Red Hat Enterprise Linux 6 LVM (Logical Volume Manager) 관리

LVM 관리자 가이드
엮음 1

Landmann

Red Hat Enterprise Linux 6 LVM (Logical Volume Manager) 관리

LVM 관리자 가이드
엮음 1

Landmann
rlandmann@redhat.com

초록

다음 부분에서는 클러스터 환경에서의 LVM 실행에 관한 내용을 다루고 있는 LVM 논리 볼륨 관리자에 대해 설명합니다.

차례

개요

1. 가이드 설명

다음 부분에서는 클러스터 환경에서의 LVM 실행에 관한 내용을 다루고 있는 LVM (Logical Volume Manager)에 대해 설명합니다.

2. 사용자

이 문서는 Linux 운영 체제에서 시스템을 관리하고 있는 시스템 관리자를 위한 것입니다. 따라서 Red Hat Enterprise Linux 6 및 GFS2 파일 시스템 관리에 능숙해야 합니다.

3. 소프트웨어 버전

표 1. 소프트웨어 버전

소프트웨어	설명
RHEL 6	RHEL6 및 그 이상 버전을 말함
GFS2	RHEL5 및 그 이상 버전의 GFS2를 말함

4. 관련 문서

Red Hat Enterprise Linux 사용에 관한 보다 자세한 내용은 다음 문서 자료에서 참조하시기 바랍니다.

- *설치 가이드* — Red Hat Enterprise Linux 6 설치 관련 내용을 다루고 있습니다.
- *운용 가이드* — Red Hat Enterprise Linux 6 활용, 설정, 관리 관련 내용을 다루고 있습니다.
- *스토리지 관리 가이드* — Red Hat Enterprise Linux 6의 스토리지 장치 및 파일 시스템을 효과적으로 관리하는 방법에 대한 내용을 다루고 있습니다.

Red Hat Enterprise Linux 6 용 고가용성 추가 기능 및 장애 복구형 스토리지 추가 기능에 대한 자세한 내용은 다음 자료에서 참조하십시오:

- *고가용성 추가 기능 개요* — 고가용성 추가 기능에 대한 높은 수준의 개요를 다루고 있습니다.
- *클러스터 관리* — Red Hat 고가용성 추가 기능을 설치, 설정, 관리에 대한 내용을 다루고 있습니다.
- *GFS 2 (Global File System 2): 설정 및 관리* — 장애 복구형 스토리지 추가 기능에 들어있는 Red Hat GFS 2 (Red Hat Global File System 2) 설치, 설정, 관리에 관한 내용을 다루고 있습니다.
- *DM Multipath* — Red Hat Enterprise Linux 6의 장치 맵퍼 멀티패스 (Device-Mapper Multipath) 기능 사용에 관한 내용을 다루고 있습니다.
- *로드 밸런서 관리* — 실제 서버 그룹 전체에 걸쳐 IP 부하를 분산하기 위해 LVS (Linux Virtual Servers)를 제공하는 일련의 통합 소프트웨어 구성 요소로 로드 밸런서 추가 기능을 사용하여 고성능 시스템 및 서비스 설정에 대한 정보를 제공합니다.
- *릴리즈 노트* — Red Hat 제품의 최신 릴리즈에 관한 내용을 다루고 있습니다.

고가용성 추가 기능 문서 및 기타 다른 Red Hat 문서는 HTML, PDF, RPM 버전으로 Red Hat Enterprise Linux 문서 CD 및 http://www.redhat.com/docs/ 온라인 상에서 보실 수 있습니다.

5. 피드백을 보내 주십시오!

문서 내용 개선을 위한 제안이 있거나 오자를 발견했을 경우 언제든지 알려 주시기 바랍니다. **Red Hat Enterprise Linux 6** 제품과 구성 요소 **doc-Logical_Volume_Manager**에 대한 리포트를 버그질라 (Bugzilla)에 제출해 주시면 됩니다. (http://bugzilla.redhat.com/) 버그 리포트를 작성하실 때, 문서의 식별자: **Logical_Volume_Manager_Administration(EN)-6 (2011-05-19-15:20)**를 반드시 기입해 주시기 바랍니다.

문서 내용 개선을 위한 제안이 있으실 경우, 최대한 명확히 설명해 주시기 바랍니다. 오류를 발견하셨다면 저희가 쉽게 식별할 수 있도록 섹션 번호와 주위의 문장들을 함께 보내주시기 바랍니다.

6. 문서화 규정

이 메뉴얼에서는 특정 단어 및 구문을 강조 표시하여 특정 정보 부분에 주의를 집중시키기 위해 문서화 규정을 사용하고 있습니다.

PDF 및 문서 편집에서 이 메뉴얼은 Liberation 글꼴 모음에 있는 서체를 사용합니다. 시스템에 Liberation 글꼴 모음이 설치되어 있을 경우 이는 HTML 편집에서도 사용되지만 설치되어 있지 않을 경우, 다른 동일한 서체로 나타나게 됩니다. 알림: Red Hat Enterprise Linux 5 및 이후 버전에는 기본값으로 Liberation 글꼴 모음이 들어 있습니다.

6.1. 표기 규정

네 가지 표기 규정을 사용하여 특정 단어 및 구문에 주의를 집중시킵니다. 이러한 규정 및 적용 방식은 다음과 같습니다.

고정폭 굵은체

쉘 명령, 파일 이름 및 경로를 포함한 시스템 입력을 강조하기 위해 사용됩니다. 키 및 키 조합을 강조하기 위해 사용되기도 합니다. 예:

> 현재 작업 중인 디렉토리에 있는 **my_next_bestselling_novel** 파일 내용을 확인하려면, 쉘 프롬프트에서 **cat my_next_bestselling_novel** 명령을 입력하고 **Enter** 키를 눌러 명령을 실행합니다.

위에서 파일 이름, 쉘 명령, 키 모두는 고정폭 굵은체로 나타나 있어 내용과 구별될 수 있습니다.

키 조합은 키 조합의 각 부분을 더하기(+) 기호로 연결하여 개별 키과 구별되게 할 수 있습니다. 예:

> **Enter** 키를 눌러 명령을 실행합니다.
>
> **Ctrl+Alt+F2**를 눌러 가상 터미널로 전환합니다.

첫 번째 예에서는 눌러야 하는 특정 키를 강조하고 있습니다. 두 번째 예에서는 키 조합 (동시에 눌러야 하는 세 개의 키 묶음)을 강조하고 있습니다.

소스 코드를 설명해야 할 경우, 문장에서 언급된 클래스 이름, 방식, 기능, 변수 이름 및 반환값은 위와 같이 **고정폭 굵은체**로 나타나게 됩니다. 예:

> 파일 관련 클래스에는 파일 시스템의 경우 **filesystem**, 파일의 경우 **file**, 디렉토리의 경우 **dir**가 포함됩니다. 각각의 클래스에는 자체의 권한 설정이 있습니다.

가변폭 굵은체

이는 프로그램 이름; 대화 상자 텍스트; 레이블된 버튼; 체크 박스 및 라디오 버튼 레이블; 메뉴 제목; 하부 메뉴 제목을 포함하여 시스템에 있는 단어 또는 구문을 나타냅니다. 예:

> 주 메뉴 바에서 **시스템 → 기본설정 → 마우스**를 선택하여 **마우스 기본 설정**을 시작합니다.

버튼 탭에서, **왼손 잡이 마우스** 체크 상자를 선택하고 **닫기**를 클릭하여 주요 마우스 버튼을 왼쪽에서 오른쪽으로 전환합니다 (왼손 잡이일 경우 보다 적절하게 마우스 사용을 할 수 있게 함).

gedit 파일에 특수 문자를 입력하려면 주 메뉴 바에서 **프로그램** → **보조 프로그램** → **문자 표**를 선택합니다. 그 다음으로, **문자 표** 메뉴 바에서 **찾기** → **찾기...**를 선택하고 **찾기** 필드에 문자를 입력하고 **다음**을 클릭합니다. 찾기한 문자가 **문자 표**에 강조 표시되어 나타납니다. 강조 표시된 문자를 두 번 클릭하여 **복사할 텍스트** 필드에 나타나면 **복사** 버튼을 클릭합니다. 편집 중인 문서로 돌아가 **gedit** 메뉴 바에서 **편집** → **붙여넣기**를 선택합니다.

위의 내용에는 프로그램 이름, 다양한 시스템 메뉴 이름 및 항목; 특정 프로그램 메뉴 이름; GUI 인터페이스에 있는 버튼 및 텍스트가 포함되어 있으며, 텍스트와 구별 가능하도록 모두 가변폭 굵은체로 되어 있습니다.

고정폭 굵은 이텔릭체 또는 ***가변폭 굵은 이텔릭체***

고정폭 굵은체이던 가변폭 굵은체이던지 간에 이텔릭체가 추가될 경우 이는 교체 또는 변경 가능한 텍스트를 나타내는 것입니다. 글자 그대로 입력하지 말아야 할 텍스트나 또는 상황에 따라 변경해야 하는 텍스트의 경우 이텔릭체로 나타냅니다. 예:

ssh를 사용하여 원격 컴퓨터에 연결하려면, 쉘 프롬프트에 **ssh *username@domain.name***을 입력합니다. 원격 컴퓨터가 **example.com**이고 사용자 이름이 john일 경우, **ssh john@example.com**을 입력합니다.

mount -o remount *file-system* 명령은 지정한 파일 시스템을 다시 마운트합니다. 예를 들어, **/home** 파일 시스템을 다시 마운트하려면 **mount -o remount /home** 명령을 사용합니다.

현재 설치된 패키지 버전을 보려면, **rpm -q *package*** 명령을 사용합니다. 그러면 다음과 같은 값이 출력됩니다: ***package-version-release***.

사용자 이름, 도메인 이름, 파일 시스템, 패키지, 버전 및 릴리즈는 굵은 이탤릭체로 표시되는 점에 유의하십시오. 각 단어는 자리 표시자로 명령을 실행할 때 입력하는 텍스트나 도는 시스템에 의해 나타나는 텍스트 중 하나입니다.

작업 제목을 표시하기 위한 기본적인 사용을 제외하고 중요한 새로운 용어를 처음 사용할 때 이텔릭체로 표시합니다. 예:

Publican은 *DocBook* 발행 시스템입니다.

6.2. 인용문 규정

터미널 출력 결과 및 소스 코드 목록은 주위의 문장에서 잘 보이는 위치에 설정됩니다.

터미널로 보내진 출력 결과는**mono-spaced roman**에 설정되어 다음과 같이 나타납니다:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

소스 코드 목록도 **mono-spaced roman**에 설정되지만, 다음과 같이 구문 강조가 추가되어 있습니다:

```
static int kvm_vm_ioctl_deassign_device(struct kvm *kvm,
                  struct kvm_assigned_pci_dev *assigned_dev)
{
          int r = 0;
          struct kvm_assigned_dev_kernel *match;

          mutex_lock(&kvm->lock);

          match = kvm_find_assigned_dev(&kvm->arch.assigned_dev_head,
                                        assigned_dev->assigned_dev_id);
          if (!match) {
                  printk(KERN_INFO "%s: device hasn't been assigned before, "
                    "so cannot be deassigned\n", __func__);
                  r = -EINVAL;
                  goto out;
          }

          kvm_deassign_device(kvm, match);

          kvm_free_assigned_device(kvm, match);

out:
          mutex_unlock(&kvm->lock);
          return r;
}
```

6.3. 알림 및 경고

마지막으로, 3 종류의 시각적 스타일을 사용하여 간과될 수 있는 정보에 주의를 집중시킵니다.

참고

알림에서는 현재 작업에 대한 도움말, 지름길 또는 대안적 방법을 제공합니다. 알림 내용을 무시해도 상관없지만 효율적으로 작업할 수 있는 방법을 놓칠 수 있습니다.

중요

중요 상자에서는 현재 세션에만 적용되는 설정을 변경하거나 업데이트를 적용하기 전 다시 시작해야 하는 서비스와 같이 간과하기 쉬운 세부 사항을 제공합니다. 중요 상자를 무시해도 데이터를 손실하게 되지 않지만 문제를 일으킬 수 있습니다.

주의

경고는 무시해서는 안됩니다. 경고를 무시할 경우 대부분 데이터가 손실될 수 있습니다.

1장. LVM 논리 볼륨 관리자

다음에서는 Red Hat Enterprise Linux 6 초기 릴리즈 및 후속 릴리즈의 새로운 LVM 논리 볼륨 관리자 기능에 대해 요약 설명합니다. 그 후 LVM (Logical Volume Manager) 구성 요소의 높은 수준의 개요를 다루고 있습니다.

1.1. 새로운 기능 및 변경된 기능

다음 부분에서는 Red Hat Enterprise Linux 6 초기 및 후속 릴리즈에 포함된 LVM 논리 볼륨 관리자의 새롭고 변경된 기능에 대해 설명합니다.

1.1.1. Red Hat Enterprise Linux 6.0에서 새롭게 변경된 기능

Red Hat Enterprise Linux 6.0에는 다음과 같은 문서, 기능 업데이트 및 변경 사항이 포함되어 있습니다.

- **lvm.conf** 파일의 **activation** 부분에 있는 **mirror_image_fault_policy**와 **mirror_log_fault_policy** 매개 변수로 장치 장애 발생시 미러 논리 볼륨의 동작 방법을 정의할 수 있습니다. 이 매개 변수가 **remove**로 설정될 경우, 시스템은 잘못된 장치를 삭제하고 이 장치 없이 실행하려 합니다. 이 매개 변수가 **allocate**로 설정될 경우, 시스템을 잘못된 장치를 삭제하고 잘못된 장치 대신 새로운 장치를 할당하려 합니다. 장치 대신 적당한 장치나 공간이 할당되지 않을 경우, 이는 **remove** 정책과 같이 동작합니다. LVM 미러 실패 정책에 대한 자세한 내용은 4.4.3.1절. “미러 논리 볼륨 실패 정책 ”에서 참조하십시오.
- Red Hat Enterprise Linux 6 릴리즈에서 Linux I/O 스택은 벤더 제공 I/O 제한 내용을 처리하기 위해 강화되었습니다. 이는 LVM을 포함하여 스토리지 관리 도구가 데이터 배치 및 액세스를 최적화할 수 있게 합니다. 이러한 지원 사항은 **lvm.conf** 파일에 있는 **data_alignment_detection** 및 **data_alignment_offset_detection**의 기본값을 변경하여 비활성화할 수 있으나 이를 비활성화하는 것은 권장 사항이 아닙니다.

  LVM에서의 데이터 정렬과 **data_alignment_detection** 및 **data_alignment_offset_detection**의 기본값 변경에 대한 내용은 **/etc/lvm/lvm.conf** 파일에 있는 줄단위 문서를 참조하십시오. 이는 부록 B. *LVM 설정 파일* 에서도 문서화되어 있습니다. Red Hat Enterprise Linux 6에서 I/O 스택 및 I/O 제한에 대한 일반적인 지원 내용은 *스토리지 관리 가이드 (Storage Administration Guide)*를 참조하십시오.
- Red Hat Enterprise Linux 6에서 장치 맵퍼 (Device Mapper)는 **udev** 통합에 대해 직접 지원합니다. 이는 장치 맵퍼를 LVM 장치를 포함하여 장치 맵퍼 장치와 관련된 모든 **udev** 프로세싱에 장치 맵퍼를 동기화합니다. **udev** 장치 관리자에 대한 장치 맵퍼 지원에 대한 자세한 내용은 A.3절. “udev 장치 관리자에 대해 장치 매퍼(Device Mapper) 지원 ”에서 참조하십시오.
- Red Hat Enterprise Linux 6 릴리즈에서 **lvconvert --repair** 명령을 사용하여 디스크 장애 발생 후 미러를 복구할 수 있습니다. 이는 일관성있는 상태로 미러를 다시 불러옵니다. **lvconvert --repair** 명령에 대한 자세한 내용은 4.4.3.3절. “미러 논리 장치 복구 ”에서 참조하십시오.
- Red Hat Enterprise Linux 6 릴리즈에서 **lvconvert** 명령의 **--merge** 옵션을 사용하여 본래 볼륨으로 스냅샷을 합칠 수 있습니다. 스냅샷을 합치는 방법에 대한 자세한 내용은 4.4.5절. “스냅샷 볼륨 합치기 ”에서 참조하십시오.
- Red Hat Enterprise Linux 6 릴리즈에서 **lvconvert** 명령의 **--splitmirrors** 인수를 사용하여 새 논리 볼륨을 구성하기 위해 미러 논리 볼륨의 중복된 이미지를 분리할 수 있습니다. 이 옵션 사용에 대한 자세한 내용은 4.4.3.2절. “미러 논리 볼륨의 중복된 이미지 분리하기 ”에서 참조하십시오.
- 미러 논리 장치를 생성할 때 **lvcreate** 명령의 **--mirrorlog mirrored** 인수를 사용하여 미러되는 미러 논리 장치에 대해 미러 로그를 생성합니다. 이 옵션 사용에 대한 자세한 내용은 4.4.3절. “미러 볼륨 생성 ”에서 참조하십시오.

1.1.2. Red Hat Enterprise Linux 6.1에서 새롭게 변경된 기능

Red Hat Enterprise Linux 6.1에는 다음과 같은 문서, 기능 업데이트 및 변경 사항이 포함되어 있습니다.

- Red Hat Enterprise Linux 6.1 릴리즈에서는 미러 논리 볼륨의 스냅샷 논리 볼륨 생성을 지원합니다. 선형 또는 스트라이프 논리 볼륨의 스냅샷을 생성하는 것과 마찬가지로 미러 볼륨의 스냅샷을 만들 수 있습니다. 스냅샷 볼륨을 생성하는 방법은 4.4.4절. “스냅샷 볼륨 생성 ”에서 참조하십시오.
- LVM 볼륨을 확장할 때 **lvextend** 명령의 **--alloc cling** 옵션을 사용하여 **cling** 할당 정책을 지정할 수 있습니다. 이러한 정책에서는 동일한 물리 볼륨에 있는 공간이 기존 논리 볼륨의 마지막 세그먼트로 선택됩니다. 물리 볼륨에 공간이 충분하지 않고 태그 목록이 **lvm.conf** 파일에 정의되어 있는 경우, LVM은 태그가 물리 볼륨에 부착되어 있는지를 확인하고 기존 익스텐트와 새 익스텐트 간에 물리 볼륨 태그를 일치시키려고 합니다.

  lvextend 명령의 **--alloc cling** 옵션을 사용하여 LVM 미러 볼륨 확장에 대한 내용은 4.4.12.2절. “**cling** 할당 정책을 사용하여 논리 볼륨 확장 ”에서 참조하십시오.
- 단일 **pvchange**, **vgchange**, **lvchange** 명령으로 여러 **--addtag** 및 **--deltag** 인수를 지정할 수 있습니다. 객체 태그의 추가 및 제거에 대한 내용은 C.1절. “객체 태그 추가 및 삭제 ”에서 참조하십시오.
- LVM 객체 태그에서 사용할 수 있는 문자 목록이 확대되고 태그에는 "/", "=", "!", ":", "#", and "&" 문자를 사용할 수 있습니다. LVM 객체 태그에 대한 내용은 부록 C. *LVM 객체 태그* 에서 참조하십시오.
- 단일 논리 볼륨에 있는 RAID0 (스트라이핑)과 RAID1 (미러링)을 결합할 수 있습니다. 논리 볼륨을 만들고 동시에 미러 수 (**--mirrors X**)와 스트라이프 수 (**--stripes Y**)를 지정하면 미러 장치의 구성 장치가 스프라이프됩니다. 미러 논리 불륨을 생성하는 내용은 4.4.3절. “미러 볼륨 생성 ”에서 참조하십시오.
- Red Hat Enterprise Linux 6.1 릴리즈에서는 클러스터 논리 볼륨에서 지속적인 데이터 백업을 생성해야 할 경우 볼륨을 독단적으로 활성화하여 스냅샷을 생성할 수 있습니다. 노드에서 논리 볼륨을 독단적으로 활성화하는 방법에 대한 내용은 4.7절. “클러스터에 있는 개별적 노드에서 논리 볼륨 활성화 ”에서 참조하십시오.

1.2. 논리 볼륨

볼륨 관리는 물리 스토리지 이상의 추상적 레이어를 생성하여, 논리 스토리지 볼륨을 생성할 수 있게 합니다. 이는 직접적으로 물리 스토리지를 사용하는것 보다 더 많은 방법에서 광대한 유연성을 제공합니다. 논리 볼륨으로 물리 디스크 크기에 제한을 두지 않을 수 있습니다. 이에 더하여 하드웨어 스토리지 설정이 소프트웨어에서 숨겨져 있어 어플리케이션을 정지시키거나 또는 파일 시스템을 마운트 해제하지 않고 크기 조정 및 이동 가능하게 됩니다. 이는 운영 비용을 줄일 수 있습니다.

논리 볼륨은 물리 스토리지를 직접 사용하여 다음과 같은 장점을 제공합니다:

- 유연한 용량

  논리 볼륨을 사용할 때, 디스크 및 파티션을 단일 논리 볼륨으로 모을 수 있으므로 파일 시스템을 여러 디스크에 걸쳐 늘릴 수 있습니다.
- 크기 조정 가능한 스토리지 풀

  기본 디스크 장치를 다시 포맷하거나 파티션하지 않고 간단한 소프트웨어 명령으로 논리 볼륨 크기를 늘이거나 줄일 수 있습니다.
- 온라인 데이터 재배치

  보다 새롭고 빠른 장애 복구형 스토리지 하부 시스템을 배치하기 위해 시스템이 활성화되어 있는 동안 데이터를 옮길 수 있습니다. 디스크가 사용되고 있는 동안 데이터를 디스크에 재배치할 수 있습니다. 예를 들어, 디스크를 삭제하기 전 핫 스왑이 가능한 디스크 비우기를 할 수 있습니다.
- 편의에 따라 장치 이름 지정

  논리 스토리지 볼륨은 사용자 정의 그룹에서 관리되며, 편의에 따라 이름을 지정할 수 있습니다.
- 디스크 스트라이핑

  두 개 이상의 디스크에 걸쳐 데이터를 스트라이핑하는 논리 볼륨을 생성할 수 있습니다. 이는 데이터 처리량을 급격히 상승시킬 수 있습니다.
- 미러 볼륨

논리 볼륨은 편리한 방법으로 데이터에 미러를 설정하게 합니다.

- 볼륨 스냅샷

  논리 볼륨을 사용하여, 지속적인 백업을 위해 장치 스냅샷을 찍거나 또는 실제 데이터에 영향을 미치지 않고 변경 효과를 테스트할 수 있습니다.

이러한 기능을 LVM에서 구현하는 방법은 이 문서의 나머지 부분에서 설명합니다.

1.3. LVM 아키텍처 개요

Linux 운영 체제의 Red Hat Enterprise Linux 4 릴리즈의 경우, 본래의 LVM1 논리 볼륨 관리자는 LVM1보다 더 일반적인 커널 프레임워크를 갖고 있는 LVM2로 대체되었습니다. LVM2는 LVM1이상으로 기능이 개선되었습니다 이는 다음과 같습니다.

- 유연한 용량
- 보다 효율적인 메타데이터 스토리지
- 보다 향상된 복구 포맷
- 새로운 ASCII 메타데이터 포맷
- 메타데이터로 원자 변경
- 메타데이터의 이중 복사본

LVM2는 스냅샷 및 클러스터 지원을 제외하고 LVM1과 역 호환성이 있습니다. **vgconvert** 명령으로 LVM1 포맷에서 LVM2 포맷으로 볼륨 그룹을 변환할 수 있습니다. LVM 메타데이터 포맷을 변환하는 방법에 대한 내용은 **vgconvert**(8) 맨 페이지를 참조하시기 바랍니다.

LVM 논리 볼륨의 기본 물리 스토리지 단위는 파티션이나 전체 디스크와 같은 블록 장치입니다. 이러한 장치는 LVM *물리 볼륨* (PV)으로 초기화되어야 합니다.

LVM 논리 볼륨을 생성하려면 물리 볼륨이 *볼륨 그룹* (VG)으로 통합되어야 합니다. 이는 LVM 논리 볼륨 (LV)을 할당할 수 있는 디스크 공간의 풀을 생성합니다. 이러한 절차는 디스크가 파티션으로 나뉘어지는 방법과 유사합니다. 논리 볼륨은 파일 시스템 및 어플리케이션 (예: 데이터베이스)으로 사용됩니다.

그림 1.1. “LVM 논리 볼륨 구성 요소 ”에서는 LVM 논리 볼륨의 구성 요소를 보여줍니다:

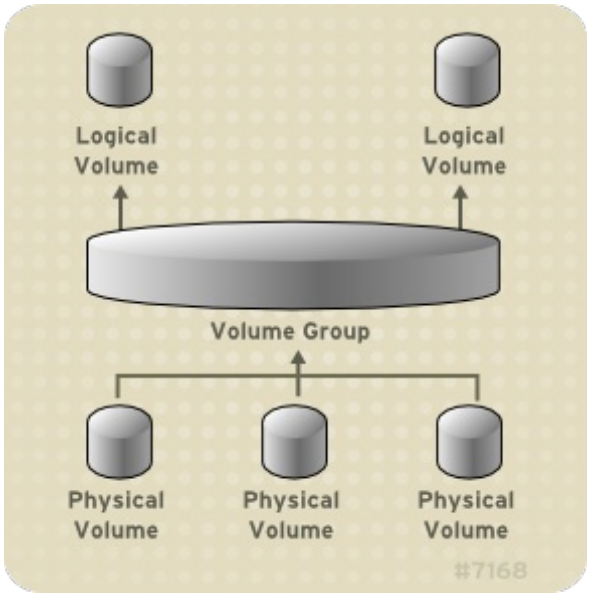

그림 1.1. LVM 논리 볼륨 구성 요소

LVM 논리 볼륨의 구성 요소에 대한 자세한 내용은 2장. *LVM 구성 요소* 에서 확인하시기 바랍니다.

1.4. CLVM (Clustered Logical Volume Manager)

CLVM (Clustered Logical Volume Manager)은 LVM으로의 클러스터 확장 모음입니다. 이는 LVM을 사용하여 공유 스토리지를 (예: SAN 상에서) 관리하기 위해 컴퓨터의 클러스터를 허용합니다. CLVM은 장애 복구형 스토리지 추가 기능 (Resilient Storage Add-On)의 일부분입니다.

시스템 요구 사항에 따라 CLVM을 사용해야 할 지에 대한 여부:

- 시스템의 하나의 노드만 논리 볼륨으로 설정하는 스토리지에 액세스를 필요로할 경우, CLVM 확장을 사용하지 않고 LVM을 사용할 수 있으며 노드와 함께 생성된 논리 볼륨은 노드에서 모두 로컬로 됩니다.
- 스토리지를 액세스하는 단일 노드 만이 활성화되는 장애 조치를 위해 클러스터 시스템을 사용할 경우, 고가용성 논리 볼륨 관리 에이전트 (HA-LVM)를 사용해야 합니다. HA-LVM에 대한 자세한 내용은 *Red Hat Cluster 설정 및 관리* 를 참조하십시오.
- 하나 이상의 클러스터 노드가 활성 노드 사이에서 공유되는 스토리지로의 액세스를 필요로할 경우, CLVM을 사용해야 합니다. CLVM은 논리 볼륨이 설정되는 동안 물리 스토리지로의 액세스를 잠금하여 사용자가 공유 스토리지에 논리 볼륨을 설정하게 하며, 공유 스토리지를 관리하기 위해 클러스터화된 잠금 서비스를 사용합니다.

CLVM을 사용하려면, **clmvd** 데몬을 포함한 고가용성 추가 기능 (High Availability Add-On) 및 장애 복구형 스토리지 추가 기능 (Resilient Storage Add-On) 소프트웨어가 실행되고 있어야 합니다. **clmvd** 데몬은 LVM으로의 주요 클러스링 확장 데몬입니다. **clvmd** 데몬은 각각의 클러스터 컴퓨터에서 실행하여 클러스터에 있는 LVM 메타데이터 업데이트를 분산하고, 동일한 논리 볼륨 관점에서 각각의 클러스터 컴퓨터를 보여줍니다. 고가용성 추가 기능 설치 및 관리에 관한 내용은 *Red Hat Cluster 설정 및 관리*를 참조하십시오.

부팅시 **clmvd** 데몬이 시작되었는지를 확인하기위해, 다음과 같이 **clvmd** 서비스에서 **chkconfig ... on** 명령을 실행할 수 있습니다:

```
# chkconfig clvmd on
```

clvmd 데몬이 시작되지 않을 경우, 다음과 같이 **clvmd** 서비스에서 **service ... start** 명령을 실행할 수 있습니다:

```
# service clvmd start
```

클러스터 환경에서 LVM 논리 볼륨을 생성하는 것은 단일 노드에서 LVM 논리 볼륨을 생성하는 것과 동일합니다. 4장. *CLI 명령을 사용한 LVM 관리* 및 7장. *LVM GUI를 통한 LVM 관리*에서 볼 수 있듯이 LVM 명령 자체에서나 LVM 그래픽 사용자 인터페이스는 차이가 없습니다. 클러스터에 생성할 LVM 볼륨을 활성화하려면, 클러스터 인프라를 반드시 실행하고 클러스터는 쿼터에 도달해야 합니다.

기본값으로 공유 스토리지에서 CLVM과 함께 생성된 논리 볼륨은 공유 스토리지로 액세스하는 모든 컴퓨터에서 볼 수 있습니다. 하지만, 모든 공유 장치가 클러스터에 있는 하나의 노드에서만 보일 경우에도 논리 볼륨을 생성할 수 있습니다. 또한 논리 볼륨의 상태를 논리 볼륨 그룹에서 클러스터 볼륨 그룹으로 변경할 수도 있습니다. 보다 자세한 내용은 4.3.2절. "클러스터에서 볼륨 그룹 생성 " 및 4.3.7절. "볼륨 그룹의 매개 변수 변경 "에서 확인하시기 바랍니다.

경고

공유 스토리지에 CLVM으로 볼륨 그룹을 생성할 때, 클러스터에 있는 모든 노드가 볼륨 그룹을 구성하는 물리 볼륨에 액세스했는지 확인해야 합니다. 스토리지에 액세스한 일부 노드가 있는 대칭적인 클러스터 설정은 지원되지 않습니다.

그림 1.2. "CLVM 개요 "에서는 클러스터에 있는 CLVM 개요를 보여줍니다.

