# Managing Storage

## Logical Volume Management

This is the recommended way to manage storage in the enterprise as it allows for growth and flexibility in your filesystem management.

3 components make up the LVM framework: Physical Volumes, Volume Groups and Logical Volumes.

> !!! NOTE !!!
> Do not confuse Logical Partitions and Logical Volumes, these are separate entities.

### Physical Volumes

These are partitions or entire disks which have been identified as being part of the LVM framework.

To create an entire disk as a PV, you may use the command pvcreate. If your goal is to initialize only a portion of a disk to use as a PV then you need to create a partition and initialize the device file representing your partition as a PV.

```
# pvcreate /dev/sdc
Writing physical volume data to disk "/dev/sdc"
Physical volume "/dev/sdc" successfully created
```

We can view all PV's using the command pvs or to get slightly more detail use pvscan

```
# pvs
PV         VG       Fmt  Attr PSize PFree
/dev/sda2  vg_gpg1 lvm2 a--  5.37g   0
/dev/sdc           lvm2 a--  1.00g 1.00g
```

or

```
# pvscan
PV /dev/sda2   VG vg_gpg1              lvm2 [5.37 GiB / 0       free]
PV /dev/sdc                lvm2 [1.00 GiB]
Total: 2 [6.37 GiB] / in use: 1 [5.37 GiB] / in no VG: 1 [1.00 GiB]
```

!!! TIP !!!
All the PV related commands are prefixed with pv so to see all the PV related commands we type in pv<TAB><TAB>.

Volume Groups

A grouping of PV's is called a Volume Group and this collective entity is given a name. We like prefixing our VG's using VG_ but this is not a requirement. Why we do this is to easily identify which items are VG's and we do so in uppercase so that they stand out.

To create a VG we need to specify at least 1 PV to add to it, so to create a VG called VG_DB and add the PV /dev/sdc to it, we could use:

```
# vgcreate VG_DB /dev/sdc
Volume group "VG_DB" successfully created
```

On creation of a VG, a device file is created called /dev/*<VG NAME>*

Now to see all VG's on our system we could use the command vgs:

```
# vgs
VG          #PV  #LV  #SN  Attr      VSize VFree
VG_DB       1    0    0    wz--n-  1020.00m 1020.00m
vg_gpg1     1    2    0    wz--n-    5.37g       0
```

To view the properties of a newly created VG_DB we use the command vgdisplay:

```
# vgdisplay VG_DB
--- Volume group ---
VG Name               VG_DB
System ID
Format                lvm2
Metadata Areas        1
Metadata Sequence No  1
VG Access             read/write
VG Status             resizable
MAX LV                0
Cur LV                0
Open LV               0
Max PV                0
Cur PV                1
```

| | |
|---|---|
| Act PV | 1 |
| VG Size | 1020.00 MiB |
| PE Size | 4.00 MiB |
| Total PE | 255 |
| Alloc PE / Size | 0 / 0 |
| Free PE / Size | 255 / 1020.00 MiB |
| VG UUID | OV2mj7-1MBm-VVTV-CF1M-t5LP-ncsO-Xy4rQ3 |

As you can see, we have not yet created any Logical Volumes out of the space provided by this Volume Group.

This indicates that we have 1 PV added to this VG so far, we of course are able to add more.

1020 MiB is the total amount of space in MiB which is available for us to use to allocate to Logical Volumes.

Inside the LVM framework, data cannot be organized into cylinders (as when you store data on a filesystem on a single hard drive). Because our VG may consist of multiple PV (and therefore multiple hard drives) we need another way to organize data and this is done by physical extents. Each physical extent by default is 4 MiB and may be changed at VG creation time using the -s option.

We have 255 extents which make up our VG given the PV's added to it. As you can now see (Total PE) * (PE Size) = (VG Size)

This shows us how many PE have been allocated as well as the size in MiB that it represents.

Finally we can see how many free PE we have as well as the size in MiB that it represents.

Now that we have a VG, we are able to create as many Logical Volumes as we like, as long as we have sufficient space (identified by extents) inside the VG.

!!! TIP !!!
As with the PV commands, all the VG related commands are prefixed with vg so to see all the VG related commands we type in lv<TAB><TAB>.

