Logical Volume Manager

Table of Contents:

Introduction to the Logical Volume Manager (LVM)

LVM stands for Logical Volume Manager. The logical volume manager introduces extra layers of abstraction between the disks or storage devices presented to a Linux system and the file systems placed on those disks or storage devices.

Before we get into those layers of abstraction and how to use them, let's talk about why you would want to use LVM in the first place.

Why use LVM?

Flexible Capacity

One benefit of using LVM is that you can create file systems that extend across multiple storage devices. With LVM, you can aggregate multiple storage devices into a single logical volume.

Easily Resize Storage While Online

LVM allows you to expand or shrink filesystems in real-time while the data remains online and fully accessible. Without LVM you would have to reformat and repartition the underlying storage devices. Of course, you would have to take the file system offline to perform that work. LVM eliminates this problem.

Online Data Relocation

LVM also allows you to easily migrate data from one storage device to another while online. For example, if you want to deploy newer, faster, or more resilient storage, you can move your existing data from the current storage devices to the new ones while your system is active.

Convenient Device Naming

Instead of using abstract disk numbers, you can use human-readable device names of your choosing. Instead of wondering what data is on /dev/sdb, you can name your data with a descriptive name.

Disk Striping

With LVM, you can stripe data across two or more disks. This can dramatically increase throughput by allowing your system to read data in parallel.

Data Redundancy / Data Mirroring

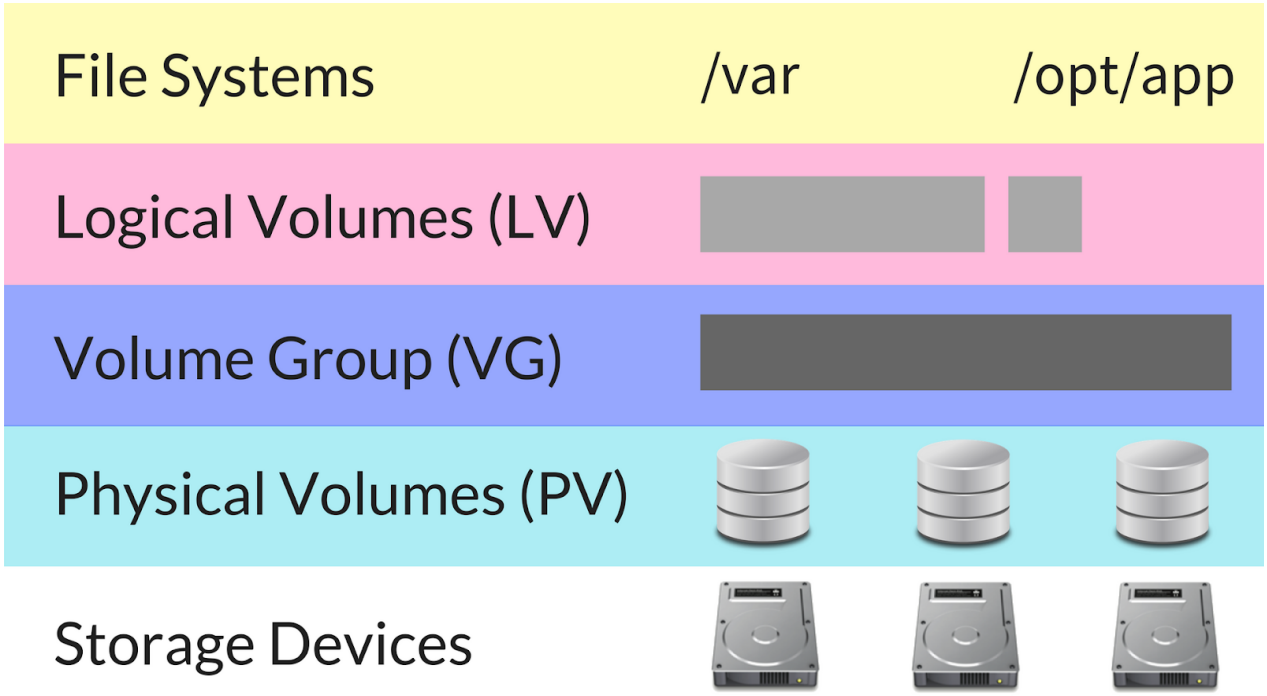
If you want to increase fault tolerance and reliability, use LVM to mirror your data so that you always have more than one copy of your data. Using LVM mirroring prevents single points of failure. If one storage device fails, your data can be accessed via another storage device. You can then fix or replace the failed storage device to restore your mirror, without downtime.