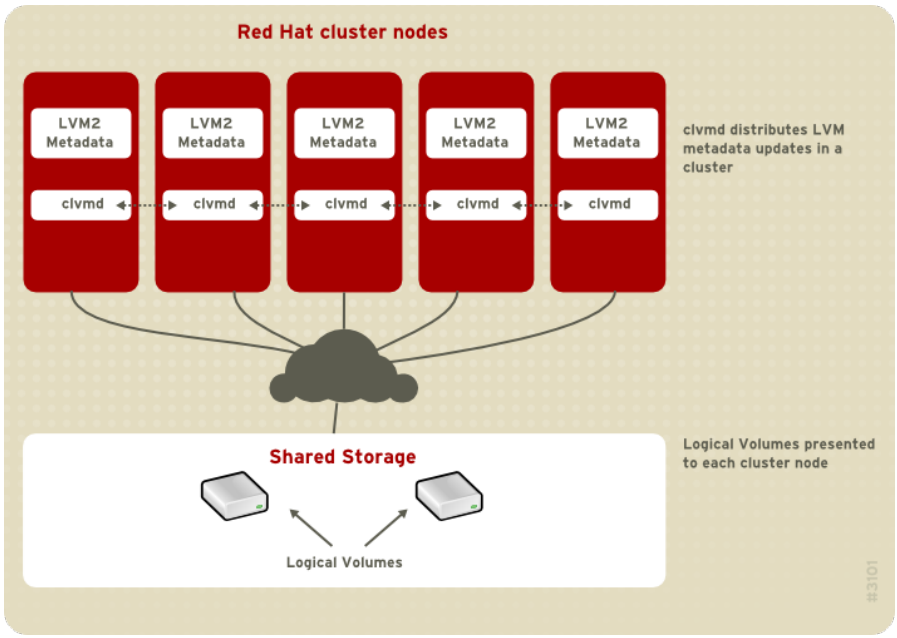

그림 1.2. CLVM 개요

참고

CLVM은 클러스터 전역 잠금 기능을 위해 **lvm.conf** 파일로 변경해야 합니다. 클러스터 잠금 기능을 지원하기 위해 **lvm.conf** 파일을 설정하는 방법은 **lvm.conf** 파일 자체 내에서 제공합니다. **lvm.conf** 파일에 관한 내용은 부록 B. *LVM 설정 파일* 에서 살펴보시기 바랍니다.

1.5. 문서 개요

이 문서의 나머지 부분에서는 다음과 같은 내용을 다루고 있습니다:

- 2장. *LVM 구성 요소* 에서는 LVM 논리 볼륨을 구성하는 구성 요소를 설명합니다.
- 3장. *LVM 관리 개요* 에서는 LVM CLI (Command Line Interface) 명령이나 LVM GUI (Graphical User Interface)를 사용하여 LVM 논리 볼륨을 설정하기 위해 실행해야 하는 기본적인 단계에 대한 개요를 설명합니다.
- 4장. *CLI 명령을 사용한 LVM 관리* 에서는 논리 볼륨을 생성 및 관리하기 위해 LVM CLI 명령으로 실행할 수 있는 개별적 관리 작업 내용을 요약 설명합니다.
- 5장. *LVM 설정 예* 에서는 다양한 LVM 설정 예를 설명합니다.
- 6장. *LVM 문제 해결* 에서는 다양한 LVM 문제 해결을 위한 방법을 설명합니다.
- 7장. *LVM GUI를 통한 LVM 관리*에서는 LVM GUI 실행에 관해 요약 설명합니다.

- 부록 A. *장치 매퍼 (Device Mapper)*에서는 논리 볼륨 및 물리 볼륨을 맵핑하기 위해 LVM이 사용하는 장치 맵퍼 (Device Mapper)를 설명합니다.
- 부록 B. *LVM 설정 파일* 에서는 LVM 설정 파일을 설명합니다.
- 부록 C. *LVM 객체 태그* 에서는 LVM 대상 태그 및 호스트 태그를 설명합니다.
- 부록 D. *LVM 볼륨 그룹 메타데이터* 에서는 LVM 볼륨 그룹 메타데이터를 설명하며, LVM 볼륨 그룹에 해당하는 메타 데이터 견본 복사본이 포함되어 있습니다.

2장. LVM 구성 요소

다음 부분에서는 LVM 논리 볼륨의 구성 요소에 대해 설명합니다.

2.1. 물리 볼륨

LVM 논리 볼륨의 기본적인 물리 스토리지 단위는 파티션 또는 전체 디스크와 같은 블록 장치입니다. LVM 논리 볼륨 용으로 장치를 사용하려면 물리 볼륨 (PV)으로 초기화해야 합니다. 물리 볼륨으로 블록 장치를 초기화하면 장치 시작 부근에 레이블이 위치하게 됩니다.

기본값으로 LVM 레이블은 두 번째 512 바이트 섹터에 위치하게 됩니다. 처음 4 개의 섹터 중 아무곳에 레이블을 두어 이러한 기본값을 덮어쓸 수 있습니다. 이는 LVM 볼륨이 섹터의 다른 사용자와 공존하게 합니다.

시스템 부팅시 순서없이 장치가 나열되었지만 LVM 레이블로 물리 장치를 올바르게 식별하고 장치를 순서대로 나열할 수 있습니다. LVM 레이블은 재부팅 후에도 클러스터를 통해 지속적으로 남아있게 됩니다.

LVM 레이블로 LVM 물리 볼륨 장치를 식별합니다. 이에는 물리 볼륨에 해당하는 임의 고유 식별자 (UUID)가 있으며, 바이트 단위로 블록 장치 크기를 저장하고 장치 상에서 LVM 메타 데이터를 저장할 장소를 기록합니다.

LVM 메타데이터에는 시스템에 있는 LVM 볼륨 그룹의 설정 정보가 들어 있습니다. 기본값으로 메타데이터의 동일한 복사본은 볼륨 그룹 안의 모든 물리 볼륨에 있는 모든 메타 데이터 영역에 보존됩니다. LVM 메타데이터는 ASCII로 저장됩니다.

현재 LVM은 각각의 물리 볼륨 상에 메타 데이터의 동일한 복사본 0, 1 또는 2개를 저장하게 합니다. 기본값은 복사본 1개입니다. 물리 볼륨에 메타데이터 사본 수를 설정하면 나중에 이를 변경할 수 없습니다. 첫 번째 사본은 레이블 바로 다음의 장치 시작 부분에 저장됩니다. 두 번째 사본이 있을 경우, 이는 장치 마지막 부분에 위치하게 됩니다. 실수로 다른 디스크에 작성하여 디스크 시작 부분이 덮어 쓰기되었을 경우, 장치 마지막에 있는 메타데이터의 두 번째 사본이 메타 데이터를 복구하게 합니다.

LVM 메타 데이터 및 메타 데이터 매개변수 변경에 관한 자세한 내용은 부록 D. *LVM 볼륨 그룹 메타데이터*에서 참조하시기 바랍니다.

2.1.1. LVM 물리 볼륨 레이아웃

그림 2.1. “물리적 볼륨 레이아웃” 에서는 LVM 물리 볼륨의 레이아웃을 보여주고 있습니다. LVM 레이블은 두 번째 섹터에 위치하게 되며, 다음으로 메타 데이터 영역이 있고, 그 다음으로 장치에서 사용 가능한 공간이 있게됩니다.

참고

Linux 커널에서 (이 문서 전반에 걸쳐), 섹터는 512 바이트 크기로 되어 있다고 간주합니다.

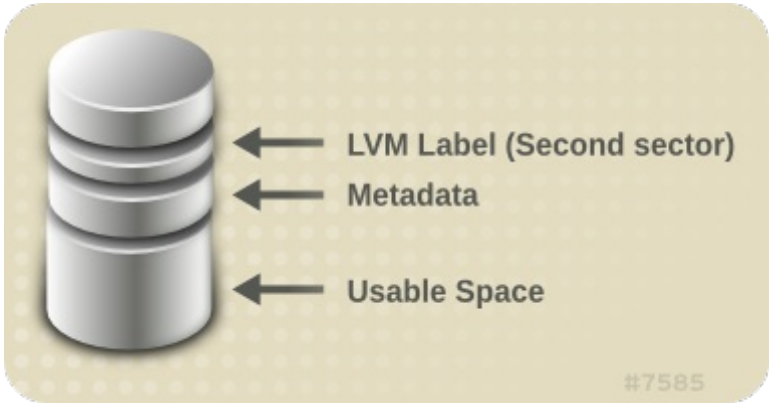

그림 2.1. 물리적 볼륨 레이아웃

2.1.2. 디스크에서 다중 파티션

LVM은 디스크 파티션에서 물리 볼륨을 생성하게 합니다. 일반적으로 다음과 같은 이유로 LVM 물리 볼륨으로 레이블하기 위해 전체 디스크를 커버하는 단일 파티션을 생성할 것을 권장합니다:

- 관리 용이

  실제 디스크가 한 번만 나타날 경우 시스템의 하드웨어를 추적하기 쉽습니다. 특히 디스크에 문제가 있을 경우 실제로 그러합니다. 또한, 단일 디스크에 있는 다중 물리 볼륨은 부팅시 알려지지 않은 파티션 유형에 대해 커널 경고를 할 수 있습니다.

- 스트라이핑 실행

  LVM은 두 개의 물리 볼륨이 동일한 물리 디스크에 있다고 알려줄 수 없습니다. 두 개의 물리 볼륨이 동일한 물리 디스크에 있을 때 스트라이프 논리 볼륨을 생성할 경우, 이는 같은 디스크에 있는 다른 파티션에 위치하게 되어 실행 기능을 저하시킵니다.

권장 사항은 아니지만, 디스크를 별개의 LVM 물리 볼륨으로 나누어야 하는 특정한 경우가 있을 수 있습니다. 예를 들어, 기존 시스템을 LVM 볼륨으로 이전해야 할 때 디스크가 있는 시스템에서 파티션의 데이터를 옮겨야 할 수 도 있고, 또한 용량이 큰 디스크가 있어 관리를 위해 하나 이상의 볼륨 그룹을 만들고자 할 경우 디스크를 파티션해야 합니다. 디스크에 한 개 이상의 파티션이 있고 이러한 파티션이 같은 볼륨 그룹에 있을 경우, 스트라이프 볼륨을 생성할 때 어떤 파티션을 논리 볼륨에 포함시킬지를 지정해야 합니다.

2.2. 볼륨 그룹

물리 볼륨은 볼륨 그룹 (VG)으로 통합됩니다. 이는 논리 볼륨을 할당할 수 있는 디스크 공간의 풀을 생성합니다.

볼륨 그룹에서 할당을 위한 디스크 공간은 익스텐트라는 고정된 크기위 단위로 나뉘어 집니다. 익스텐트는 물리 볼륨에서 공간을 할당할 수 있는 가장 작은 단위로, 이를 물리 익스텐트라고도 부릅니다.

논리 볼륨은 물리 익스텐트와 같은 크기의 논리 익스텐트로 할당됩니다. 따라서 익스텐트 크기는 볼륨 그룹에 있는 모든 논리 볼륨과 같습니다. 볼륨 그룹은 논리 익스텐트를 물리 익스텐트로 맵핑합니다.

2.3. LVM 논리적 볼륨

LVM에서 볼륨 그룹은 논리 볼륨으로 나뉘어 집니다. *선형 (linear)* 볼륨, *스트라이프 (striped)* 볼륨, *미러 (mirrored)* 볼륨이라는 세 가지 유형의 LVM 논리 볼륨이 있습니다. 다음 부분에서는 이에 대해 설명합니다.

2.3.1. 선형 (Linear) 볼륨

선형 (linear) 볼륨은 하나의 논리 볼륨으로 물리 볼륨을 모읍니다. 예를 들어, 두 개의 60GB 디스크가 있을 경우, 한 개의 120GB 논리 볼륨을 생성할 수 있습니다. 물리 스토리지는 연결되어 집니다.

선형 볼륨을 생성하여 논리 볼륨 물리 익스텐트 영역을 순서대로 지정합니다. 예를 들어, 그림 2.2. "익스텐트 맵핑 " 에서 보여주듯이 1에서 99까지의 논리 익스텐트는 하나의 물리 볼륨으로 맵핑할 수 있고 100에서 198까지의 논리 익스텐트는 두 번째 물리 볼륨으로 맵핑할 수 있습니다. 어플리케이션의 관점에서는 198 익스텐트 크기로된 하나의 장치만이 있게 됩니다.

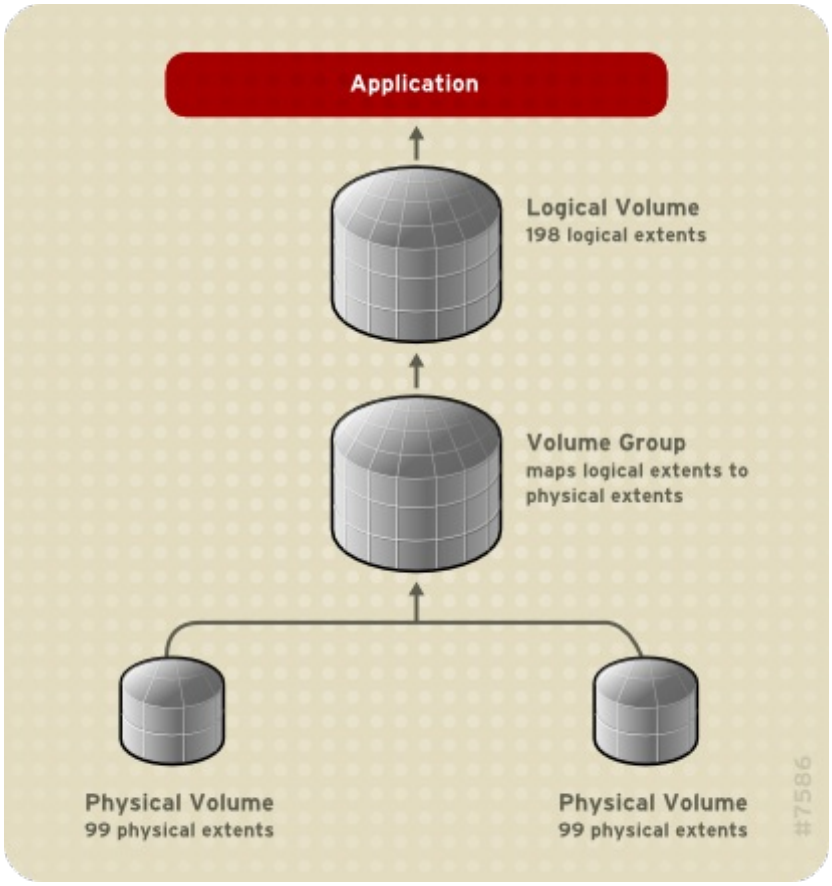

그림 2.2. 익스텐트 맵핑

논리 볼륨으로 이루어진 물리 볼륨이 같은 크기일 필요는 없습니다. 그림 2.3. "동일하지 않은 물리 볼륨이 있는 선형 (Linear) 볼륨 "에서는 4MB의 물리 익스텐트 크기와 함께 **VG1** 볼륨 그룹을 보여주고 있습니다. 이러한 볼륨 그룹에는 **PV1** 및 **PV2**라는 두 개의 물리 볼륨이 들어 있습니다. 익스텐트 크기로 물리 볼륨은 4MB 단위로 나뉘어 집니다. 예에서, **PV1**은 100 익스텐트 크기 (400MB)로 **PV2**는 200 익스텐트 크기 (800MB)로 되어 있습니다. 1에서 300 사이의 익스텐트 크기 (4MB to 1200MB)로 선형 볼륨을 만들 수 있습니다. 예에서는 **LV1**이라는 선형 볼륨이 300 익스텐트 크기로 되어 있습니다.

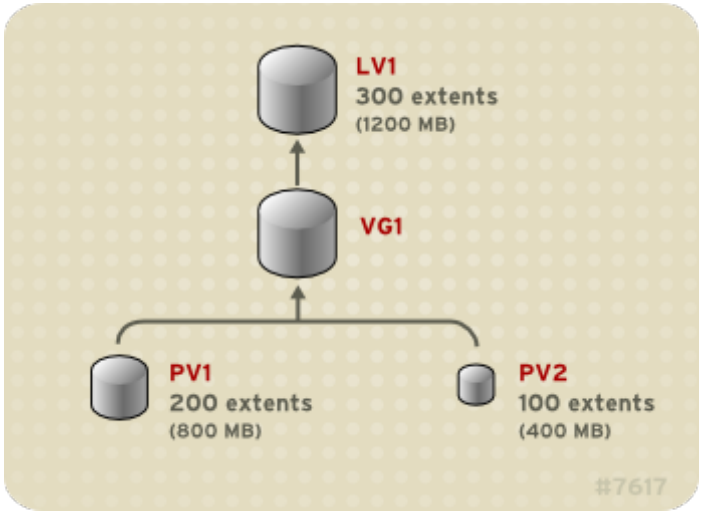

그림 2.3. 동일하지 않은 물리 볼륨이 있는 선형 (Linear) 볼륨

물리 익스텐트 풀에서 원하는 크기의 선형 논리 볼륨을 하나 이상 설정할 수 있습니다. 그림 2.4. "다중 논리 볼륨 "에서는 그림 2.3. "동일하지 않은 물리 볼륨이 있는 선형 (Linear) 볼륨 "에서와 동일한 볼륨 그룹을 보여주고 있지만, 이러한 경우 두개의 논리 볼륨이 볼륨 그룹에서 나뉘어 졌습니다: 250 익스텐트 크기 (1000MB)로 된 **LV1** 및 50 익스텐트 크기 (200MB)로 된 **LV2**.

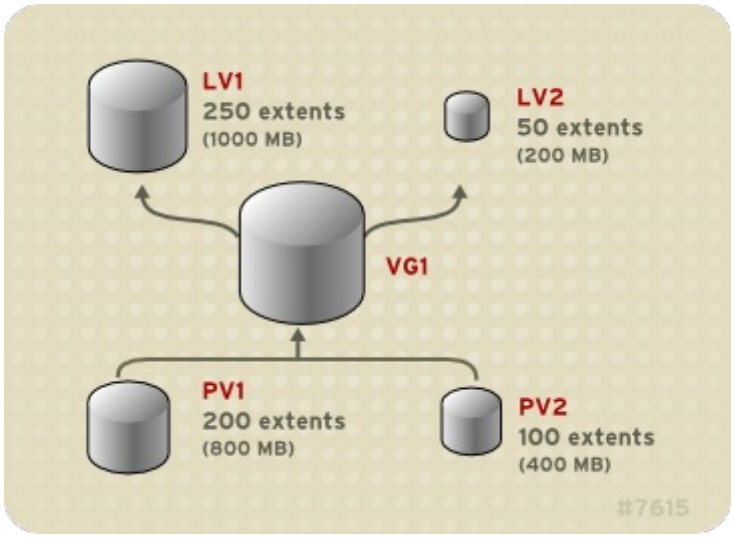

그림 2.4. 다중 논리 볼륨

2.3.2. 스트라이프 (Striped) 논리 볼륨

LVM 논리 볼륨에 데이터를 기록할 때, 파일 시스템은 기본적인 물리 볼륨에 데이터를 기록합니다. 스트라이프 (striped) 논리 볼륨을 생성하여 데이터가 물리 볼륨에 기록되는 방식을 조절할 수 있습니다. 대량의 순차적 읽기 및 쓰기 작업의 경우, 데이터 I/O의 효율성을 향상시킬 수 있습니다.

스트라이핑은 라운드 라운드 (round-round) 방식에서 미리 지정된 물리 볼륨의 수에 데이터를 작성하여 성능을 향상시킵니다. 스트라이핑을 사용하여 I/O는 병렬로 실행될 수 있습니다. 이는 경우에 따라 스트라이프에 있는 각각의 추가 물리 볼륨에 대해 선형에 가까운 성능 향상을 초래할 수 있습니다.

다음의 그림에서는 세 개의 물리 볼륨에 걸쳐 스트라이프된 데이터를 보여주고 있습니다.:

- 데이터의 첫 번째 스트라이프는 PV1에 작성됩니다

- 데이터의 두 번째 스트라이프는 PV2에 작성됩니다
- 데이터의 세 번째 스트라이프는 PV3에 작성됩니다
- 데이터의 네 번째 스트라이프는 PV1에 작성됩니다

스트라이프 논리 볼륨에서 스트라이프의 크기는 익스텐트의 크기를 초과할 수 없습니다.

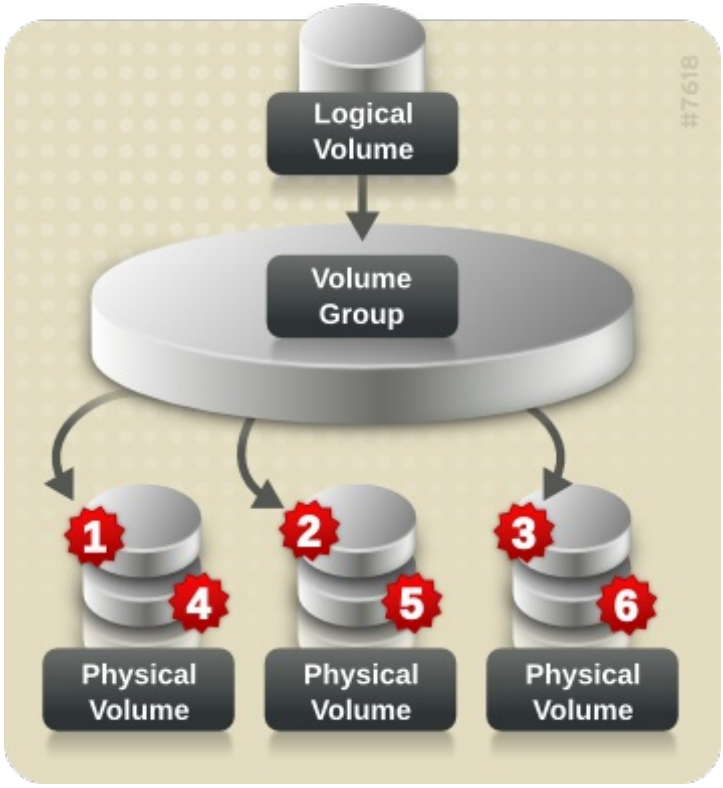

그림 2.5. 세 개의 PV를 통한 스트라이핑 데이터

스트라이프 논리 볼륨은 첫 번째 세트의 마지막에 다른 장치 세트를 연결하여 확장될 수 있습니다. 하지만, 스트라이프 논리 볼륨을 확장하려면, 스트라이프를 지원하기 위해 볼륨 그룹으로 된 기본적인 물리 볼륨에 충분한 여유 공간이 있어야 합니다. 예를 들어, 전체 볼륨 그룹에 사용할 양방향 스크라이프 볼륨이 있을 경우, 볼륨 그룹에 단일 물리 볼륨을 추가하여 스트라이프 볼륨을 확장할 수 없습니다. 대신, 볼륨 그룹에 최소 두 개의 물리 볼륨을 추가해야 합니다. 스트라이프 볼륨을 확장하는 방법에 관한 자세한 내용은 4.4.12.1절. “스트라이프 볼륨 확장 ”에서 참조하시기 바랍니다.

2.3.3. 미러 (Mirrored) 논리 볼륨

미러는 다른 장치에 있는 데이터의 동일한 복사본을 저장합니다. 데이터가 하나의 장치에 기록되었을 때, 이 데이터는 미러되어 두 번째 장치에도 기록됩니다. 이는 장치에 장애가 발생할 경우 데이터를 보호하게 됩니다. 하나의 미러 leg에 장애가 발생할 경우, 논리 볼륨은 선형 볼륨으로 되어 액세스 가능하게 됩니다.

LVM은 미러 볼륨을 지원합니다. 미러 논리 볼륨을 생성할 때 LVM은 기본적인 물리 볼륨에 기록된 데이터가 별개의 물리 볼륨으로 미러되었는 지를 확인합니다. LVM으로 다중 미러를 사용하여 미러 논리 볼륨을 생성할 수 있습니다.

LVM 미러는 일반적으로 512KB 크기로된 영역으로 복사되도록 장치를 나눕니다. LVM은 어떤 부분이 미러를 사용하여 동기화되었는 지를 추적하기 위해 사용된 로그를 보관합니다. 이러한 로그는 재부팅 후에도 없어지지 않도록 디스크나 메모리로 보관될 수 있습니다.

그림 2.6. “미러 (Mirrored) 논리 볼륨 ”에서는 하나의 미러로 된 미러 논리 볼륨을 보여줍니다. 이러한 설정에서 로그는 디스크에 저장됩니다.

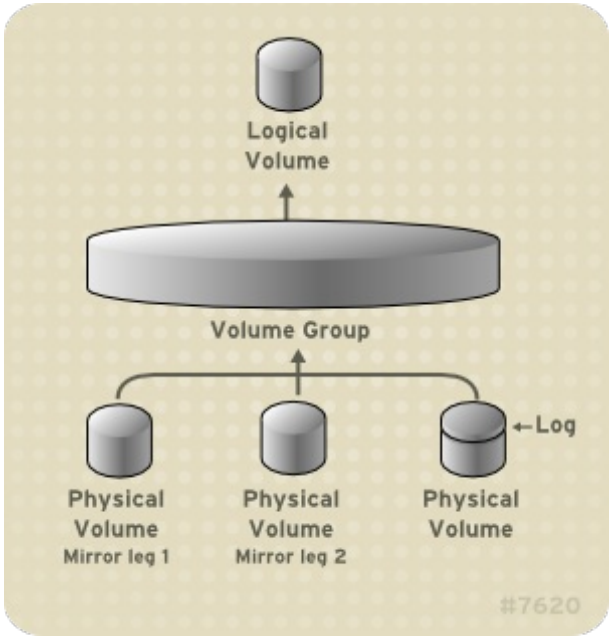

그림 2.6. 미러 (Mirrored) 논리 볼륨

미러 생성 및 수정에 관한 내용은 4.4.3절. “미러 볼륨 생성 ”에서 참조하시기 바랍니다.

2.3.4. 스냅샷 볼륨

LVM 스냅샷 기능은 서비스의 장애를 초래하지 않고 특정 경우에 장치의 가상 이미지를 생성하게 합니다. 스냅샷을 찍은 후 원래의 장치에 변경 사항이 있을 경우, 스냅샷 기능은 변경 되기 이전에 변경된 데이터 영역의 복사본을 만들어 장치의 상태를 재생할 수 있습니다.

참고

LVM 스냅샷은 클러스터에 있는 노드에서 지원되지 않습니다. 클러스터 볼륨 그룹에서 스냅샷 볼륨을 생성할 수 없습니다.

참고

LVM 스냅샷은 LVM 미러 논리 볼륨을 지원하지 않습니다.

스냅샷이 생성된 후에 이는 변경된 데이터 영역만을 복사하기 때문에, 스냅샷 기능에는 최소 스토리지 용량이 있어야 합니다. 예를 들어, 자주 업데이트되지 않는 원본의 경우, 원본 용량의 3-5 %면 스냅샷을 보관하기에 충분합니다.

참고

파일 시스템의 스냅샷 복사본은 가상 복사본으로 파일 시스템에 대한 실제적인 미디어 백업이 아닙니다. 스탭샷은 백업 절차에 대한 대체 기능을 제공하지 않습니다.

스냅샷 크기는 본래 볼륨에 변경된 사항을 저장하기 위해 제외시켜 둔 공간을 제어합니다. 예를 들어, 스냅샷

을 만들어 본래 볼륨을 완전하게 덮어쓰기할 경우, 스냅샷은 변경 사항을 저장하기 위해 최소 본래 볼륨만큼 크게 됩니다. 예상되는 변경 수준에 따라 스냅샷을 측정해야 합니다. 즉, **/usr**과 같은 대부분의 읽기 전용 볼륨의 일시적인 스냅샷은 **/home**과 같이 쓰기 작업이 많은 오래 보관되는 스냅샷 볼륨 보다 더 적은 공간이 필요하게 됩니다.

스냅샷 볼륨이 꽉찼을 경우, 스냅샷은 정지됩니다. 이는 본래 볼륨에 있는 변경 사항을 추적할 수 없기 때문입니다. 스냅샷의 크기를 수시로 확인해야 합니다. 스냅샷은 크기를 재조정할 수 있지만, 스토리지 용량이 있을 경우, 스냅샷이 정지되지 않도록 스냅샷 볼륨의 크기를 증가시킬 수 있습니다. 반대로 필요보다 스냅샷 볼륨이 클 경우, 다른 논리 볼륨에 사용할 수 있도록 여유 공간을 두기 위해 스냅샷 볼륨 크기를 줄일 수 도 있습니다.

스냅샷 파일 시스템을 생성할 때, 원래 파일 시스템에 완전 읽기 및 쓰기 액세스를 할 수 있습니다. 스냅샷에서 일부분이 변경될 경우, 변경된 부분이 표시되고 원래 볼륨에서 복사되지 않습니다.

다음에서는 스냅샷 기능 사용법을 설명합니다:

- 대부분 지속적으로 데이터를 업데이트하는 라이브 시스템을 중지시키지 않고 논리 볼륨에서 백업을 실행해야 할 경우 스냅샷을 찍습니다.
- 스냅샷 파일 시스템에서 **fsck** 명령을 실행하여 파일 시스템 무결성을 확인하고 본래의 파일 시스템이 파일 시스템 복구를 필요로하는 지에 대해 결정할 수 있습니다.
- 스냅샷은 읽기/쓰기가 되기 때문에, 스냅샷을 찍어 테스트를 실행하고, 실제 데이터는 보존하여 프로덕션 데이터에 대한 어플리케이션을 테스트할 수 있습니다.
- Red Hat 가상화와 함께 사용하기 위해 LVM 볼륨을 만들 수 있습니다. LVM 스냅샷을 사용하여 가상 게스트 이미지의 스냅샷을 생성할 수 있습니다. 이러한 스냅샷은 최소한의 추가 스토리지로 새 게스트를 만들거나 기존 게스트를 수정하기 위한 편리한 방법을 제공합니다. 가상 게스트의 LVM 스냅샷에 대한 보다 자세한 내용은 *Red Hat Enterprise Linux 가상화 가이드*에서 참조하십시오.

스냅샷 볼륨 생성에 대한 내용은 4.4.4절. “스냅샷 볼륨 생성 ”에서 참조하시기 바랍니다.

Red Hat Enterprise Linux 6 릴리즈에서 **lvconvert** 명령의 **--merge** 옵션을 사용하여 스냅샷을 본래 볼륨으로 합칠 수 있습니다. 이 기능을 사용함에 있어서 데이터나 파일이 손상되었을 경우 시스템 롤백을 실행합니다. 그렇지 않을 경우 이전 상태로 시스템을 복구해야 합니다. 스냅샷 볼륨을 합친 후, 논리 볼륨은 본래 볼륨의 이름, 부 버전 번호, UUID를 갖게 되고 합쳐진 스냅샷은 삭제됩니다. 이 옵션 사용에 대한 자세한 내용은 4.4.5절. “스냅샷 볼륨 합치기 ”에서 참조하십시오.

3장. LVM 관리 개요

다음 부분에서는 LVM 논리 볼륨 설정에 사용되는 관리 절차에 대한 개요를 다루고 있습니다. 이는 일반적인 절차 이해를 위한 것입니다. 일반적인 LVM 설정 절차의 단계별 예는 5장. *LVM 설정 예* 에서 참조하시기 바랍니다.

LVM 관리에 사용할 수 있는 CLI 명령에 대한 설명은 4장. *CLI 명령을 사용한 LVM 관리* 에서 참조하시기 바랍니다. 또는 7장. *LVM GUI를 통한 LVM 관리*에 있는 LVM GUI를 참조하셔도 됩니다.

3.1. 클러스터에 LVM 볼륨 생성

클러스터 환경에서 논리 볼륨을 생성하기 위해, LVM에서의 클러스터링 확장 모음인 CLVM (Clustered Logical Volume Manager)을 사용합니다. 이러한 확장으로 컴퓨터의 클러스터가 LVM을 사용하여 공유 스토리지 (예: SAN에서)를 관리하게 합니다. CLVM을 사용하려면, 1.4절. “CLVM (Clustered Logical Volume Manager) ”에서 설명하고 있듯이, 부팅시 **clmvd** 데몬을 포함한 고가용성 추가 기능 및 장애 복구형 스토리지 추가 기능 소프트웨어를 반드시 시작해야 합니다.

