항목에 대해 일치하는 항목을 지정하는 동안 전체 항목에서 명령을 실행하면 전체 항목이 처리됩니다.**If you execute a command on a whole item while specifying selection criteria that match an item from that whole item is processed.** 예를 들어 해당 볼륨 그룹에서 하나 이상의 항목을 선택하는 동안 볼륨 그룹을 변경하면 전체 볼륨 그룹이 선택됩니다. 이 예에서는 볼륨 그룹 **lvol1** 의 일부인 논리 볼륨 **lvol1**을 선택합니다. 볼륨 그룹의 모든 논리 볼륨이 처리됩니다.*

```
# lvs -o name,vg_name
  LV    VG
  root  fedora
  swap  fedora
  lvol1 vg
  lvol2 vg
  lvol3 vg
  lvol4 vg
  lvol5 vg
  lvol6 vg
  pool  vg

# vgchange -ay -S 'lv_name=lvol1'
  7 logical volume(s) in volume group "vg" now active
```

*다음 예제에서는 더 복잡한 선택 기준 문을 보여줍니다. 이 예에서 모든 논리 볼륨에는 **origin** 역할이 있고 이름이 **lvol[456]**이거나 논리 볼륨 크기가 **5GB** 이상인 경우 모든 논리 볼륨에 **mytag** 태그가 지정됩니다.*

```
# lvchange --addtag mytag -S '(role=origin && lv_name=~lvol[456]) || lv_size > 5g'
  Logical volume "root" changed.
  Logical volume "lvol5" changed.
```

부록 D. LVM 오브젝트 태그

LVM 태그는 동일한 유형의 LVM2 개체를 그룹화하는 데 사용할 수 있는 단어입니다. 태그는 물리 볼륨, 볼륨 그룹, 논리 볼륨과 같은 오브젝트에 연결할 수 있습니다. 태그는 클러스터 구성의 호스트에 연결할 수 있습니다.

태그는 PV, VG 또는 LV 인수 대신 명령줄에 제공할 수 있습니다. 모호성을 방지하려면 태그 앞에 @ 앞에 추가해야 합니다. 각 태그는 명령줄에서 위치로 예상되는 유형인 해당 태그를 소유하는 모든 오브젝트로 교체하여 확장됩니다.

LVM 태그는 최대 1024자의 문자열입니다. LVM 태그는 하이픈으로 시작할 수 없습니다.

유효한 태그는 제한된 범위의 문자만 구성할 수 있습니다. 허용되는 문자는 [A-Za-z0-9_+.-]입니다. Red Hat Enterprise Linux 6.1 릴리스에서는 허용되는 문자 목록이 확장되었으며 태그에는 /, =, !, :, # 및 & 문자가 포함될 수 있습니다.

볼륨 그룹의 오브젝트만 태그할 수 있습니다. 볼륨 그룹에서 제거된 경우 물리 볼륨은 태그를 잃게 됩니다. 이는 볼륨 그룹 메타데이터의 일부로 저장되고 물리 볼륨이 제거될 때 삭제되기 때문입니다.

다음 명령은 database 태그가 있는 모든 논리 볼륨을 나열합니다.

```
# lvs @database
```

다음 명령은 현재 활성 상태인 호스트 태그를 나열합니다.

```
# lvm tags
```

D.1. 오브젝트 태그 추가 및 제거

물리 볼륨에서 태그를 추가하거나 삭제하려면 pvchange 명령의 --addtag 또는 --deltag 옵션을 사용합니다.

볼륨 그룹에서 태그를 추가하거나 삭제하려면 Cryostat change 또는 Cryostat create 명령의 --addtag 또는 --deltag 옵션을 사용합니다.

논리 볼륨에서 태그를 추가하거나 삭제하려면 **lvchange** 또는 **lvcreate** 명령의 **--addtag** 또는 **--deltag** 옵션을 사용합니다.

단일 **pvchange, Cryostatchange** 또는 **lvchange** 명령에 **--addtag** 및 **--deltag** 인수를 여러 개 지정할 수 있습니다. 예를 들어 다음 명령은 **T9** 및 **T10** 태그를 삭제하고 **T13** 및 **T14** 태그를 볼륨 그룹 권한 부여에 추가합니다.