Snapshots

LVM gives you the ability to create point-in-time snapshots of your filesystems. This can be perfect for when you need consistent backups. For example, you could pause writes to a database, take a snapshot of the logical volume where the database data resides, then resume writes to the database. That way you ensure your data is in a known-good state when you perform the backup of the snapshot.

Layers of Abstraction in LVM

As I mentioned earlier, the logical volume manager introduces extra layers of abstraction between the storage devices and the file systems placed on those storage devices.



The first layer of abstraction is physical volumes. These are storage devices that are used by LVM. The name is a bit of a legacy name. To be clear, these storage devices do not have to be physical. They just have to be made available to the Linux operating system. In other words, as long as Linux sees the device as a block storage device, it can be used as a physical volume (PV). Physical hard drives, iSCSI devices, SAN disks, and so on can be PVs. You can allocate an entire storage device as a PV or you can partition a storage device and use just that one partition as a PV.

The next layer of abstraction is the Volume Group. A volume group is made up of one or more PVs. You can think of a volume group as a pool of storage. If you want to increase the size of the pool, you simply add more PVs. Keep in mind that you can have different types of storage in the same volume group if you want. For example, you could have some PVs that are backed by hard drives and other PVs that are backed by san disks. We'll talk about how this can come in handy when performing data migrations later (in this lesson).

The next layer of abstraction is the Logical Volume layer. Logical Volumes are created from a volume group. File systems are created on Logical Volumes. Without LVM you would create a file system on a disk partition, but with LVM you create a file system on a logical volume. As long as there is free space in the Volume Group, logical volumes can be extended. You can also shrink logical volumes to reclaim unused space if you want, but typically you'll find yourself extending logical volumes.

Logical Volume Creation Process

At a high level, the process for creating a logical volume is this:

1. Create one or more physical volumes.
2. Create a volume group from those one or more physical volumes.
3. Finally, you can create one or more logical volumes from the volume group.

Creating Physical Volumes

Because we are going to be working on system administration type of tasks that will require root privileges, I'm going to switch to the root user first.

```
su -
```

Before you can create a physical volume, you need to know what storage devices are available. The easiest way to do that is with the `lvmdiskscan` command:

```
lvmdiskscan
```

This command shows you all the storage devices that have the ability to be used with LVM. At least one of these storage devices is currently being used to host the root filesystem, but the output we see here doesn't tell us which one that is. We could guess, but let's don't do that. Let's find out some more information about our block devices with the "`lsblk`" command.

```
lsblk
```

Here you can see that `/dev/sda1` is actually mounted on `/`. It contains the root filesystem. If you want to display the full paths to the devices shown with `lsblk`, use the `-p` option:

```
lsblk -p
```

Know it's clear that `sda1` is really `/dev/sda1`. We can further confirm by using the `df` command which shows disk space usage. I like using the `-h` option to force df to display sizes in a human readable format.

```
df -h
```

So, lvmdiskscan is a great way to see what disks are available. You can also use the `fdisk -l` command to see the storage attached to your Linux system.

```
fdisk -l
```

I prefer `lvmdiskscan`, but again, there are multiple ways to view the attached storage devices.

```
lvmdiskscan
```

To create a PV, a physical volume, we use the pvcreate command and give it the path to the storage device.

```
pvcreate /dev/sdb
```

The pvcreate command initializes the disk for use by the logical volume manager. You can view the list of pvs with the pvs command.

```
pvs
```

Creating Volume Groups

We can use our PV to create our VG, or volume group. I'm going to pretend that I'm creating this volume group for an application, so I'm going to name it vg_app. I like to start the name of all my volume groups with "vg_". This is not required. You can name your volume groups anything you want, but I like the convention of "vg_".

```
vgcreate vg_app /dev/sdb
```

The view the volume groups, use the vgs command:

```
vgs
```

It shows that we have 1 physical volume. The size of the volume group is 50GB and we have 50GB free in the volume group. If we look at our pvs with the pvs command, we'll now see what VG our PV belongs to.

```
pvs
```

Creating Logical Volumes

Now the we have a pool of storage in a volume group, we can use that storage to create a logical volume. Let's create a logical volume that is 20 GB in size and name it lv_data. Again, I like the convention of "lv_" as part of the logical volume name, but you don't have to do that.

```
lvcreate -L 20G -n lv_data vg_app
```

Just like we can view physical volumes with the `pvs` command and view volume groups with the `vgs` command, we can also view logical volumes with the `lvs` command.

```
lvs
```

You can also view your logical volumes with the lvdisplay command.

```
lvdisplay
```

The reason you would want to use the lvdisplay command is that it provides different output. For example, notice the LV Path: `/dev/vg_app/lv_data`.