클러스터 환경에서 LVM 논리 볼륨을 생성하는 것은 단일 노드에서 LVM 논리 볼륨을 생성하는 것과 동일합니다. LVM 명령 자체나 LVM GUI 인터페이스에서의 다른점은 없습니다. 클러스터에 생성하려는 LVM 볼륨을 활성화하려면, 클러스터 인프라가 반드시 실행되고 있어야 하며 클러스터는 정족수(quorum)에 달해 있어야 합니다.

CLVM은 클러스터 전역 잠금 기능을 위해 **lvm.conf** 파일로 변경해야 합니다. 클러스터 잠금 기능을 지원하기 위해 **lvm.conf** 파일을 설정하는 방법은 **lvm.conf** 파일 자체 내에서 제공합니다. **lvm.conf** 파일에 관한 내용은 부록 B. *LVM 설정 파일* 에서 살펴보시기 바랍니다.

기본값으로 공유 스토리지에서 CLVM과 함께 생성된 논리 볼륨은 공유 스토리지로 액세스하는 모든 컴퓨터에서 볼 수 있습니다. 하지만, 클러스터에 있는 하나의 노드에서만 모든 공유 장치가 보일 경우에도 볼륨 그룹을 생성할 수 있습니다. 또한 볼륨 그룹의 상태를 논리 볼륨 그룹에서 클러스터 볼륨 그룹으로 변경할 수도 있습니다. 보다 자세한 내용은 4.3.2절. “클러스터에서 볼륨 그룹 생성 ” 및 4.3.7절. “볼륨 그룹의 매개 변수 변경 ”에서 확인하시기 바랍니다.

경고

공유 스토리지에 CLVM으로 볼륨 그룹을 생성할 때, 클러스터에 있는 모든 노드가 볼륨 그룹을 구성하는 물리 볼륨에 액세스했는지 확인해야 합니다. 스토리지에 액세스한 일부 노드가 있는 대칭적인 클러스터 설정은 지원되지 않습니다.

고가용성 추가 기능 설치 및 클라스터 인프라를 설정하는 방법에 대한 내용은 *클러스터 관리*에서 참조하시기 바랍니다.

클러스터에 미러 논리 볼륨 생성에 관한 예시는 5.5절. “클러스터에 미러 LVM 논리 볼륨 생성 ”에서 참조하십시오.

3.2. 논리 볼륨 생성에 관한 개요

다음 부분에서는 LVM 논리 볼륨을 생성하기 위한 실행 절차에 대해 요약하여 설명합니다.

1. 물리 볼륨으로서 LVM 볼륨으로 사용할 파티션을 초기화합니다 (이는 파티션을 레이블함)
2. 볼륨 그룹을 생성합니다.
3. 논리 볼륨을 생성합니다.

논리 볼륨을 생성한 후에 파일 시스템을 생성하고 마운트할 수 있습니다. 이 문서의 예에서는 GFS2 파일 시스템을 사용합니다.

참고

GFS2 파일 시스템이 독립형 시스템이나 클러스터 설정 부분으로 구현될 수 있어도, Red Hat Enterprise 6 릴리즈에서 Red Hat은 단일 노드 파일 시스템으로 GFS2 사용을 지원하지 않습니다. Red Hat은 클러스터 파일 시스템의 스냅샷을 마운트하기 위해 (예: 백업 용도) 단일 노드 GFS2 파일 시스템을 지속적으로 지원하게 됩니다.

1. **mkfs.gfs2** 명령을 사용하여 논리 볼륨에 GFS2 파일 시스템을 생성합니다.
2. **mkdir** 명령을 사용하여 새로운 마운트 지점을 생성합니다. 클러스터된 시스템에서 클러스터에 있는 모든 노드에 마운트 지점을 생성합니다.
3. 파일 시스템을 마운트합니다. 시스템에 있는 각각의 노드에 대한 **fstab** 파일에 행을 추가할 수 있습니다.

다른 방법으로, LVM GUI를 사용하여 GFS2 파일 시스템을 생성하고 마운트할 수 있습니다.

LVM 볼륨을 생성하는 것은 LVM 설정 정보에 해당하는 저장 장치가 볼륨이 생성된 곳이 아닌 물리 볼륨에 있기 때문에 장치 독립적입니다. 저장 장치를 사용하는 서버는 로컬 복사본을 갖게 되지만 물리 볼륨에 있는 장치에서 다시 생성할 수 있습니다. LVM 버전이 호환가능할 경우 물리 볼륨을 다른 서버에 부착할 수 있습니다.

3.3. 논리 볼륨에 파일 시스템 늘리기

논리 볼륨에 파일 시스템을 늘리기 위해 다음과 같은 절차를 실행합니다:

1. 새 물리 볼륨을 생성합니다.
2. 새 물리 볼륨을 포함시키기 위해 늘리려는 파일 시스템과 함께 논리 볼륨이 있는 볼륨 그룹을 확장합니다.
3. 새 물리 볼륨을 포함시키기 위해 논리 볼륨을 확장합니다.
4. 파일 시스템을 늘립니다.

볼륨 그룹에 할당되지 않은 충분한 공간이 있을 경우, 1 단계와 2 단계를 실행하지 않고 이 공간을 사용하여 논리 볼륨을 확장할 수 있습니다.

3.4. 논리 볼륨 백업

메타데이터 백업 및 메타데이터 아카이브는 **lvm.conf** 파일에서 비활성화되지 않는 한 모든 볼륨 그룹 및 논리 볼륨 설정 변경 시 자동으로 생성됩니다. 기본값으로 메타데이터 백업은 **/etc/lvm/backup** 파일에 저장되며 메타데이터 아카이브는 **/etc/lvm/archive** 파일에 저장됩니다. **/etc/lvm/archive** 파일에 저장된 메타데이터 아카이브가 얼마나 오래 동안 얼마나 많은 아카이브 파일을 저장할 지는 **lvm.conf** 파일에서 설정할 수 있는 매개변수에 의해 결정됩니다. 매일 실행되는 시스템 백업에는 **/etc/lvm** 디렉토리의 내용이 들어 있어야 합니다.

메타 데이터 백업에서는 논리 볼륨에 있는 사용자 및 시스템 데이터를 백업하지 않음에 유의합니다.

vgcfgbackup 명령을 사용하여 **/etc/lvm/backup** 파일에 메타데이터를 수동으로 백업할 수 있습니다. **vgcfgrestore** 명령으로는 메타데이터를 복구할 수 있습니다. **vgcfgbackup** 및 **vgcfgrestore** 명령은 4.3.12절. “볼륨 그룹 메타 데이터 백업 ”에 설명되어 있습니다.

3.5. 로깅

모든 메세지 출력 결과는 독자적으로 로깅 레벨을 선택하여 로깅 모듈을 통과합니다:

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

로깅 레벨은 부록 B. *LVM 설정 파일* 에 설명된 **/etc/lvm/lvm.conf** 파일에 설정되어 있습니다.

4장. CLI 명령을 사용한 LVM 관리

다음 부분에서는 논리 볼륨을 생성하고 관리하기 위해 LVM CLI (Command Line Interface) 명령으로 실행할 수 있는 개별적 관리 작업에 대해 요약하여 설명합니다.

참고

클러스터 환경 용으로 LVM 볼륨을 생성하거나 수정할 경우, **clvmd** 데몬을 실행하고 있는 지를 확인해야 합니다. 자세한 내용은 3.1절. “클러스터에 LVM 볼륨 생성 ”에서 확인하십시오.

4.1. CLI 명령 사용

LVM CLI 명령에는 몇 가지 일반적인 기능이 있습니다.

명령행 인수에서 크기를 요청할 때 항상 단위를 명확하게 명시해야 합니다. 단위를 명시하지 않을 경우, 주로 KB 또는 MB와 같은 기본값이 단위로 됩니다. LVM CLI 명령은 소수를 허용하지 않습니다.

명령행 인수에서 단위를 지정할 때, LVM은 대소문자 구분을 하지 않습니다; 예를 들어, M 또는 m으로 지정하는 것은 동일하며 2의 배수 (1024의 배수)가 사용됩니다. 하지만, 명령에서 **--units** 인수를 지정하면, 소문자는 1024의 배수로된 단위를 가리키는 반면 대문자는 1000의 배수로된 단위를 가리킵니다.

명령이 볼륨 그룹 또는 논리 볼륨 이름을 인수로 갖는 것에서, 완전 경로명은 옵션으로 됩니다. **vg0** 볼륨 그룹에 있는 **lvol0** 논리 볼륨은 **vg0/lvol0**으로 지정될 수 있습니다. 여기서 볼륨 그룹 목록이 필요하지만 이는 비어있게 되고, 모든 볼륨 그룹 목록은 대체됩니다. 논리 볼륨 목록이 필요하지만 볼륨 그룹이 주어지고, 볼륨 그룹에 있는 모든 논리 볼륨 목록은 대체됩니다. 예를 들어, **lvdisplay vg0** 명령은 **vg0** 볼륨 그룹에 있는 모든 논리 볼륨을 보여줍니다.

모든 LVM 명령은 보다 더 상세한 출력 결과를 얻기 위해 여러번 입력 할 수 있는 **-v** 인수를 허용합니다. 예를 들어, 다음의 예에서는 **lvcreate** 명령의 기본 출력 결과를 보여주고 있습니다.

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

다음의 예에서는 **-v** 인수와 함께 사용된 **lvcreate** 명령의 출력 결과를 보여주고 있습니다.

```
# 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
```

명령 실행에 관한 보다 상세한 출력 결과를 디스플레이하기 위해 **-vv**, **-vvv**, **-vvvv** 인수를 사용할 수 있습니다. **-vvvv** 인수로 최대 출력 결과를 볼 수 있습니다. 다음의 예에서는 **-vvvv** 인수와 함께 사용된 **lvcreate** 명령에 대해 처음 몇 줄의 출력 결과 만을 보여주고 있습니다.

```
# 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
...
```

--help 인수로 LVM CLI 명령에 해당하는 도움말을 볼 수 있습니다.

```
commandname --help
```

명령에 해당하는 맨 페이지를 보려면 **man** 명령을 실행합니다:

```
man commandname
```

man lvm 명령은 LVM에 관한 일반적인 온라인 정보를 보여줍니다.

모든 LVM 객체는 UUID에 의해 내부적으로 참조하게 되며, 이는 객체를 생성할 때 지정됩니다. 볼륨 그룹의 일부분인 **/dev/sdf**라는 물리 볼륨을 삭제하여 할 경우 유용할 수 있으며, 이를 다시 플러그인하면 **/dev/sdk**에서 찾을 수 있습니다. LVM은 장치 이름이 아닌 UUID에 의해 물리 볼륨을 확인하기 때문에 이는 계속 물리 볼륨을 찾게 됩니다. 물리 볼륨을 생성할 때 물리 볼륨의 UUID를 지정하는 방법에 대한 내용은 6.4절. “물리 볼륨 메타 데이터 복구 ”에서 참조하시기 바랍니다.

4.2. 물리 볼륨 관리

다음 부분에서는 물리 볼륨 관리 실행을 위한 명령에 대해 설명합니다.

4.2.1. 물리 볼륨 생성

다음 부분에서는 물리 볼륨 생성에 사용되는 명령에 대해 설명합니다.

4.2.1.1. 파티션 유형 설정

물리 볼륨에 대해 전체 디스크를 사용할 경우, 디스크에는 파티션 테이블이 없어야 합니다. DOS 디스크 파티션의 경우, 파티션 id는 **fdisk** 또는 **cfdisk**를 사용하여 0x8e로 설정되어야 합니다. 전체 디스크 장치의 경우 파티션 테이블만이 삭제되어야 하며, 사실상 이는 디스크 상의 모든 데이터를 삭제하게 됩니다. 다음의 명령을 사용하여 첫 번째 섹터를 제로(zero)로 지정하여 기존 파티션 테이블을 삭제할 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. 초기화 작업은 파일 시스템을 포맷하는 것과 유사합니다.

다음 명령으로 LVM 물리 볼륨으로 사용할 **/dev/sdd1**, **/dev/sde1**, **/dev/sdf1**를 초기화합니다.

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

전체 디스크가 아닌 파티션을 초기화하려면: 파티션에서 **pvcreate** 명령을 실행합니다. 다음 예에서는 LVM 논리 볼륨의 일부분으로 차후에 사용할 LVM 물리 볼륨으로서 **/dev/hdb1** 파티션을 초기화하고 있습니다.

```
pvcreate /dev/hdb1
```

4.2.1.3. 블록 장치 스캐닝

다음의 예에서 볼 수 있듯이 **lvmdiskscan** 명령으로 물리 볼륨으로 사용된 블록 장치를 확인할 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 보기

LVM 물리 볼륨 속성 보기에 사용할 수 명령에는 다음의 세 가지가 있습니다: **pvs**, **pvdisplay**, **pvscan**.

pvs 명령으로 물리 볼륨 당 하나의 행으로 설정 가능한 형식의 물리 볼륨 정보를 알 수 있습니다. **pvs** 명령으로 포맷 제어를 할 수 있고 스트립핑에 유용합니다. 출력 결과를 사용자 설정하기 위해 **pvs** 명령을 사용하는 방법에 대한 내용은 4.8절. “LVM 용 사용자 설정 리포트 ”에서 참조하시기 바랍니다.

pvdisplay 명령으로 각각의 물리 볼륨에 대해 상세 출력을 할 수 있습니다. 이는 고정 포맷으로 물리 볼륨의 속성 (크기, 익스텐트, 볼륨 그룹 등)을 보여줍니다.

다음의 예에서는 단일 물리 볼륨에 대한 **pvdisplay** 명령의 출력 결과를 보여 주고 있습니다

```
# pvdisplay
  --- Physical volume ---
  PV Name               /dev/sdc1
  VG Name               new_vg
  PV Size               17.14 GB / not usable 3.40 MB
  Allocatable           yes
  PE Size (KByte)       4096
  Total PE              4388
  Free PE               4375
  Allocated PE          13
  PV UUID               Joqlch-yWSj-kuEn-IdwM-01S9-XO8M-mcpsVe
```

pvscan 명령으로 물리 볼륨에 해당하는 시스템에 있는 모든 LVM 블록 장치를 스캔합니다.

다음의 명령에서는 검색된 모든 물리 장치를 보여주고 있습니다:

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

lvm.conf에서 필터를 지정하여 특정 물리 볼륨을 스캔하지 않게 할 수 있습니다. 어떤 장치를 스캔할 지를 제어하기 위한 필터 사용에 관한 자세한 내용은 4.5절. “필터로 LVM 장치 스캔 제어 ”에서 참조하시기 바랍니다.

4.2.3. 물리 볼륨에서 할당을 허용하지 않음

pvchange 명령을 사용하여 한 개 이상의 물리 볼륨의 여유 공간에 있는 물리 익스텐트의 할당을 허용하지 않을 수 있습니다. 디스크 오류가 있을 경우나 물리 볼륨을 삭제할 경우 이러한 작업이 필요하게 됩니다.

다음의 명령으로 **/dev/sdk1**에 물리 익스텐트의 할당을 허용하지 않습니다.

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

pvchange 명령의 **-xy** 인수를 사용하여 할당이 허용되지 않은 부분에 대해 할당을 허용할 수 있습니다.

4.2.4. 물리 볼륨 크기 조정

기본 블록 장치의 크기를 변경해야 할 경우, **pvresize** 명령을 사용하여 새로운 크기로 LVM을 업데이트합니다. LVM이 물리 볼륨을 사용하고 있는 동안에도 이 명령을 실행할 수 있습니다.

4.2.5. 물리 볼륨 삭제

장치를 더이상 LVM으로 사용할 필요가 없을 경우, **pvremove** 명령으로 LVM 레이블을 삭제할 수 있습니다. **pvremove** 명령을 실행하면 빈 물리 볼륨에 있는 LVM 메타 데이터를 제로(zero)로 만듭니다.

삭제하려는 물리 볼륨이 볼륨 그룹의 일부분일 경우, 4.3.6절. “볼륨 그룹에서 물리 볼륨 삭제 ”에서 설명하고 있듯이 **vgreduce** 명령으로 볼륨 그룹에서 이를 삭제해야 합니다.

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

4.3. 볼륨 그룹 관리

다음 부분에서는 볼륨 그룹 관리를 실행하기 위한 명령에 대해 설명합니다.

4.3.1. 볼륨 그룹 생성

한 개 이상의 물리 볼륨에서 볼륨 그룹을 생성하려면 **vgcreate** 명령을 사용합니다. **vgcreate** 명령으로 새 볼륨 그룹을 생성하고 최소 하나의 물리 볼륨을 추가합니다.

다음 명령으로 **/dev/sdd1** 및 **/dev/sde1** 물리 볼륨이 들어있는 **vg1**라는 볼륨 그룹을 생성합니다.

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

볼륨 그룹을 생성하기 위해 물리 볼륨을 사용한 경우, 기본값으로 디스크 공간이 4MB 익스텐트로 나뉘어집니다. 이러한 익스텐트는 크기에 있어서 논리 볼륨이 확장 또는 축소될 수 있는 최소 크기입니다. 익스텐트의 크기가 클 경우 논리 볼륨의 I/O 실행에 영향을 미치지 않습니다.

기본값 익스텐트 크기가 적합하지 않을 경우, **vgcreate** 명령에 **-s** 옵션을 사용하여 익스텐트 크기를 지정할 수 있습니다. **vgcreate** 명령의 **-p** 및 **-l** 인수를 사용하여 볼륨 그룹이 갖을 수 있는 물리 볼륨 또는 논리 볼륨의 수에 제한을 둘 수 있습니다.

기본값으로 볼륨 그룹은 동일한 물리 볼륨에 병렬 스트라이프를 배치하지 않는 것과 같은 일반적인 규칙에 따라 물리 익스텐트를 할당합니다. 이는 **normal** 할당 정책입니다. **vgcreate** 명령의 **--alloc** 인수를 사용하여 **contiguous**, **anywhere**, **cling**의 할당 정책을 지정할 수 있습니다.

contiguous 정책에서는 새 익스텐트가 기존 익스텐트에 근접해야 합니다. 할당 요청을 만족시키기에 충분한 여유 익스텐트가 있지만 **normal** 할당 정책에서는 이를 사용하고자 하지 않을 경우, 같은 물리 볼륨에 두 개의 스트라이프가 위치하여 실행 속도가 줄어들어도 **anywhere** 할당 정책이 이를 사용합니다. **cling** 정책은 논리 볼륨의 동일한 스트라이프에 있는 기존 익스텐트와 같은 물리 볼륨에 있는 새 익스텐트를 위치하게 합니다. 이러한 정책은 **vgchange** 명령으로 변경될 수 있습니다.

cling 정책을 LVM 태그와 함께 사용하여 LVM 볼륨을 확장할 때 사용하기 위한 추가 물리 볼륨을 지정하는 방법에 대한 내용은 4.4.12.2절. “**cling** 할당 정책을 사용하여 논리 볼륨 확장 ”에서 참조하십시오.

일반적으로 **normal** 외의 할당 정책은 특별한 형태로 익스텐트를 할당해야 하는 것과 같이 특별한 경우에만 요청됩니다.

LVM 볼륨 그룹 및 기본 논리 볼륨은 다음과 같은 레이아웃으로 **/dev** 디렉토리에 있는 장치 특수 파일 디렉토리 트리에 포함됩니다.

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

예를 들어, 각각의 볼륨 그룹에 **lvo1**, **lvo2**, **lvo3**라는 세 개의 논리 볼륨을 갖는 **myvg1** 및 **myvg2**라는 두 개의 볼륨 그룹을 생성할 경우, 여섯 개의 장치 특수 파일이 생성됩니다:

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

64 비트 CPU에서 LVM의 최대 장치 크기는 8 EB입니다.

4.3.2. 클러스터에서 볼륨 그룹 생성

vgcreate 명령으로 단일 노드에 생성하는 것과 같이 클러스터 환경에 볼륨 그룹을 생성합니다.

기본값으로 공유 스토리지에서 CLVM으로 생성된 볼륨 그룹은 공유 스토리지로 액세스하는 모든 컴퓨터에서 볼 수 있습니다. 하지만 **vgcreate** 명령의 **-c n**을 사용하여 클러스터에 있는 하나의 노드에서만 볼 수 있는 로컬 볼륨 그룹을 생성할 수 있습니다.

다음과 같은 명령은 클러스터 환경에서 실행하면 명령이 실행된 곳에서 노드로 로컬 볼륨 그룹을 생성합니다. 이는 **/dev/sdd1** 및 **/dev/sde1** 물리 볼륨이 들어있는 **vg1**라는 로컬 볼륨을 생성합니다.

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

기존 볼륨 그룹이 로컬로 또는 **vgchange** 명령의 **-c** 옵션으로 클러스터할 지를 변경할 수 있습니다. 자세한 내용은 4.3.7절. “볼륨 그룹의 매개 변수 변경 ”에서 설명하고있습니다.

vgs 명령을 사용하여 기존 볼륨 그룹이 클러스터화된 볼륨 그룹인지를 확인할 수 있으며, 볼륨이 클러스터되어 있을 경우 이는 **c** 속성을 표시합니다. 다음 명령은 **VolGroup00** 및 **testvg1** 볼륨 그룹의 속성을 표시합니다. 예에서**VolGroup00**는 클러스터되어 있지 않는 반면, **testvg1**는 **Attr** 헤딩 아래 **c** 속성에 의해 표시되어 클러스터됩니다.

```
[root@doc-07]# vgs
  VG            #PV #LV #SN Attr   VSize  VFree
  VolGroup00      1   2   0 wz--n- 19.88G    0
  testvg1         1   1   0 wz--nc 46.00G 8.00M
```

vgs 명령에 대한 보다 자세한 내용은 4.3.4절. “볼륨 그룹 보기 ” 4.8절. “LVM 용 사용자 설정 리포트 ”, 및 **vgs** 맨 페이지를 참조하십시오.

4.3.3. 볼륨 그룹에 물리 볼륨 추가

기존 볼륨 그룹에 물리 볼륨을 추가하려면 **vgextend** 명령을 사용합니다. **vgextend** 명령으로 하나 이상의 여유 물리 볼륨을 추가하여 볼륨 그룹의 용량을 늘립니다.

다음 명령으로 볼륨 그룹 **vg1**에 물리 볼륨 **/dev/sdf1**을 추가합니다.

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

4.3.4. 볼륨 그룹 보기

LVM 볼륨 그룹의 속성을 보려면 다음과 같은 두 가지 명령을 사용할 수 있습니다: **vgs** 및 **vgdisplay**

vgscan 명령으로 볼륨 그룹에 해당하는 모든 디스크를 스캔하고 LVM 캐시 파일을 다시 작성할 수 도 있지만 볼륨 그룹을 볼 수 도 있습니다. **vgscan** 명령에 관한 내용은4.3.5절. “캐시 파일 작성을 위해 볼륨 그룹에 해당하는 디스크 보기 ”에서 참조하시기 바랍니다.

vgs 명령은 볼륨 그룹 당 하나의 행을 보여주어 설정 가능한 형식으로 볼륨 그룹 정보를 제공합니다. **vgs** 명령으로 많은 포맷을 제어할 수 있고 스크립팅에 유용하게 사용할 수 있습니다. 출력 결과를 사용자 설정하기 위한 **vgs** 명령 사용에 관한 내용은 4.8절. “LVM 용 사용자 설정 리포트 ”에서 참조하시기 바랍니다.

vgdisplay 명령으로 고정된 형식으로 볼륨 그룹 속성 (예: 크기, 익스텐트, 물리 볼륨 수 등)을 볼 수 있습니다. 다음 예에서는 **new_vg** 볼륨 그룹에 해당하는 **vgdisplay** 명령의 출력 결과를 보여 주고 있습니다. 볼륨 그룹을 지정하지 않을 경우, 기존의 모든 볼륨 그룹이 나타납니다.

```
# 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
```

4.3.5. 캐시 파일 작성을 위해 볼륨 그룹에 해당하는 디스크 보기

vgscan 명령으로 LVM 물리 볼륨 및 볼륨 그룹을 찾는 시스템에서 지원되는 모든 디스크 장치를 스캔합니다. 이는 현재 LVM 장치 목록을 관리하는 **/etc/lvm/.cache**에 LVM 캐시 파일을 작성합니다.

LVM은 시스템 시작시 또는 LVM이 실행되는 동안 즉, **vgcreate** 명령을 실행할 때나 LVM이 불일치되는 사항을 발견하였을 때 **vgscan** 명령을 자동으로 실행합니다.

참고

하드웨어 설정을 변경하거나 노드에서 장치를 추가 또는 삭제할 때 **vgscan** 명령을 수동으로 실행해야 할 수 있습니다. 이는 시스템 부팅시 나타나지 않는 시스템의 새로운 장치를 볼 수 있게 하기 위함입니다. 예를 들어, SAN 상의 시스템에 새 디스크를 추가할 때나 또는 물리 볼륨으로 레이블된 새 디스크를 핫플러그 (hotplug)할 때 이러한 기능이 필요할 수 도 있습니다.

특수 장치를 스캔하지 않도록 제한을 두기 위해 **lvm.conf** 파일에 필터를 지정할 수 있습니다. 장치 스캔 제어를 위한 필터 사용에 관한 내용은 4.5절. “필터로 LVM 장치 스캔 제어 ”에서 참조하시기 바랍니다.

다음 예에서는 **vgscan** 명령의 출력 결과를 보여주고 있습니다.

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

4.3.6. 볼륨 그룹에서 물리 볼륨 삭제

볼륨 그룹에서 사용되지 않는 물리 볼륨을 삭제하려면 **vgreduce** 명령을 사용합니다. **vgreduce** 명령으로 하나 이상의 비어있는 물리 볼륨을 삭제하여 볼륨 그룹 용량을 줄입니다. 이는 다른 볼륨 그룹에서 사용되도록 또는 시스템에서 삭제되도록 물리 볼륨을 삭제합니다.

볼륨 그룹에서 물리 볼륨을 삭제하기 전에 **pvdisplay** 명령으로 물리 볼륨이 다른 논리 볼륨에 의해 사용되지 않는 지를 확인할 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 사용되고 있을 경우 **pvmove** 명령으로 데이터를 다른 물리 볼륨으로 옮겨야 합니다. 그 후 **vgreduce** 명령을 사용하여 물리 볼륨을 삭제합니다:

다음 명령으로 **my_volume_group** 볼륨 그룹에서 **/dev/hda1** 물리 볼륨을 삭제합니다.

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

4.3.7. 볼륨 그룹의 매개 변수 변경

4.3.8절. “볼륨 그룹 활성화 및 비활성화 ”에 설명되어 있듯이, **vgchange** 명령은 볼륨 그룹을 활성화 및 비활성화하기 위해 사용됩니다. 이 명령을 사용하여 기존 볼륨 그룹의 여러 볼륨 그룹 매개 변수를 변경할 수 있습니다.

다음 명령으로 **vg00** 볼륨 그룹의 논리 볼륨을 최대 128까지로 변경할 수 있습니다.

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

볼륨 그룹 매개 변수 설명에 대해 **vgchange** 명령으로 변경할 수 있습니다. **vgchange**(8) 맨 페이지를 참조하시기 바랍니다.

4.3.8. 볼륨 그룹 활성화 및 비활성화

볼륨 그룹을 생성하면 이는 기본값으로 활성화됩니다. 즉 볼륨 그룹에 있는 논리 볼륨은 액세스 가능하고 변경될 가능성이 있습니다.

볼륨 그룹을 비활성화시켜 커널에 알려지지 않게 해야 할 경우가 있습니다. 볼륨 그룹을 비활성화 또는 활성화하려면 **vgchange** 명령의 **-a** (**--available**) 인수를 사용합니다.

다음 예에서는 **my_volume_group** 볼륨 그룹을 비활성화하고 있습니다.

```
vgchange -a n my_volume_group
```

클러스터 잠금 기능이 활성화되어 있을 경우, 'e'를 추가하여 배타적으로 하나의 노드에서 볼륨 그룹을 활성화 또는 비활성화하거나 또는 'l'를 추가하여 로컬 노드에서만 볼륨 그룹을 활성화 또는 비활성화합니다. 단일 호스트 스냅샷이 있는 논리 볼륨은 한 번에 하나의 노드에서만 사용될 수 있으므로 항상 배타적으로 활성화됩니다.

4.4.8절. “논리 볼륨 그룹의 매개 변수 변경 ”에서 설명하듯이 **lvchange** 명령을 사용하여 개별적 논리 볼륨을 비활성화할 수 있습니다. 클러스터에 있는 개별적 노드에서 논리 볼륨을 활성화하는 방법은 4.7절. “클러

스터에 있는 개별적 노드에서 논리 볼륨 활성화 "에서 참조하시기 바랍니다.

4.3.9. 볼륨 그룹 삭제

논리 볼륨이 포함되어 있지 않은 볼륨 그룹을 삭제하려면, **vgremove** 명령을 사용합니다.

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

4.3.10. 볼륨 그룹 나누기

볼륨 그룹의 물리 볼륨을 나누어 새 볼륨 그룹을 생성하려면, **vgsplit** 명령을 사용합니다.

논리 볼륨은 볼륨 그룹에서 분리할 수 없습니다. 각각의 기존 논리 볼륨은 이전 볼륨 그룹이나 새 볼륨 그룹 형태로 논리 볼륨에 있어야 합니다. 하지만, 필요에 따라 **pvmove** 명령을 사용하여 강제로 분리할 수 있습니다.

다음의 예에서는 원래의 볼륨 그룹 **bigvg**에서 새로운 볼륨 그룹 **smallvg**을 나누고 있습니다.

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

4.3.11. 볼륨 그룹 합치기

두 개의 볼륨 그룹을 하나의 단일 볼륨 그룹으로 통합하려면, **vgmerge** 명령을 사용합니다. 볼륨의 물리 익스텐트 크기가 동일하고 양 쪽의 볼륨 그룹의 물리 볼륨 및 논리 볼륨 요약이 destination 볼륨 그룹 제한에 맞을 경우 비활성화 "source" 볼륨을 활성화 또는 비활성화 "destination" 볼륨과 통합할 수 있습니다.

다음의 명령으로 비활성화 볼륨 그룹 **my_vg**를 런타임 상세 정보를 제공하는 활성화 또는 비활성화 볼륨 그룹 **databases**으로 합칠 수 있습니다.

```
vgmerge -v databases my_vg
```

4.3.12. 볼륨 그룹 메타 데이터 백업

메타 데이터 백업 및 메타 데이터 아카이브는 **lvm.conf** 파일에서 비활성화되지 않는 한 모든 볼륨 그룹 및 논리 볼륨 설정 변경 시 자동으로 생성됩니다. 기본값으로 메타 데이터 백업은 **/etc/lvm/backup**에 저장되고 메타 데이터 아카이브는 **/etc/lvm/archives**에 저장됩니다. **vgcfgbackup** 명령으로 **/etc/lvm/backup** 파일에 수동으로 메타 데이터를 백업할 수 있습니다.

vgcfrestore 명령을 사용하여 볼륨 그룹에 있는 아카이브에서 모든 물리 볼륨으로 볼륨 그룹의 메타 데이터를 복구합니다.

vgcfgrestore 명령을 사용하여 물리 볼륨 메타 데이터를 복구하는 예는 6.4절. "물리 볼륨 메타 데이터 복구 "에서 참조하시기 바랍니다.

