gentoo linux™ **(/)** **Wiki**

LVM

From Gentoo Wiki
Jump to:navigation Jump to:search

Not to be confused with LLVM (/wiki/LLVM).

LVM (**L**ogical **V**olume **M**anager) allows administrators to create meta devices that provide an abstraction layer between a file system and the physical storage that is used underneath. The meta devices (on which file systems are placed) are *logical volumes*, which use storage from storage pools called *volume groups*. A volume group is provisioned with one or more *physical volumes* which are the true devices on which the data is stored.

Physical volumes can be partitions, whole hard drives grouped as JBOD (**J**ust a **B**unch **O**f **D**isks), RAID systems, iSCSI, Fibre Channel, eSATA etc.

Resources
🏠 Home (https://sourceware.org/lvm2/)
🖴 Package information (https://packages.gentoo.org/packages/sys-fs/lvm2)
W Wikipedia (https://en.wikipedia.org/wiki/Logical%20Volume%20Manager%20%28Linux%29)
⑂ GitWeb (https://sourceware.org/git/?p=lvm2.git)

Contents

Installation

LVM is handled by both kernel-level drivers and user space applications to manage the LVM configuration.

Kernel

Activate the following kernel options:

| KERNEL | **linux-4.9 Enabling LVM** |

```
Device Drivers  --->
   Multiple devices driver support (RAID and LVM)  --->
      <*> Device mapper support
         <*> Crypt target support
         <*> Snapshot target
         <*> Mirror target
         <*> Multipath target
            <*> I/O Path Selector based on the number of in-flight I/Os
            <*> I/O Path Selector based on the service time
```

> **⊓ Note**
> Not everything needs to be enabled; some of the options are only needed for LVM2 Snapshots and LVM2 Thin
> Snapshots (/wiki/LVM#LVM2_snapshots_and_thin_snapshots), LVM2 Mirrors (/wiki/LVM#Mirrored_volumes),
> LVM2 RAID 0/Stripeset (/wiki/LVM#Striping_.28RAID0.29) and encryption.

USE flags

USE flags for sys-fs/lvm2 (https://packages.gentoo.org/packages/sys-fs/lvm2) ⊠	User-land ...
lvm (https://packages.gentoo.org/useflags/lvm)	Build all of LVM2 including daemons and tools like lvchange, not just the device-mapper library (for other packages to use). If your system uses LVM2 for managing disks and partitions, enable this flag.
lvm2create-initrd (https://packages.gentoo.org/useflags/lvm2create-initrd)	Install lvm2create_initrd script and pull in sys-apps/makedev for the /sbin/MAKEDEV command
readline (https://packages.gentoo.org/useflags/readline)	Enable support for libreadline, a GNU line-editing library that almost everyone wants
sanlock (https://packages.gentoo.org/useflags/sanlock)	Enable lvmlockd with support for sanlock
More information about USE flags (/wiki/Handbook:AMD64/Working/USE)	Data provided by the Gentoo Package Database (https://packages.gentoo.org) · Last update: 2022-12-20 20:38

Emerge

After reviewing the USE flags, ask Portage to install the sys-fs/lvm2 (https://packages.gentoo.org/packages/sys-fs/lvm2) package:

```
root # emerge --ask sys-fs/lvm2
```

Configuration

Configuring LVM is done on several levels:

1. LV, PV and VG management through the management utilities;
2. LVM subsystem fine-tuning through the configuration file;
3. Service management at the distribution level;
4. Setup through an initial ram file system (initramfs).

Management of the logical and physical volumes as well as the volume groups is handled through the Usage chapter.

LVM configuration file

LVM has an extensive configuration file at `/etc/lvm/lvm.conf`. Most users will not need to modify settings in this file in order to start using LVM.

Service management

Gentoo provides the LVM service to automatically detect and activate the volume groups and logical volumes.

The service can be managed through the init system.

openrc

To start LVM manually:

```
root # /etc/init.d/lvm start
```

To start LVM at boot time:

```
root # rc-update add lvm boot
```

systemd

To start lvm manually:

```
root # systemctl start lvm2-monitor.service
```
To start LVM at boot time:

```
root # systemctl enable lvm2-monitor.service
```

Using LVM in an initramfs

Most bootloaders (/wiki/Bootloader) cannot boot from LVM directly - neither GRUB legacy nor LILO can. GRUB2 can boot from an LVM linear logical volume, mirrored logical volume and possibly some kinds of RAID logical volumes. No bootloader currently supports thin logical volumes.

For that reason, it is recommended to use a non-LVM /boot partition and mount the LVM root from an initramfs (/wiki/Initramfs). An initramfs can be generated automatically through genkernel (/wiki/Genkernel), dracut (/wiki/Dracut) or manually (https://forums.gentoo.org/viewtopic-t-1068042-start-4.html) as Custom Initramfs (/wiki/Custom_Initramfs):

- **genkernel** can boot from all types except thin volumes (as it neither builds nor copies the sys-block/thin-provisioning-tools (https://packages.gentoo.org/packages/sys-block/thin-provisioning-tools) binaries from the build host) and maybe RAID10 (RAID10 support requires LVM2 2.02.98, but genkernel builds 2.02.89, however if static binaries are available it can copy those);
- **dracut** should boot all types, but only includes thin support in the initramfs if the host being run on has a thin root.

Genkernel

Emerge sys-kernel/genkernel (https://packages.gentoo.org/packages/sys-kernel/genkernel). The static USE flag may also be enabled on the package sys-fs/lvm2 (https://packages.gentoo.org/packages/sys-fs/lvm2) so that genkernel will use the system binaries (otherwise it will build its own private copy). The following example will build only an initramfs (not an entire kernel) and enable support for LVM.

```
root # genkernel --lvm initramfs
```
The genkernel manpage outlines other options depending on system requirements.