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

D.2. 호스트 태그

클러스터 구성에서 구성 파일에 호스트 태그를 정의할 수 있습니다. **tags** 섹션에 **hosttags = 1** 을 설정하면 시스템의 호스트 이름을 사용하여 호스트 태그가 자동으로 정의됩니다. 이를 통해 모든 시스템에서 복제할 수 있는 공통 구성 파일을 사용하여 파일의 동일한 복사본을 보유할 수 있지만, 호스트 이름에 따라 시스템마다 동작이 다를 수 있습니다.

구성 파일에 대한 자세한 내용은 부록 B. LVM 구성 파일 을 참조하십시오.

각 호스트 태그에 대해 추가 구성 파일이 있는 경우 이 파일을 읽습니다. **lvm_hosttag.conf.** 해당 파일이 새 태그를 정의하면 추가 설정 파일이 읽을 파일 목록에 추가됩니다.

예를 들어 구성 파일의 다음 항목은 항상 **tag1** 을 정의하고 호스트 이름이 **host1** 인 경우 **tag2** 를 정의합니다.

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

D.3. 태그를 사용하여 활성화 제어

해당 호스트에서 특정 논리 볼륨만 활성화해야 하는 구성 파일을 지정할 수 있습니다. 예를 들어 다음 항목은 활성화 요청(예: **Cryostat change -ay**)에 대한 필터 역할을 하며 해당 호스트의 메타데이터에 **database** 태그가 있는 모든 논리 볼륨 또는 볼륨 그룹만 활성화합니다.

```
activation { volume_list = ["vg1/lvol0", "@database" ] }
```

메타데이터 태그가 해당 시스템의 호스트 태그와 일치하는 경우에만 일치하는 특수 일치 "@*"가 있습니다.

다른 예로 클러스터의 모든 머신에 구성 파일에 다음 항목이 있는 상황을 고려하십시오.

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

***host db 2** 에서만 **Cryostat1/lvol 2**를 활성화하려면 다음을 수행합니다.*

1. *클러스터의 모든 호스트에서 **lvchange --addtag @db2 Cryostat1/lvol2** 를 실행합니다.*
2. ***lvchange -ay Cryostat1/lvol2** 를 실행합니다.*

이 솔루션은 볼륨 그룹 메타데이터 내에 호스트 이름을 저장하는 작업이 포함됩니다.

부록 E. LVM 볼륨 그룹 메타데이터

볼륨 그룹의 구성 세부 정보를 메타데이터라고 합니다. 기본적으로 메타데이터의 동일한 사본은 볼륨 그룹 내의 모든 물리 볼륨의 모든 메타데이터 영역에서 유지 관리됩니다. LVM 볼륨 그룹 메타데이터는 ASCII로 저장됩니다.

볼륨 그룹에 많은 물리 볼륨이 포함된 경우 메타데이터의 중복 복사본을 많이 갖는 것은 비효율적입니다. pvcreate 명령의 --metadatacopies 0 옵션을 사용하여 메타데이터 사본 없이 물리 볼륨을 생성할 수 있습니다. 메타데이터의 수를 선택하면 물리 볼륨에 포함할 복사본을 지정하면 나중에 변경할 수 없습니다. 0개의 복사본을 선택하면 구성 변경에 대한 업데이트가 더 빨라질 수 있습니다. 그러나 항상 모든 볼륨 그룹에 메타데이터 영역이 있는 물리 볼륨이 하나 이상 포함되어야 합니다(파일 시스템에 볼륨 그룹 메타데이터를 저장할 수 있는 고급 구성 설정을 사용하지 않는 경우). 향후 볼륨 그룹을 분할하려면 모든 볼륨 그룹에 하나 이상의 메타데이터 사본이 필요합니다.

핵심 메타데이터는 ASCII에 저장됩니다. 메타데이터 영역은 원형 버퍼입니다. 새 메타데이터가 이전 메타데이터에 추가된 후 포인터가 업데이트됩니다.

pvcreate 명령의 --metadatasize 옵션을 사용하여 메타데이터 영역의 크기를 지정할 수 있습니다. 물리 볼륨과 수백의 수의 논리 볼륨이 포함된 볼륨 그룹의 경우 기본 크기가 너무 작을 수 있습니다.

E.1. 물리 볼륨 레이블

기본적으로 pvcreate 명령은 물리 볼륨 레이블을 2번째 512바이트 섹터에 배치합니다. 이 레이블은 물리 볼륨 레이블을 스캔하는 LVM 툴이 처음 4개 섹터를 검사하므로 선택적으로 처음 4개의 섹터에 배치될 수 있습니다. 물리 볼륨 레이블은 LABELONE 문자열로 시작됩니다.