Logical Volumes
This is what we've been building up to, a flexible storage subsystem which can be allocated a filesystem of your choosing. LV's occupy extents (space) of a VG. So you can create a LV of less than or equal to the amount of free PE as viewed using the command vgdisplay VG_DB

Our objective now is to create a LV which uses 20 Physical extents of the VG called VG_DB

(each PE is 4MiB in size so that means our LV will be 80MiB in size) and name it LV_PGSQL. We like prefixing our LV's with an uppercase LV_ so that we could easily separate the multiple elements that make up our LVM framework.

```
# lvcreate -n LV_PGSQL -l 20 VG_DB
```

To see all LV's inside VG_DB we use:

```
# lvs VG_DB
LV      VG          Attr       LSize  Pool Origin Data%  Move Log Copy%  Convert
LV_     PGSQL       VG_DB      -wi-a--- 80.00m
```

To view detailed information about LV_PGSQL which is inside VG_DB we use the command:

```
# lvdisplay VG_DB/LV_PGSQL
--- Logical volume ---
LV Path                /dev/VG_DB/LV_PGSQL
LV Name                LV_PGSQL
VG Name                VG_DB
LV UUID                C93cDe-3Ew9-Nb5O-Moqj-9Uji-Cl1w-l9qNNa
LV Write Access        read/write
LV Creation host, time elsa.enterpriselinux.pro, 2013-02-26 21:00:36 -0700
LV Status              available
# open                 0
LV Size                80.00 MiB
Current LE             20
Segments               1
Allocation             inherit
Read ahead sectors     auto
- currently set to     256
Block device           253:2
```

The device file which represents the Logical Volume itself.

The size of the LV is 80MiB

This represents the number of Logical Extents which make up our LV (each LE is equal in size to a PE which is a property of the VG and was left at the default of 4MiB).

Segments indicate the number of PV which contributes extents to this LV.

Before we can use this LV for storage we need to allocate it a filesystem using the mkfs command:

```
# mkfs -t ext4 /dev/VG_DB/LV_PGSQL
mke2fs 1.41.12 (17-May-2010)
---truncated---
```

Now let's create a mountpoint for this filesystem:

```
# mkdir -p /db/pgsql
```

And finally, let's make sure that this filesystem is persistently mounted to /db/pgsql by adding the following entry to /etc/fstab:

```
/dev/VG_DB/LV_PGSQL       /db/pgsql                ext4    defaults        0 0
```

Let's reprocess /etc/fstab to make sure that our entry is correct and that the filesystem is mounted accordingly:

```
# mount -a
```

!!! NOTE !!!
No feedback when running mount -a is a good thing! If there was some form of feedback then it means that something was not correctly processed inside /etc/fstab

## Adding More Disks

Remember that your LV can only be as big as the number of free Physical Extents which make up your VG. If you're running out of space and wish to add more PE, you may add additional Physical Volumes to a Volume Group using the vgextend command. This is a 2 step process.

1. Initialize storage as a PV using pvcreate
2. Add newly initialized PV to the existing VG

Step 1:

```
# pvcreate /dev/sdd
Writing physical volume data to disk "/dev/sdd"
Physical volume "/dev/sdd" successfully created
```

Step 2:

```
# vgextend VG_DB /dev/sdd
Volume group "VG_DB" successfully extended
```

Now when we run the vgs command against VG_DB we see that there are 2 PV which make up this VG.

```
# vgs VG_DB
VG        #PV #LV #SN Attr   VSize VFree
VG_DB   2   1   0 wz--n- 1.99g 1.91g
```

## Growing a Logical Volume

This is a common task for many system administrators and we have to be mindful that this is a 2 step process which is completed online. That's right, you do not have to unmount your filesystem and you users may continue working as this is a completely transparent process.

1. Extend the LV
2. Grow the filesystem

Recall that our VG_DB/LV_PGSQL is 80 MiB or 20 extents in size and we will now grow it to be 160 MiB (40 extents).