The initrd will require parameters to tell it how to start LVM, and they are supplied the same way as other kernel parameters. For example:

FILE   `/etc/default/grub`   **Adding dolvm as a kernel boot parameter**

```
GRUB_CMDLINE_LINUX="dolvm"
```

Dracut

The sys-kernel/dracut (https://packages.gentoo.org/packages/sys-kernel/dracut) package was ported from the RedHat project and serves a similar tool for generating an initramfs. For details, please refer to Dracut (/wiki/Dracut). Generally, the following command will generate a usable default initramfs.

```
root # dracut -a lvm
```
The initrd will require parameters to tell it how to start LVM, and they are supplied the same way as other kernel parameters. For example:

FILE   `/etc/default/grub`   **Adding LVM support to the kernel boot parameters**

```
GRUB_CMDLINE_LINUX="rd.lvm.vg=vol00"
```

For a comprehensive list of LVM options within **dracut** please see the section in the Dracut Manual (https://www.ke rnel.org/pub/linux/utils/boot/dracut/dracut.html#_lvm).

Usage

LVM organizes storage in three different levels as follows:

- hard drives, partitions, RAID systems or other means of storage are initialized as physical volumes (PVs)
- Physical Volumes (PV) are grouped together in Volume Groups (VG)
- Logical Volumes (LV) are managed in Volume Groups (VG)

PV (Physical Volume)

Physical Volumes are the actual hardware or storage system LVM builds up upon.

Partitioning

> **Note**
> Using separate partitions for provisioning storage to volume groups is only needed if it is not desired to use the entire disk for a single LVM volume group. If the entire disk can be used, then skip this and initialize the entire hard drive as a physical volume.

The partition type for *LVM* is *8e* (Linux LVM).

For instance, to set the type through **fdisk** for a partition on /dev/sda:

```
root # fdisk /dev/sda
```

In **fdisk**, create partitions using the `n` key and then change the partition type with the `t` key to *8e*.

Create PV

Physical volumes can be created / initialized with the **pvcreate** command.

For instance, the following command creates a physical volume on the first primary partition of /dev/sda and /dev/sdb:

```
root # pvcreate /dev/sd[ab]1
```

List PV

With the **pvdisplay** command, an overview of all active physical volumes on the system can be obtained.

```
root # pvdisplay
  --- Physical volume ---
  PV Name               /dev/sda1
  VG Name               volgrp
  PV Size               160.01 GiB / not usable 2.31 MiB
  Allocatable           yes
  PE Size               4.00 MiB
  Total PE              40962
  Free PE               4098
  Allocated PE          36864
  PV UUID               3WHAz3-dh4r-RJ0E-5o6T-9Dbs-4xLe-inVwcV

  --- Physical volume ---
  PV Name               /dev/sdb1
  VG Name               volgrp
  PV Size               160.01 GiB / not usable 2.31 MiB
  Allocatable           yes
  PE Size               4.00 MiB
  Total PE              40962
  Free PE               40962
  Allocated PE          0
  PV UUID               b031x0-6rej-BcBu-bE2C-eCXG-jObu-0Boo0x
```

If more physical volumes should be displayed, then **pvscan** can detect inactive physical volumes and activate those.

root # pvscan

```
  PV /dev/sda1   VG volgrp        lvm2 [160.01 GiB / 16.01 GiB free]
  PV /dev/sdb1   VG volgrp        lvm2 [160.01 GiB / 160.01 GiB free]
  Total: 2 [320.02 GB] / in use: 2 [320.02 GiB] / in no VG: 0 [0]
```

Remove PV

LVM automatically distributes the data onto all available physical volumes (unless told otherwise) but in a linear approach. If a requested logical volume (within a volume group) is smaller than the amount of free space on a single physical volume, then all space for the logical volume is claimed on that (single) physical volume in a contiguous manner. This is done for performance reasons.

If a physical volume needs to be removed from a volume group, the data first needs to be moved away from the physical volume. With the **pvmove** command, all data on a physical volume is moved to other physical volumes within the same volume group.

root # pvmove -v /dev/sda1

Such an operation can take a while depending on the amount of data that needs to be moved. Once finished, there should be no data left on the device. Verify with **pvdisplay** that the physical volume is no longer used by any logical volume.

The next step is to remove the physical volume from the volume group using **vgreduce** after which the device can be "deselected" as a physical volume using **pvremove**:

root # vgreduce vg0 /dev/sda1 && pvremove /dev/sda1

VG (Volume Group)

A volume group (VG) groups a number of physical volumes and show up as /dev/VG_NAME in the device file system. The name of a volume group is chosen by the administrator.

Create VG

The following command creates a volume group called *vg0* with two physical volumes assigned to it: /dev/sda1 and /dev/sdb1.

root # vgcreate vg0 /dev/sd[ab]1

List VG

To list all active volume groups, use the **vgdisplay** command:

root # vgdisplay

```
  --- Volume group ---
  VG Name               vg0
  System ID
  Format                lvm2
  Metadata Areas        1
  Metadata Sequence No  8
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                6
  Open LV               6
  Max PV                0
  Cur PV                1
  Act PV                1
  VG Size               320.02 GiB
  PE Size               4.00 MiB
  Total PE              81924
  Alloc PE / Size       36864 / 144.00 GiB
  Free  PE / Size       45056 /176.01 GiB
  VG UUID               mFPXj3-DdPi-7YJ5-9WKy-KA5Y-Vd4S-Lycxq3
```

If volume groups are missing, use the **vgscan** command to locate volume groups:

```
root # vgscan
```

```
  Reading all physical volumes.  This may take a while...
  Found volume group "vg0" using metadata type lvm2
```

Extend VG

Volume groups group physical volumes, allowing administrators to use a pool of storage resources to allocate to file systems. When a volume group does not hold enough storage resources, it is necessary to extend the volume group with additional physical volumes.