물리 볼륨 레이블에는 다음이 포함됩니다.

- 물리 볼륨 UUID
- 블록 장치의 크기(바이트)
- 데이터 영역 위치 NULL 종료 목록
- 잘못된 메타데이터 영역 위치 목록

메타데이터 위치는 오프셋 및 크기(바이트)로 저장됩니다. 레이블에는 약 15개의 위치에 대한 공간이 있지만, LVM 툴에서는 현재 3개, 즉 단일 데이터 영역과 최대 두 개의 메타데이터 영역을 사용합니다.

E.2. 메타데이터 콘텐츠

볼륨 그룹 메타데이터에는 다음이 포함됩니다.

- How and when it was created에 대한 정보
- 볼륨 그룹에 대한 정보

볼륨 그룹 정보에는 다음이 포함됩니다.

- 이름 및 고유 ID
- 메타데이터가 업데이트될 때마다 증가되는 버전 번호입니다.
- 읽기/쓰기 또는 다시 가능 속성과 같은 모든 속성
- 물리 볼륨 수 및 포함할 수 있는 논리 볼륨 수에 대한 관리 제한
- 확장 크기(512바이트로 정의된 섹터 단위)
- 볼륨 그룹을 구성하는 물리 볼륨의 순서가 지정되지 않은 목록, 각각 다음을 사용합니다.
  - UUID: 이를 포함하는 블록 장치를 결정하는 데 사용됩니다.
  - 물리 볼륨이 할당 가능인지 여부와 같은 모든 속성

- *물리 볼륨(섹터) 내에서 첫 번째 확장 영역의 시작 부분 오프셋*
- *확장 영역 수*

- *논리 볼륨의 순서가 지정되지 않은 목록, 각각으로 구성됩니다.*
  - *정렬된 논리 볼륨 세그먼트 목록입니다. 각 세그먼트마다 메타데이터에는 정렬된 물리 볼륨 세그먼트 또는 논리 볼륨 세그먼트에 적용된 매핑이 포함됩니다.*

E.3. 샘플 메타데이터

*다음은 **myvg** 라는 볼륨 그룹의 **LVM** 볼륨 그룹 메타데이터의 예를 보여줍니다.*

```
# 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
                ]
            }
```

```
            }
        }
    }
```

부록 F. 개정 내역

고침 4.0-2	**Wed Aug 7 2019**	**Steven Levine**
7.7 GA 게시에 대한 문서 준비.		
고침 3.0-2	**Thu Oct 4 2018**	**Steven Levine**
7.6 GA 게시에 대한 문서 준비.		
고침 2.0-2	**Thu Mar 15 2018**	**Steven Levine**
7.5 GA 게시에 대한 문서 준비.		
고침 2.0-1	**Thu Dec 14 2017**	**Steven Levine**
7.5 베타 게시에 대한 문서 준비.		
고침 1.0-11	**Wed Jul 19 2017**	**Steven Levine**
7.4 GA 게시의 문서 버전입니다.		
고침 1.0-9	**Mon May 15 2017**	**Steven Levine**
7.4 Beta 게시에 대한 문서 준비.		
고침 1.0-7	**Mon Mar 27 2017**	**Steven Levine**
7.3의 업데이트 버전		
고침 1.0-5	**Mon Oct 17 2016**	**Steven Levine**
7.3 GA 게시의 버전		
고침 1.0-4	**Wed Aug 17 2016**	**Steven Levine**
7.3 베타 게시에 대한 문서 준비.		
고침 0.3-4	**Mon Nov 9 2015**	**Steven Levine**
7.2 GA 게시를 위한 문서 준비		
고침 0.3-2	**Wed Aug 19 2015**	**Steven Levine**
7.2 베타 게시에 대한 문서 준비.		
고침 0.2-7	**Mon Feb 16 2015**	**Steven Levine**
7.1 GA 버전		
고침 0.2-6	**Thu Dec 11 2014**	**Steven Levine**
7.1 베타 릴리스 버전		
고침 0.1-22	**Mon Jun 2 2014**	**Steven Levine**
7.0 GA 버전		
고침 0.1-1	**Wed Jan 16 2013**	**Steven Levine**
Red Hat Enterprise Linux 6 버전의 문서에서 분기했습니다.		