That's the format: `/dev` followed by the volume group named followed by the logical volume name. It's easy to tell that the `lv_data` logical volume belongs to the `vg_app` volume group.

Creating File Systems

Now that we have a logical volume, we can treat it like we would a normal disk partition. So, let's put a file system on our logical volume and mount it.

```
mkfs -t ext4  /dev/vg_app/lv_data
```

```
mkdir /data
```

```
mount /dev/vg_app/lv_data /data
```

```
df -h /data
```

Now let's create a place to store the actual application files for our application. Let's call it's LV `lv_app` and give it 5GB of space.

```
lvcreate -L 5G -n lv_app vg_app
```

Now you can see we have two logical volumes:

```
lvs
```

Again, we can put a file system on this logical volume and mount it.

```
mkfs -t ext4  /dev/vg_app/lv_app
```

```
mkdir /app
```

Again, we treat this logical volume just like any other partition. Let's put it in the `/etc/fstab` so that it gets mounted at boot time.

```
vi /etc/fstab
```

```
/dev/vg_app/lv_app /app ext4 defaults 0 0
```

```
:wq!
```

```
mount /app
```

```
df -h /app
```

```
df -h
```

You also access your logical volume through the device mapper path as shown in the df output. For example, `/dev/vg_app/lv_app` can be accessed via `/dev/mapper/vg_app-lv_app`.

```
ls -l /dev/vg_app/lv_app
```

```
ls -l /dev/mapper/vg_app-lv_app
```

```
ls -l /dev/dm-1
```

Be aware the the `/dev/dm-1` might change between reboots, so don't refer to it directly when mounting or configure `/etc/fstab` and so on. Again, use the path as displayed in the lvdisplay output which is `/dev/vg-name/lv-name`.

```
lvdisplay
```

Let's see how much space we have left in our volume group:

```
vgs
```

This shows we have 1 PV, 2 LVs, and 25GB free space. Let's also pretend that our fictional application generates logs and we want to keep those log files separate from the other data. So, we need to create a logical volume for those logs. If we use -L 25g to specify the size we are are going to get an error. Let me show you and then explain why.

```
lvcreate -L 25g -n lv_logs vg_app
```

```
  Volume group "vg_app" has insufficient free space (6399 extents): 6400 required.
```

We're about to wade off into the weeds and dive into some very fine details. If you're more of a big-picture type of person, just bear with me for the next couple of minutes. If you like to know every detail, then you're going to love the next couple of minutes.

Logical Extents and Physical Extents

There is yet another layer of abstraction that we haven't talked about. Well, two layers of abstraction, really. Each of our LVs is actually divided up into LEs, which stands for logical extents. Or if we look at it from the other direction, a collection of Logical Extents makes up a Logical Volume. This is how LVM allows us to expand or shrink a logical volume -- the logical volume manager just changes the number of underlying Logical Extents for the logical volume. Anyway, to view information about these LEs, use the lvdisplay command.

```
lvdisplay
```

Look at the line that begins with "Current LE". Those are the number of logical extents that comprise this logical volume. That number is 1,280. You can use the "-m" option to show a map of the logical volume.

```
lvdisplay -m
```

This map view tells us that the logical extents for the `lv_app` LV resize on the `/dev/sdb` disk. Like an LV is divided into LE, a PV is divided into PE, physical extents. There is a one-to-one mapping of LEs to PEs.

Let's look at the map from the view of the disk, using the `pvdisplay -m` command:

```
pvdisplay -m
```

Physical extents 0 through 5119 belong to the `lv_data` logical volume. They correspond to the 0 through 5119 logical extents of that LV.

Physical extents 5120 through 6399 belong to the `lv_app` logical volume. They correspond to the 0 through 1279 logical extents of that LV.

Finally, we see we that PE 6,400 to 12,798 are unused. This is actually the space we want to use for our new logical volume.

If we back up just a bit, we can see the size of each of the PEs and LEs is 4MB. That's on the PE size line. Just below the PE line is the total number of PE, which for this PV is 12,799.

If we do the math on that: 4MB * 12799 we end up with 51,196 MB, which is 4MB shy of 50GB. You'll also notice on the "PV size line" a message that reads "not usable 4.00 MiB". The reason about 4MB of space is not useable is do to LVM metadata, alignment to disk sectors and so on. More or less, you can think of these as rounding errors. They are really insignificant in the grand scheme of things. The most important thing is to know how to get around the error we saw when we tried to create our new logical volume. Let's go back to that `lvcreate` command:

```
 lvcreate -L 25g -n lv_logs vg_app
```

```
  Volume group "vg_app" has insufficient free space (6399 extents): 6400
required.
```

So know you know why this error message is being given and exactly what it means. We're asking for 25 gb of space which works out to 6,400 extents, but we only have 6,399 extends available to allocate. Bt the way, to be thorough, you can see the number of extents available in the volume group by using the vgdisplay command.

```
vgdisplay
```

On the "Free PE / Size" line we see that we have 6,399 extents available in our volume group.