The next example extends the volume group *vg0* with a physical volume at /dev/sdc1:

```
root # vgextend vg0 /dev/sdc1
```
Remember that the physical volume first needs to be initialized as such!

Reduce VG

If physical volumes need to be removed from the volume group, all data still in use on the physical volume needs to be moved to other physical volumes in the volume group. As seen before, this is handled through the **pvmove** command, after which the physical volume can be removed from the volume group using **vgreduce**:

```
root # pvmove -v /dev/sdc1
```
```
root # vgreduce vg0 /dev/sdc1
```

Remove VG

If a volume group is no longer necessary (or, in other words, the storage pool that it represents is no longer used and the physical volumes in it need to be freed for other purposes) then the volume group can be removed with **vgremove**. This only works if no logical volume is defined for the volume group, and all but one physical volume have already been removed from the pool.

```
root # vgremove vg0
```

LV (Logical Volume)

Logical volumes are the final meta devices which are made available to the system, usually to create file systems on. They are created and managed in volume groups and show up as /dev/VG_NAME/LV_NAME. Like with volume groups, the name used for a logical volume is decided by the administrator.

Create LV

To create a logical volume, the **lvcreate** command is used. The parameters to the command consist out of the requested size for the logical volume (which cannot be larger than the amount of free space in the volume group), the volume group from which the space is to be claimed and the name of the logical volume to be created.

In the next example, a logical volume named *lvol1* is created from the volume group named *vg0* and with a size of 150MB:

```
root # lvcreate -L 150M -n lvol1 vg0
```
It is possible to tell **lvcreate** to use all free space inside a volume group. This is done through the -l option which selects the amount of *extents* rather than a (human readable) size. Logical volumes are split into *logical extents* which are data chunks inside a volume group. All extents in a volume group have the same size. With the -l option **lvcreate** can be asked to allocate all free extents:

```
root # lvcreate -l 100%FREE -n lvol1 vg0
```
Next to *FREE* the *VG* key can be used to denote the entire size of a volume group.

List LV

To list all logical volumes, use the **lvdisplay** command:

```
root # lvdisplay
```
If logical volumes are missing, then the **lvscan** command can be used to scan for logical volumes on all available volume groups.

```
root # lvscan
```

Extend LV

When a logical volume needs to be expanded, then the **lvextend** command can be used to grow the allocated space for the logical volume.

For instance, to extend the logical volume *lvol1* to a total of 500 MB:

```
root # lvextend -L500M /dev/vg0/lvol1
```
It is also possible to use the size to be added rather than the total size:

```
root # lvextend -L+350MB /dev/vg0/lvol1
```
An extended volume group does not immediately provide the additional storage to the end users. For that, the file system on top of the volume group needs to be increased in size as well. Not all file systems allow online resizing, so check the documentation for the file system in question for more information.

For instance, to resize an ext4 file system to become 500MB in size:

```
root # resize2fs /dev/vg0/lvol1 500M
```
For certain file systems, **lvresize** extends the logical volume and the file system in one step. For instance, to extend the logical volume *lvol1* and resize the ext4 file system:

```
root # lvresize --resizefs --size +350GB /dev/vg0/lvol1
```

Reduce LV

If a logical volume needs to be reduced in size, first shrink the file system itself. Not all file systems support online shrinking.

For instance, ext4 does not support online shrinking so the file system needs to be unmounted first. It is also recommended to do a file system check to make sure there are no inconsistencies:

```
root # umount /mnt/data
```
```
root # e2fsck -f /dev/vg0/lvol1
```
```
root # resize2fs /dev/vg0/lvol1 150M
```
With a reduced file system, it is now possible to reduce the logical volume as well:

```
root # lvreduce -L150M /dev/vg0/lvol1
```

LV Permissions

LVM supports permission states on the logical volumes.

For instance, a logical volume can be set to *read only* using the **lvchange** command:

```
root # lvchange -p r /dev/vg0/lvol1
```
```
root # mount -o remount /dev/vg0/lvol1
```
The remount is needed as the change is not enforced immediately.

To mark the logical volume as writable again, use the *rw* permission bit:

```
root # lvchange -p rw /dev/vg0/lvol1 && mount -o remount /dev/vg0/lvol1
```

Remove LV

Before removing a logical volume, make sure it is no longer mounted:

```
root # umount /dev/vg0/lvol1
```
Deactivate the logical volume so that no further write activity can take place:

```
root # lvchange -a n /dev/vg0/lvol1
```
With the volume unmounted and deactivated, it can now be removed, freeing the extents allocated to it for use by other logical volumes in the volume group:

```
root # lvremove /dev/vg0/lvol1
```

Features

LVM provides quite a few interesting features for storage administrators, including (but not limited to)

- thin provisioning (over-committing storage)
- snapshot support

- volume types with different storage allocation methods

Thin provisioning

Recent versions of LVM2 (2.02.89) support "thin" volumes. Thin volumes are to block devices what sparse files (https://en.wikipedia.org/wiki/Sparse_file) are to file systems. Thus, a thin logical volume within a pool can be "over-committed": its presented size can be larger than the allocated size - it can even be larger than the pool itself. Just like a sparse file, the extents are allocated as the block device gets populated. If the file system has *discard* support extents are freed again as files are removed, reducing space utilization of the pool.

Within LVM, such a thin pool is a special type of logical volume, which itself can host logical volumes.

Creating a thin pool

> ⚠ **Warning**
> If an overflow occurs within the thin pool metadata, then the pool will be corrupted. **LVM cannot recover from this**.

> 🗋 **Note**
> If the thin pool gets exhausted, any process that would cause the thin pool to allocate more (unavailable) extents will be stuck in "killable sleep" state until either the thin pool is extended or the process recieves SIGKILL.