4.3.13. 볼륨 그룹 이름 변경

vgrename 명령을 사용하여 기존의 볼륨 그룹 이름을 변경합니다.

다음의 명령으로 기존의 **vg02** 볼륨 그룹을 **my_volume_group**으로 이름을 변경합니다.

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

```
vgrename vg02 my_volume_group
```

4.3.14. 다른 시스템으로 볼륨 그룹 이동

전체 LVM 볼륨 그룹을 다른 시스템으로 옮길 수 있습니다. 이를 실행하기 위해 **vgexport** 및 **vgimport** 명령을 사용할 것을 권장합니다.

vgexport 명령은 물리 볼륨을 분리시키는 시스템에 비활성화 볼륨 그룹을 액세스 불가능하게 합니다. **vgexport** 명령으로 이를 비활성화시킨 후 **vgimport** 명령으로 볼륨 그룹이 시스템에 다시 액세스할 수 있게 합니다.

하나의 시스템에서 다른 시스템으로 볼륨 그룹을 이동하려면 다음과 같은 절차를 실행합니다:

1. 볼륨 그룹에서 어떤 사용자도 활성화 볼륨에 있는 파일에 액세스하지 않도록 한 후, 논리 볼륨을 마운트 해제합니다.
2. **vgchange** 명령의 **-a n** 인수를 사용하여 볼륨 그룹을 비활성화시키면, 볼륨 그룹에서 더이상 작업이 이루어지지 않게 됩니다.
3. 볼륨 그룹을 내보내기 위해 **vgexport** 명령을 사용합니다. 삭제된 볼륨 그룹이 시스템에 의해 액세스되지 않게 합니다.

   볼륨 그룹을 내보낸 후, 다음의 예처럼 **pvscan** 명령을 실행할 때 물리 볼륨은 내보내어진 볼륨 그룹에 있는 것처럼 나타납니다.

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

   시스템이 종료되면, 볼륨 그룹으로 구성된 디스크를 언플러그하고 이를 새로운 시스템에 연결할 수 있습니다.
4. 디스크가 새로운 시스템에 플로그될 경우, **vgimport** 명령을 사용하여 볼륨 그룹을 읽어온 후, 새로운 디스크에서 액세스 가능하게 합니다.
5. **vgchange** 명령의 **-a y** 인수를 사용하여 볼륨 그룹을 활성화합니다.
6. 파일 시스템을 마운트하여 사용 가능하게 합니다.

4.3.15. 볼륨 그룹 디렉토리 재생성

vgmknodes 명령을 사용하여 볼륨 그룹 디렉토리 및 논리 볼륨 특정 파일을 재생성합니다. 이러한 명령은 활성 논리 볼륨에 필요한 **/dev** 디렉토리에 있는 LVM2 특정 파일을 확인합니다. 이는 손실된 특정 파일을 생성하고 사용되지 않는 파일을 삭제합니다.

vgscan 명령에 **mknodes** 인수를 지정하여 **vgmknodes** 명령을 **vgscan** 명령과 통합할 수 있습니다.

4.4. 논리 볼륨 관리

다음 부분에서는 다양한 논리 볼륨 관리를 위해 실행되는 명령에 대해 설명합니다.

4.4.1. 선형 논리 볼륨 생성

논리 볼륨을 생성하기 위해 **lvcreate** 명령을 사용하비다. 논리 볼륨에 해당하는 이름을 지정하지 않을 경우, 기본값으로 **lvol#** 이름이 사용되며 여기서 **#**는 논리 볼륨의 내부 번호로 대체합니다.

논리 볼륨 생성 시, 논리 볼륨은 볼륨 그룹으로된 물리 볼륨에 있는 여유 익스텐트를 사용하여 볼륨 그룹에

서 나뉘어 집니다. 일반적으로 논리 볼륨은 기본 물리 볼륨에 있는 여유 공간을 사용합니다. 논리 볼륨을 수정하는 것이 자유로우며 물리 볼륨에서 공간을 재할당합니다.

다음의 명령으로 **vg1** 볼륨 그룹에 10GB 크기의 논리 볼륨을 생성합니다.

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

다음 명령으로 **/dev/testvg/testlv** 블록 장치를 생성하여, **testvg** 볼륨 그룹에 **testlv**라는 1500MB 선형 논리 볼륨을 생성합니다.

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

다음 명령으로 **vg0** 볼륨 그룹에 여유 익스텐트에서 **gfslv**라는 50GB의 논리 볼륨을 생성합니다.

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

lvcreate 명령의 **-l** 인수를 사용하여 익스텐트에서 논리 볼륨의 크기를 지정할 수 있습니다. 이러한 인수를 사용하여 논리 볼륨에 사용할 볼륨 그룹의 퍼센트도 지정할 수 있습니다. 다음 명령으로 **testvol** 볼륨 그룹에서 전체 공간의 60%를 사용하는 **mylv**라는 논리 볼륨을 생성합니다.

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

lvcreate 명령의 **-l** 인수를 사용하여 논리 볼륨 크기로서 볼륨 그룹에 있는 나머지 여유 공간의 퍼센트를 지정할 수 있습니다. 다음의 명령으로 **testvol** 볼륨 그룹에 있는 모든 할당되지 않은 공간을 사용하는 **yourlv**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 생성하기 위한 다른 방법으로는 **vgdisplay** 명령을 사용하여 "Total PE" 크기를 찾아 이를 **lvcreate** 명령에 입력값으로 사용하는 것입니다.

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

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

물리 볼륨이 삭제되어야 할 경우 논리 볼륨을 생성하기 위해 사용된 기본적인 물리 볼륨이 중요하게 되므로 논리 볼륨을 생성할 때 이러한 가능성을 염두해 두어야 합니다. 볼륨 그룹에서 물리 볼륨을 삭제하는 방법에 대한 내용은 4.3.6절. “볼륨 그룹에서 물리 볼륨 삭제 ”에서 참조하시기 바랍니다.

볼륨 그룹에 있는 특정 물리 볼륨에서 할당된 논리 볼륨을 생성하기 위해 물리 볼륨이나 **lvcreate** 명령 행의 마지막에 있는 볼륨을 지정합니다. 다음 명령으로 **/dev/sdg1** 물리 볼륨에서 할당된 **testvg** 볼륨 그룹에 **testlv**라는 논리 볼륨을 생성합니다.

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

논리 볼륨으로 사용할 물리 볼륨의 익스텐트를 지정할 수 있습니다. 다음의 예에서는 물리 볼륨**/dev/sda1**의 익스텐트 0 부터 24 까지에서 그리고 **testvg** 볼륨 그룹에 있는 물리 볼륨 **/dev/sdb1**의 익스텐트 50 부터 124 까지에서 선형 논리 볼륨을 생성하고 있습니다.

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

다음의 예에서는 물리 볼륨 **/dev/sda1**의 익스텐트 0에서 25까지에서 선형 논리 볼륨을 생성하고 그 후 익스텐트 100까지 배열합니다.

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

논리 볼륨의 익스텐트가 할당되는 방법에 대한 기본 정책은 **inherit**되는 것이며, 이는 볼륨 그룹에 대해서도 같은 정책이 적용됩니다. 이러한 정책은 **lvchange** 명령으로 변경될 수 있습니다. 할당 정책에 관한 자세한 내용은 4.3.1절. “볼륨 그룹 생성 ”에서 참조하시기 바랍니다.

4.4.2. 스트라이프 (striped) 볼륨 생성

순차적으로 대량 읽기 및 쓰기 작업을 할 경우, 스트라이프 논리 볼륨을 생성하면 데이터 I/O의 효율성을 향상시킬 수 있습니다. 스트라이프 볼륨에 관한 일반적인 내용은 2.3.2절. “스트라이프 (Striped) 논리 볼륨 ”에서 참조하시기 바랍니다.

스트라이프 논리 볼륨을 생성할 때, **lvcreate** 명령의 **-i** 인수를 사용하여 스트라이프 수를 지정합니다. 이는 얼마나 많은 물리 볼륨 및 논리 볼륨을 스트라이프할 지를 결정합니다. 스트라이프 수는 볼륨 그룹에 있는 물리 볼륨의 수 보다 커서는 안됩니다. (**--alloc anywhere** 인수가 사용되지 않을 경우).

스트라이프 논리 볼륨으로된 기본 물리 장치는 크기가 다르며, 스트라이프 볼륨의 최대 크기는 크기가 가장 작은 기본 장치에 의해 결정됩니다. 예를 들어, two-legged 스트라이프에서, 최대 크기는 가장 작은 장치 크기의 두 배가 되고, three-legged 스트라이프에서 최대 크기는 가장 작은 장치 크기의 세 배가 됩니다.

다음의 명령으로 2 개의 물리 볼륨에 걸쳐 64kB로된 스트라이프 논리 볼륨을 생성합니다. 논리 볼륨은 50 GB 크기에, **gfslv**라는 이름으로, **vg0** 볼륨 그룹에서 나누어져 있습니다.

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

선형 볼륨과 마찬가지로, 스트라이프 용으로 사용할 물리 볼륨의 익스텐트를 지정할 수 있습니다. 다음의 명령으로 두 개의 물리 볼륨에 걸쳐 스트라이프하는 100 익스텐트 크기의 스트라이프 볼륨을 생성하고, **stripelv**라고 이름을 정한 후, **testvg** 볼륨 그룹에 둡니다. 스트라이프는 **/dev/sda1**의 0-49 섹터와 **/dev/sdb1**의 50-99 섹터를 사용하게 됩니다.

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

4.4.3. 미러 볼륨 생성

클러스터에서의 미러 LVM 논리 볼륨

클러스터에 미러 LVM 논리 볼륨을 생성하기 위해 단일 노드에 미러 LVM 논리 볼륨을 생성하는 것과 동일한 명령 및 절차가 필요합니다. 하지만 클러스터에 미러 LVM 볼륨을 생성하려면 클러스터 및 클러스터 미러 인프라를 실행하고, 클러스터는 쿼터에 도달해야 하며, **lvm.conf** 파일에 있는 잠금 유형은 클러스터 잠금 기능을 활성화하기 위해 3.1절. “클러스터에 LVM 볼륨 생성 ”에 설명되어 있듯이 직접적으로 또는 **lvmconf** 명령을 실행하여 올바르게 설정되어 있어야 합니다.
클러스터에 있는 여러 노드에서 빠른 시간에 연속하여 여러 LVM 미러를 생성하고 변환 명령을 실행하려고 하면 이 명령의 백로그가 발생할 수 있습니다. 이는 요청된 작업이 시간 초과된 후 실패할 가능성이 있습니다. 이 문제를 해결하기 위해 클러스터 미러 생성 명령은 클러스터의 단일 노드에서 실행하는 것이 좋습니다.

미러 볼륨 생성 시, **lvcreate** 명령의 **-m** 인수를 사용하여 만들 데이터 사본 수를 지정합니다. **-m1**을 지정하면 하나의 미러를 생성하고, 선형 논리 볼륨 및 하나의 복사본이라는 두 개의 파일 시스템 복사본을 생성합니다. 이와 유사하게, **-m2**를 지정하면 두 개의 미러를 생성하고, 세 개의 파일 시스템 복사본을 생성합니다.

다음의 명령으로 단일 미러와 함께 미러 논리 볼륨을 생성합니다. 볼륨은 50 GB 크기에 **mirrorlv**라는 이름으로, **vg0** 볼륨 그룹에서 나눕니다:

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

LVM 미러는 기보값으로 512KB 크기인 영역으로 복사된 장치를 나눕니다. **lvcreate** 명령의 **-R** 인수를 사용하여 영역 크기를 MB 단위로 지정할 수 있습니다. 또한 **lvm.conf** 파일에 있는 **mirror_region_size** 설정을 편집하여 기본값 영역 크기를 변경할 수 있습니다.

참고

클러스터 인프라의 제한으로 인해 1.5TB 이상의 클러스터 미러는 기본값 영역 크기인 512KB로 생성될 수 없습니다. 대용량 미러가 필요한 사용자는 기본값 크기보다 큰 영역 크기로 늘리셔야 합니다. 영역 크기를 늘리지 않을 경우 LVM 생성이 중단되거나 다른 LVM 명령이 중단될 수 있습니다.
1.5TB 이상의 미러에 대해 영역 크기를 지정하기 위한 일반적인 지침으로 미러 크기를 TB 단위로 할 수 있으며 번호를 2로 지정할 수 있습니다. 여기서 **lvcreate** 명령에 **-R** 인수로 이 번호를 사용합니다. 예를 들어, 미러 크기가 1.5TB인 경우, **-R 2**로 지정할 수 있으며, 미러 크기가 3TB인 경우에는 **-R 4**로 지정할 수 있습니다. 미러 크기가 5TB인 경우, **-R 8**로 지정할 수 있습니다.
다음의 명령으로 2MB 영역 크기로된 미러 물리 볼륨을 생성합니다:

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

LVM은 미러로 어떤 영역이 동기화되었는 지를 추적하기 위해 사용되는 로그를 관리합니다. 이러한 로그는 기본값으로 디스크에 저장되어, 재부팅 후에도 영구적으로 보관되며 미러가 재부팅할 때마다 다시 동기화할 필요가 없는지를 확인합니다. **--mirrorlog core** 인수로 이러한 로그가 메모리에 저장되도록 지정할 수 있습니다; 이는 추가 로그 장치가 필요하지 않게 하지만, 재부팅할 때 마다 전체 미러를 동기화해야 합니다.

다음의 명령으로 **bigvg** 볼륨 그룹에서 미러 논리 볼륨을 생성합니다. 논리 볼륨은 **ondiskmirvol**라는 이름으로 하나의 단일 미러를 갖게 됩니다. 볼륨은 12MB 크기가 되며 메모리에 미러 로그를 보관합니다.

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

미러 로그는 미러 leg가 생성되어 있는 장치와 다른 장치에 생성됩니다. 하지만, **vgcreate** 명령의 **--alloc anywhere** 인수를 사용하여 미러 leg 중 하나로 동일한 장치에 미러 로그를 생성할 수 있습니다. 이는 성능을 저하시킬 수 도 있지만, 기본 장치기 두 개 밖에 없을 경우에도 미러를 생성할 수 있게 합니다.

다음의 명령으로 단일 미러와 함께 미러 논리 볼륨을 생성합니다. 여기서 미러 로그는 미러 leg 중 하나로 동일한 장치에 있습니다. 예에서, **vg0** 볼륨 그룹은 두 개의 장치로만 구성되어 있습니다. 이러한 명령이 생성한 미러 볼륨은 500 메가 바이트 크기에 **mirrorlv**라는 이름으로 **vg0** 볼륨 그룹에서 나누어 집니다.

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

참고

클러스터된 미러로 미러 로그 관리는 현시점의 최하위 클러스터 ID를 갖는 클러스터 노드의 완전한 책임이 됩니다. 따라서 클러스터 미러 로그를 보유하는 장치는 클러서트의 하부 집합에서 사용할 수 없는 경우, 최하위의 ID를 갖는 클러스터 노드가 미러 로그에 액세스를 유지하고 있는 한 클러스터된 미러는 영향을 받지 않고 작업을 지속할 수 있습니다. 미러는 영향을 받지 않기 때문에 자동 수정 작업 (복구)도 발생하지 않습니다. 최하위 ID의 클러스터 노드가 미러 로그에 액세스할 수 없게 되면 (다른 노드에서 로그로의 액세스 가능 여부와 관계없이) 자동 작업이 작동하게 됩니다.

자체적으로 미러된 미러 로그를 생성하기 위해 **--mirrorlog mirrored** 옵션을 지정할 수 있습니다. 다음의 명령으로 **bigvg** 볼륨 그룹에서 미러 논리 볼륨을 생성합니다. 논리 볼륨은 **twologvol**라는 이름으로 하나의 단일 미러를 갖게 됩니다. 볼륨은 12MB 크기가 되며 개별적 장치에 보관되는 각각의 로그와 함께 미러 로그는 미러됩니다.

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

마찬가지로 표준 미러 로그와 함께 **vgcreate** 명령의 **--alloc anywhere** 인수를 사용하여 미러 leg와 동일한 장치에 이중 미러 로그를 생성할 수 있습니다. 이는 성능을 저하시킬 수 도 있지만, 이는 미러 leg 외에 분리된 장치에 보관하기 위해 각 로그에 대해 기본 장치가 충분하지 않을 경우에도 이중 미러 로그를 생성할 수 있게 합니다.

미러가 생성되면, 미러 영역은 동기화됩니다. 미러가 용량이 클 경우, 동기화하는데 시간이 더 오래 걸릴 수도 있습니다. 재생될 필요가 없는 새로운 미러를 생성할 경우, **nosync** 인수를 지정하여 첫 번째 장치에서의 초기 동기화할 필요가 없음을 나타냅니다.

미러 leg 및 로그 용으로 사용할 장치와 사용할 장치의 익스텐트를 지정할 수 있습니다. 특정 디스크에 로그를 강제하려면, 로그가 위치할 디스크에 정확히 하나의 익스텐트를 지정합니다. LVM은 명령행에 나열될 장치 순서를 따를 필요가 없습니다. 물리 볼륨이 나열되어 있을 경우, 할당될 공간 만이 해당하게 됩니다. 열거되어 있는 이미 할당된 물리 익스텐트는 무시하게 됩니다.

다음의 명령은 미러되지 않은 단일 로그와 단일 미러로 미러 논리 볼륨을 생성하고 있습니다. 볼륨은 500MB 크기에 **mirrorlv**라는 이름으로 **vg0** 볼륨 그룹에서 나뉘어져 있습니다. 첫 번째 미러 leg는 **/dev/sda1** 장치에 위치하게 되고, 두 번째 미러 leg는 **/dev/sdb1** 장치에 위치하게 되며, 미러 로그는 **/dev/sdc1**에 위치하게 됩니다.

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

다음의 명령은 단일 미러로 미러 논리 볼륨을 생성하고 있습니다. 볼륨은 500 MB 크기에 **mirrorlv**라는 이름으로 **vg0** 볼륨 그룹에서 나뉘어 집니다. 첫 번째 미러 leg는 **/dev/sda1** 장치의 익스텐트 0에서 499까지 위치하게 되고, 두 번째 미러 leg는 **/dev/sdb1** 장치의 익스텐트 0에서 499까지 위치하게 되며, 미러 로그는 **/dev/sdc1** 장치의 익스텐트 0에서 시작합니다. 이는 1MB 익스텐트입니다. 지정한 익스텐트가 이미 할당되어 있을 경우 이를 무시하게 됩니다.

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

참고

Red Hat Enterprise Linux 6.1 릴리즈에서는 단일 논리 볼륨에서 RAID0 (스트라이핑)과 RAID1 (미러링)을 통합할 수 있습니다. 논리 볼륨을 생성하고 동시에 미러 수 (**--mirrors X**)와 스트라이프 수 (**--stripes Y**)를 지정하면 미러 장치의 구성 장치가 스트라이프됩니다.

4.4.3.1. 미러 논리 볼륨 실패 정책

lvm.conf 파일의 **activation** 부분에 있는 **mirror_image_fault_policy** 및 **mirror_log_fault_policy** 매개 변수로 장치 실패시 미러 논리 볼륨 작동 방식을 지정할 수 있습니다. 이러한 매개 변수가 **remove**로 설정될 경우, 시스템은 잘못된 장치를 제거하고 이 장치 없이 실행됩니다. 이러한 매개 변수가 **allocate**로 설정될 경우, 시스템은 잘못된 장치를 제거하고 잘못된 장치 대신 새로운 장치를 할당합니다. 적절한 장치나 공간이 할당될 수 없을 경우 이러한 정책은 **remove** 정책과 같이 작동합니다.

기본값으로 **mirror_log_fault_policy** 매개 변수는 **allocate**로 설정되어 있습니다. 로그에 대해 이러한 정책을 사용하는 것이 신속하며 충돌 및 재부팅을 통한 동기화 상태를 기억하게 하는 기능을 관리합니다. 이 정책을 **remove**로 설정하였을 경우, 로그 장치가 실패했을 때 미러는 내부 메모리 로그를 사용하도록 전환되며 미러는 충돌 및 재부팅을 통한 동기화 상태를 기억하지 않게 되고 전체 미러는 다시 동기화됩니다.

기본값으로 **mirror_image_fault_policy** 매개 변수는 **remove**로 설정되어 있습니다. 이 정책으로 미러 이미지가 실패할 경우 복사본이 하나만 남아있을 때 미러는 미러되지 않은 장치로 전환하게 됩니다. 미러 장치에 대해 이 정책을 **allocate**로 설정하면, 미러가 장치에 다시 동기화해야 하기 때문에 처리 시간이 소요되지만 이로 인해 장치의 미러 특성을 유지할 수 있게 됩니다.

참고

LVM 미러 장치에 장애가 발생하면, 두 단계의 복구 절차가 실행됩니다. 첫 번째 단계에서는 장애가 발생한 장치의 제거가 이루어 집니다. 이는 선형 장치로 미러가 축소되는 결과를 초래할 수 있습니다. 두 번째 단계에서는 **mirror_log_fault_policy** 매개 변수가 **allocate**로 설정되는 경우, 장애가 발생한 장치를 대체하려 합니다. 하지만 두 번째 단계에서 다른 장치가 사용 가능한 경우 미러로 장애와 관련이 없는 이전에 사용하고 있는 장치가 선택된다고 보장할 수 없다는 점에 유의하십시오. LVM 미러 장애 발생 시 수동으로 복구하는 방법에 대한 내용은 6.3절. “LVM 미러 장애 복구 ”에서 참조하십시오.

4.4.3.2. 미러 논리 볼륨의 중복된 이미지 분리하기

미러 논리 볼륨의 중복된 이미지를 분리하여 새로운 논리 볼륨을 구성할 수 있습니다. 이미지를 분리하려면 **lvconvert** 명령의 **--splitmirrors** 인수를 사용하여 분리할 중복된 이미지 수를 지정합니다. 명령의 **--name** 인수를 사용하여 새롭게 분리된 논리 볼륨의 이름을 지정해야 합니다.

다음 명령으로 **vg/lv**라는 미러 논리 볼륨에서 **copy**라는 새로운 논리 볼륨을 분리할 수 있습니다. 새로운 논리 볼륨에는 두 개의 미러 leg가 들어 있게 됩니다. 예에서 LVM은 분리한 장치를 선택하고 있습니다.

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

분리할 장치를 지정할 수 있습니다. 다음 명령으로 **vg/lv** 미러 논리 볼륨에서 **copy**라는 새로운 논리 볼륨을 분리할 수 있습니다. 새 논리 볼륨에는 **/dev/sdc1** 및 **/dev/sde1** 장치로 구성된 두 개의 미러 leg가 들어 있게 됩니다.

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

4.4.3.3. 미러 논리 장치 복구

lvconvert --repair 명령을 사용하여 디스크 실패 후 미러를 복구할 수 있습니다. 이는 안정적인 상태로 미러를 되돌려 놓습니다. **lvconvert --repair** 명령은 대화식 명령으로 시스템이 실패한 장치를 대체하기 위한 여부를 나타낼지에 대해 묻습니다.

- 프롬프트를 생략하고 실패한 장치 모드를 대체하려면 명령행에서 **-y** 옵션을 지정합니다.
- 프롬프트를 생략하고 실패한 장치 중 어느것도 대체하지 않으려면 명령행에서 **-f** 옵션을 지정합니다.
- 프롬프트를 생략하고 미러 이미지 및 미러 로그에 에대 다른 대체 정책을 적용하려면 **--use-policies** 인수를 지정하여 **lvm.conf** 파일에 **mirror_log_fault_policy**와 **mirror_device_fault_policy**를 지정하여 장치 대체 정책을 사용할 수 있습니다.

4.4.3.4. 미러 볼륨 설정 변경

lvconvert 명령을 사용하여 미러 볼륨에서 선형 볼륨으로 또는 선형 볼륨에서 미러 볼륨으로 논리 볼륨을 변환할 수 있습니다. 이 명령을 사용하여 **corelog**와 같은 기존 논리 볼륨의 다른 미러 매개 변수를 재설정할 수 도 있습니다.

논리 볼륨을 미러 볼륨으로 변환할 경우, 기본적으로 기존 볼륨에 미러 leg를 생성해야 합니다. 즉 이는 볼륨 그룹에 미러 leg 및 미러 로그에 필요한 공간 및 장치가 있어야 함을 의미합니다.

미러 leg가 손실되었을 경우, LVM은 볼륨을 선형 볼륨으로 변환하여 미러 중복없이 계속 볼륨에 액세스하게 합니다. 미러 leg를 대체한 후, 이를 복구하기 위해 **lvconvert** 명령을 사용합니다. 이러한 절차는 6.3절. “LVM 미러 장애 복구 ”에 설명되어 있습니다.

다음의 명령으로 선형 논리 볼륨 **vg00/lvol1**을 미러 논리 볼륨으로 변환합니다.

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

다음의 명령으로 미러 leg를 삭제하여 **vg00/lvol1** 미러 논리 볼륨을 선형 논리 볼륨으로 변환합니다.

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

4.4.4. 스냅샷 볼륨 생성

스냅샷 볼륨을 생성하기 위해 **lvcreate** 명령의 **-s** 인수를 사용합니다. 스냅샷 볼륨은 쓰기 가능합니다.

참고

LVM 스냅샷은 클러스터에 있는 노드에서 지원되지 않습니다. 클러스터 볼륨 그룹에 스냅샷 볼륨을 생성할 수 없습니다. 하지만 Red Hat Enterprise Linux 6.1 릴리즈에서 클러스터 논리 볼륨에 지속적인 데이터 백업을 생성해야 할 경우, 볼륨을 독단적으로 활성화하여 스냅샷을 생성할 수 있습니다. 노드에서 논리 볼륨을 독단적으로 활성화하는 내용은 4.7절. “클러스터에 있는 개별적 노드에서 논리 볼륨 활성화 ”에서 참조하십시오.

참고

Red Hat Enterprise Linux 6.1 릴리즈에서는 미러 논리 볼륨에 대해 LVM 스냅샷을 지원합니다.

다음의 명령으로 100MB 크기에 **/dev/vg00/snap**라는 이름의 스냅샷 논리 볼륨을 생성합니다. 이는 **/dev/vg00/lvol1**라는 초기 논리 볼륨의 스냅샷을 생성합니다. 초기 논리 볼륨에 파일 시스템이 들어 있을 경우, 초기 파일 시스템이 업데이트되는 동안 백업을 실행하기 위해 파일 시스템의 컨텐츠를 액세스함으로 임시 디렉토리에 스냅샷 논리 볼륨을 마운트할 수 있습니다.

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

스냅샷 논리 볼륨을 생성한 후에, **lvdisplay** 명령으로 초기 볼륨을 지정하면 모든 스냅샷 논리 볼륨 목록 및 상태 (활성 또는 비활성)가 들어있는 출력 결과가 나타납니다.

다음의 예에서는 **/dev/new_vg/newvgsnap** 스냅샷 볼륨이 생성된 **/dev/new_vg/lvol0** 논리 볼륨 상태를 보여주고 있습니다.

```
# 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
```

lvs 명령은 기본값으로 각각의 스냅샷 볼륨으로 사용된 초기 볼륨 및 스냅샷 볼륨의 현재 퍼센트를 보여줍니다. 다음의 예에서는 **/dev/new_vg/newvgsnap** 스냅샷 볼륨이 생성된 **/dev/new_vg/lvol0** 논리 볼륨이 들어있는 시스템에 해당하는 **lvs** 명령의 기본 출력 결과를 보여주고 있습니다.

```
# 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
```

참고

초기 볼륨이 변경되어 스냅샷 크기가 늘어났기 때문에 **lvs** 명령으로 정기적으로 스냅샷 볼륨의 퍼센트를 모니터하여 스냅샷이 채워지지 않았는 지를 확인합니다. 초기 스냅샷의 변경되지 않은 부분에 작성할 경우 스냅샷을 손상시키지 않고 성공적으로 실행될 수 없으므로 스냅샷이 100%로 채워졌을 경우에는 완전히 손실될 수 있습니다.

4.4.5. 스냅샷 볼륨 합치기

Red Hat Enterprise Linux 6 릴리즈에서 **lvconvert** 명령의 **--merge** 옵션을 사용하여 스냅샷을 원래 볼륨으로 합치기할 수 있습니다. 원래 볼륨과 스냅샷 볼륨 모두가 열리지 않을 경우, 합치기가 바로 시작됩니다. 그렇지 않을 경우, 원래 볼륨 또는 스냅샷 볼륨이 처음으로 활성화되어 모두가 종료되었을 경우 합치기가 시작됩니다. root 파일 시스템과 같이 종료할 수 없는 원래 볼륨으로 스냅샷 볼륨을 합치기할 경우 이는 다음 번에 원래 볼륨이 활성화될 때 까지 연기됩니다. 합치기가 시작되면 논리 볼륨에는 원래 볼륨의 이름 마이너

번호 및 UUID가 나타나게 됩니다. 합치기 작업이 진행되고 있을 경우 원래 볼륨으로의 읽기 또는 쓰기 작업은 합쳐진 스냅샷으로 이동한 것처럼 나타납니다. 합치기 작업이 완료되면 합쳐진 스냅샷은 삭제됩니다.

다음의 명령으로 스냅샷 볼륨 **vg00/lvol1_snap**을 원래의 볼륨으로 통합합니다.

```
lvconvert --merge vg00/lvol1_snap"
```

명령행에 여러 스냅샷을 지정할 수 있습니다. 또는 LVM 객체 태그를 사용하여 원래 볼륨으로 합쳐진 여러 스냅샷을 지정할 수 있습니다. 다음의 예에서는 **vg00/lvol1**, **vg00/lvol2**, **vg00/lvol3** 논리 볼륨이 **@some_tag**로 태그되어 있습니다. 다음의 명령으로 이러한 세가지 볼륨에 대한 스냅샷 논리 볼륨을 순차적으로 **vg00/lvol1**, **vg00/lvol2**, **vg00/lvol3**으로 합치기할 수 있습니다. **--background** 옵션이 사용되었을 경우 모든 스냅샷 논리 볼륨 합치기 작업은 병렬로 시작됩니다.

```
lvconvert --merge @some_tag"
```

LVM 객체 태그에 대한 내용은 부록 C. *LVM 객체 태그* 에서 참조하십시오. **lvconvert --merge** 명령에 대한 자세한 내용은 **lvconvert**(8) man 페이지를 참조하십시오.

4.4.6. 영구 장치 번호

주 장치 및 부 장치 번호는 모듈을 읽어올 때 할당됩니다. 몇몇 어플리케이션은 블록 장치가 같은 장치 (주 장치 및 부 장치) 번호로 활성화될 경우 가장 잘 작동합니다. 다음의 인수를 사용하여 **lvcreate** 및 **lvchange** 명령으로 이를 지정할 수 있습니다:

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

주 장치 번호가 다른 장치에 이미 할당되지 않도록 하기 위해 큰 주 장치 번호를 사용합니다.

NFS를 사용하여 파일 시스템을 내보내기 할 경우, 내보내기 영역에 **fsid** 매개 변수를 지정하면 LVM 내에 있는 영구 장치 번호를 설정할 필요가 없게 됩니다.