At this point we can use the "-l" option to lvcreate to specify the size in number of extents.

```
lvcreate --help
```

```
      {-l|--extents LogicalExtentsNumber[%{VG|FREE|ORIGIN}] |
       -L|--size LogicalVolumeSize[bBsSkKmMgGtTpPeE]}
```

So, we could run `lvcreate -l 6399`. Or we could use a percentage. One common pattern is to use 100% of the free remaining space like this:

```
lvcreate -l 100%FREE -n lv_logs vg_app
```

```
lvs
```

That's how you can squeeze every possible bit of space out of your volume group and put it into a logical volume.

Extending Volume Groups

Let's say that the `lv_data` logical volume is getting full and we need to extend it. If we look at our volume group, we'll see we've already allocated all our space to the existing logical volumes.

```
vgs
```

```
VFREE = 0
```

In this case, we need to extend the volume group itself before we can extend the logical volume within that volume group. So, let's repeat our initial process of looking for disks with the `lvmdiskscan` command

```
lvmdiskscan
```

It looks like sdc is free, let's use that.

```
pvcreate /dev/sdc
```

Now we can add this PV to our VG with the vgextend command.

```
vgextend vg_app /dev/sdc
```

```
vgs
```

Now if we look at the VFREE column it shows 50g of free space.

Let's look at our pvs.

```
pvs
```

We have one pv that is complete full and one that is completely free. Before we extend our logical volume into this free space, let's look at the space from the file system level.

```
df -h /data
```

It shows that the file system size is 20G.

Extending Logical Volumes

Now, let's use the lvextend command to add 5GB of space to that logical volume. In addition to increasing the size of the logical volume, we also need to grow the file system on that logical volume to fill up this new space. That's what the `-r` option is for.

```
lvextend -L +5G -r /dev/vg_app/lv_data
```

Let's see if we have increased the size of the file system:

```
df -h /data
```

If you forget to use the `-r` option to lvextend to perform the resize, you'll have to do that after the fact. Let me demonstrate that.

```
lvextend -L +5G  /dev/vg_app/lv_data
```

If we look at the lv size, it will have increased:

```
lvs
```

However, the file system is still the same size, because we forgot to use the `-r` option.

```
df -h /data
```

Extending File Systems

To fix this you'll have to use a resize tool for the specific filesystem you're working with. For ext file systems, that tool is `resize2fs`. We give it the path the the underlying device, which is a logical volume in our case.

```
resize2fs /dev/vg_app/lv_data
```

Now, the file system has be extended to fill up the available space.

```
df -h /data
```

Let's look at the map for this logical volume:

```
lvdisplay -m /dev/vg_app/lv_data
```

We see that some of the extents live on `/dev/sdb` while other extents live on `/dev/sdc`. Again, this is what makes LVM so powerful and flexible. You have have one file system that spans multiple storage devices. That way, file system sizes aren't limited to disk sizes.

Using Multiple Volume Groups

Just to demonstrate that you can have more than one volume group per Linux system, I'm going to create another vg called `vg_safe`. I'm going to pretend that I need the data stored in this vg to be very safe. Of course, vgs are made up of pvs, so lets create a couple of pvs for our new vg.

```
lvmdiskscan
```

We have two more disks that we can use. We've been creating one pv at a time, but you can give the pvcreate command a list of storage devices like this:

```
pvcreate /dev/sdd /dev/sde
```

```
pvs
```

Likewise, we can give vgcreate a list of PVs that we want in the volume group.

```
vgcreate vg_safe /dev/sdd /dev/sde
```

```
vgs
```

Creating Mirrored Logical Volumes

Let's create a mirrored logical volume. This will ensure that an exact copy of the data will be stored on two different storage devices.

```
lvcreate -m 1 -L 5G -n lv_secrets vg_safe
```

```
lvs
```

Here, the Copy%Sync column indicates if the mirror is synced. If it's at 100%, then all the data is synchronized. If you were to replace a failed disk, for a example, the mirror would need to resync the data from the existing copy to the new disk. While the copy was in progress the percent would be less than 100, and when it finished syncing the data, it would be at 100%.