Each thin pool has metadata associated with it, which is added to the thin pool size. LVM will compute the size of the metadata based on the size of the thin pool as the minimum of *pool_chunks * 64 bytes* or 2MiB, whichever is larger. The administrator can select a different metadata size as well.

To create a thin pool, add the `--type thin-pool --thinpool thin_pool` options to **lvcreate**:

```
root # lvcreate -L 150M --type thin-pool --thinpool thin_pool vg0
```
The above example creates a thin pool called *thin_pool* with a total size of 150 MB. This is the real allocated size for the thin pool (and thus the total amount of actual storage that can be used).

To explicitly ask for a certain metadata size, use the `--metadatasize` option:

```
root # lvcreate -L 150M --poolmetadatasize 2M --type thin-pool --thinpool thin_pool vg0
```
Due to the metadata that is added to the thin pool, the intuitive way of using all available size in a volume group for a logical volume does not work (see LVM bug 812726 (https://bugzilla.redhat.com/show_bug.cgi?id=812726)):

```
root # lvcreate -l 100%FREE --type thin-pool --thinpool thin_pool vg0
```

```
 Insufficient suitable allocatable extents for logical volume thin_pool: 549 more required
```

Note the thin pool does not have an associated device node like other LV's.

Creating a thin logical volume

A *thin logical volume* is a logical volume inside the thin pool (which itself is a logical volume). As thin logical volumes are *sparse*, a virtual size instead of a physical size is specified using the `-V` option:

```
root # lvcreate -T vg0/thin_pool -V 300M -n lvol1
```
In this example, the (thin) logical volume *lvol1* is exposed as a 300MB-sized device, even though the underlying pool only holds 150MB of real allocated storage.

It is also possible to create both the thin pool as well as the logical volume inside the thin pool in one command:

```
root # lvcreate -T vg0/thin_pool -V 300M -L150M -n lvol1
```

Listing thin pools and thin logical volumes

Thin pools and thin logical volumes are special types of logical volumes, and as such as displayed through the **lvdisplay** command. The **lvscan** command will also detect these logical volumes.

Extending a thin pool

> ⚠ **Warning**
> As of LVM2 2.02.89, the metadata size of the thin pool cannot be expanded, it is fixed at creation

The thin pool is expanded like a non-thin logical volume using `lvextend`. For instance:

```
root # lvextend -L500M vg0/thin_pool
```

Extending a thin logical volume

A thin logical volume is expanded just like a regular one:

```
root # lvextend -L1G vg0/lvol1
```
Note that the `lvextend` command uses the `-L` option (or `-l` if extent counts are used) and not a "virtual size" option as was used during the creation.

Reducing a thin pool

Currently, LVM cannot reduce the size of the thin pool. See LVM bug 812731 (https://bugzilla.redhat.com/show_bug. cgi?id=812731).

Reducing a thin logical volume

Thin logical volumes are reduced just like regular logical volumes.

For instance:

```
root # lvreduce -L300M vg0/lvol1l
```
Note that the `lvreduce` command uses the `-L` option (or `-l` if extent counts are used) and not a "virtual size" option as was used during the creation.

Removing thin pools

Thin pools cannot be removed until all the thin logical volumes inside it are removed.

When a thin pool no longer services any thin logical volume, it can be removed through the `lvremove` command:

```
root # lvremove vg0/thin_pool
```

LVM2 snapshots and thin snapshots

A snapshot is a logical volume that acts as copy of another logical volume. It displays the state of the original logical volume at the time of snapshot creation.

> ⚠ **Warning**
> Since the logical snapshot volume also gets the same filesystem *LABEL* and *UUID*, be sure the `/etc/fstab` file or initramfs **does not** contain entries for these filesystems using the `LABEL=` or `UUID=` syntax. Otherwise you might end up with the snapshot being mounted instead of the (intended) original logical volume.

Creating a snapshot logical volume

A snapshot logical volume is created using the `-s` option to `lvcreate`. Snapshot logical volumes are still given allocated storage as LVM "registers" all changes made to the original logical volume and stores these changes in the allocated storage for the snapshot. When querying the snapshot state, LVM will start from the original logical volume and then check all changes registered, "undoing" the changes before showing the result to the user.

A snapshot logical volume henceforth "growths" at the rate that changes are made on the original logical volume. When the allocated storage for the snapshot is completely used, then the snapshot will be removed automatically from the system.

```
root # lvcreate -l 10%VG -s -n 20140412_lvol1 /dev/vg0/lvol1
```
The above example creates a snapshot logical volume called *20140412_lvol1*, based on the logical volume *lvol1* in volume group *vg0*. It uses 10% of the space (extents actually) allocated to the volume group.

Accessing a snapshot logical volume

Snapshot logical volumes can be mounted like regular logical volumes. They are even not restricted to read-only operations - it is possible to modify snapshots and thus use it for things such as testing changes before doing these on a "production" file system.

As long as snapshot logical volumes exist, the regular/original logical volume cannot be reduced in size or removed.

LVM thin snapshots

> **□ Note**
> A thin snapshot can only be taken on a thin pool for a thin logical volume. The thin device mapper target supports thin snapshots of read-only non-thin logical volumes, but the LVM2 tooling does not support this. However, it is possible to create a regular (non-thin) snapshot logical volume of a thin logical volume.

To create a thin snapshot, the `lvcreate` command is used with the `-s` option. No size declaration needs to be passed on:

```
root # lvcreate -s -n 20140413_lvol1 /dev/vg0/lvol1
```
Thin logical volume snapshots have the same size as their original thin logical volume, and use a physical allocation of 0 just like all other thin logical volumes.

> **❶ Important**
> If -l or -L is specified, a snapshot will still be created, but the resulting snapshot will be a regular snapshot, not a thin snapshot.