Step 1:

```
# lvextend -L 160M /dev/VG_DB/LV_PGSQL
Extending logical volume LV_PGSQL to 160.00 MiB
Logical volume LV_PGSQL successfully resized
```

Step 2:

```
# resize2fs /dev/VG_DB/LV_PGSQL
resize2fs 1.41.12 (17-May-2010)
Filesystem at /dev/VG_DB/LV_PGSQL is mounted on /db/pgsql; on-line resizing required
old desc_blocks = 1, new_desc_blocks = 1
Performing an on-line resize of /dev/VG_DB/LV_PGSQL to 163840 (1k) blocks.
The filesystem on /dev/VG_DB/LV_PGSQL is now 163840 blocks long.
```

Now let's verify that the VG_DB/LV_PGSQL is in fact 160 MiB:

```
# lvdisplay VG_DB/LV_PGSQL
```

```
--- Logical volume ---
LV Path                /dev/VG_DB/LV_PGSQL
LV Name                LV_PGSQL
VG Name                VG_DB
LV UUID                C93cDe-3Ew9-Nb5O-Moqj-9Uji-Cl1w-l9qNNa
LV Write Access        read/write
LV Creation host, time elsa.enterpriselinux.pro, 2013-02-26 21:00:36 -0700
LV Status              available
# open                 1
LV Size                160.00 MiB
Current LE             40
Segments               1
```

## Reducing a Logical Volume

This is quite a rare task for systems administrators but if it needs to be performed, it is imperative that you know that this is an OFFLINE process and that 4 steps are required. Of course, you can only reduce the LV to as much as what is used.

1. Unmount the filesystem
2. Verify that the filesystem is clean before reducing its size
3. Reduce the filesystem size
4. Reduce the Logical Volume size

Our VG_DB/LV_PGSQL is currently 160 MiB in size and we want to take it down to 20 MiB.

Step 1:

```
# umount /dev/VG_DB/LV_PGSQL
```

This accomplishes our first step which is to take the LV offline so that it's not in use.

Step 2:

```
# fsck -f /dev/VG_DB/LV_PGSQL
fsck from util-linux-ng 2.17.2
e2fsck 1.41.12 (17-May-2010)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/mapper/VG_DB-LV_PGSQL: 11/40960 files (9.1% non-contiguous), 10819/163840 blocks
```

Now that we've verified that the filesystem is clean and is error free we may proceed.

Step 3:

```
# resize2fs /dev/VG_DB/LV_PGSQL 40M
resize2fs 1.41.12 (17-May-2010)
Resizing the filesystem on /dev/VG_DB/LV_PGSQL to 40960 (1k) blocks.
The filesystem on /dev/VG_DB/LV_PGSQL is now 40960 blocks long.
```

We're almost there! Our LV has had the filesystem reduced to 40 MiB so the final step is to reduce the LV size.

Step 4:

```
# lvreduce -L 40M /dev/VG_DB/LV_PGSQL
WARNING: Reducing active logical volume to 40.00 MiB
THIS MAY DESTROY YOUR DATA (filesystem etc.)
Do you really want to reduce LV_PGSQL? [y/n]: y
Reducing logical volume LV_PGSQL to 40.00 MiB
Logical volume LV_PGSQL successfully resized
```

The real success is to insure that the filesystem is available when it is remounted, so let's go ahead and do that.

```
# mount -a
```

Remember the tip earlier? If the mount -a runs error free, then all was good. But just in case you're paranoid, here's a final verification:

```
# lvdisplay VG_DB/LV_PGSQL
--- Logical volume ---
LV Path                /dev/VG_DB/LV_PGSQL
LV Name                LV_PGSQL
VG Name                VG_DB
LV UUID                C93cDe-3Ew9-Nb5O-Moqj-9Uji-Cl1w-l9qNNa
LV Write Access        read/write
LV Creation host, time elsa.enterpriselinux.pro, 2013-02-26 21:00:36 -0700
LV Status              available
# open                 1
LV Size                40.00 MiB
Current LE             10
Segments               1
---truncated---
```

## Removing a disk from the LVM framework

Firstly, we need to make sure that any data on the PV which we want to remove (in our case this is /dev/sdc) is relocated to another PV. We may achieve this by using the command:

```
# pvmove /dev/sdc /dev/sdd
/dev/sdc: Moved: 100.0%
```

Now that the data has been moved, we may reduce the LV by removing the PV from the VG:

```
# vgreduce VG_DB /dev/sdc
(no output)
```

At this stage, the PV /dev/sdc is still part of the LVM framework so we can remove it by using the command:

```
# pvremove /dev/sdc
(no output)
```

!!! TIP !!!
The trend continues! So to see all the LV related commands, remember that they're all prefixed with lv, type in lv<TAB><TAB>.