4.4.7. 논리 볼륨 크기 조정

논리 볼륨의 크기를 줄이려면, **lvreduce** 명령을 사용합니다. 논리 볼륨에 파일 시스템이 들어 있을 경우, 먼저 파일 시스템을 줄여서 (또는 LVM GUI를 사용하여) 최소한 논리 볼륨 크기가 파일 시스템 크기 만큼 되도록 합니다.

다음 명령을 사용하여 3 개의 논리 익스텐트로 **vg00** 볼륨 그룹에서 **lvol1** 논리 볼륨의 크기를 줄입니다.

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

4.4.8. 논리 볼륨 그룹의 매개 변수 변경

논리 볼륨의 매개 변수를 변경하려면 **lvchange** 명령을 사용합니다. 변경 가능한 매개 변수의 목록은 **lvchange**(8) 맨 페이지에서 참조하시기 바랍니다.

lvchange 명령을 사용하여 논리 볼륨을 활성화 및 비활성화시킬 수 있습니다. 동시에 볼륨 그룹에 있는 모든 논리 볼륨을 활성화 및 비활성화시키기 위해 4.3.7절. “볼륨 그룹의 매개 변수 변경 ”에서 설명하고 있듯이 **vgchange** 명령을 사용합니다.

다음의 명령으로 **vg00** 볼륨 그룹에 있는 **lvol1** 볼륨이 읽기 전용이 되도록 권한을 변경합니다.

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

4.4.9. 논리 볼륨 이름 변경

기존 논리 볼륨 이름을 변경하기 위해 **lvrename** 명령을 사용합니다.

다음의 명령으로 **vg02** 볼륨 그룹에 있는 **lvold** 논리 볼륨을 **lvnew**로 이름을 변경할 수 있습니다.

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

```
lvrename vg02 lvold lvnew
```

클러스터에 있는 개별적 노드에서 논리 볼륨을 활성화하는 방법에 대한 보다 자세한 내용은4.7절. “클러스터에 있는 개별적 노드에서 논리 볼륨 활성화 ”에서 참조하시기 바랍니다.

4.4.10. 논리 볼륨 삭제

비활성화 논리 볼륨을 삭제하려면 **lvremove** 명령을 사용합니다. 현재 논리 볼륨이 마운트되어 있을 경우 삭제 하기전 논리 볼륨을 마운트 해제해야 합니다. 또한 클러스터 환경에서도 논리 볼륨을 삭제하기전 이를 비활성화시켜야 합니다.

다음 명령으로 **testvg** 볼륨 그룹에서 **/dev/testvg/testlv** 논리 볼륨을 삭제합니다. 이러한 경우 논리 볼륨은 비활성화되지 않음에 유의하시기 바랍니다.

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

lvchange -an 명령을 사용하여 논리 볼륨을 삭제하기 전 이를 비활성화시킬 수 있으며, 이러한 경우, 활성 논리 볼륨을 삭제하기를 원하는 지에 대한 여부를 확인하는 프롬프트가 나타나지 않습니다.

4.4.11. 논리 볼륨 보기

LVM 논리 볼륨의 속성 보기에 사용할 수 있는 명령에는 다음과 같은 세 가지가 있습니다: **lvs**, **lvdisplay**, **lvscan**.

lvs 명령은 하나의 논리 볼륨 당 하나의 행을 나타나게 하여 설정 가능한 형식으로 논리 볼륨 정보를 제공합니다. **lvs** 명령으로 포맷 관리를 할 수 있으며, 스크립팅에 유용합니다. 출력 결과를 사용자 설정하기 위해 **lvs** 명령을 사용하는 방법에 관한 내용은 4.8절. “LVM 용 사용자 설정 리포트 ”에서 참조하시기 바랍니다.

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

다음의 명령으로 **vg00**에 있는 **lvol2**의 속성을 볼 수 있습니다. 스냅샷 논리 볼륨이 초기 논리 볼륨 용으로 생성되었을 경우, 이러한 명령은 스냅샷 논리 볼륨 및 논리 볼륨의 (활성 또는 비활성) 상태에 대한 목록을 보여줍니다.

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

다음의 예에서와 같이 **lvscan** 명령으로 시스템에 있는 모든 논리 볼륨을 찾고 이를 나열합니다.

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

4.4.12. 논리 볼륨 늘리기

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

논리 볼륨을 확장할 때 얼마 정도를 확장할 지 또는 확장한 후 논리 볼륨이 얼마 정도가 되기를 원하는 지를 나타내야 합니다.

다음의 명령으로 **/dev/myvg/homevol** 논리 볼륨을 12 GB 까지 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
```

다음의 명령으로 **/dev/myvg/homevol** 논리 볼륨에 다른 GB를 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 볼륨 크기를 확장할 익스텐트 수를 지정할 수 있습니다. 또한 이러한 인수를 사용하여 볼륨 그룹의 퍼센트나 볼륨 그룹에 있는 남아있는 여유 공간의 퍼센트도 지정할 수 있습니다. 다음의 명령으로 **testlv**라는 논리 볼륨을 확장하여 **myvg** 볼륨 그룹에 있는 할당되지 않은 모든 공간을 채웁니다.

```
[root@tng3-1 ~]# lvextend -l +100%FREE /dev/myvg/testlv
  Extending logical volume testlv to 68.59 GB
  Logical volume testlv successfully resized
```

논리 볼륨을 확장한 후 해당 파일 시스템의 크기를 확장해야 합니다.

기본값으로 대부분의 파일 시스템 크기 조정 도구는 기본적인 논리 볼륨의 크기가 되도록 파일 시스템 크기를 확장하므로 두 가지 명령에 대해 같은 크기로 지정해야 할 지를 염려하지 않아도 됩니다.

4.4.12.1. 스트라이프 볼륨 확장

스트라이프 논리 볼륨 크기를 확장하기 위해, 스트라이트를 지원할 수 있는 볼륨 그룹으로된 기본 물리 볼륨에 충분한 여유 공간이 있어야 합니다. 예를 들어, 전체 볼륨 그룹을 사용하는 두 가지 방식의 스트라이프가 있을 경우, 볼륨 그룹에 단일 물리 볼륨을 추가하는 것으로 스트라이프를 확장할 수 없습니다. 대신, 볼륨 그룹에 최소 두 개의 물리 볼륨을 추가해야 합니다.

예를 들어, 다음의 **vgs** 명령으로 나타난 두 개의 기본 물리 볼륨으로 된 **vg** 볼륨 그룹이 있다고 가정해 봅시다.

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

볼륨 그룹에 있는 전체 공간을 사용하여 스트라이프를 생성할 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)
```

현재 볼륨 그룹에는 더이상 여유 공간이 없음에 유의합니다.

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

다음 명령으로 볼륨 그룹에 다른 물리 볼륨을 추가하면, 135G의 추가 공간이 생깁니다.

```
# 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 필요하기 때문에 볼륨 그룹의 전체 크기 만큼 스트라이프 논리 볼륨을 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
```

스트라이프 논리 볼륨을 확장하려면, 다른 물리 볼륨을 추가한 후 논리 볼륨을 확장해야 합니다. 예에서 볼륨 그룹에 추가된 두 개의 물리 볼륨으로 논리 볼륨을 볼륨 그룹의 전체 크기 만큼 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
```

스트라이프 논리 볼륨을 확장하기 위해 기본 물리 장치가 충분하지 않을 경우, 익스텐션을 스트라이프하지 않아도 상관 없을 경우라면 볼륨을 확장할 수 있습니다. 이는 실행이 불균등하게 되게 할 수 있습니다. 논리 볼륨에 여유 공간을 추가할 때, 기본값 실행은 기존 논리 볼륨의 마지막 세그먼트의 동일한 스트라이핑 매개변수를 사용하지만 이러한 매개 변수를 덮어쓰기할 수 도 있습니다. 다음의 예에서는 초기 **lvextend** 명령이 실패한 후 남아있는 여유 공간을 사용하기 위해 기존 스트라이프 논리 볼륨을 확장하고 있습니다.

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

4.4.12.2. cling 할당 정책을 사용하여 논리 볼륨 확장

LVM 볼륨을 확장할 때 **lvextend** 명령의 **--alloc cling** 옵션을 사용하여 **cling** 할당 정책을 지정할 수 있습니다. 이러한 정책은 기존 논리 볼륨의 마지막 세그먼트로 동일한 물리 볼륨에 있는 공간이 선택됩니다. 물리 볼륨에 공간이 충분하지 않고 태그 목록이 **lvm.conf** 파일에 정의되어 있는 경우, LVM은 태그가 물리 볼륨에 부착되어 있는지를 확인하고 기존 익스텐트와 새 익스텐트 간의 물리 볼륨 태그를 일치시키려 합니다.

예를 들어, 논리 볼륨이 단일 볼륨 그룹에 있는 두 사이트 간에 미러링되어 있으면, 위치에 따라 물리 볼륨을 @site1 및 @site2로 태그하고 **lvm.conf** 파일에 다음과 같은 행을 지정하여 물리 볼륨을 태그할 수 있습니다:

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

물리 볼륨의 태그 정보는 부록 C. *LVM 객체 태그*에서 참조하십시오.

다음 예제에서는 다음과 같은 행을 추가하도록 **lvm.conf** 파일이 변경되어 있습니다:

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

또한 이 예제에서는 **/dev/sdb1**, **/dev/sdc1**, **/dev/sdd1**, **/dev/sde1**, **/dev/sdf1**, **/dev/sdg1**, **/dev/sdh1** 물리 볼륨으로 구성된 **taft** 볼륨 그룹을 생성합니다. 이러한 물리 볼륨은 **A**, **B**, **C**로 태그될 수 있습니다. 예제에서는 **C** 태그를 사용하지 않지만 미러 leg에 사용할 물리 볼륨을 선택하기 위해 LVM이 이러한 태그를 사용하고 있음을 보여주고 있습니다.

```
[root@taft-03 ~]# pvs -a -o +pv_tags /dev/sd[bcdefgh]1
  PV         VG   Fmt  Attr PSize   PFree   PV Tags
  /dev/sdb1  taft lvm2 a-   135.66g 135.66g A
  /dev/sdc1  taft lvm2 a-   135.66g 135.66g B
  /dev/sdd1  taft lvm2 a-   135.66g 135.66g B
  /dev/sde1  taft lvm2 a-   135.66g 135.66g C
  /dev/sdf1  taft lvm2 a-   135.66g 135.66g C
  /dev/sdg1  taft lvm2 a-   135.66g 135.66g A
  /dev/sdh1  taft lvm2 a-   135.66g 135.66g A
```

다음 명령으로 **taft** 볼륨 그룹에서 100G 미러 볼륨을 생성합니다.

```
[root@taft-03 ~]# lvcreate -m 1 -n mirror --nosync -L 100G taft
```

다음의 명령으로 미러 leg 및 미러 로그로 사용될 장치를 표시합니다.

```
[root@taft-03 ~]# lvs -a -o +devices
  LV                VG        Attr   LSize   Log         Copy%  Devices
  mirror            taft      Mwi-a- 100.00g mirror_mlog 100.00
mirror_mimage_0(0),mirror_mimage_1(0)
  [mirror_mimage_0] taft      iwi-ao 100.00g                     /dev/sdb1(0)
  [mirror_mimage_1] taft      iwi-ao 100.00g                     /dev/sdc1(0)
  [mirror_mlog]     taft      lwi-ao   4.00m                     /dev/sdh1(0)
```

다음 명령은 미러 leg가 동일한 태그를 갖는 물리 볼륨을 사용하여 확장되어야 함을 나타내기 위해 **cling** 할당 정책을 사용하여 미러 볼륨의 크기를 확장합니다.

```
[root@taft-03 ~]# lvextend --alloc cling -L +100G taft/mirror
  Extending 2 mirror images.
  Extending logical volume mirror to 200.00 GiB
  Logical volume mirror successfully resized
```

다음에 표시된 명령은 leg로 동일한 태그를 갖는 물리 볼륨을 사용하여 미러 leg를 확장하는 것을 보여줍니다. 다음에 표시 명령 레지로 동일한 태그 물리적 볼륨을 사용하여 미러 레그가 확장되는 것을 보여줍니다. **C** 태그를 갖는 물리 볼륨은 무시되는 점에 유의하십시오.

```
[root@taft-03 ~]# lvs -a -o +devices
  LV                VG        Attr   LSize   Log         Copy%  Devices
  mirror            taft      Mwi-a- 200.00g mirror_mlog  50.16
mirror_mimage_0(0),mirror_mimage_1(0)
  [mirror_mimage_0] taft      Iwi-ao 200.00g                     /dev/sdb1(0)
  [mirror_mimage_0] taft      Iwi-ao 200.00g                     /dev/sdg1(0)
  [mirror_mimage_1] taft      Iwi-ao 200.00g                     /dev/sdc1(0)
  [mirror_mimage_1] taft      Iwi-ao 200.00g                     /dev/sdd1(0)
  [mirror_mlog]     taft      lwi-ao   4.00m                     /dev/sdh1(0)
```

4.4.13. 논리 볼륨 축소하기

논리 볼륨의 크기를 축소하기 위해, 먼저 파일 시스템을 마운트 해제합니다. 그 후 **lvreduce** 명령을 사용하여 볼륨을 축소할 수 있습니다. 볼륨을 축소한 후 파일 시스템을 다시 마운트합니다.

경고

볼륨 자체를 축소하기 전에 파일 시스템의 크기를 축소해야 합니다. 그렇지 않을 경우 데이터를 손실할 수 도 있습니다.

논리 볼륨을 축소하여 볼륨 그룹에 있는 다른 논리 볼륨에 할당하기 위해 볼륨 그룹의 일부분에 여유 공간을 둡니다.

다음의 에에서는 **vg00** 볼륨 그룹에 있는 **lvol1** 논리 볼륨을 크기를 3 개의 논리 익스텐트로 축소하고 있습니다.

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

4.5. 필터로 LVM 장치 스캔 제어

시작 시, **vgscan** 명령이 실행되어 LVM 레이블을 찾아 시스템에 있는 블록 장치를 스캔하고, 이중 어떤 것이 물리 볼륨인지를 확인하며 메타데이터를 읽고 볼륨 그룹 목록을 작성합니다. 물리 볼륨 명은 시스템에 있는 각 노드의 캐시 파일 **/etc/lvm/.cache**에 저장됩니다. 그 후의 명령은 파일을 읽어 다시 스캔되지 않도록 합니다.

lvm.conf 설정 파일에서 필터를 설정하여 어떤 LVM 장치가 스캔하게 할 지를 제어할 수 있습니다. **lvm.conf**파일에 있는 필터는 **/dev** 디렉토리에 있는 장치 이름에 적용하기 위해 일련의 간단한 정규 표현식으로 되어 있어 스캔하여 발견한 각각의 블록 장치를 허용할 지 또는 거부할 지를 결정합니다.

다음에서는 어떤 LVM 장치가 스캔하게 할 지를 제어하기 위한 필터 사용의 예를 보여주고 있습니다. 정교 표현식이 완전 경로 이름과 자유롭게 일치되어 있으므로 이는 최상의 활용 예가 아닐 수 도 있음에 유의하시기 바랍니다. 예를 들어, **a/loop/**는 **a/.*loop.*/** 와 동일하고 **/dev/solooperation/lvol1**과 일치될 수 있습니다.

다음의 필터는 발견된 모든 장치를 추가합니다. 설정 파일에 필터가 설정되지 않았을 경우 기본값으로 실행됩니다:

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

드라이브에 미디어가 없을 경우 지연을 방지하기 위해 다음의 필터로 cdrom 장치를 삭제합니다:

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

다음의 필터는 모든 루프를 추가하고 다른 모든 볼록 장치를 삭제합니다:

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

다음의 필터는 모든 루프 및 IDE를 추가하고 기타 다른 모든 블록 장치를 삭제합니다:

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

다음의 필터는 첫 번째 IDE 장치에 있는 파티션 8을 추가하고 기타 다른 모든 불록 장치를 삭제합니다:

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

lvm.conf 파일에 대한 보다 자세한 내용은 *부록 B. LVM 설정 파일* 및 **lvm.conf**(5) 맨 페이지를 참조하시기 바랍니다.

4.6. 온라인 데이터 재배치

시스템에서 **pvmove** 명령을 사용하고 있는 동안 데이터를 이동시킬 수 있습니다.

pvmove 명령으로 데이터를 나누어 섹션으로 이동시키고 각각의 섹션을 이동시키기 위해 임시 미러를 생성합니다. **pvmove** 명령 실행에 관한 내용은 **pvmove**(8) 맨 페이지를 참조하시기 바랍니다.

다음의 명령으로 물리 볼륨 **/dev/sdc1**에 할당된 모든 공간을 볼륨 그룹에 있는 물리 볼륨의 여유 공간으로 옮깁니다.

```
pvmove /dev/sdc1
```

다음의 명령으로 **MyLV** 논리 볼륨의 익스텐트를 이동시킵니다.

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

pvmove 명령이 실행하는 데 시간이 오래 걸릴 수 있으므로, 포그라운드에서의 진행 상태를 볼 수 없게 백그라운드에서 명령을 실행할 수 있습니다. 다음의 명령으로 **/dev/sdc1** 물리 볼륨에 할당된 모든 익스텐트를 백그라운드에 있는 **/dev/sdf1**로 옮깁니다.

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

다음의 명령은 5초 간격으로 이동 상태를 퍼센트로 보고합니다.

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

4.7. 클러스터에 있는 개별적 노드에서 논리 볼륨 활성화

클러스터 환경에 LVM을 설치했을 경우, 하나의 노드에 배타적으로 논리 볼륨을 활성화시켜야 할 수 도 있습니다.

하나의 노드에 배타적으로 논리 볼륨을 활성화시키기 위해, **lvchange -aey** 명령을 사용합니다. 다른 방법으로 **lvchange -aly** 명령을 사용하여 배타적이지는 않지만 하나의 노드에 논리 볼륨만을 활성화시킬 수 있습니다. 그 후 이를 추가 노드에서 동시에 활성화시킬 수 있습니다.

*부록 C. LVM 객체 태그*에서 설명하고 있듯이, LVM 태그를 사용하여 개별적 노드에 논리 볼륨을 활성화시킬

수 도 있습니다. 부록 B. *LVM 설정 파일* 에서 설명하고 있듯이, 설정 파일에 있는 노드 활성화도 지정할 수 있습니다.

4.8. LVM 용 사용자 설정 리포트

pvs, **lvs**, **vgs** 명령으로 LVM 객체에 대한 간결하고 사용자 설정 가능한 리포트를 만들 수 있습니다. 이러한 명령으로 생성된 리포트에는 각각의 객체에 대해 하나의 행으로된 출력 결과가 포함되어 있습니다. 각 행에는 객체와 관련하여 영역 속성 목록이 순서대로 나열됩니다. 물리 볼륨, 볼륨 그룹, 논리 볼륨, 물리 볼륨 세그먼트, 논리 볼륨 세그먼트로 보고할 객체를 선택합니다.

다음 부분에서는 다음과 같은 내용을 다루고 있습니다:

- 생성된 리포트의 포맷을 제어하기 위해 사용할 수 있는 명령 인수에 대한 요약
- 각각의 LVM 객체에 대해 선택할 수 있는 영역 목록
- 생성된 리포트를 정렬하는데 사용할 수 있는 명령 인수에 대한 요약
- 리포트 출력 결과 단위 지정을 위한 지시사항

4.8.1. 포맷 제어

pvs, **lvs** 또는 **vgs** 명령을 사용할 지는 영역 보기의 기본 설정 및 정렬 순서에 의해 결정됩니다. 다음과 같은 인수로 이러한 명령의 출력 결과를 제어할 수 있습니다:

- **-o** 인수를 사용하여 기본값 이외에 표시 영역에 있는 것을 다른 것으로 변경할 수 있습니다. 예를 들어 다음과 같은 출력 결과는 **pvs** 명령에 대해 기본값이 나타납니다 (물리 볼륨에 관한 정보를 보여줌).

```
# 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
```

  다음의 명령은 물리 볼륨 이름 및 크기 만이 나타나게 합니다.

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

- 덧샘 기호 (+)와 함께 출력 결과에 영역을 추가할 수 있습니다. 이는 -o 인수와 함께 사용됩니다.
  다음의 예에서는 기본 영역에 더하여 물리 볼륨의 UUID를 보여주고 있습니다.

```
# 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 추가하는 것에는 몇몇 추가 영역이 포함됩니다. 예를 들어, **pvs -v** 명령은 기본 영역에 더하여 **DevSize** 및 **PV UUID** 영역을 보여주고 있습니다.

```
# 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 헤드 라인을 삭제합니다. 이는 스크립트를 작성할 경우 유용할 수 있습니다.

  다음의 예에서는 **pv_name** 인수와 함께 **--noheadings** 인수를 사용하고 있습니다. 이는 모든 물리 볼륨 목록을 생성하게 됩니다.

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

- **--separator *separator*** 인수는 ***separator***를 사용하여 각각의 영역을 구분합니다.

  다음의 예에서는 등호 (=)로 **pvs** 명령의 기본 출력 결과 란을 구분하고 있습니다.

```
# pvs --separator =
  PV=VG=Fmt=Attr=PSize=PFree
  /dev/sdb1=new_vg=lvm2=a-=17.14G=17.14G
  /dev/sdc1=new_vg=lvm2=a-=17.14G=17.09G
  /dev/sdd1=new_vg=lvm2=a-=17.14G=17.14G
```

  separator 인수를 사용할 때 출력 결과 란을 정렬하기 위해, **--aligned** 인수와 관련하여 **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
```

lvs 또는 **vgs** 명령의 **-P** 인수를 사용하여 출력 결과에 나타나지 않는 실패한 볼륨에 관한 정보를 볼 수 있습니다. 이러한 인수 영역의 출력 결과에 대한 내용은 6.2절. “실패한 장치에 있는 정보 보기 ”에서 참조하시기 바랍니다.

디스플레이 인수의 전체 목록은 **pvs**(8), **vgs**(8), **lvs**(8) 맨 페이지에서 참조하시기 바랍니다.

볼륨 그룹 영역은 물리 볼륨 (및 물리 볼륨 세그먼트) 영역이나 논리 볼륨 (및 논리 볼륨 세그먼트) 영역과 함께 혼용될 수 있지만, 물리 볼륨 및 논리 볼륨 영역은 혼용될 수 없습니다. 예를 들어, 다음의 명령은 각각의 물리 볼륨에 해당하는 하나의 행으로된 출력 결과를 보여주고 있습니다.

```
# 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. 객체 선택

다음 부분에는 **pvs**, **vgs**, **lvs** 명령과 함께 LVM 객체에 관한 정보를 나열한 표가 있습니다.

편의에 따라, 명령의 기본값과 영역 이름이 일치할 경우, 영역 이름 접두부가 사용되지 않을 수 있습니다. 예를 들어, **pvs** 명령에서, **name**은 **pv_name**을 의미하며, **vgs** 명령에서는 **name**은 **vg_name**을 의미합니다.

다음의 명령을 실행하는 것은 **pvs -o pv_free** 명령을 실행하는 것과 동일합니다.

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

pvs 명령

표 4.1. “pvs 표시 영역 ”에는 헤더에 있는 영역 이름 및 영역 설명과 함께 **pvs** 명령의 디스플레이 인수가 나열되어 있습니다.

표 4.1. pvs 표시 영역

인수	헤더	설명
dev_size	DevSize	물리 볼륨이 생성된 기본적인 장치의 크기
pe_start	1st PE	기본 장치에서 첫 번째 물리 익스텐트의 시작으로의 오프셋
pv_attr	Attr	물리 볼륨 상태: 할당 가능(a) 또는 내보내어짐(X)
pv_fmt	Fmt	물리 볼륨의 메타데이터 포맷 (**lvm2** 또는 **lvm1**)
pv_free	PFree	물리 볼륨에 남아있는 여유 공간
pv_name	PV	물리 볼륨명
pv_pe_alloc_count	Alloc	사용된 물리 익스텐트 수
pv_pe_count	PE	물리 익스텐트 수
pvseg_size	SSize	물리 볼륨의 세그먼트 크기
pvseg_start	Start	물리 볼륨 세그먼트의 물리 익스텐트 시작
pv_size	PSize	물리 볼륨의 크기
pv_tags	PV Tags	물리 볼륨에 부착된 LVM 태그
pv_used	Used	현재 물리 볼륨에서 사용되는 공간
pv_uuid	PV UUID	물리 볼륨의 UUID

pvs 명령으로 **pv_name**, **vg_name**, **pv_fmt**, **pv_attr**, **pv_size**, **pv_free** 영역의 기본값을 볼 수 있습니다. 이는 **pv_name**에 의해 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 사용하여 기본값 영역 보기에 **dev_size**, **pv_uuid** 영역을 추가합니다.

```
# 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
```

각각의 물리 볼륨 세그먼트에 관한 정보를 보기 위해 **pvs** 명령의 **--segments** 인수를 사용할 수 있습니다. 세그먼트는 익스텐트 그룹입니다. 세그먼트 보기는 논리 볼륨이 나뉘어졌는 지를 확인하는데 유용합니다.

pvs --segments 명령으로 **pv_name**, **vg_name**, **pv_fmt**, **pv_attr**, **pv_size**, **pv_free**, **pvseg_start**, **pvseg_size** 영역의 기본값을 볼 수 있습니다. 이는 물리 볼륨에서 **pv_name** 및 **pvseg_size**에 의해 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
```

LVM 물리 볼륨으로 초기화되지 않은 LVM에 의해 발견된 장치를 확인하기 위해 **pvs -a** 명령을 사용할 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
```

vgs 명령

표 4.2. “vgs 표시 영역 ”에는 헤더에 있는 영역 이름 및 영역 설명과 함께 **vgs** 명령의 디스플레이 인수가 나열되어 있습니다.

표 4.2. vgs 표시 영역

인수	헤더	설명
lv_count	#LV	볼륨 그룹이 있는 논리 볼륨 수
max_lv	MaxLV	볼륨 그룹에서 허용하는 최대 논리 볼륨 수 (무제한일 경우 0)
max_pv	MaxPV	볼륨 그룹에서 허용하는 최대 물리 볼륨 수 (무제한일 경우 0)
pv_count	#PV	볼륨 그룹을 지정하는 물리 볼륨 수
snap_count	#SN	볼륨 그룹에 있는 스냅샷 수
vg_attr	Attr	볼륨 그룹의 상태: 쓰기 가능(w), 읽기 전용(r), 크기 조정 가능(z), 내보내어짐(x), 부분(p), 클러스터됨(c).
vg_extent_count	#Ext	볼륨 그룹에서 물리 익스텐트 수
vg_extent_size	Ext	볼륨 그룹에서 물리 익스텐트 크기
vg_fmt	Fmt	볼륨 그룹의 메타데이터 포맷 (**lvm2** 또는 **lvm1**)
vg_free	VFree	볼륨 그룹에 남아있는 여유 공간 크기
vg_free_count	Free	볼륨 그룹에 있는 여유 물리 익스텐트 수
vg_name	VG	볼륨 그룹명
vg_seqno	Seq	볼륨 그룹 버전을 나타내는 번호
vg_size	VSize	볼륨 그룹의 크기
vg_sysid	SYS ID	LVM1 시스템 ID
vg_tags	VG Tags	볼륨 그룹에 부착된 LVM 태그
vg_uuid	VG UUID	볼륨 그룹의 UUID

vgs 명령으로 **vg_name**, **pv_count**, **lv_count**, **snap_count**, **vg_attr**, **vg_size**, **vg_free** 영역의 기

본값을 볼 수 있습니다. 이는 **vg_name**에 의해 정렬됩니다.

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

vgs 명령과 함께 **-v** 인수를 사용하여 기본값 영역 보기에 **vg_extent_size**, **vg_uuid** 영역을 추가합니다.

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

lvs 명령

표 4.3. “lvs 표시 영역 ”에는 헤더에 있는 영역 이름 및 영역 설명과 함께 **lvs** 명령의 디스플레이 인수가 나열되어 있습니다.

표 4.3. lvs 표시 영역

인수	헤더	설명
chunksize **chunk_size**	Chunk	스냅샷 볼륨에서 단위 크기
copy_percent	Copy%	미러 논리 볼륨의 동기화 퍼센트: 물리 익스텐트가 **pv_move** 명령과 함께 이동할 경우 사용
devices	Devices	논리 볼륨을 만드는 기본 장치: 물리 볼륨, 논리 볼륨, 물리 익스텐트 및 논리 익스텐트 시작
lv_attr	Attr	논리 볼륨의 상태. 논리 볼륨 비트 속성은 다음과 같습니다: 비트 1: 볼륨 유형: 미러 (m), 초기 동기화하지 않은 미러 (M), 초기 볼륨 (o), (p)vmove, 스냅샷 (s), 유효하지 않은 스냅샷 (S), 가상 (v) 비트 2: 권한: 쓰기 가능(w), 읽기 전용(r) 비트 3: 할당 정책: (c)ontiguous, (n)ormal, (a)nywhere, (i)nherited. 할당 변경에 대해 볼륨이 현재 잠금 상태일 경우 이는 대문자로 나타납니다. 예를 들어, **pvmove** 명령이 실행 중일 경우 비트 4: 고정된 부 장치 (m) 비트 5: 상태: 활성화 (a), 중지 (s), 잘못된 스냅샷 (I), 잘못되어 중지된 스냅샷 (S), 테이블없이 나타나는 장치 맵핑 (d), 비활성화 테이블과 나타나는 장치 맵핑 (i) 비트 6: 장치 오픈 (o)
lv_kernel_major	KMaj	논리 볼륨의 실제 주 장치 번호 (비활성 상태일 경우 -1)
lv_kernel_minor	KMIN	논리 볼륨의 실제 부 장치 번호 (비활성 상태일 경우 -1)
lv_major	Maj	논리 볼륨의 영구 주 장치 번호 (지정되지 않았을 경우 -1로 됨)
lv_minor	Min	논리 볼륨의 영구 부 장치 번호 (지정되지 않았을 경우 -1로 됨)
lv_name	LV	논리 볼륨 이름
lv_size	LSize	논리 볼륨 크기
lv_tags	LV Tags	논리 볼륨에 부착된 LVM 태그
lv_uuid	LV UUID	논리 볼륨의 UUID
mirror_log	Log	미러 로그가 있는 장치
modules	Modules	논리 볼륨을 사용하기 위해 알맞은 커널 장치 매퍼 대상 필요
move_pv	Move	**pvmove** 명령으로 생성된 임시적 논리 볼륨의 물리 볼륨 소스
origin	Origin	스냅샷 볼륨의 초기 장치
regionsize	Region	미러 논리 볼륨의 단위 크기

region_size		
seg_count	#Seg	논리 볼륨에서 세그먼트의 수
seg_size	SSize	논리 볼륨에서 세그먼트의 크기
seg_start	Start	논리 볼륨에서 세그먼트의 오프셋
seg_tags	Seg Tags	논리 볼륨의 세그먼트에 부착된 LVM 태그
segtype	Type	논리 볼륨의 세그먼트 유형 (예: 미러, 스트라이프, 선형)
snap_percent	Snap%	사용되고 있는 스냅샷 볼륨의 퍼센트
stripes	#Str	논리 볼륨에서 스트라이프 또는 미러 수
stripesize **stripe_size**	Stripe	스트라이프 논리 볼륨에서 스트라이트의 단위 크기

lvs 명령으로 **lv_name**, **vg_name**, **lv_attr**, **lv_size**, **origin**, **snap_percent**, **move_pv**, **mirror_log**, **copy_percent** 영역의 기본값을 볼 수 있습니다. 이는 기본값으로 볼륨 그룹과 함께 **vg_name** 및 **lv_name**에 의해 정렬됩니다.

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

lvs 명령과 함께 **-v** 인수를 사용하여 기본값 영역 보기에 **seg_count**, **lv_major**, **lv_minor**, **lv_kernel_major**, **lv_kernel_minor**, **lv_uuid**영역을 추가합니다.

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

lvs 명령의 **--segments** 인수를 사용하여 세그먼트 정보에 중점을 둔 기본값 정보를 볼 수 있습니다. **segments** 인수를 사용할 때, **seg** 접두부는 옵션값입니다. **lvs --segments** 명령으로 **lv_name**, **vg_name**, **lv_attr**, **stripes**, **segtype**, **seg_size** 영역의 기본값을 볼 수 있습니다. 이는 기본값으로 볼륨 그룹 내에서 **vg_name**, **lv_name**에 의해 정렬되고, 논리 볼륨 내에서는 **seg_start**에 의해 정렬됩니다. 논리 볼륨이 분리된 경우, 명령의 출력 결과가 이를 보여줍니다.