It is also possible to take snapshots of snapshots:

```
root # lvcreate -s -n 1_20140413_lvol1 /dev/vg0/20140413_lvol1
```
Thin snapshots have several advantages over regular snapshots. First, thin snapshots are independent of their original logical volume once created. The original logical volume can be shrunk or deleted without affecting the snapshot. Second, thin snapshots can be efficiently created recursively (snapshots of snapshots) without the "chaining" overhead of regular recursive LVM snapshots.

Rolling back to snapshot state

To rollback the logical volume to the version of the snapshot, use the following command:

```
root # lvconvert --merge /dev/vg0/20140413_lvol1
```
This might take a couple of minutes, depending on the size of the volume. Please note that the rollback will only happen once the parent logical volume is offline. Hence a reboot might be required.

> **❶ Important**
> The snapshot will disappear and this change is not revertible

Rolling back thin snapshots

For thin volumes, `lvconvert --merge` does not work. Instead, delete the original logical volume and rename the snapshot:

```
root # umount /dev/vg0/lvol1
```
```
root # lvremove /dev/vg0/lvol1
```
```
root # lvrename vg0/20140413_lvol1 lvol1
```

Different storage allocation methods

LVM supports different allocation methods for storage:

- Linear volumes (which is the default);
- Mirrored volumes (in a more-or-less active/standby setup);
- Striping (RAID0);
- Mirrored volumes (RAID1 - which is more an active/active setup);
- Striping with parity (RAID4 and RAID5);
- Striping with double parity (RAID6);

- Striping and mirroring (RAID10).

Linear volumes

Linear volumes are the most common kind of LVM volumes. LVM will attempt to allocate the logical volume to be as physically contiguous as possible. If there is a physical volume large enough to hold the entire logical volume, then LVM will allocate it there, otherwise it will split it up into as few pieces as possible.

The commands introduced earlier on to create volume groups and logical volumes create linear volumes.

Because linear volumes have no special requirements, they are the easiest to manipulate and can be resized and relocated at will. If a logical volume is allocated across multiple physical volumes, and any of the physical volumes become unavailable, then that logical volume cannot be started anymore and will be unusable.

Mirrored volumes

LVM supports *mirrored* volumes, which provide fault tolerance in the event of drive failure. Unlike RAID1, there is no performance benefit - all reads and writes are delivered to a single side of the mirror.

To keep track of the mirror state, LVM requires a *log* to be kept. It is recommended (and often even mandatory) to position this log on a physical volume that does not contain any of the mirrored logical volumes. There are three kind of logs that can be used for mirrors:

1. **Disk** is the default log type. All changes made are logged into extra metadata extents, which LVM manages. If a device fails, then the changes are kept in the log until the mirror can be restored again.
2. **Mirror** logs are **disk** logs that are themselves mirrored.
3. **Core** mirror logs record the state of the mirror in memory only. LVM will have to rebuild the mirror every time it is activated. This type is useful for temporary mirrors.

To create a logical volume with a single mirror, pass the *-m 1* argument (to select standard mirroring) with optionally `--mirrorlog` to select a particular log type:

```
root # lvcreate -m 1 --mirrorlog mirror -l 40%VG --nosync -n lvol1 vg0
```
The *-m 1* tells LVM to create one (additional) mirror, so requiring 2 physical volumes. The `--nosync` option is an optimization - without it LVM will try synchronize the mirror by copying empty sectors from one logical volume to another.

It is possible to create a mirror of an existing logical volume:

```
root # lvconvert -m 1 -b vg0/lvol1
```
The `-b` option does the conversion in the background as this can take quite a while.

To remove a mirror, set the number of mirrors (back) to 0:

```
root # lvconvert -m0 vg0/lvol1
```
If part of the mirror is unavailable (usually because the disk containing the physical volume has failed), the volume group will need to be brought up in degraded mode:

```
root # vgchange -ay --partial vg0
```
On the first write, LVM will notice the mirror is broken. The default policy ("remove") is to automatically reduce/break the mirror according to the number of pieces available. A 3-way mirror with a missing physical volume will be reduced to 2-way mirror; a 2-way mirror will be reduced to a regular linear volume. If the failure is only transient, and the missing physical volume returns after LVM has broken the mirror, the mirrored logical volume will need to be recreated on it.

To recover the mirror, the failed physical volume needs to be removed from the volume group, and a replacement physical volume needs to be added (or if the volume group has a free physical volume, it can be created on that one). Then the mirror can be recreated with **lvconvert** at which point the old physical volume can be removed from the volume group:

```
root # vgextend vg0 /dev/sdc1
```
```
root # lvconvert -b -m 1 --mirrorlog disk vg0/lvol1
```
```
root # vgreduce --removemissing vg0
```
It is possible to have LVM recreate the mirror with free extents on a different physical volume if one side fails. To accomplish that, set `mirror_image_fault_policy` to *allocate* in `lvm.conf`.

Thin mirrors

It is not (yet) possible to create a mirrored thin pool or thin volume. It is possible to create a mirrored thin pool by creating a normal mirrored logical volume and then converting the logical volume to a thin pool with **lvconvert**. 2 logical volumes are required: one for the thin pool and one for the thin metadata; the conversion process will merge them into a single logical volume.

> **⚠ Warning**
> LVM 2.02.98 or above is required for this to work properly. Prior versions are either not capable or will segfault and corrupt the volume group. Also, conversion of a mirror into a thin pool **destroys** all existing data in the mirror!

```
root # lvcreate -m 1 --mirrorlog mirrored -l40%VG -n thin_pool vg0
root # lvcreate -m 1 --mirrorlog mirrored -L4MB -n thin_meta vg0
root # lvconvert --thinpool vg0/thin_pool --poolmetadata vg0/thin_meta
```

Striping (RAID0)

Instead of a linear volume, where multiple contiguous physical volumes are appended, it possible to create a *striped* or *RAID0* volume for better performance. This will alternate storage allocations across the available physical volumes.