## Working with iSCSI

iSCSI is a client-server framework which provides SCSI functionality over TCP/IP. Basically, it allows you to import storage which is being provided by another server and your local computer will see it as local SCSI storage. This means that a device like /dev/sdb may appear to be local but is in fact being provided across the network to you by a server. The server which provides you with iSCSI storage is called the target and the client which connects to it is called an initiator.

In this class, we will focus on connecting to an iSCSI target - in other words, we're going to setup the initiator.

In order for you to configure an iSCSI initiator, you need to install the iscsi-initiator-utils package using the following command:

```
$ yum install -y iscsi-initiator-utils
```

SUSE NOTE: zypper install -y

UBUNTU NOTE: apt-get install -y open-iscsi

Now that we have the required software installed, we're able to connect to our iSCSI target.

The process is simple and well documented in the man page for the iscsiadm command.

Step 1: Discover the storage being provided by your iSCSI target.

Given the iSCSI target called elp.enterpriselinux.pro, we run the following to see what targets (storage units) are available for our computer:

```
# iscsiadm --mode discoverydb --type sendtargets --portal elp.enterpriselinux.pro --discover
```

The output of the above should provide you with the names of the targets which you can connect to.

```
166.78.126.71:3260,1 iqn.2013-09.pro.enterpriselinux:storage
```

The name of our storage is iqn.2013-09.pro.enterpriselinux:storage which is made up of

IQN = iSCSI Qualified Name
2013-09 = The year and month in which the storage was created
pro.enterpriselinux = Our domain name written backwards
storage = The name of the target

The next step is to login to this particular target using the following command:

```
# iscsiadm --mode node --targetname iqn.2013-09.pro.enterpriselinux:storage --portal 166.78.126.71:3260 --login
```

Once you have logged in you can confirm that the new disk is seen by your computer as local storage even though it is being remotely accessed using TCP/IP.

## Cool filesystem tools

blkid

Determines the UUID of devices so that you can accurately identify a filesystem.

Example:

```
# blkid /dev/sda1
/dev/sda1: UUID="c9fb705d-19eb-4493-ab72-0eafceb34045" TYPE="ext4"
```

lsblk

This is a real gem of a command and is new to EL6. It lists all your block devices in tree view along with their respective mount points, filesystem types and labels.

Example:

```
# lsblk -f
NAME                      FSTYPE          LABEL MOUNTPOINT
sr0
sda
├─sda1                    ext4                  /boot
└─sda2                    LVM2_member
├─VolGroup-lv_root (dm-0) ext4                  /
└─VolGroup-lv_swap (dm-1) swap                  [SWAP]
vda                       LVM2_member
├─vda1
├─vda5
├─vda6
├─vda7
├─vda8
├─vda9
├─vda10
├─vda11
├─vda12
├─vda13
├─vda14
├─vda15
└─VG_DATA-LV_PRIV (dm-2)
└─private (dm-3)          ext2      FOO   /excalibur
```

lsscsi

Lists information about currently connected SCSI devices.

findmnt

Another awesome command which will display your mounted filesystems and their properties.

tune2fs

This is used to manipulate the properties of an EXT based filesystem. With it one can allocate a label to the filesystem, implement default mount options (thereby eliminating the need to explicitly specify the mount option on each server's /etc/fstab file), set the frequency of how often fsck checks your filesystem, specify journal options and much more!

Examples:

Let's change a the label of /dev/sda1

```
# tune2fs BOOT /dev/sda1
```

Now let's set the acl mount option as a default for /dev/sda4

```
# tune2fs -o acl /dev/sda4
```

Similarly, to remove a mount option put a caret (^) in front of that option as follows:

```
# tune2fs -o ^user_xattr /dev/sda4
```

To force fsck (e2fsck) to check the /dev/sda4 filesystem every 20 mounts:

```
# fsck -c 20 /dev/sda4
```

---

Lab Activiy

If you have 2 or more flash drives to spare, connect them to your system and create a Volume Group called VG_ELP with half the flash drives. Out of the space provided by that volume group use 50% of the space to create a Logical Volume called LV_TLC1. Format it with the

ext4 filesystem and make sure that it is persistently mounted to /elp/tlc1.

Now grow LV_TLC1 to use 100% of the space provided by VG_ELP.

Next, add the other half of the flash drives to VG_ELP and create another Logical Volume called LV_TLC2 which is formatted with the ext4 filesystem and have it persistently mounted to /elp/tlc1.

Reduce LV_TLC1 to 25% of the original size when you are done.