```
# 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 사용하여 기본값 영역 보기에 **seg_start**, **stripesize**, **chunksize** 영역을 추가합니다.

```
# 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
```

다음의 예에서는 하나의 논리 볼륨이 설정된 시스템에서 **lvs** 명령의 기본 출력값을 보여주고 있으며, 다음으로 **segments** 인수와 함께 **lvs** 명령의 기본 출력값을 보여주고 있습니다.

```
# 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 리포트 정렬

일반적으로 **lvs**, **vgs**, **pvs** 명령의 출력 결과가 올바르게 정렬되기 이전에 이를 내부적으로 저장해야 합니다. **--unbuffered** 인수를 지정하여 결과가 출력되자 마자 정렬되지 않은 출력 결과 보기를 할 수 있습니다.

다른 목록의 정렬 순서를 지정하려면 아무 리포트 명령에 **-O** 인수를 사용합니다. 출력 결과 자체에 이러한 영역을 포함시킬 필요는 없습니다.

다음의 예에서는 물리 볼륨명, 크기, 여유 공간을 나타내는 **pvs** 명령의 출력 결과를 보여주고 있습니다.

```
# 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
```

다음의 예에서는 여유 공간 영역에 따라 정렬된 동일한 출력 결과를 보여주고 있습니다.

```
# 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
```

다음의 예에서는 정렬하고 있는 영역 보기를 할 필요가 없는 경우를 보여주고 있습니다.

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

역정렬하여 보려면 **-**와 함께 **-O** 인수 뒤에 지정한 부분이 선행되어야 합니다.

```
# 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 리포트 보기에 해당하는 단위를 지정하려면, 리포트 명령의 **--units** 인수를 사용합니다. 바이트 (b), 킬로바이트 (k), 메가바이트 (m), 기가바이트 (g), 테라바이트(t), 엑사바이트(e), 페터바이트(p), 인간 판독 가능(h) 등을 지정할 수 있습니다. **lvm.conf** 파일의 **global** 섹션에서 **units** 매개 변수를 지정하여 기본값을 덮어쓰기할 수 있습니다.

다음의 예에서는 기본값 GB가 아닌 MB로 **pvs** 명령의 출력 결과를 보여주고 있습니다.

```
# 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
```

기본값으로 단위는 2의 배수 (1024의 배수)로 나타납니다. 단위를 대문자 (B, K, M, G, T, H)로 지정하여 단위가 1000의 배수로 나타나도록 할 수 있습니다.

다음의 명령은 기본값인 1024의 배수로서 출력 결과를 보여주고 있습니다.

```
# 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
```

다음의 명령은 1000의 배수로서의 출력 결과를 보여주고 있습니다.

```
#  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
```

(512 바이트로 지정된) 섹터 또는 사용자 정의 단위를 지정할 수 있습니다.

다음의 예에서는 섹터 수로 **pvs** 명령의 출력 결과를 보여주고 있습니다.

```
# 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
```

다음의 예에서는 4MB 단위로 된 **pvs** 명령의 출력 결과를 보여주고 있습니다.

```
# 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
```

5장. LVM 설정 예

다음 부분에서는 기본적인 LVM 설정 예를 다루고 있습니다.

5.1. 세 개의 디스크에 LVM 논리 볼륨 생성

예에서는 **/dev/sda1**, **/dev/sdb1**, **/dev/sdc1**에 있는 디스크로 구성된 **new_logical_volume**이라는 LVM 논리 볼륨을 생성합니다.

5.1.1. 물리 볼륨 생성

볼륨 그룹에 있는 디스크를 사용하려면, 이를 LVM 물리 볼륨으로 레이블합니다.

경고

이 명령으로 **/dev/sda1**, **/dev/sdb1**, **/dev/sdc1**에 있는 데이터를 삭제합니다.

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

5.1.2. 볼륨 그룹 생성

다음 명령으로 **new_vol_group** 볼륨 그룹을 생성합니다.

```
[root@tng3-1 ~]# vgcreate new_vol_group /dev/sda1 /dev/sdb1 /dev/sdc1
  Volume group "new_vol_group" successfully created
```

새 볼륨 그룹의 속성을 보기 위해 **vgs** 명령을 사용할 수 있습니다.

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

5.1.3. 논리 볼륨 생성

다음의 명령으로 **new_vol_group** 볼륨 그룹에서 **new_logical_volume** 논리 볼륨을 생성합니다. 예에서는 볼륨 그룹의 2GB를 사용하여 논리 볼륨을 생성합니다.

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

5.1.4. 파일 시스템 생성

다음 명령으로 논리 볼륨에 GFS2 파일 시스템을 생성합니다.

```
[root@tng3-1 ~]# mkfs.gfs2 -plock_nolock -j 1
/dev/new_vol_group/new_logical_volume
This will destroy any data on /dev/new_vol_group/new_logical_volume.

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

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

Syncing...
All Done
```

다음 명령으로 논리 볼륨을 마운트하고 파일 시스템 디스크 공간 사용량을 보고합니다.

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

5.2. 스트라이프 (Striped) 논리 볼륨 생성

예에서는 **/dev/sda1**, **/dev/sdb1**, **/dev/sdc1**에 있는 디스크의 데이터를 스트라이프로 나누는 **striped_logical_volume**이라는 LVM 스트라이프 논리 볼륨을 생성합니다.

5.2.1. 물리 볼륨 생성

볼륨 그룹에서 사용할 디스크를 LVM 물리 볼륨으로 레이블합니다.

경고

이 명령으로 **/dev/sda1**, **/dev/sdb1**, **/dev/sdc1**에 있는 데이터를 삭제합니다.

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

5.2.2. 볼륨 그룹 생성

다음 명령으로 **volgroup01** 볼륨 그룹을 생성합니다.

```
[root@tng3-1 ~]# vgcreate volgroup01 /dev/sda1 /dev/sdb1 /dev/sdc1
  Volume group "volgroup01" successfully created
```

새 볼륨 그룹의 속성을 보기 위해 **vgs** 명령을 사용할 수 있습니다.

```
[root@tng3-1 ~]# vgs
  VG                #PV #LV #SN Attr   VSize  VFree
  volgroup01          3   0   0 wz--n- 51.45G 51.45G
```

5.2.3. 논리 볼륨 생성

다음 명령으로 **volgroup01** 볼륨 그룹에서 **striped_logical_volume** 스트라이프 논리 볼륨을 생성합니다. 예에서는 3개의 스트라이프 및 4 KB의 스트라이프 한 개와 함께 2 GB의 논리 볼륨을 생성하고 있습니다.

```
[root@tng3-1 ~]# lvcreate -i3 -I4 -L2G -nstriped_logical_volume volgroup01
  Rounding size (512 extents) up to stripe boundary size (513 extents)
  Logical volume "striped_logical_volume" created
```

5.2.4. 파일 시스템 생성

다음 명령으로 논리 볼륨에 GFS2 파일 시스템을 생성합니다.

```
[root@tng3-1 ~]# mkfs.gfs2 -plock_nolock -j 1
/dev/volgroup01/striped_logical_volume
This will destroy any data on /dev/volgroup01/striped_logical_volume.

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

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

Syncing...
All Done
```

다음 명령으로 논리 볼륨을 마운트하고 파일 시스템 디스크 공간 사용량을 보고합니다.

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

5.3. 볼륨 그룹 나누기

예에서 기존 볼륨 그룹은 세 개의 물리 볼륨으로 구성되어 있습니다. 물리 볼륨에 사용되지 않은 공간이 충분할 경우, 새 디스크를 추가하지 않고 새로운 볼륨 그룹을 생성할 수 있습니다.

초기 설정에서, **mylv** 논리 볼륨은 **/dev/sda1**, **/dev/sdb1**, **/dev/sdc1**이라는 세 개의 물리 볼륨으로 구성된 **myvol** 볼륨 그룹에서 생성됩니다.

이러한 생성 절차를 마친 후, **myvg** 볼륨 그룹은 **/dev/sda1** 및 **/dev/sdb1**로 구성됩니다. 두 번째 볼륨 그

룹인 **yourvg**는 **/dev/sdc1**로 구성됩니다.

5.3.1. 여유 공간 지정

pvscan 명령으로 현재 볼륨 그룹에서 사용할 수 있는 여유 공간 크기를 지정할 수 있습니다.

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

5.3.2. 데이터 이동

/dev/sdc1에서 사용된 모든 물리 익스텐트를 **pvmove** 명령을 사용하여 **/dev/sdb1**로 옮길 수 있습니다. **pvmove** 명령이 실행되는 데 시간이 오래 걸릴 수 도 있습니다.

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

데이터를 옮긴 후, **/dev/sdc1**에 있는 모든 여유 공간을 확인할 수 있습니다.

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

5.3.3. 볼륨 그룹 나누기

새 볼륨 그룹 **yourvg**를 생성하려면, **vgsplit** 명령을 사용하여 **myvg** 볼륨 그룹을 나눕니다.

볼륨 그룹을 나누기 전, 논리 볼륨은 비활성화되어 있어야 합니다. 파일 시스템이 마운트되어 있을 경우, 논리 볼륨을 비활성화시키기 전 파일 시스템을 마운트 해제해야 합니다.

lvchange 명령이나 또는 **vgchange** 명령을 사용하여 논리 볼륨을 비활성화시킬 수 있습니다. 다음의 명령에서는 **mylv** 논리 볼륨을 비활성화하고 **myvg** 볼륨 그룹에서 **yourvg** 볼륨 그룹을 나눈 뒤, **/dev/sdc1** 물리 볼륨을 새 볼륨 그룹 **yourvg**로 옮기고 있습니다.

```
[root@tng3-1 ~]# lvchange -a n /dev/myvg/mylv
[root@tng3-1 ~]# vgsplit myvg yourvg /dev/sdc1
  Volume group "yourvg" successfully split from "myvg"
```

vgs 명령을 사용하여 두 개의 볼륨 그룹의 속성을 확인할 수 있습니다.

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

5.3.4. 새 논리 볼륨 생성

새 볼륨 그룹을 생성한 후, 새 논리 볼륨 **yourlv**를 생성할 수 있습니다.

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

5.3.5. 파일 시스템 생성 및 새로운 논리 볼륨 마운트하기

새 논리 볼륨에 파일 시스템을 만들어 이를 마운트할 수 있습니다.

```
[root@tng3-1 ~]#  mkfs.gfs2 -plock_nolock -j 1 /dev/yourvg/yourlv
This will destroy any data on /dev/yourvg/yourlv.

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

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

Syncing...
All Done

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

5.3.6. 원래의 논리 볼륨을 활성화하고 마운트하기

mylv 논리 볼륨을 비활성화시켰을 경우, 마운트하기 전 이를 다시 활성화시켜야 합니다.

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

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

5.4. 논리 볼륨에서 디스크 삭제하기

다음 예에서는 기존 논리 볼륨에서 디스크를 교체하거나 또는 다른 볼륨으로 디스크를 사용하여 디스크를 삭제하는 방법을 보여 주고 있습니다. 디스크를 삭제하려면, 먼저 LVM 물리 볼륨에 있는 익스텐트를 다른 디스크나 또는 디스크 모음으로 옮겨야 합니다.

5.4.1. 기존의 물리 볼륨으로 익스텐트 옮기기

예에서 논리 볼륨은 **myvg** 볼륨 그룹에 있는 네 개의 물리 볼륨으로 할당됩니다.

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

/dev/sdb1의 익스텐트를 삭제하고자 할 경우, 이를 볼륨 그룹에서 삭제할 수 있습니다.

볼륨 그룹에 있는 다른 물리 볼륨에 여유 익스텐트가 충분할 경우, 삭제하고자 하는 장치에서 다른 옵션 없이 **pvmove** 명령을 실행하면 익스텐트는 다른 장치로 할당됩니다.

```
[root@tng3-1 ~]# pvmove /dev/sdb1
  /dev/sdb1: Moved: 2.0%
 ...
  /dev/sdb1: Moved: 79.2%
 ...
  /dev/sdb1: Moved: 100.0%
```

pvmove 명령을 실행 완료하면, 익스텐트는 다음과 같이 분배됩니다:

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

vgreduce 명령을 사용하여 볼륨 그룹에서의 **/dev/sdb1** 물리 볼륨을 삭제합니다.

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

현재 디스크는 물리적으로 다른 사용자에게 삭제 또는 할당될 수 있습니다.

5.4.2. 새 디스크로 익스텐트 옮기기

예에서, 논리 볼륨은 **myvg** 볼륨 그룹에 있는 세 개의 물리 볼륨으로 할당되며 이는 다음과 같습니다:

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

/dev/sdb1의 익스텐트를 새 장치 **/dev/sdd1**로 옮기려 합니다.

5.4.2.1. 새 물리 볼륨 생성하기

/dev/sdd1에서 새 물리 볼륨을 생성합니다.

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

5.4.2.2. 새 물리 볼륨을 볼륨 그룹에 추가하기

기존의 **myvg** 볼륨 그룹에 **/dev/sdd1**을 추가합니다.

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

5.4.2.3. 데이터 이동

pvmove 명령을 사용하여 **/dev/sdb1**에서 **/dev/sdd1**로 데이터를 이동합니다.

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

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

5.4.2.4. 볼륨 그룹에서 기존의 물리 볼륨을 삭제하기

/dev/sdb1의 데이터를 삭제한 후에, 볼륨 그룹에서 이를 삭제할 수 있습니다.

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

현재 다른 볼륨 그룹으로 디스크를 재할당하거나 시스템에서 디스크를 삭제할 수 있습니다.

5.5. 클러스터에 미러 LVM 논리 볼륨 생성

클러스터에 미러 LVM 논리 볼륨을 생성하기 위해 단일 노드에서 미러 LVM 논리 볼륨을 생성하는 것과 동일한 명령 및 절차가 필요합니다. 하지만 클러스터에 미러 LVM 볼륨을 생성하려면 클러스터 및 클러스터 미러 인프라를 실행하고, 클러스터는 쿼터에 도달해야 하며, **lvm.conf** 파일에 있는 잠금 유형은 클러스터 잠금 기능을 활성화하기 위해 3.1절. “클러스터에 LVM 볼륨 생성 ”에 설명되어 있듯이 직접적으로 또는 **lvmconf** 명령을 실행하여 올바르게 설정되어 있어야 합니다.

다음 절차에서는 클러스터에 미러 LVM 볼륨을 생성합니다. 먼저 생성 절차에서는 클러스터 서비스가 설치되어 실행되고 있는지를 확인한 후 미러 볼륨을 생성합니다.

1. 클러스터에 있는 모든 노드에 의해 공유되는 미러 논리 볼륨을 생성하기 위해 잠금 기능 유형은 클러스터의 모든 노드에 있는 **lvm.conf**에 올바르게 설정되어 있어야 합니다. 기본값으로 잠금 기능 유

형은 로컬에 설정되어 있습니다. 이를 변경하려면, 클러스터의 각 노드에서 다음과 같은 명령을 실행하여 클러스터 잠금 기능을 활성화합니다:

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

2. 클러스터 논리 볼륨을 생성하려면, 클러스터 인프라가 반드시 설정되어 있어야 하고 클러스터에 있는 모든 노드에서 실행되고 있어야 합니다. 다음의 예에서는 문제가 발생했던 노드에서 **clvmd** 데몬이 실행되고 있는지를 확인합니다:

```
[root@doc-07 ~]# ps auxw | grep clvmd
root     17642  0.0  0.1 32164 1072 ?        Ssl  Apr06   0:00 clvmd -T20 -t
90
```

다음 명령으로 클러스터의 로컬 상태 보기를 합니다:

```
[root@example-01 ~]# cman_tool services
fence domain
member count  3
victim count  0
victim now    0
master nodeid 2
wait state    none
members       1 2 3

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

3. **cmirror** 패키지가 설치되었는지를 확인합니다.
4. **cmirrord** 서비스를 시작합니다.

```
[root@hexample-01 ~]# service cmirrord start
Starting cmirrord:                                         [  OK  ]
```

5. 미러를 생성합니다. 먼저 물리 볼륨을 생성합니다. 다음 명령으로 세 개의 물리 볼륨을 생성합니다. 물리 볼륨 중 두 개는 미러의 leg로 사용되며 세번째 물리 볼륨에는 미러 로그가 들어 있게 됩니다.

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

6. 볼륨 그룹을 생성합니다. 다음 예에서는 이전 단계에서 생성된 세 개의 물리 볼륨으로 구성된 **vg001** 볼륨 그룹을 생성합니다.

```
[root@doc-07 ~]# vgcreate vg001 /dev/xvdb1 /dev/xvdb2 /dev/xvdc1
  Clustered volume group "vg001" successfully created
```

vgcreate 명령 출력 결과에서는 볼륨 그룹이 클러스터됨을 가리킴에 유의합니다. 볼륨 그룹이 **vgs** 명령으로 클러스터됨을 확인할 수 있습니다. 이러한 명령은 볼륨 그룹의 속성을 표시합니다. 볼륨 그룹이 클러스터될 경우 이는 c 속성이 표시됩니다.

```
[root@doc-07 ~]# vgs vg001
  VG       #PV #LV #SN Attr   VSize  VFree
  vg001      3   0   0 wz--nc 68.97G 68.97G
```

7. 미러 논리 볼륨을 생성합니다. 예에서는 **vg001** 볼륨 그룹에서 **mirrorlv** 논리 볼륨을 생성하고 있습니다. 이 볼륨은 하나의 미러 leg를 갖습니다. 예에서는 논리 볼륨에 어떤 물리 볼륨 익스텐트를 사용할 지를 지정하고 있습니다.

```
[root@doc-07 ~]# lvcreate -l 1000 -m1 vg001 -n mirrorlv /dev/xvdb1:1-1000
/dev/xvdb2:1-1000 /dev/xvdc1:0
  Logical volume "mirrorlv" created
```

lvs 명령을 사용하여 미러 생성 진행 상태를 확인할 수 있습니다. 다음의 예에서는 미러가 완료되었을 때 47%에서 91%로 그리고 100%까지 동기화되는 상태를 보여주고 있습니다.

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

미러 완료 상태는 시스템 로그에 기록됩니다:

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

8. **-o +devices** 옵션과 함께 **lvs** 명령을 사용하여 미러 leg를 구성하는 장치와 함께 미러 설정을 확인할 수 있습니다. 예에서는 논리 볼륨이 두개의 선형 이미지와 하나의 로그로 구성되어 있음을 확인할 수 있습니다.

```
[root@doc-07 ~]# lvs -a -o +devices
  LV                 VG         Attr   LSize  Origin Snap%  Move Log
Copy%  Convert Devices
  mirrorlv           vg001      mwi-a-  3.91G
mirrorlv_mlog 100.00         mirrorlv_mimage_0(0),mirrorlv_mimage_1(0)
  [mirrorlv_mimage_0] vg001     iwi-ao  3.91G
/dev/xvdb1(1)
  [mirrorlv_mimage_1] vg001     iwi-ao  3.91G
/dev/xvdb2(1)
  [mirrorlv_mlog]    vg001      lwi-ao  4.00M
/dev/xvdc1(0)
```

lvs 명령의 **seg_pe_ranges** 옵션을 사용하여 데이터 레이아웃을 확인할 수 있습니다. 이 옵션을 사용하여 레이아웃이 올바르게 이중화되어 있는지를 확인할 수 있습니다. 이 명령의 출력 결과에서 **lvcreate** 및 **lvresize** 명령이 입력으로 얻는 것과 동일한 형식의 PE 범위를 확인할 수 있습니다.

```
[root@doc-07 ~]# lvs -a -o +seg_pe_ranges --segments
  PE Ranges
  mirrorlv_mimage_0:0-999 mirrorlv_mimage_1:0-999
  /dev/xvdb1:1-1000
  /dev/xvdb2:1-1000
  /dev/xvdc1:0-0
```

참고

LVM 미러 볼륨의 leg 중 하나에 문제가 발생할 경우 이를 복구하는 방법에 대한 내용은 6.3절. “LVM 미러 장애 복구 ”에서 확인하십시오.

6장. LVM 문제 해결

다음 부분에서는 다양한 LVM 문제를 해결하는 방법에 대해 다루고 있습니다.

6.1. 문제 해결 진단

명령이 예상했던 데로 작동하지 않을 경우, 다음과 같은 방법으로 진단할 수 있습니다:

- 상세한 출력 결과를 얻기 위해 명령에 **-v**, **-vv**, **-vvv**, **-vvvv** 인수를 사용합니다.
- 논리 볼륨 활성화와 관련된 문제일 경우, 설정 파일의 '로그(log)' 섹션에서 '활성화 (activation) = 1'을 설정하고 **-vvvv** 인수와 함께 명령을 실행합니다. 이에 대한 출력 결과를 확인한 후에 매개 변수를 0으로 다시 설정하여 메모리 부족시 잠금 상태로 인한 문제가 발생하지 않게 합니다.
- 진단 목적으로 정보 덤프를 제공하는 **lvmdump** 명령을 실행합니다. 자세한 내용은 **lvmdump**(8) 맨 페이지에서 참조하시기 바랍니다.
- 추가 시스템 정보를 위해 **lvs -v**, **pvs -a**, **dmsetup info -c** 명령을 실행합니다.
- **/etc/lvm/backup** 파일에서 메타 데이터의 마지막 백업을 **/etc/lvm/archive** 파일에서는 아카이브된 버전을 검사합니다.
- **lvm dumpconfig** 명령을 실행하여 최신 설정 정보를 확인합니다.
- 어떤 장치에 물리 볼륨이 있는 지에 대한 기록을 알기 위해 **/etc/lvm** 디렉토리에서 **.cache** 파일을 확인합니다.

6.2. 실패한 장치에 있는 정보 보기

lvs 또는 **vgs** 명령의 **-P** 인수를 사용하여 출력 결과에서 나타나지 않는 실패한 볼륨에 관한 정보를 확인합니다. 이러한 인수는 메타데이터가 내부적으로 완전하게 일치하지 않아도 실행하게 합니다. 예를 들어, **vg** 볼륨 그룹으로된 장치 중 하나가 실패했을 경우, **vgs** 명령을 실행하면 다음과 같은 출력 결과가 나타납니다.

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

vgs 명령의 **-P** 인수를 지정한 경우, 볼륨 그룹은 사용 가능하지 않게되지만 실패한 장치에 관한 상세 정보를 볼 수 있습니다.

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

예에서, 실패한 장치는 볼륨 그룹에 있는 선형 (linear) 및 스트라이프 (striped) 논리 볼륨을 실패하게 합니다. **-P** 인수없이 **lvs** 명령을 실행하면 다음과 같은 출력 결과가 나타납니다.

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

-P 인수를 사용하여 실패한 논리 볼륨을 확인합니다.

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

다음 예에서는 미러 논리 볼륨 중 하나의 leg가 실패했을 경우 **-P** 인수와 함께 사용된 **pvs** 및 **lvs** 명령에 대한 출력 결과를 보여줍니다.

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

```
[root@link-08 ~]# lvs -a -o +devices -P
  Partial mode. Incomplete volume groups will be activated read-only.
  LV                 VG    Attr   LSize   Origin Snap%  Move Log
Copy%  Devices
  my_mirror          corey mwi-a- 120.00G                    my_mirror_mlog
1.95 my_mirror_mimage_0(0),my_mirror_mimage_1(0)
  [my_mirror_mimage_0] corey iwi-ao 120.00G
unknown device(0)
  [my_mirror_mimage_1] corey iwi-ao 120.00G
/dev/sdb1(0)
  [my_mirror_mlog]     corey lwi-ao   4.00M
/dev/sdd1(0)
```

6.3. LVM 미러 장애 복구

다음 부분에서는 물리 볼륨의 기본 장치가 정지하여 LVM 미러 볼륨의 leg 중 하나에 장애가 발생하고 **mirror_log_fault_policy** 매개 변수가 **remove**로 설정되어 있어 미러를 수동으로 다시 구축해야 하는 상태에서 복구하는 방법에 대해 설명합니다. **mirror_log_fault_policy** 매개 변수 설정에 대한 보다 자세한 내용은 6.3절. “LVM 미러 장애 복구 ”에서 참조하십시오.

미러 leg가 실패할 경우, LVM은 미러 볼륨을 선형 볼륨으로 전환하여, 미러 중복 없이 이전과 같이 계속 실행되게 합니다. 이 때, 대체 물리 장치로서 사용할 미러를 다시 구축하기 위해 시스템에 새 디스크 장치를 추가할 수 있습니다.

다음의 명령으로 미러로 사용될 물리 볼륨을 생성합니다.

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

다음의 명령으로 **vg** 볼륨 그룹 및 미러된 볼륨 **groupfs**를 생성합니다.

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

lvs 명령을 사용하여 미러 볼륨의 레이아웃과 미러 leg 및 미러 로그에 대한 기본 장치를 확인합니다. 첫 번째 미러의 예는 아직 완전히 동기화되지 않았습니다; **Copy%** 란이 100.00이라고 나타날 때 까지 기다리셔야 합니다.

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

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

예에서 주요 **/dev/sda1** 미러 leg 작업에 장애가 발생하였습니다. 미러 볼륨에 쓰기 작업을 실행할 경우 LVM이 장애가 발생한 미러를 감지하게 됩니다. 이러한 경우, LVM은 미러를 단일 선형 (linear) 볼륨으로 전

환합니다. 전환 작업을 위해 **dd** 명령을 실행합니다.

```
[root@link-08 ~]# dd if=/dev/zero of=/dev/vg/groupfs count=10
10+0 records in
10+0 records out
```

lvs 명령을 사용하여 현재 장치가 선형 (linear) 장치로 되어 있는 지를 확인할 수 있습니다. 실패한 디스크의 경우 I/O 오류가 발생합니다.

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

이 때 논리 볼륨을 계속 사용할 수 있지만 미러 중복이 없게 됩니다.

미러 볼륨을 다시 구축하려면, 손상된 드라이브를 교체하고 쿨리 볼륨을 다시 생성합니다. 새 디스크로 교체하지 않고 같은 디스크를 사용하면, **pvcreate** 명령을 실행했을 때 "일치하지 않음 (inconsistent)"이라는 경고 메세지가 나타납니다. **vgreduce --removemissing** 명령을 실행하여 이러한 경고 메세지가 나타나지 않게 할 수 있습니다.

```
[root@link-08 ~]# pvcreate /dev/sdi[12]
  Physical volume "/dev/sdi1" successfully created
  Physical volume "/dev/sdi2" successfully created

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

다음으로 새로운 물리 볼륨으로 원래의 볼륨 그룹을 확장합니다.

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

[root@link-08 ~]# pvscan
  PV /dev/sdb1   VG vg   lvm2 [67.83 GB / 67.10 GB free]
  PV /dev/sdb2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdc1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdc2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdd1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdd2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sde1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sde2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdf1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdf2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdg1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdg2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdh1   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/sdh2   VG vg   lvm2 [67.83 GB / 67.83 GB free]
  PV /dev/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 (linear) 볼륨을 원래의 미러 상태로 전환합니다.

```
[root@link-08 ~]# lvconvert -m 1 /dev/vg/groupfs /dev/sdi1 /dev/sdb1 /dev/sdc1
  Logical volume mirror converted.
```

lvs 명령을 사용하여 미러가 복구되었는지를 확인할 수 있습니다.

```
[root@link-08 ~]# lvs -a -o +devices
  LV                VG   Attr   LSize   Origin Snap%  Move Log          Copy%
Devices
  groupfs           vg   mwi-a- 752.00M                    groupfs_mlog 68.62
groupfs_mimage_0(0),groupfs_mimage_1(0)
  [groupfs_mimage_0] vg   iwi-ao 752.00M
/dev/sdb1(0)
  [groupfs_mimage_1] vg   iwi-ao 752.00M
/dev/sdi1(0)
  [groupfs_mlog]     vg   lwi-ao   4.00M
/dev/sdc1(0)
```

6.4. 물리 볼륨 메타 데이터 복구

물리 볼륨의 볼륨 그룹 메타 데이터 영역이 뜻하지 않게 덮어쓰기 되었거나 삭제되었을 경우, 메타 데이터 영역이 올바르지 않다는 오류 메세지나 또는 시스템이 특정 UUID로 물리 볼륨을 찾을 수 없다는 오류 메세지를 받게 됩니다. 손실된 메타 데이터와 같은 UUID를 지정하여 물리 볼륨에 있는 새로운 메타 데이터 영역을 작성하여 이를 복구할 수 있습니다.

경고

작동 중인 LVM 논리 볼륨으로 이러한 작업을 실행할 수 없습니다. UUID를 잘못 지정할 경우 데이터를 손실하게 됩니다.

다음의 예에서는 메타 데이터가 없거나 충돌할 경우 나타날 수 있는 출력 결과를 보여주고 있습니다.

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

/etc/lvm/archive 디렉토리에서 확인하여 덮어 쓰기된 물리 볼륨에 해당하는 UUID를 찾을 수 있습니다. 해당 볼륨 그룹의 아카이브된 LVM 메타 데이터에 해당하는 ***VolumeGroupName_xxxx*.vg** 파일을 찾습니다.

다른 방법으로 비활성화된 볼륨을 찾아 **partial** (**-P**) 인수를 설정하여 손상된 볼륨 그룹에 대한 UUID를 찾을 수 있습니다.

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

pvcreate 명령의 **--uuid** 및 **--restorefile** 인수를 사용하여 물리 볼륨을 복구합니다. 다음의 예에서는 위의 UUID와 함께 , **FmGRh3-zhok-iVI8-7qTD-S5BI-MAEN-NYM5Sk**를 사용하여 물리 볼륨으로 **/dev/sdh1** 장치를 레이블하고 있습니다. 이 명령으로 볼륨 그룹에 최근 아카이브된 메타 데이터, **VG_00050.vg**에 있는 메타데이터 정보를 사용하여 물리 볼륨 레이블을 복구합니다. **restorefile** 인수는 **pvcreate** 명령을 지시하여 볼륨 그룹에 있는 이전 물리 볼륨과 호환하는 새 물리 볼륨을 만들어, 새로운 메타 데이터가 이전 물리 볼륨이 들어있는 데이터에 배치되지 않는지를 확인합니다. (예를 들어, 본래의 **pvcreate** 명령이 메타 데이터 위치를 제어하는 명령행 인수를 사용하거나 또는 다른 기본값을 사용하는 다른 버전의 소프트웨어를 사용하여 기존 물리 볼륨이 생성된 경우에 발생할 수 있음).**pvcreate** 명령은 LVM 메타 데이터 영역만을 덮어쓰기하고 기존의 데이터 영역에는 영향을 미치지 않습니다.

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

vgcfgrestore 명령을 사용하여 볼륨 그룹의 메타 데이터를 복구할 수 있습니다.