To create a striped volume over three physical volumes:

```
root # lvcreate -i 3 -l 20%VG -n lvol1_stripe vg0
```

> Using default stripesize 64.00 KiB

The `-i` option indicates over how many physical volumes the striping should be done.

It is possible to mirror a stripe set. The `-i` and `-m` options can be combined to create a striped mirror:

```
root # lvcreate -i 2 -m 1 -l 10%VG vg0
```
This creates a 2 physical volume stripe set and mirrors it on 2 different physical volumes, for a total of 4 physical volumes. An existing stripe set can be mirrored with `lvconvert`.

A thin pool can be striped like any other logical volume. All the thin volumes created from the pool inherit that settings - do not specify it manually when creating a thin volume.

It is not possible to stripe an existing volume, nor reshape the stripes across more/less physical volumes, nor to convert to a different RAID level/linear volume. A stripe set can be mirrored. It is possible to extend a stripe set across additional physical volumes, but they must be added in multiples of the original stripe set (which will effectively linearly append a new stripe set).

Mirroring (RAID1)

Unlike RAID0, which is striping, RAID1 (/wiki/Raid1_with_LVM_from_scratch) is mirroring, but implemented differently than the original LVM mirror. Under RAID1, reads are spread out across physical volumes, improving performance. RAID1 mirror failures do not cause I/O to block because LVM does not need to break it on write.

Any place where an LVM mirror could be used, a RAID1 mirror can be used in its place. It is possible to have LVM create RAID1 mirrors instead of regular mirrors implicitly by setting *mirror_segtype_default* to *raid1* in `lvm.conf`.

> **⚠ Warning**
> LVM RAID1 mirroring is not supported by GRUB before version 2.02. Ensure the latest version is installed with grub-install or the system may become unbootable if the grub files are contained in the LVM RAID1.

To create a logical volume with a single mirror:

```
root # lvcreate -m 1 --type raid1 -l 40%VG --nosync -n lvm_raid1 vg0
```
Note the difference for creating a mirror: There is no *mirrorlog* specified, because RAID1 logical volumes do not have an explicit mirror log - it built-in to the logical volume.

It is possible to convert an existing logical volume to RAID1:

```
root # lvconvert -m 1 --type raid1 -b vg0/lvol1
```
To remove a RAID1 mirror, set the number of mirrors to 0:

```
root # lvconvert -m0 vg0/lvm_raid1
```
If part of the RAID1 is unavailable (usually because the disk containing the physical volume has failed), the volume group will need to be brought up in degraded mode:

```
root # vgchange -ay --partial vg0
```

Unlike an LVM mirror, writing does NOT break the mirroring. If the failure is only transient, and the missing physical volume returns, LVM will resync the mirror by copying cover the out-of-date segments instead of the entire logical volume. If the failure is permanent, then the failed physical volume needs to be removed from the volume group, and a replacement physical volume needs to be added (or if the volume group has a free physical volume, it can be created on a different PV). The mirror can then be repaired with `lvconvert`, and the old physical volume can be removed from the volume group:

```
root # vgextend vg0 /dev/sdc1
root # lvconvert --repair -b vg0/lvm_raid1
root # vgreduce --removemissing vg0
```

Thin RAID1

It is not (yet) possible to create a RAID1 thin pool or thin volume. It is possible to create a RAID1 thin pool by creating a normal mirrored logical volume and then converting the logical volume to a thin pool with `lvconvert`. 2 logical volumes are required: one for the thin pool and one for the thin metadata; the conversion process will then merge them into a single logical volume.

> **⚠ Warning**
> LVM 2.02.98 or above is required for this to work properly. Prior versions are either not capable or will segfault and corrupt the VG. Also, conversion of a RAID1 into a thin pool **destroys** all existing data in the mirror!

```
root # lvcreate -m 1 --type raid1 -l40%VG -n thin_pool vg0
root # lvcreate -m 1 --type raid1 -L4MB -n thin_meta vg0
root # lvconvert --thinpool vg0/thin_pool --poolmetadata vg00/thin_meta
```

Striping with parity (RAID4 and RAID5)

> **🗋 Note**
> Striping with parity requires at least 3 physical volumes.

RAID0 is not fault-tolerant - if any of the physical volumes fail then the logical volume is unusable. By adding a parity stripe to RAID0 the logical volume can still function if a physical volume is missing. A new physical volume can then be added to restore fault tolerance.

Stripsets with parity come in 2 flavors: RAID4 and RAID5. Under RAID4, all the parity stripes are stored on the same physical volume. This can become a bottleneck because all writes hit that physical volume, and it gets worse the more physical volumes are in the array. With RAID5, the parity data is distributed evenly across the physical volumes so none of them become a bottleneck. For that reason, RAID4 is rare and is considered obsolete/historical. In practice, all stripesets with parity are RAID5.

```
root # lvcreate --type raid5 -l 20%VG -i 2 -n lvm_raid5 vg0
```
Only the data physical volumes are specified with -i, LVM adds one to it automatically for the parity. So for a 3 physical volume RAID5, *-i 2* is passed on and not *-i 3*.

When a physical volume fails, then the volume group will need to be brought up in degraded mode:

```
root # vgchange -ay --partial vg0
```
The volume will work normally at this point, however this degrades the array to RAID0 until a replacement physical volume is added. Performance is unlikely to be affected while the array is degraded - although it does need to recompute its missing data via parity, it only requires simple XOR for the parity block with the remaining data. The overhead is negligible compared to the disk I/O.

To repair the RAID5:

```
root # lvconvert --repair vg0/lvm_raid5
root # vgreduce --removemissing vg0
```
It is possible to replace a still working physical volume in RAID5 as well:

```
root # lvconvert --replace /dev/sdb1 vg0/lvm_raid5
root # vgreduce vg0 /dev/sdb1
```

The same restrictions of stripe sets apply to stripe sets with parity as well: it is not possible to enable striping with parity on an existing volume, nor reshape the stripes with parity across more/less physical volumes, nor to convert to a different RAID level/linear volume. A stripe set with parity can be mirrored. It is possible to extend a stripe set with parity across additional physical volumes, but they must be added in multiples of the original stripe set with parity (which will effectively linearly append a new stripe set with parity).