Let's look at the `lvs` command with the `-a` option which stands for all.

```
lvs -a
```

The logical volume we created is actually RAID 1. So a mirror and raid 1 are the same thing. Anyway, when you create a RAID logical volume, LVM creates a metadata subvolume that is one extent in size for every data or parity subvolume in the array. In this example, creating a 2-way RAID1 array results in two metadata subvolumes (`lv_secrets_rmeta_0` and `lv_secrets_rmeta_1`) and two data subvolumes (`lv_secrets_rimage_0` and `lv_secrets_rimage_1`).

Let's look at the map with pvdisplay

```
pvdisplay -m
```

At the bottom of the screen you see that `lv_secrets_rmeta_1` and `lv_secrets_rimage_1` reside on the `/dev/sde` disk. It's scrolled off the top of your screen, but the output above that is for the `/dev/sdd` disk. That's where `lv_secrets_rmeta_0` and `lv_secrets_rimage_0` live.

To be clear, we operate on this logical volume using the name we gave it. For example, let's create a file system on that logical volume and mount it.

```
mkfs -t ext4 /dev/vg_safe/lv_secrets
```

```
mkdir /secrets
```

```
mount /dev/vg_safe/lv_secrets /secrets
```

```
df -h /secrets
```

So we write to `/dev/vg_safe/lv_secrets` and let the logical volume manager handle the mirroring. We just use this file system like any other.

Deleting Logical Volumes, Volume Groups, and Physical Volumes

Before we wrap up this lesson, I want to show you how undo your changes or how to delete logical volumes, volume groups, and physical volumes.

Let's unmount the file system that we just mounted.

```
umount /secrets
```

Now, we can remove the underlying logical volume.

```
lvremove /dev/vg_safe/lv_secrets
```

If you want to remove a pv from a vg, use the vgreduce command. First, let's show we have two pvs in our vg:

```
vgs
```

```
vgreduce vg_safe /dev/sde
```

```
vgs
```

```
pvs
```

Now that pv if free to be used in another volume group if we want to. Or if, you don't use you return free it up from LVM with the `pvremove` command.

```
pvremove /dev/sde
```

```
pvs
```

Let's finish destroying the `vg_safe` volume group with `vgremove`.

```
vgremove vg_safe
```

```
vgs
```

```
pvs
```

Now we can pvremove /dev/sdd:

```
pvremove /dev/sdd
```

```
pvs
```

Migrating Data While Online

Early I mentioned how easy it is to move data from one storage device to another with LVM. Let's say that the storage device attached to our system at /dev/sde is faster and has more space. Let's say we want to move the data that currently resides on /dev/sdb to that new disk. To do that, we'll just add /dev/sde to the volume group and migrate the data over.

First we pvcreate the device.

```
pvcreate /dev/sde
```

Now we add it to the volume group.

```
vgextend vg_app /dev/sde
```

```
pvs
```

Finally, we use the pvmove command to move all the data from /dev/sdb to /dev/sde.

```
pvmove /dev/sdb /dev/sde
```

Once the pvmove command is complete, all the data the used to live on /dev/sdb now lives on /dev/sde. And the whole time this was happening, any logical volumes and file systems that where on /dev/sdb remained online and available throughout this entire process. There's no need to take an outage for this process.

Let's look at the pvs command.

```
pvs
```

Now /dev/sdb is unused. We can take a closer look with pvdisplay:

```
pvdisplay /dev/sdb
```

We see that "Allocated PE" is zero. Now that we're done with this disk we can remove it from the volume group with vgreduce.

```
vgreduce vg_app /dev/sdb
```

And finally, `pvremove` it.

```
pvremove /dev/sdb
```

```
pvs
```

Notice that our physical volumes are of different sizes. You don't have to use the exact same underlying devices with the same sizes in the same volume group. You can mix and match your storage as you want or need.

LVM Review and Summary

In this lesson you learned that LVM adds layers of abstraction between storage devices and file systems. These layers of abstraction include Physical Volumes, Volume Groups, and Logical Volumes.

You also learned how to configure LVM, starting with the `pvcreate` command to configure physical volumes, the `vgcreate` command to configure volume groups, and the `lvcreate` command to create logical volumes. From there, you treated the logical volumes like you would any other disk partitions. You created a file system on the logical volume and mounted it like any other file system.

From there you learned how to extend logical volume using `lvextend`. When you needed to add more capacity to a volume group, you learned how to do so with the `vgextend` command.

You also learned how to create mirrored logical volumes by use the `-m` option to the `lvcreate` command.

Finally, you learned how to remove logical volumes with the `lvremove` command, remove physical volumes from volume groups with the `vgreduce` command, remove volume groups with `vgremove`, and remove physical volumes with the `pvremove` command.