```
[root@link-07 backup]# vgcfgrestore VG
  Restored volume group VG
```

논리 볼륨을 나타낼 수 있습니다.

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

다음의 명령으로 볼륨을 활성화하고 활성 볼륨을 나타낼 수 있습니다.

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

디스크 상의 LVM 메타 데이터가 오버라이드된 것 만큼 많은 공간을 차지할 경우, 이 명령으로 물리 볼륨을 복구할 수 있습니다. 메타 데이터를 오버라이드한 것이 메타 데이터 영역을 지나갈 경우 볼륨에 있는 데이터에 영향을 미치게 됩니다. **fsck** 명령을 사용하여 데이터를 복구할 수 있습니다.

6.5. 손실된 물리 볼륨 대체

물리 볼륨이 실패하거나 대체되어야할 경우, 기존 볼륨 그룹에서 손실된 물리 볼륨을 대체하기 위해 6.4절. “물리 볼륨 메타 데이터 복구”에서 설명하고 있듯이 물리 볼륨 메타 데이터 복구하는 것과 같은 절차로 새 물리 볼륨을 레이블할 수 있습니다. **vgdisplay** 명령의 **--partial** 및 **--verbose** 인수를 사용하여 UUID 및 더 이상 존재하지 않는 물리 볼륨의 크기를 확인합니다. 같은 크기의 다른 물리 볼륨을 대체하고자 할 경우, **--restorefile** 및 **--uuid** 인수와 함께 **pvcreate** 명령을 사용하여 손실된 물리 볼륨과 같은 UUID를 사용하는 새로운 장치를 초기화할 수 있습니다. 그 후 **vgcfgrestore** 명령으로 볼륨 그룹의 메타 데이터를 복구합니다.

6.6. 볼륨 그룹에서 손실된 물리 볼륨 제거

물리 볼륨이 손실된 경우, **vgchange** 명령의 **--partial** 인수를 사용하여 볼륨 그룹에 있는 남아있는 다른 물리 볼륨을 활성화할 수 있습니다. **vgreduce** 명령의 **--removemissing** 인수를 사용하여 볼륨 그룹에서 물리 볼륨이 사용된 모든 논리 볼륨을 삭제할 수 있습니다.

삭제할 내용을 확인하기 위해 **--test** 인수와 함께 **vgreduce** 명령을 실행할 것을 권장합니다.

대부분의 LVM 실행과 같이, **vgcfgrestore** 명령을 사용하여 이전 상태로 볼륨 그룹 메타데이터를 복구할 경우, **vgreduce** 명령으로 역실행가능합니다. 예를 들어, **--test** 인수 없이 **vgreduce** 명령의 **--removemissing** 인수를 사용하여 보관하고자 했던 논리 볼륨이 삭제된 것을 발견했을 경우, 물리 볼륨을 대체하여 이전 상태로 볼륨 그룹을 복구하기 위해 다른 **vgcfgrestore** 명령을 사용하실 수 있습니다.

6.7. 논리 볼륨에 대해 불충분한 여유 익스텐트

vgdisplay 또는 **vgs** 명령의 출력 결과에 기반하여 익스텐트가 충분하다고 생각되어 논리 볼륨을 생성할 경우 "여유 익스텐트가 충분하지 않음"이라는 오류 메세지가 나타날 수 있습니다. 이는 판독 가능한 출력 결과를 위해 명령이 소수 2 자리로 나타나기 때문입니다. 논리 볼륨 크기를 정확하게 지정하려면 바이트 배수 대신 여유 물리 익스텐트 수를 사용합니다.

기본값으로 **vgdisplay** 명령에는 여유 물리 익스텐트를 나타내는 출력 결과 행이 포함됩니다.

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

다른 방법으로 **vgs** 명령의 **vg_free_count** 및 **vg_extent_count** 인수를 사용하여 여유 익스텐트 및 총 익스텐트 수를 나타낼 수 있습니다.

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

8780 여유 물리 익스텐트와 함께 바이트대신 익스텐트를 사용하기 위해 소문자 l 인수를 사용하여 다음과 같은 명령을 실행할 수 있습니다:

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

이는 볼륨 그룹에 있는 모든 여유 익스텐트를 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
```

다른 방법으로, **lvcreate** 명령의 **-l** 인수를 사용하여 볼륨 그룹에 있는 남아있는 여유 공간의 퍼센트를 사용하기 위해 논리 볼륨을 확장할 수 있습니다. 보다 자세한 내용은 4.4.1절. “선형 논리 볼륨 생성 ”에서 참조하시기 바랍니다.

7장. LVM GUI를 통한 LVM 관리

CLI (Command Line Interface)에 더하여, LVM은 LVM 논리 볼륨 설정에 사용할 수 있는 GUI (Graphical User Interface)를 제공합니다. **`system-config-lvm`** 명령을 사용하여 이러한 유틸리티를 불러올 수 있습니다. *스토리지 관리 가이드*의 LVM 부분에서는 이러한 유릴리티를 사용하여 LVM 논리 볼륨을 설정하는 방법에 대해 단계적으로 설명하고 있습니다.

장치 매퍼 (Device Mapper)

장치 매퍼 (Device Mapper)는 볼륨 관리를 위한 프레임워크를 제공하는 커널 드라이버로 논리 볼륨으로 사용될 맵화된 장치 생성에 대한 일반적인 방법을 제공합니다. 이는 볼륨 그룹이나 메타 데이터 포맷에 대해 명확하게 알고 있지 못합니다.

장치 매퍼는 보다 높은 수준의 테크놀로지 기반을 제공합니다. LVM에 더하여, 장치-매퍼 멀티 패스 및 **dmraid** 명령은 장치 매퍼를 사용합니다. 장치 매퍼로의 애플리케이션 인터페이스는 **ioctl** 시스템 호출입니다. 사용자 인터페이스는 **dmsetup** 명령입니다.

LVM 논리 볼륨은 장치 매퍼(Device Mapper)를 사용하여 활성화됩니다. 각각의 논리 볼륨은 맵핑된 장치로 전환되고, 각각의 세그먼트는 장치를 설명하는 맵핑 테이블에 있는 행으로 전환됩니다. 장치 매퍼는 선형 (linear) 맵핑, 스트라이프 (striped) 맵핑, 에러 (error) 맵핑을 포함하여 다양한 맵핑 대상을 지원합니다. 예를 들어, 두 개의 디스크는 선형 맵핑과 병행하여 각각의 디스크에 하나씩 하나의 논리 볼륨으로 연결될 수 있습니다. LVM2가 볼륨을 생성할 때, 이는 **dmsetup** 명령으로 쿼리 가능한 기본 장치 맵퍼를 생성합니다. 맵핑 테이블에서 장치 포맷에 관한 내용은 A.1절. “장치 테이블 맵핑 ”에서 참조하십시오. 장치를 쿼리하기 위해 **dmsetup** 명령을 사용하는 방법에 관한 내용은 A.2절. “dmsetup 명령 ”에서 참조하십시오.

A.1. 장치 테이블 맵핑

맵핑된 장치는 지원되는 장치 테이블 매핑을 사용하여 장치의 논리 섹터의 각 영역을 맵핑하는 방법을 지정하는 테이블에 의해 정의됩니다. 맵핑된 장치의 테이블은 다음과 같은 형식의 행 목록에서 구성됩니다:

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

장치 매퍼 테이블의 첫 번째 행에서, ***start*** 매개 변수는 0과 동일해야 합니다. 하나의 행에 있는 ***start*** + ***length*** 매개 변수는 다음 행의 ***start***와 동일해야 합니다. 어떤 맵핑 매개변수가 맵핑 테이블의 행에 지정되어야 하는 가는 어떤 ***mapping*** 유형이 해당 행에 지정되는 가에 따라 결정됩니다.

장치 매퍼 크기는 항상 섹터에서 지정됩니다 (512 바이트).

장치 맵퍼에서 맵핑 매개변수로 장치가 지정될 경우, 이는 파일 시스템 (예: **/dev/hda**)에서 장치 이름에 의해서나 또는 ***major*:*minor*** 형식의 major 및 minor 번호에 의해 참조될 수 있습니다. major:minor 형식은 경로명 검색을 피할 수 있기 때문에 선호됩니다.

장치의 매핑 테이블 예제는 다음과 같습니다. 이 테이블에는 4 개의 선형 대상이있습니다:

```
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 첫 번째 2 개의 매개 변수는 세그먼트 시작점 블록 및 세그먼트의 길이입니다. 다음 키워드는 맵팽 대상으로, 예에서 모든 경우 **linear**가 됩니다. 나머지 행은 **linear** 대상에 대한 매개 변수로 구성됩니다.

다음의 하부 섹션에서는 맵핑 형식을 설명합니다:

- linear
- striped
- mirror
- snapshot 및 snapshot-origin
- error
- zero

- multipath
- crypt

A.1.1. 선형 맵핑 대상

선형 맵핑 대상은 다른 블록 장치에 연속적인 범위의 블록을 맵핑합니다. 선형 대상 형식은 다음과 같습니다:

```
start length linear device offset
```

start

가상 장치에서 시작점 블록

length

세그먼트 길이

device

블록 장치, 파일 시스템에서 장치 이름에 의해 또는 ***major*:*minor*** 형식에서 major 및 minor 번호에 의해 참조됩니다

offset

장치에서 맵핑의 시작점 오프셋

다음 예제에서는 가상 장치의 시작점 블록이 0, 세그먼트 길이가 1638400, major:minor 번호 쌍이 8:2, 장치의 시작점 오프셋이 41146992인 선형 대상을 보여주고 있습니다.

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

다음의 예에서는 **/dev/hda**로 지정된 장치 매개 변수와 함께 선형 대상을 보여주고 있습니다.

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

A.1.2. 스트라이프 맵핑 대상

스트라이프 맵핑 대상은 물리적 장치 전역에서 스트라이프를 지원합니다. 이는 스트라이프 수, 스트라이핑 chunk size, 장치 이름 및 섹터 쌍의 목록을 인수로 사용합니다. 스트라이프 대상 형식은 다음과 같습니다:

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

각각의 스트라이프에 대해 ***device*** 및 ***offset*** 매개 변수 모음 하나가 있습니다.

start

가상 장치에서 시작점 블록

length

세그먼트 길이

#stripes

가상 장치 용 스트라이프 수

chunk_size

다음으로 전환하기 전 까지 각각의 스트라이프에 작성된 섹터 수량; 컨널 페이지 크기 만큼 큰 것으로 최소 2 개의 전력이 되어야 합니다

device

블록 장치, 파일 시스템에서 장치 이름에 의해서나 또는 ***major*:*minor*** 형식에서 major 및 minor 번호에 의해 참조됩니다.

offset

장치에서 맵핑의 시작점 오프셋

다음의 예에서는 세 개의 스트라이프 및 128 chunk size를 갖는 스트라이프 대상을 보여주고 있습니다:

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

0

가상 장치에서 시작점 블록

73728

세그먼트 길이

striped 3 128

128 블록의 chunk size를 갖는 세 개의 장치를 통해 스트라이프

8:9

첫 번째 장치의 major:minor 번호

384

첫 번째 장치에서 맵핑의 시작점 오프셋

8:8

두 번째 장치의 major:minor 번호

384

두 번째 장치에서 맵핑의 시작점 오프셋

8:7

세 번째 장치의 major:minor 번호

9789824

세 번째 장치에서 맵핑의 시작점 오프셋

다음의 예제에서는 256 KiB chunk와 함께 major 및 minor 번호 대신 파일 시스템에 있는 장치 이름에 의해 지정된 장치 매개 변수를 갖는 2 개의 스트라이프에 대한 스트라이프 대상을 보여주고 있습니다.

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

A.1.3. 미러 맵핑 대상

미러 맵핑 대상은 미러 논리 장치의 맵핑을 지원합니다. 미러 대상의 형식은 다음과 같습니다:

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

start

가상 장치에서 시작점 블록

length

세그먼트 길이

log_type

가능한 로그 유형 및 인수는 다음과 같습니다:

core

미러는 로컬로 되고 미러 로그는 코어 메모리에 저장됩니다. 이러한 로그 유형은 1 - 3 인수를 갖습니다:

regionsize [**no**]**sync**] [**block_on_error**]

disk

미러는 로컬로 되고 미러 로그는 디스크에 저장됩니다. 이러한 로그 유형은 2 - 4 인수를 갖습니다:

logdevice regionsize [**no**]**sync**] [**block_on_error**]

clustered_core

미러는 클러스터되고 미러 로그는 코어 메모리에 저장됩니다. 이러한 로그 유형은 2 - 4 인수를 갖습니다:

regionsize UUID [**no**]**sync**] [**block_on_error**]

clustered_disk

미러는 클러스터되고 미러 로그는 디스크에 저장됩니다. 이러한 로그 유형은 3 - 5 인수를 갖습니다:

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

LVM은 어떤 영역이 미러와 동기화하는 지를 기록하기 위해 사용하는 로그를 유지합니다.

regionsize 인수는 이러한 영역의 크기를 지정합니다.

클러스터된 환경에서, ***UUID*** 인수는 미러 로그 장치와 관련된 고유한 식별자이므로 로그 상태는 클러스터 전역에서 유지될 수 있습니다.

[no]sync 인수 옵션은 "in-sync" 또는 "out-of-sync"로 미러를 지정하기 위해 사용될 수 있습니다. **block_on_error** 인수는 미러에 대해 오류를 무시하는 것이 아니라 오류를 처리하도록 지시하는데 사용됩니다.

#log_args

맵핑에서 지정될 로그 인수의 수

logargs

미러에 대한 로그 인수; 제공되는 로그 인수 수량은 ***#log-args*** 매개 변수에 의해 지정되고 유효한 로그 인수는 ***log_type*** 매개 변수에 의해 결정됩니다.

#devs

미러에서 leg 수; 각 leg에 대해 장치 및 오프셋이 지정됩니다.

device

각 미러 leg에 대한 블록 장치, 파일 시스템에 있는 장치 이름으로 참조 또는 ***major*:*minor*** 형식에 있는 major 및 minor 번호에 의해 참조. 블록 장치 및 오프셋은 각각의 미러 leg에 대해 지정되며, ***#devs*** 매개 변수에 의해 표시됩니다.

offset

장치에서 맵핑의 시작점 오프셋. 블록 장치 및 오프셋은 ***#devs*** 매개 변수에 의해 나타나는 각 미러 leg에 대해 지정됩니다.

다음의 예제에서는 디스트레 저장된 미러 로그와 함께 클러스터된 미러에 대한 미러 맵핑 대상을 보여주고 있습니다.

```
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 시작점 블록

52428800

세그먼트 길이

mirror clustered_disk

미러가 클러스터되어 있고 미러 로그가 디스크 상에서 유지되고 있음을 지정하는 로그 유형을 갖는 미러 대상 \t

4

4 개의 미러 로그 인수가 계속됩니다

253:2

로그 장치의 major:minor 번호

1024

동기화하고 있는 기록을 보관하기 위해 미러 로그가 사용하는 영역 크기

UUID

클러스터를 통해 로그 정보를 관리하기 위한 미러 로그 장치의 UUID

block_on_error

미러는 오류에 대응해야 합니다

3

미러에서 leg 수

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

미러의 각 leg를 구성하는 장치에 대한 major:minor 번호 및 오프셋

A.1.4. snapshot 및 snapshot-origin 맵핑 대상

볼륨의 첫 번째 LVM 스냅샷을 생성할 때, 네 개의 장치 맵퍼가 사용됩니다:

1. 소스 볼륨의 기존 맵핑 테이블이 들어 있는 **linear** 맵핑을 갖는 장치
2. 소스 볼륨에 대해 COW (copy-on-write) 장치로 사용되는 **linear** 맵핑을 갖는 장치; 각각의 쓰기 작업에 대해, 기존 데이터는 각 스냅샷의 COW 장치에 저장되어 가시적 컨텐츠가 변경되지 않게 보관됩니다 (COW 장치가 채워질 때 까지).
3. 가시적 스냅샷 볼륨인 #1과 #2를 결합한 **snapshot** 맵핑을 갖는 장치
4. "기존" 볼륨 (이는 기존 소스 볼륨에 의해 사용되는 장치 번호를 사용합니다). 이러한 볼륨의 테이블은 장치 #1에서 "snapshot-origin" 맵핑하여 교체됩니다.

이러한 장치를 생성하기 위해 고정된 이름 지정 체계를 사용합니다. 예를 들어, 다음과 같은 명령을 사용하여 **base**라는 LVM 볼륨을 생성하고 해당 볼륨에 **snap** 라는 스냅샷 볼륨을 생성할 수 있습니다.

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

이는 4 개의 장치를 만들어 내며, 이는 다음의 명령으로 볼 수 있습니다:

```
# 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 시작점 블록

length

세그먼트 길이

origin

스냅샷의 기본 볼륨

일반적으로 **snapshot-origin**은 이를 기반으로 하는 하나 이상의 스냅샷을 가지고 있습니다. 읽기 작업은 백업 장치에 직접 맵핑됩니다. 각각의 쓰기 작업의 경우 기존 데이터는 각각의 스냅샷의 COW 장치에 저장되어 COW 장치가 채워질 때 까지 가시적 컨텐츠가 변경되지 않게 보관됩니다.

snapshot 대상의 형식은 다음과 같습니다.

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

start

가상 장치에서 시작점 블록

length

세그먼트 길이

origin

스냅샷의 기본 볼륨

COW-device

변경된 데이터 청크가 저장된 장치

P|N

P (Persistent) 또는 N (Not persistent); 재부팅 후 스냅샷이 유지되고 있는지에 대한 여부를 나타

냅니다. 임시 스냅샷 (N)의 경우 많은 데이터를 디스크에 저장할 수 없으며, 커널에 의해 메모리에 저장할 수 있습니다.

chunksize

COW 장치에 저장될 변경된 데이터 청크의 섹터 크기

다음의 예제에서는 254:11 원본 장치를 갖는 **snapshot-origin** 대상을 보여주고 있습니다.

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

다음 예제에서는 254:11 원본 장치와 254:12 COW 장치를 갖는 **snapshot** 대상을 보여주고 있습니다. 이러한 스냅샷 장치는 재부팅 후에도 지속되며 COW 장치에 저장된 데이터의 청크 크기는 16 섹터입니다.

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

A.1.5. 오류 맵핑 대상

오류 맵핑 대상과 함께 맵핑된 섹터로의 I/O 작업은 모두 실패합니다.

오류 맵핑 대상은 테스트 용으로 사용할 수 있습니다. 장애시 장치가 어떻게 작동하는지를 테스트하려면, 장치 중간에 잘못된 섹터로 맵핑된 장치를 생성하거나 또는 미러의 leg를 옮겨 비우기하여 leg를 오류 대상으로 교체합니다.

오류 대상은 장애가 있는 장치의 위치에서, 시간 제한을 피하는 방법으로 실제 장치에서 다시 시작하는데 사용될 수 있습니다. 이는 장애 발생 시 LVM 메타데이터를 재구성하는 동안 중간 대상으로서 역할을 수행합니다.

error 맵핑 대상은 ***start*** 및 ***length*** 이외에 추가 매개 변수를 갖지 않습니다.

다음의 예제에서는 **error** 대상을 보여주고 있습니다.

```
0 65536 error
```

A.1.6. zero 맵핑 대상

zero 맵핑 대상은 **/dev/zero**와 동등한 블록 장치입니다. 이러한 맵핑으로의 읽기 작업을 영 블록을 반환합니다. 맵핑에 작성된 데이터는 삭제되지만 쓰기 작업은 성공합니다. **zero** 맵핑 대상은 ***start*** 및 ***length*** 매개 변수 이외에 추가 매개 변수를 갖지 않습니다.

다음의 예제에서는 16Tb 장치 용 **zero** 대상을 보여주고 있습니다.

```
0 65536 zero
```

A.1.7. 멀티패스 맵핑 대상

멀티패스 맵핑 대상은 멀티패스된 장치의 맵핑을 지원합니다. **multipath** 대상의 형식은 다음과 같습니다:

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

각 경로 그룹에 대한 ***pathgroupargs*** 매개 변수 모음이 하나 있습니다.

start

가상 장치에서 시작점 블록

length

세그먼트 길이

#features

멀티패스 기능의 수로 이러한 기능은 다음에 표시됩니다. 이러한 매개 변수가 영이되면, ***feature*** 매개 변수가 없게 되고 다음의 장치 맵핑 매개 변수는 ***#handlerargs***가 됩니다. 현재 지원되는 멀티패스 기능은 **queue_if_no_path** 하나 입니다. 이는 사용 가능한 경로가 없을 경우 현재 멀티패스된 장치는 I/O 작업을 대기열로 하도록 설정됨을 의미합니다.

예를 들어, **multipath.conf** 파일의 **no_path_retry** 옵션이 지정된 일련의 시도 횟수가 경로를 사용한 후 전체 경로가 실패로 표시될 때 까지 I/O 작업을 대기열로 하도록 설정되어 있을 경우, 맵핑은 전체 경로 검사기가 지정된 검사를 실패할 때 까지 다음과 같이 나타나게 됩니다.

```
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
```

전체 경로 검사기가 지정된 검사를 실패한 후, 맵핑은 다음과 같이 나타나게 됩니다.

```
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 인수 수량으로 이러한 인수가 그 다음에 표시됩니다. 하드웨어 처리기는 경로 그룹을 전환하거나 또는 I/O 오류를 처리할 때 하드웨어 특정 작업을 실행하기 위해 사용되는 모듈을 지정합니다. 이것이 0으로 설정되어 있을 경우, 다음 매개 변수는 ***#pathgroups***이 됩니다.

#pathgroups

경로 그룹의 수량. 경로 그룹은 멀티패스된 장치가 로드 밸런스를 수행하는 경로 모음입니다. 각각의 경로 그룹에 대해 ***pathgroupargs*** 매개 변수 모음이 하나 있습니다.

pathgroup

시도할 다음 경로 그룹

pathgroupsargs

각 경로 그룹은 다음과 같은 인수로 구성되어 있습니다:

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

경로 그룹에는 각 경로에 대한 경로 인수 모음 하나가 있습니다.

pathselector

경로 그룹에서 어떤 경로를 다음 I/O 작업에 사용할 지를 결정하기 위해 사용되고 있는 알고리즘을 지정합니다.

#selectorargs

멀티패스 맵핑에서 이러한 인수를 따르는 경로 선택기 인수의 수량. 현재 이러한 인수 값은 항상 0입니다.

#paths

경로 그룹에 있는 경로 수량

#pathargs

이 그룹의 각 경로에 지정된 경로 인수의 수량. 현재 이러한 숫자는 항상 ***ioreqs*** 인수인 1로 되어 있습니다.

device

경로의 블록 장치 번호, ***major*:*minor*** 형식으로 major와 minor 번호에 의해 참조됩니다

ioreqs

현재 그룹의 다음 경로로 전환하기 전 이러한 경로로 라우팅하기 위한 I/O 요청 수.

그림 A.1. “멀티패스 맵핑 대상 ”에서는 두 개의 경로 그룹이 있는 멀티패스 대상 형식을 보여주고 있습니다.

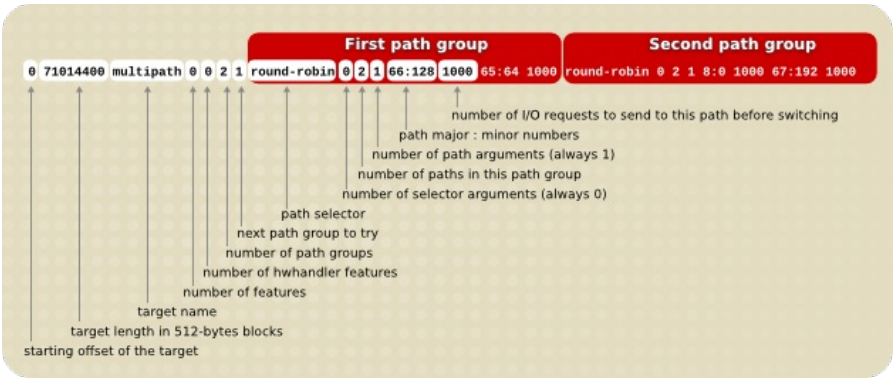

그림 A.1. 멀티패스 맵핑 대상

다음 예제에서는 동일한 멀티패스 장치에 대한 순수한 장애 조치 대상의 정의를 보여주고 있습니다. 이러한 대상에는 경로 그룹 당 경로 하나만 열려있는 4 개의 경로 그룹이 있어 멀티패스된 장치는 한 번에 하나의 경로만을 사용하게 됩니다.

```
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
```

다음의 예제에서는 동일한 멀티패스 장치에 대해 완전하게 확산된 (multibus) 대상을 보여주고 있습니다. 이러한 대상에는 모든 경로를 포함하는 하나의 경로 그룹만 있습니다. 이러한 설정에서 멀티패스는 경로 전체에 부하를 균등하게 확산합니다.

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

멀티패싱에 관한 보다 자세한 내용은 *Device Mapper Multipath 사용* 문서를 참조하십시오.

A.1.8. crypt 맵핑 대상

crypt 대상은 지정된 장치를 통해 전송된 데이터를 암호화합니다. 이는 커널 Crypto API를 사용합니다.

crypt 대상의 형식은 다음과 같습니다:

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

start

가상 장치에서 시작점 블록

length

세그먼트 길이

cipher

Cipher는 ***cipher[-chainmode]-ivmode[:iv options]***로 구성되어 있습니다.

cipher

사용 가능한 Ciphers는 **/proc/crypto**에 나열되어 있습니다 (예: **aes**)

chainmode

항상 **cbc**를 사용합니다. **ebc**는 사용하지 않습니다; 이는 초기 백터 (IV)를 사용하지 않습니다.

ivmode[:iv options]

IV는 초기 벡터 (initial vector)로 암호를 변경하는 데 사용됩니다. IV 모드는 **plain** 또는 **essiv:hash**입니다. **-plain**의 ***ivmode***는 IV로서 섹터 번호 (및 IV 오프셋)를 사용합니다. **-essiv**의 ***ivmode***는 워터마크의 약점을 피하기 위해 강화되었습니다.

key

암호화 키, 16 진법으로 공급

IV-offset

초기 백터 (IV) 오프셋

device

블록 장치, 파일 시스템에서 장치 이름에 의해 또는 ***major*:*minor*** 형식에서 major 및 minor 번호에 의해 참조됩니다

offset

장치에서 맵핑의 시작점 오프셋

다음은 **crypt** 대상의 예제입니다.

```
0 2097152 crypt aes-plain 0123456789abcdef0123456789abcdef 0 /dev/hda 0
```

A.2. dmsetup 명령

dmsetup 명령은 장치 매퍼와 통신하기 위한 명령행 래퍼 (wrapper)입니다. LVM 장치에 관한 일반적인 시스템 정보는 다음의 하부 섹션에서 설명하고 있듯이 **dmsetup** 명령의 **info**, **ls**, **status**, **deps** 옵션을 사용하여 참조하시기 바랍니다.

dmsetup명령의 추가 옵션 및 기능에 관한 내용은 **dmsetup**(8) 맨 페이지에서 참조하십시오.

A.2.1. dmsetup info 명령

dmsetup info *device* 명령은 Device Mapper 장치에 관한 요약 정보를 제공합니다. 장치 이름을 지정하지 않으셨을 경우, 출력 결과에는 현재 설정된 모든 Device Mapper 장치에 관한 정보가 나타납니다. 장치를 지정하셨을 경우, 이 명령은 해당 장치에 대한 정보만을 제공합니다.

dmsetup info 명령은 다음과 같은 범주에 있는 정보를 제공합니다:

Name

장치의 이름입니다. LVM 장치는 하이픈으로 구별된 볼륨 그룹 이름 및 논리 볼륨 이름으로 나타납니다. 원래 이름에서의 하이픈은 두 개의 하이픈으로 변환됩니다.

State

가능한 장치 상태는 **SUSPENDED**, **ACTIVE**, **READ-ONLY**가 됩니다. **dmsetup suspend** 명령은 장치 상태를 **SUSPENDED**로 설정합니다. 장치가 일시 정지 상태가 되어 있을 경우, 해당 장치의 모든 I/O 작업이 중지됩니다. **dmsetup resume** 명령은 장치의 상태를 **ACTIVE**로 복원합니다.

Read Ahead

읽기 작업이 실행되고 있는 열린 파일에 대해 시스템을 미리 읽어오는 데이터 블록의 수량입니다. 기본값으로 커널은 자동으로 적절한 값을 선택합니다. **--readahead** option of the **dmsetup** 명령을 사용하여 이러한 값을 변경할 수 있습니다.

Tables present

이 카테고리의 가능한 상태 값은 **LIVE** 및 **INACTIVE** 입니다. **INACTIVE** 상태는 테이블 상태가 **LIVE**가 되도록 **dmsetup resume** 명령이 장치 상태를 **ACTIVE**로 복원할 때 교체될 수 있게 테이블이 로딩되는 것을 나타냅니다. 자세한 내용은 **dmsetup** 맨 페이지를 참조하십시오.

Open count

open reference count는 장치가 열린 횟수를 나타냅니다. **mount** 명령은 장치를 엽니다.

Event number

수신된 현재 이벤트 수. **dmsetup wait *n*** 명령을 실행하면 수신될 때 까지 호출을 차단하여 사용자는 n 번째 이벤트를 기다리게 합니다.

Major, minor

Major 및 minor 장치 번호

Number of targets

장치를 구성하는 부분의 수량입니다. 예를 들어, 3 개의 디스크를 통과하는 선형 장치는 3 개의 대상을 갖게 됩니다. 디스크의 시작과 끝 지점으로 구성되는 중간 지점이 없는 선형 장치는 2 개의 대상을 갖게 됩니다.

UUID

장치의 UUID

다음의 예제에서는 **dmsetup info** 명령의 일부 출력 결과를 보여주고 있습니다.

```
[root@ask-07 ~]# dmsetup info
Name:              testgfsvg-testgfslv1
State:             ACTIVE
Read Ahead:        256
Tables present:    LIVE
Open count:        0
Event number:      0
Major, minor:      253, 2
Number of targets: 2
UUID: LVM-K528WUGQgPadNXYcFrrf9LnPlUMswgkCkpgPIgYzSvigM7SfeWCypddNSWtNzc2N
...
Name:              VolGroup00-LogVol00
State:             ACTIVE
Read Ahead:        256
Tables present:    LIVE
Open count:        1
Event number:      0
Major, minor:      253, 0
Number of targets: 1
UUID: LVM-tOcS1kqFV9drb0X1Vr8sxeYP0tqcrpdegyqj5lZxe45JMGlmvtqLmbLpBcenh2L3
```

A.2.2. dmsetup ls 명령

dmsetup ls 명령으로 맵핑된 장치의 장치 이름을 나열할 수 있습니다. **dmsetup ls --target *target_type*** 명령을 사용하여 최소 한개의 특정 유형의 대상을 갖는 장치를 나열할 수 있습니다. **dmsetup ls** 명령의 다른 옵션은 **dmsetup** 맨 페이지를 참조하십시오.