Thin RAID5 logical volumes

It is not (yet) possible to create stripe set with parity (RAID5) thin pools or thin logical volumes. It is possible to create a RAID5 thin pool by creating a normal RAID5 logical volume and then converting the logical volume into a thin pool with **lvconvert**. 2 logical volumes are required: one for the thin pool and one for the thin metadata; the conversion process will merge them into a single logical volume.

> **⚠ Warning**
> LVM 2.02.98 or above is required for this to work properly. Prior versions are either not capable or will segfault and corrupt the VG. Also, coversion of a RAID5 LV into a thin pool **destroys** all existing data in the LV!

```
root # lvcreate --type raid5 -i 2 -l20%VG -n thin_pool vg0
root # lvcreate --type raid5 -i 2 -L4MB -n thin_meta vg0
root # lvconvert --thinpool vg0/thin_pool --poolmetadata vg00/thin_meta
```

Striping with double parity (RAID6)

> **🗋 Note**
> RAID6 requires at least 5 physical volumes.

RAID6 is similar to RAID5, however RAID6 can survive up to **two** physical volume failures, thus offering more fault tolerance than RAID5 at the expense of extra physical volumes.

```
root # lvcreate --type raid6 -l 20%VG -i 3 -n lvm_raid6 vg00
```

Like RAID5, the `-i` option is used to specify the number of physical volumes to stripe, excluding the 2 physical volumes for parity. So for a 5 physical volume RAID6, pass on `-i 3` and not `-i 5`.

Recovery for RAID6 is the same as RAID5.

> **🗋 Note**
> Unlike RAID5 where parity block is cheap to recompute vs disk I/O, this is only half true in RAID6. RAID6 uses 2 parity stripes: One stripe is computed the same way as RAID5 (simple XOR). The second parity stripe is much harder to compute - see [https://www.kernel.org/pub/linux/kernel/people/hpa/raid6.pdf (https://www.kernel.org/pub/linux/kernel/people/hpa/raid6.pdf)

Thin RAID6 logical volumes

It is not (yet) possible to create a RAID6 thin pool or thin volumes. It is possible to create a RAID6 thin pool by creating a normal RAID6 logical volume and then converting the logical volume into a thin pool with **lvconvert**. 2 logical volumes are required: one for the thin pool and one for the thin metadata; the conversion process will merge them into a single logical volume.

> **⚠ Warning**
> LVM 2.02.98 or above is required for this to work properly. Prior versions are either not capable or will segfault and corrupt the VG. Also, conversion of a RAID6 LV into a thin pool **destroys** all existing data in the LV!

```
root # lvcreate --type raid6 -i 2 -l20%VG -n thin_pool vg0
root # lvcreate --type raid6 -i 2 -L4MB -n thin_meta vg0
root # lvconvert --thinpool vg0/thin_pool --poolmetadata vg0/thin_meta
```

LVM RAID10

> **🗋 Note**
> RAID10 requires at least 4 physical volumes. Also LVM syntax requires the number of physical volumes be multiple of the numbers stripes and mirror, even though RAID10 format does not

RAID10 is a combination of RAID0 and RAID1. It is more powerful than RAID0+RAID1 as the mirroring is done at the stripe level instead of the logical volume level, and therefore the layout doesn't need to be symmetric. A RAID10 volume can tolerate at least a single missing physical volume, and possibly more.

> **⬜ Note**
> LVM currently limits RAID10 to a single mirror.

```
root # lvcreate --type raid10 -l 1020 -i 2 -m 1 --nosync -n lvm_raid10 vg0
```
Both the `-i` and `-m` options are specified: `-i` is the number of stripes and `-m` is the number of mirrors. Two stripes and 1 mirror requires 4 physical volumes.

Thin RAID10

It is not (yet) possible to create a RAID10 thin pool or thin volumes. It is possible to create a RAID10 thin pool by creating a normal RAID10 logical volume and then converting the logical volume into a thin pool with **lvconvert**. 2 logical volumes are required: one for the thin pool and one for the thin metadata; the conversion process will merge them into a single logical volume.

> **⚠ Warning**
> Conversion of a RAID10 logical volume into a thin pool **destroys** all existing data in the logical volume!

```
root # lvcreate -i 2 -m 1 --type raid10 -l 1012 -n thin_pool vg0
root # lvcreate -i 2 -m 1 --type raid10 -l 6 -n thin_meta vg0
root # lvconvert --thinpool vg0/thin_pool --poolmetadata vg0/thin_meta
```

Experimenting with LVM

It is possible to experiment with LVM without using real storage devices. To accomplish this, loopback devices are created.

First make sure to have the loopback module loaded.

```
root # modprobe -r loop && modprobe loop max_part=63
```

> **⬜ Note**
> If loopback support is built into the kernel, then use `loop.max_part=63` as boot option.

Next configure LVM to not use udev (/wiki/Udev) to scan for devices:

FILE **/etc/lvm/lvm.conf** **Disabling udev in LVM config**

```
obtain_device_list_from_udev = 0
```

> **❶ Important**
> This is for testing only, make sure to change the setting back when dealing with real devices since it is much faster to use udev!