다음 예제에서는 현재 설정된 맵핑 장치의 장치 이름을 나열하는 명령을 보여주고 있습니다.

```
[root@ask-07 ~]# dmsetup ls
testgfsvg-testgfslv3    (253, 4)
testgfsvg-testgfslv2    (253, 3)
testgfsvg-testgfslv1    (253, 2)
VolGroup00-LogVol01     (253, 1)
VolGroup00-LogVol00     (253, 0)
```

다음 예제에서는 현재 설정된 미러 맵핑의 장치 이름을 나열하는 명령을 보여주고 있습니다.

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

멀티패스나 다른 장치 매퍼 디바이스에 스택되는 LVM 설정은 정렬하기에 복잡할 수 있습니다. **dmsetup ls** 명령은 다음의 예에서와 같이 장치간의 의존 관계를 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 특정 장치에 있는 각각의 대상에 대한 상태 정보를 제공합니다. 장치 이름을 지정하지 않으셨을 경우, 출력 결과에는 현재 설정된 모든 Device Mapper 장치에 관한 정보가 나타나게 됩니다. **dmsetup status --target *target_type*** 명령을 사용하여 최소 한개의 특정 유형의 대상을 갖는 장치의 상태 만을 나열할 수 있습니다.

다음 예제에서는 현재 설정된 맵핑 장치에 있는 대상 상태를 나열하는 명령을 보여주고 있습니다.

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

A.2.4. dmsetup deps 명령

dmsetup deps *device* 명령은 지정된 장치의 맵핑 테이블에서 참조하는 장치 (major, minor) 쌍의 목록을 제공합니다. 장치 이름을 지정하지 않으셨을 경우, 출력 결과에는 현재 설정된 모든 Device Mapper 장치에 관한 정보가 나타나게 됩니다.

다음 예제에서는 현재 설정된 모든 맵핑 장치의 의존성을 나열하는 명령을 보여주고 있습니다.

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

다음 예제에서는 **lock_stress-grant--02.1722** 장치의 의존성만을 나열하는 명령을 보여주고 있습니다.

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

A.3. udev 장치 관리자에 대해 장치 매퍼(Device Mapper) 지원

udev 장치 관리자의 주요 역할은 **/dev** 디렉토리에 있는 노드의 동적 설정 방식을 제공하는 것입니다. 이러한 노드를 생성하여 사용자 공간에 있는 **udev** 규칙 응용 프로그램을 지시합니다. 이러한 규칙은 특정 장치의 추가, 제거, 변경의 결과로 커널에서 직접 전송된 **udev** 이벤트에서 실행됩니다. 이는 핫플러그 지원에 대한 편리한 중앙 메커니즘을 제공합니다.

실제 노드를 생성하는 것 외에, **udev** 장치 관리자는 자신의 이름으로된 심볼릭 링크를 생성할 수 있으며, 필요할 경우 사용자는 사용자 정의된 이름과 **/dev** 디렉토리에 디렉토리 구조를 선택할 수 있습니다.

각각의 **udev** 이벤트에는 이름, 하부 시스템, 장치 유형, 사용되는 주/부 번호, 이벤트 유형과 같은 실행되고 있는 장치에 대한 기본 정보가 들어있습니다. **udev** 규칙에서 액세스할 수 있는 **/sys** 디렉토리에 있는 모든 정보에 액세스할 수 있으며, 사용자는 이 정보를 기반으로 간단한 필터를 사용하여 정보에 따라 조건부로 규칙을 실행할 수 있습니다.

udev 장치 관리자는 중앙 관리 방식의 노드 권한 설정을 제공합니다. 사용자는 이벤트를 처리하는 동안 사용 가능한 정보로 지정한 장치의 권한을 정의하기 위해 사용자 정의된 규칙을 쉽게 추가할 수 있습니다.

udev 규칙에 직접 hook 프로그램을 추가할 수 도 있습니다. **udev** 장치 관리자는 이벤트를 처리하기 위해 필요한 프로세스를 제공하기 위해 이러한 프로그램을 호출할 수 있습니다. 또한 프로그램은 이러한 처리의 결과로 환경 변수를 내보낼 수 있습니다. 어떤 처리 결과든지 정보의 추가 소스로 규칙에 사용될 수 있습니다.

udev 라이브러리를 사용하는 소프트웨어는 사용 가능한 모든 정보와 함께 **udev** 이벤트를 받아 처리할 수 있으므로 처리 사항은 **udev** 데몬에 바인딩되지 않습니다.

A.3.1. 장치 맵퍼 (Device Mapper)로 udev 통합

RHEL 6에서 장치 맵퍼(Device Mapper)는 **udev** 통합에 대해 직접적인 지원을 제공합니다. 이는 장치 맵퍼를 LVM 장치를 포함하여 장치 맵퍼 장치와 연관된 모든 **udev** 처리와 동기화합니다. **udev** 데몬에 있는 규칙 어플리 케이션은 장치 변경의 소스인 (**dmsetup** 및 LVM 등) 프로그램과 병렁 처리 형태이기 때문에 동기화가 필요합니다. 이러한 지원 없이 이전 변경 이벤트의 결과로 **udev** 규칙에 의해 처리되는 아직 열려있는 상태의 장치를 제거하기 위한 문제가 사용자에게 자주 발생했었습니다. 특히 장치의 변경 간격이 매우 짧은 경우에 자주 발생했었습니다.

RHEL 6 릴리즈에서는 일반적으로 장치 맵퍼 (Device Mapper) 장치 및 LVM에 대해 공식적으로 지원되는 **udev** 규칙을 제공합니다. 표 A.1. “장치 맵퍼 (Device-Mapper) 장치에 대한 udev 규칙 ”에는 **/lib/udev/rules.d**에 설치된 이러한 규칙에 대해 요약되어 있습니다.

표 A.1. 장치 맵퍼 (Device-Mapper) 장치에 대한 udev 규칙

파일 이름	설명
10-dm.rules	기본적/일반적 장치 맵퍼 (Device Mapper) 규칙이 들어 있으며 **/dev/dm-*N*** 대상과 함께 **/dev/mapper**에 심볼릭 링크를 생성합니다. 여기서 N은 커널에 의해 장치에 동적으로 할당된 번호입니다. (**/dev/dm-*N***은 노드입니다) 알림: **/dev/dm-*N*** 노드는 장치에 액세스하기 위해 *절대로* 스크립트에서 사용해서는 안됩니다. 이는 N 번호가 동적으로 할당되어 장치를 활성화하는 순서에 의해 변경되기 때문입니다. 따라서, **/dev/mapper** 디렉토리에 있는 실제적인 이름을 사용해야 합니다. 이러한 레이아웃은 노드/심볼릭 링크가 생성되어야 방법에 대한 **udev** 요구 사항을 지원하는 것입니다.
11-dm-lvm.rules	LVM 장치에 적용할 규칙이 포함되어 있으며 볼륨 그룹의 논리 볼륨에 대해 심볼릭 링크를 생성합니다. 심볼릭 링크는 **/dev/dm-*N*** 대상과 함께 **/dev/*vgname*** 디렉토리에 생성됩니다. 알림: 장치 맵퍼 하부시스템의 향후 모든 규칙의 이름을 지정하기 위한 기준과 알관성을 유지하기 위해, udev 규칙은 **11-dm-*subsystem_name*.rules** 형식을 따라야 합니다. **udev** 규칙을 제공하는 **libdevmapper** 사용자는 이러한 기준을 따라야 합니다.
13-dm-disk.rules	일반적으로 모든 장치 맵퍼 (Device Mapper) 장치에 대해 적용되는 규칙이 포함되어 있으며 **/dev/disk/by-id**, **/dev/disk/by-uuid**, **/dev/disk/by-uuid** 디렉토리에 심볼릭 링크를 생성합니다.
95-dm-notify.rules	**libdevmapper** (LVM 및 **dmsetup**과 같이)를 사용하여 대기중 프로세스를 통지하기 위한 규칙이 포함되어 있습니다. 이전의 모든 규칙이 적용된 후 통지가 완료되며, **udev** 프로세스가 완료되었는지를 확인합니다. 그 후 통지된 프로세스는 다시 시작됩니다.

12-dm-permissions.rules 파일을 사용하여 사용자 설정 권한 규칙을 추가할 수 있습니다. 이 파일은 **/lib/udev/rules** 디렉토리에 설치되지 *않고* **/usr/share/doc/device-mapper-*version*** 디렉토리에서 찾을 수 있습니다. **12-dm-permissions.rules** 파일은 예에서 처럼 일부 일치되는 규칙에 기반한 권한 설정 방법에 대한 설명이 들어있는 템플릿으로 이 파일에는 일반적인 상황의 예가 들어 있습니다. 이 파일을 편집하여 **/etc/udev/rules.d** 디렉토리에 수동으로 배치하면 여기에 업데이트가 위치하게 되어 설정이 그대로 유지됩니다.

이러한 규칙은 이벤트를 처리하는 동안 다른 규칙에 의해 사용될 수 있는 모든 기본적인 변수를 설정합니다.

다음과 같은 매개 변수는 10-dm.rules에 설정되어 있습니다:

- **DM_NAME**: 장치 매퍼 장치 이름
- **DM_UUID**: 장치 매퍼 장치 UUID
- **DM_SUSPENDED**: 장치 맵퍼 (Device Mapper) 장치의 일시 정지 상태
- **DM_UDEV_RULES_VSN**: **udev** 규칙 버전 (이는 이전에 언급된 매개 변수가 공식적인 장치 맵퍼 규칙에 의해 직접 설정되어 있는지를 다른 모든 규칙이 확인하는 것입니다)

다음의 매개 변수는 **11-dm-lvm.rules**에 설정되어 있습니다:

- **DM_LV_NAME**: 논리 볼륨 이름
- **DM_VG_NAME**: 볼륨 그룹 이름
- **DM_LV_LAYER**: LVM 레이어 이름

이러한 모든 매개 변수는 **12-dm-permissions.rules** 파일에서 설명하고 있듯이 **12-dm-permissions.rules** 파일에서 사용하여 특정 장치 매퍼 (Device Mapper) 장치에 대한 권한을 정의할 수 있습니다.

A.3.2. udev를 지원하는 명령 및 인터페이스

표 A.2. “udev를 지원하기 위한 dmsetup 명령 ”에서는 **udev** 통합을 지원하는 **dmsetup** 명령에 대해 요약하고 있습니다.

표 A.2. udev를 지원하기 위한 dmsetup 명령

명령	설명
dmsetup udevcomplete	udev가 규칙 처리를 완료하고 대기 중인 프로세스의 잠금 해제를 알리는데 사용됩니다 (**95-dm-notify.rules**에 있는 **udev** 규칙에서 호출됩니다)
dmsetup udevcomplete_all	모든 대기 프로세스를 수동으로 잠금 해제하기 위한 디버깅 목적으로 사용됩니다
dmsetup udevcookies	기존의 모든 쿠키 (시스템 전역 세마포어)를 보기 위한 디버깅 목적으로 사용됩니다
dmsetup udevcreatecookie	수동으로 쿠키 (세마포어)를 생성하는데 사용됩니다. 이는 하나의 동기화 리소스 하에서 많은 프로세스를 실행하는데 유용합니다.
dmsetup udevreleasecookie	동기화 쿠키하에 있는 모든 프로세스와 관련된 모든 **udev** 처리를 위해 기다리는데 사용됩니다.

udev 통합을 지원하는 **dmsetup** 옵션은 다음과 같습니다.

--udevcookie

udev 트랜잭션에 추가하려는 모든 dmsetup 프로세스를 대상으로 정의해야 합니다. **udevcreatecookie** 및 **udevreleasecookie**과 함께 사용됩니다.

```
COOKIE=$(dmsetup udevcreatecookie)
  dmsetup command --udevcookie $COOKIE ....
  dmsetup command --udevcookie $COOKIE ....
  ....
  dmsetup command --udevcookie $COOKIE ....
dmsetup udevreleasecookie --udevcookie $COOKIE
```

--udevcookie 옵션 사용 이외에 매개 변수를 프로세스 환경으로 내보내기할 수 있습니다:

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

--noudevrules

udev 규칙 비활성화: 노드/심볼릭 링크는 **libdevmapper** 자체에 의해 생성됩니다 (이전 방식). 이는 **udev**가 올바르게 작동하지 않을 경우 디버깅 목적을 위한 옵션입니다.

--noudevsync

udev 동기화를 비활성화합니다. 또한 이는 디버깅을 목적으로 하는 것입니다.

dmsetup 및 이 명령의 옵션에 대한 자세한 내용은 **dmsetup**(8) man 페이지를 참조하십시오.

LVM 명령은 **udev** 통합을 지원하는 다음과 같은 옵션을 지원합니다:

- **--noudevrules**: **dmsetup** 명령에 대해 **udev** 규칙을 비활성화합니다.
- **--noudevsync**: **dmsetup** 명령에 대해 **udev** 동기화를 비활성화합니다.

lvm.conf 파일에는 **udev** 통합을 지원하는 다음과 같은 옵션이 포함되어 있습니다:

- **udev_rules**: 모든 LVM2 명령 전역에 걸쳐 **udev_rules**을 활성화/비활성화
- **udev_sync**: 모든 LVM 명령에 대해 **udev** 동기화를 활성화/비활성화

lvm.conf 파일 옵션에 대한 보다 자세한 내용은 **lvm.conf** 파일에 있는 인라인 주석을 참조하십시오.

LVM 설정 파일

LVM은 여러 설정 파일을 지원합니다. 시스템을 시작하면, **/etc/lvm**에 기본값으로 설정된 **LVM_SYSTEM_DIR** 환경 변수에 의해 지정된 디렉토리에서 **lvm.conf** 설정 파일을 읽어 오게 됩니다.

lvm.conf 파일은 읽어올 추가 설정 파일을 지정할 수 있습니다. 나중에 설정된 파일은 이전에 설정된 파일을 덮어쓰게 됩니다. 모든 설정 파일을 읽어온 후, 현재 사용되고 있는 설정 보기를 하려면, **lvm dumpconfig** 명령을 실행합니다.

추가 설정 파일을 읽어오는 방법에 대한 자세한 내용은 C.2절. “호스트 태그”에서 참조하시기 바랍니다.

B.1. LVM 설정 파일

다음 파일은 LVM 설정에 사용됩니다:

/etc/lvm/lvm.conf

도구로 읽혀진 중앙 설정 파일

etc/lvm/lvm_*hosttag*.conf

각각의 호스트 태그에 대해 추가 설정 파일이 있을 경우 이를 읽어 옵니다: **lvm_*hosttag*.conf**. 이 파일이 새 태크를 지정할 경우, 추가 설정 파일은 읽어 올 tiles 목록에 추가됩니다. 호스트 태그에 관한 내용은 C.2절. “호스트 태그”에서 참조하시기 바랍니다.

LVM 설정 파일에 더하여, LVM을 실행하고 있는 시스템에는 다음과 같은 파일로 LVM 시스템 설정에 영향을 미칩니다:

/etc/lvm/.cache

장치 이름 필터 캐시 파일 (설정 가능).

/etc/lvm/backup/

자동 볼륨 그룹 메타 데이터 백업 용 디렉토리 (설정 가능).

/etc/lvm/archive/

자동 볼륨 그룹 메타 데이터 아카이브 용 디렉토리 (디렉토리 경로 및 아카이브 히스토리 정보 관련 설정 가능).

/var/lock/lvm/

단일 호스트 설정에서, 병렬 프로그램 도구 실행으로 메타 데이터를 손상시키지 않게 하기 위한 잠금 파일, 클러스터 전반에 걸친 DLM이 사용됩니다.

B.2. lvm.conf 설정 파일의 예

다음은 **lvm.conf** 설정 파일의 예입니다. 사용하시는 설정 파일은 이것과 조금 다를 수 있습니다.

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


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

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

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

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

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

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

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

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

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


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

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

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

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

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

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

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

    # Advanced settings.

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

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

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

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

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

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

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

```
    data_alignment = 0

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

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

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

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

# This section allows you to configure the way in which LVM selects
# free space for its Logical Volumes.
#allocation {
#    When searching for free space to extend an LV, the "cling"
#    allocation policy will choose space on the same PVs as the last
#    segment of the existing LV.  If there is insufficient space and a
#    list of tags is defined here, it will check whether any of them are
#    attached to the PVs concerned and then seek to match those PV tags
#    between existing extents and new extents.
#    Use the special tag "@*" as a wildcard to match any PV tag.
#
#    Example: LVs are mirrored between two sites within a single VG.
#    PVs are tagged with either @site1 or @site2 to indicate where
#    they are situated.
#
#    cling_tag_list = [ "@site1", "@site2" ]
#    cling_tag_list = [ "@*" ]
#}

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

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

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

    # Should we log error and debug messages to a file?
```

```
    # By default there is no log file.
    #file = "/var/log/lvm2.log"

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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


# Miscellaneous global LVM2 settings
global {

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    # If set to 1, no operations that change on-disk metadata will be permitted.
    # Additionally, read-only commands that encounter metadata in need of repair
    # will still be allowed to proceed exactly as if the repair had been
    # performed (except for the unchanged vg_seqno).
```

```
    # Inappropriate use could mess up your system, so seek advice first!
    metadata_read_only = 0
}

activation {
    # Set to 0 to disable udev synchronisation (if compiled into the binaries).
    # Processes will not wait for notification from udev.
    # They will continue irrespective of any possible udev processing
    # in the background.  You should only use this if udev is not running
    # or has rules that ignore the devices LVM2 creates.
    # The command line argument --nodevsync takes precedence over this setting.
    # If set to 1 when udev is not running, and there are LVM2 processes
    # waiting for udev, run 'dmsetup udevcomplete_all' manually to wake them up.
    udev_sync = 1

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

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

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

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

    # Nice value used while devices suspended
    process_priority = -18

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

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

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

    # 'mirror_image_fault_policy' and 'mirror_log_fault_policy' define
    # how a device failure affecting a mirror is handled.
    # A mirror is composed of mirror images (copies) and a log.
    # A disk log ensures that a mirror does not need to be re-synced
```

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

    mirror_log_fault_policy = "allocate"
    mirror_image_fault_policy = "remove"

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

    snapshot_autoextend_threshold = 100
    snapshot_autoextend_percent = 20

    # While activating devices, I/O to devices being (re)configured is
    # suspended, and as a precaution against deadlocks, LVM2 needs to pin
    # any memory it is using so it is not paged out.  Groups of pages that
    # are known not to be accessed during activation need not be pinned
    # into memory.  Each string listed in this setting is compared against
```

```
    # each line in /proc/self/maps, and the pages corresponding to any
    # lines that match are not pinned.  On some systems locale-archive was
    # found to make up over 80% of the memory used by the process.
    # mlock_filter = [ "locale/locale-archive", "gconv/gconv-modules.cache" ]

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

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

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


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

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

    # pvmetadatacopies = 1

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

    # vgmetadatacopies = 0

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

    # pvmetadatasize = 255

    # List of directories holding live copies of text format metadata.
    # These directories must not be on logical volumes!
    # It's possible to use LVM2 with a couple of directories here,
    # preferably on different (non-LV) filesystems, and with no other
```

```
    # on-disk metadata (pvmetadatacopies = 0). Or this can be in
    # addition to on-disk metadata areas.
    # The feature was originally added to simplify testing and is not
    # supported under low memory situations - the machine could lock up.
    #
    # Never edit any files in these directories by hand unless you
    # you are absolutely sure you know what you are doing! Use
    # the supplied toolset to make changes (e.g. vgcfgrestore).

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

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

    mirror_library = "libdevmapper-event-lvm2mirror.so"

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

    snapshot_library = "libdevmapper-event-lvm2snapshot.so"

    # Full path of the dmeventd binary.
    #
    # executable = "/sbin/dmeventd"
}
```

LVM 객체 태그

LVM 태그는 같은 유형의 LVM2 객체를 모으는데 사용될 수 있는 문자입니다. 태그는 물리 볼륨, 볼륨 그룹, 논리 볼륨과 같은 객체에 부착될 수 있습니다. 클러스터 설정에서 호스트에 태그를 부착할 수 도 있습니다. 스냅샷은 태그될 수 없습니다.

PV, VG, LV 인수에 있는 명령행에 태그할 수 있습니다. 태그는 @를 앞자리에 두어 명확하게 합니다. 각각의 태그는 명령행에 있는 태그의 위치에 따라 유형을 알 수 있는 태그를 태그가 있는 모든 객체로 대체하여 확장됩니다.

Red Hat Enterprise Linux 6.1 릴리스에서 LVM 태그는 최대 1024자의 문자열입니다 (이전 릴리스에서 최대 길이는 128 자였습니다). LVM 태그는 하이픈으로 시작할 수 없습니다.

유효한 태그는 한정된 범위의 문자로만 구성됩니다. Red Hat Enterprise Linux 6.0 릴리즈에서 사용할 수 있는 문자는 [A-Za-z0-9_+.-]입니다. Red Hat Enterprise Linux 6.1 릴리즈에서는 사용 가능한 문자 목록이 확대되고 태그에는 "/", "=", "!", ":", "#", and "&"를 사용할 수 있습니다.

볼륨 그룹에 있는 객체만이 태그될 수 있습니다. 물리 볼륨이 볼륨 그룹에서 삭제될 경우 태그도 삭제됩니다; 이는 볼륨 그룹 메타데이터의 부분으로 태그가 저장되므로 물리 볼륨이 삭제될 경우 태그가 삭제되는 것입니다. 스냅샷은 태그될 수 없습니다.

다음의 명령으로 **database** 태그로된 모든 논리 볼륨을 나열합니다.

```
lvs @database
```

C.1. 객체 태그 추가 및 삭제

물리 볼륨에서 태그를 추가 또는 삭제하려면 **pvchange** 명령의 **--addtag** 또는 **--deltag** 옵션을 사용합니다.

볼륨 그룹에서 태그를 추가 또는 삭제하려면 **vgchange** 또는 **vgcreate** 명령의 **--addtag** 또는 **--deltag** 옵션을 사용합니다.

논리 볼륨에서 태그를 추가 또는 삭제하려면 **lvchange** 또는 **lvcreate** 명령의 **--addtag** 또는 **--deltag** 옵션을 사용합니다.

Red Hat Enterprise Linux 6.1 릴리즈에서는 **pvchange**, **vgchange**, **lvchange**의 단일 명령으로 **--addtag** 및 **--deltag** 인수를 지정할 수 있습니다. 예를 들어, 다음 명령에서는 태그 **T9**와 **T10**을 삭제하고 태그 **T13**과 **T14**를 볼륨 그룹 **grant**에 추가합니다.

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

C.2. 호스트 태그

클러스터 설정에서, 설정 파일에 있는 호스트 태그를 지정할 수 있습니다. **tags** 섹션에서 **hosttags = 1**을 설정했을 경우, 호스트 태그는 컴퓨터의 호스트명을 사용하여 자동으로 지정됩니다. 이는 컴퓨터에서 복사될 수 있는 일반적인 설정 파일을 사용하게 하지만, 호스트명에 따라 컴퓨터에서 다르게 작동합니다.

설정 파일에 관한 내용은 부록 B. *LVM 설정 파일*에서 참조하시기 바랍니다.

각각의 호스트 태그의 경우, 추가 설정 파일이 있을 경우 이를 읽어 옵니다: lvm_***hosttag***.conf. 이러한 파일이 새로운 태그를 지정한 경우, 읽어올 추가 설정 파일이 파일 목록에 첨가됩니다.

예를 들어, 다음과 같이 설정 파일에 있는 항목은 항상 **tag1**로 지정되고, 호스트명이 **host1**으로 지정되어

있을 경우 **tag2**로 지정됩니다.

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

C.3. 태그로 활성화 관리

해당 호스트에서 활성화되어야 하는 특정 논리 볼륨에 대한 설정 파일을 지정할 수 있습니다. 예를 들어, 다음과 같은 항목이 활성화 요청 (예: **vgchange -ay**)에 대해 필터로 작동하면 해당 호스트 상의 메타 데이터에 있는 **database** 태그로 된 **vg1/lvol0**, 모든 논리 볼륨, 볼륨 그룹만이 활성화됩니다.

```
activation { volume_list = ["vg1/lvol0", "@database" ] }
```

메타 데이터 태그가 해당 컴퓨터의 호스트 태그와 일치할 경우에만 일치하게 되는 특정 일치 "@*"가 있습니다.

다른 예로 클러스터에 있는 모든 컴퓨터의 설정 파일에 다음과 같은 항목을 갖고 있다고 가정합시다:

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

호스트에서 **vg1/lvol2** 만을 활성화하려면 다음을 실행합니다:

1. 클러스터에 있는 아무 호스트에서 **lvchange --addtag @db2 vg1/lvol2** 명령을 실행합니다.
2. **lvchange -ay vg1/lvol2** 명령을 실행합니다.

이는 볼륨 그룹 메타 데이터 안에 있는 호스트명을 저장하여야 해결될 수 있습니다.

LVM 볼륨 그룹 메타데이터

볼륨 그룹의 설정 정보를 메타데이터라고 부릅니다. 기본값으로, 메타데이터의 동일 복사본은 볼륨 그룹 안의 모든 물리 볼륨에 있는 모든 메타데이터 영역에서 관리됩니다. LVM 볼륨 그룹 메타데이터는 ASCII로 저장됩니다.

볼륨 그룹에 여러 개의 물리 볼륨이 있을 경우, 여러 개의 메타 데이터 이중 복사본을 갖고 있는 것은 비효율적입니다. **pvcreate** 명령의 **--metadatacopies 0** 옵션을 사용하여 메타 데이터 복사본 없이 물리 볼륨을 생성할 수 있습니다. 메타 데이터 복사본 수를 선택하면 물리 볼륨이 포함되며, 나중에 이를 변경할 수 없게 됩니다. 0 개의 복사본을 선택하면 설정 변경에서 보다 빠른 업데이트를 할 수 있습니다. (파일 시스템에 볼륨 그룹 메타 데이터를 저장하게 하는 고급 설정을 사용하지 않는 한) 항상 모든 볼륨 그룹에는 메타 데이터 영역과 함께 최소 하나의 물리 볼륨이 들어 있어야 함에 유의합니다. 차후에 볼륨 그룹을 나누고자 할 경우, 모든 볼륨 그룹에는 최소 하나의 메타 데이터 복사본이 있어야 합니다.

주요 메타 데이터는 ASCII에 저장됩니다. 메타 데이터 영역은 순환 버퍼입니다. 새 메타 데이터는 이전 메타 데이터에 추가되며 새 메타 데이터의 시작 포인터가 업데이트됩니다.

pvcreate 명령의 **--metadatasize**. 옵션을 사용하여 메타 데이터 영역 크기를 지정할 수 있습니다. 여러 논리 볼륨 또는 물리 볼륨이 있는 볼륨 그룹의 경우 기본값 크기는 너무 작게 됩니다.

D.1. 물리 볼륨 레이블

기본값으로 **pvcreate** 명령은 두 번째 512 바이트 섹터에서 물리 볼륨 레이블을 둡니다. 다른 방법으로 물리 볼륨 레이블을 찾는 LVM 도구가 처음 4개의 섹터를 확인하므로 이러한 레이블을 처음 네 개의 섹터 중 아무곳에 둘 수 있습니다. 물리 볼륨 레이블은 **LABELONE** 문자열로 시작합니다.

물리 볼륨 레이블에는 다음과 같은 내용이 들어있습니다:

- 물리 볼륨 UUID
- 바이트 단위로 된 블록 장치 크기
- NULL로 종료되는 데이터 영역 위치 목록
- NULL로 종료되는 메타 데이터 영역 위치 목록

메타데이터 위치는 오프셋과 바이트 단위 크기로 저장됩니다. 레이블에 약 15 개의 여유 영역이 있지만, LVM 도구는 단일 데이터 영역 및 최대 두 개의 메타데이터 영역과 같이 3 개만을 사용합니다.

D.2. 메타데이터 컨텐츠

볼륨 그룹 메타데이터 컨텐츠:

- 생성 시기 및 방법에 관한 정보
- 볼륨 그룹에 관한 정보:

볼륨 그룹 정보에는 다음과 같은 사항이 포함되어 있습니다:

- 이름 및 고유 id
- 메타데이터가 업데이트될 때마다 증가하는 버전 번호
- 기타 속성: 읽기/쓰기? 크기 조정 가능?
- 물리 볼륨 및 논리 볼륨의 수에 대한 관리 제한
- 익스텐트 크기 (512 바이트로 지정된 섹터 단위)
- 볼륨 그룹을 만드는 물리 볼륨의 비순서 목록 각각은 다음과 같이 구성되어 있습니다 :
  - 블록 장치를 지정하는데 사용되는 UUID

- 물리 볼륨의 할당 여부와 같은 속성
- (섹터에서) 물리 볼륨 안에 있는 첫 번째 익스텐트 시작에서 오프셋
- 익스텐트 수

- 논리 볼륨의 비순서 목록 각각은 다음과 같이 구성되어 있습니다
  - 논리 볼륨 세그먼트의 순서 목록. 각각의 세그먼트에 대해 메타데이터에는 물리 볼륨 세그먼트나 논리 볼륨 세그먼트의 순서 목록에 적용되는 맵핑이 들어 있습니다.

D.3. 메타데이터의 예

다음은 **myvg**라는 볼륨 그룹의 LVM 볼륨 그룹 메타데이터 예를 보여주고 있습니다.

```
# 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
                                ]
                        }
                }
        }
}
```

개정 내역

고침 1-12.400 **2013-10-31** **Rüdiger Landmann**

Rebuild with publican 4.0.0

고침 1-12 **2012-07-18** **Anthony Towns**

Rebuild for Publican 3.0

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

Red Hat Enterprise Linux 6.1의 초기 릴리즈

문제 해결: #694619
논리 볼륨을 확장하기 위한 새로운 **cling** 할당 정책을 문서화.

문제 해결: #682649
클러스터 볼륨에서 여러 미러 생성 명령의 연속 실행에 대한 경고 추가

문제 해결: #674100
dmsetup ls --tree 명령에 대한 예시 출력 결과 추가

문제 해결: #694607
단일 명령행에서 --addtag 및 --deltag 인수 사용 지원을 문서화

문제 해결: #694604
태그에서 확장 문자 목록 지원을 문서화

문제 해결: #694611
스트라이프 미러에 대한 지원을 문서화

문제 해결: #694616
미러 볼륨의 스냅샷 지원을 문서화

문제 해결: #694618
독단적으로 활성화된 클러스터 볼륨의 스냅샷 지원을 문서화

문제 해결: #682648
미러 leg가 다시 할당되면 미러 로그를 이동할 수 있는 것을 문서화

문제 해결: #661530
현재 기능을 문서화한 **cluster.conf** 예시 업데이트

문제 해결: #642400
최하위의 클러스터 ID로 클러스터 노드가 유지되는 클러스터 로그 관리에 관한 알림을 추가

문제 해결: #663462
Xen 가상 머신 모니터에 대한 오래된 참조 내용을 삭제

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

Red Hat Enterprise Linux 6의 초기 릴리즈

색인

Symbols