Create some image files which will become the storage devices. The next example uses five files for a total of about ~10GB of real hard drive space:

```
root # mkdir /var/lib/lvm_img
root # dd if=/dev/null of=/var/lib/lvm_img/lvm0.img bs=1024 seek=2097152
root # dd if=/dev/null of=/var/lib/lvm_img/lvm1.img bs=1024 seek=2097152
root # dd if=/dev/null of=/var/lib/lvm_img/lvm2.img bs=1024 seek=2097152
root # dd if=/dev/null of=/var/lib/lvm_img/lvm3.img bs=1024 seek=2097152
root # dd if=/dev/null of=/var/lib/lvm_img/lvm4.img bs=1024 seek=2097152
```
Check which loopback devices are available:

```
root # losetup -a
```
Assuming all loopback devices are available, next create the devices:

```
root # losetup /dev/loop0 /var/lib/lvm_img/lvm0.img
root # losetup /dev/loop1 /var/lib/lvm_img/lvm1.img
root # losetup /dev/loop2 /var/lib/lvm_img/lvm2.img
root # losetup /dev/loop3 /var/lib/lvm_img/lvm3.img
root # losetup /dev/loop4 /var/lib/lvm_img/lvm4.img
```

The `/dev/loop[0-4]` devices are now available to use as any other hard drive in the system (and thus be perfect for physical volumes).

> **🗅 Note**
> On the next reboot, all the loopback devices will be released and the folder `/var/lib/lvm_img` can be deleted.

Troubleshooting

LVM has a few features that already provide some level of redundancy. However, there are situations where it is possible to restore lost physical volumes or logical volumes.

vgcfgrestore utility

By default, on any change to a LVM physical volume, volume group, or logical volume, LVM2 create a backup file of the metadata in `/etc/lvm/archive`. These files can be used to recover from an accidental change (like deleting the wrong logical volume). LVM also keeps a backup copy of the most recent metadata in `/etc/lvm/backup`. These can be used to restore metadata to a replacement disk, or repair corrupted metadata.

To see what states of the volume group are available to be restored (partial output to improve readability):

```
root # vgcfgrestore --list vg00
```

```
File:          /etc/lvm/archive/vg0_00042-302371184.vg
VG name:       vg0
Description:   Created *before* executing 'lvremove vg0/lvm_raid1'
Backup Time:   Sat Jul 13 01:41:32 201
```

Recovering an accidentally deleted logical volume

Assuming the logical volume *lvm_raid1* was accidentally removed from volume group *vg0*, it is possible to recover it as follows:

```
root # vgcfgrestore -f /etc/lvm/archive/vg0_00042-302371184.vg vg0
```

> **❶ Important**
> vgcfgrestore only restores LVM metadata, *not* the data inside the logical volume. However pvremove, vgremove, and lvremove only wipe metadata, leaving any data intact. If `issue_discards` is set in `/etc/lvm/lvm.conf` though, then these command *are* destructive to data.

Replacing a failed physical volume

It possible to do a true "replace" and recreate the metadata on the new physical volume to be the same as the old physical volume:

```
root # vgdisplay --partial --verbose
```

```
--- Physical volumes ---
PV Name               /dev/loop0
PV UUID               iLdp2U-GX3X-W2PY-aSlX-AVE9-7zVC-Cjr5VU
PV Status             allocatable
Total PE / Free PE    511 / 102

PV Name               unknown device
PV UUID               T7bUjc-PYoO-bMqI-53vh-uxOV-xHYv-0VejBY
PV Status             allocatable
Total PE / Free PE    511 / 102
```

The important line here is the UUID "unknown device".

root # `pvcreate --uuid T7bUjc-PYoO-bMqI-53vh-uxOV-xHYv-0VejBY --restorefile /etc/lvm/backup/vg0 /dev/loop1`

```
Couldn't find device with uuid T7bUjc-PYoO-bMqI-53vh-uxOV-xHYv-0VejBY.
Physical volume "/dev/loop1" successfully created
```

This recreates the physical volume metadata, but not the missing logical volume or volume group data on the physical volume.

root # `vgcfgrestore -f /etc/lvm/backup/vg0 vg0`

```
Restored volume group vg0
```

This now reconstructs all the missing metadata on the physical volume, including the logical volume and volume group data. However it doesn't restore the data, so the mirror is out of sync.

root # `vgchange -ay vg0`

```
device-mapper: reload ioctl on  failed: Invalid argument
1 logical volume(s) in volume group "vg0" now active
```

root # `lvchange --resync vg0/lvm_raid1`

```
Do you really want to deactivate logical volume lvm_raid1 to resync it? [y/n]: y
```

This will resync the mirror. This works with RAID 4,5 and 6 as well.

Deactivating a logical volume

It is possible to deactivate a logical volume with the following command:

root # `umount /dev/vg0/lvol1`

root # `lvchange -a n /dev/vg0/lvol1`

It is not possible to mount the logical volume anywhere before it gets reactivated:

root # `lvchange -a y /dev/vg0/lvol1`

See also

- Device-mapper (/wiki/Device-mapper)
- Raid1 with LVM from scratch (/wiki/Raid1_with_LVM_from_scratch)
- Custom_Initramfs#LVM (/wiki/Custom_Initramfs#LVM) — Describes the creation of a custom initramfs for usage with LVM
- Custom_Initramfs/Examples#LUKS.2C_LVM.2C_Resume_from_Hibernate.2C_Script_to_Build_the_Initramfs (/wiki/Custom_Initramfs/Examples#LUKS.2C_LVM.2C_Resume_from_Hibernate.2C_Script_to_Build_the_Initramfs) — An example script

External resources

- LVM2 sourceware.org (http://sourceware.org/lvm2/)
- LVM tldp.org (http://tldp.org/HOWTO/LVM-HOWTO/)
- LVM2 Wiki redhat.com (http://sources.redhat.com/lvm2/wiki/)
- Shrinking root logical volume on-line (http://www.voleg.info/linux-shrink-root-lv.html)

Retrieved from "https://wiki.gentoo.org/index.php?title=LVM&oldid=1116647 (https://wiki.gentoo.org/index.php?title=LVM&oldid=1116647)"