# Practice M6: FHS, Disks, Filesystems, and Archives

\* NOTE: Most of the exercises included in this practice guide are not suitable for execution in WSL or Docker environment. It is recommended a virtual infrastructure to be used instead.

For the purpose of this practice, we will need at least one virtual machine with **AlmaLinux OS 9.x** (or **CentOS Stream 9**, or **Oracle Linux 9.x**, or **Rocky Linux 9.x**), **openSUSE Leap 15.x,** or **Debian 12.x** (or **Ubuntu Server 22.04/24.04**) installed. Of course, another version of the listed or another distribution can be used, but there can be some differences.

All commands that we are going to use in this practice will be accompanied by the appropriate prompt. This way, it will be easier for us to understand which user, in which folder, and on which machine is executing the command.

The next steps will be executed on an **AlmaLinux** machine. If there is a big difference in the way we issue the command, or in its result between distributions, it will be stated clearly.

Please note that each practice we start with a fresh machine or set of machines.

Please note that the output from the commands shown below may differ compared to the one you get.

## Part 1

Let's create a folder **archive** in our home folder:

[lsauser@almalinux ~]$ **mkdir archive**

[lsauser@almalinux ~]$ **cd archive**

[lsauser@almalinux archive]$ **pwd**

/home/lsauser/archive

[lsauser@almalinux archive]$

### Working with tar

Now, we can create an archive of the **/etc** folder. We will prefix the actual command with another one – **time**. This way we will be able to compare the execution time:

[lsauser@almalinux archive]$ **time sudo tar -cvf etc.tar /etc && ls -alh etc.tar**

This combination will return the time needed to create the archive and information about the resulting file:

...

real 0m0.075s

user 0m0.009s

sys 0m0.025s

-rw-r--r--. 1 root root 23M Oct 6 13:51 etc.tar

[lsauser@almalinux archive]$

*If the* ***tar*** *command is not available, then install the package with*

*[lsauser@almalinux archive]$* ***sudo dnf install tar***

*Use the package manager for your distribution.*

In fact, **tar** doesn't apply any compression on the files, it just packs them into a single file. There are other tools specialized in compression.

### Working with zip and unzip

The two utilities - **zip** and **unzip** are not always installed with the default setup. Let's install them:

[lsauser@centos archive]$ **sudo dnf install -y zip unzip**

*Use the package manager for your distribution.*

Now, we can repeat the archiving step we did earlier, but this time with **zip**:

[lsauser@almalinux archive]$ **time sudo zip -ry etc.zip /etc && ls -alh etc.zip**

If we compare the statistics, we will notice a difference at least in the size of the resulting file:

...

real 0m0.487s

user 0m0.466s

sys 0m0.016s

-rw-r--r--. 1 root root 5.3M Oct 6 13:52 etc.zip

[lsauser@almalinux archive]$

### Adding Compression to tar

Next few utilities – **gzip**, **bzip**, and **xz**, we will use in combination with **tar**. In fact, this is their typical usage. First, let's install the missing package:

[lsauser@almalinux archive]$ **sudo dnf install -y bzip2**

*Use the package manager for your distribution.*

Now, let's issue the following set of commands:

[lsauser@almalinux archive]$ **time sudo tar -czvf etc.tar.gz /etc && ls -alh etc.tar.gz**

...

[lsauser@almalinux archive]$ **time sudo tar -cjvf etc.tar.bz2 /etc && ls -alh etc.tar.bz2**

...

[lsauser@almalinux archive]$ **time sudo tar -cJvf etc.tar.xz /etc && ls -alh etc.tar.xz**

...

As we can see, each command execution took different amount of time, and produced file varying in size. By adding different options and arguments we can achieve better compression level, or faster execution.

Here are the results:

[lsauser@almalinux archive]$ **ls -alhS etc\***

-rw-r--r--. 1 root root 23M Oct 6 13:51 etc.tar

-rw-r--r--. 1 root root 5.3M Oct 6 13:52 etc.zip

-rw-r--r--. 1 root root 5.2M Oct 6 13:53 etc.tar.gz

-rw-r--r--. 1 root root 3.7M Oct 6 13:53 etc.tar.bz2

-rw-r--r--. 1 root root 3.2M Oct 6 13:53 etc.tar.xz

[lsauser@almalinux archive]$

Before continuing with next steps, we can test if our archive is working. Let's extract the contents of one of the archive files:

[lsauser@almalinux archive]$ **tar -xzvf etc.tar.gz**

...

[lsauser@almalinux archive]$ **ls -aldF etc\***

drwxr-xr-x. 76 lsauser lsauser 8192 Oct 6 13:53 etc/

-rw-r--r--. 1 root root 23203840 Oct 6 13:51 etc.tar

-rw-r--r--. 1 root root 3847260 Oct 6 13:53 etc.tar.bz2

-rw-r--r--. 1 root root 5413111 Oct 6 13:53 etc.tar.gz

-rw-r--r--. 1 root root 3299152 Oct 6 13:53 etc.tar.xz

-rw-r--r--. 1 root root 5537323 Oct 6 13:52 etc.zip

[lsauser@almalinux archive]$

Now, we can go one level up, and clean all files created so far:

[lsauser@almalinux archive]$ **cd ..**

[lsauser@almalinux archive]$ **rm -rf archive/**

### Filesystem Exploration

We can ask for the listing of the root folder:

[lsauser@almalinux ~]$ **ls -al /**

total 16

dr-xr-xr-x. 17 root root 224 Aug 31 13:09 .

dr-xr-xr-x. 17 root root 224 Aug 31 13:09 ..

lrwxrwxrwx. 1 root root 7 Oct 9 2021 bin -> usr/bin

dr-xr-xr-x. 5 root root 4096 Aug 31 13:12 boot

drwxr-xr-x. 20 root root 3000 Oct 6 13:21 dev

drwxr-xr-x. 76 root root 8192 Oct 6 13:53 etc

drwxr-xr-x. 3 root root 21 Aug 31 13:13 home

lrwxrwxrwx. 1 root root 7 Oct 9 2021 lib -> usr/lib

lrwxrwxrwx. 1 root root 9 Oct 9 2021 lib64 -> usr/lib64

drwxr-xr-x. 2 root root 6 Oct 9 2021 media

drwxr-xr-x. 2 root root 6 Oct 9 2021 mnt

drwxr-xr-x. 2 root root 6 Oct 9 2021 opt

dr-xr-xr-x. 113 root root 0 Oct 6 13:21 proc

dr-xr-x---. 2 root root 114 Aug 31 13:14 root

drwxr-xr-x. 24 root root 700 Oct 6 13:21 run

lrwxrwxrwx. 1 root root 8 Oct 9 2021 sbin -> usr/sbin

drwxr-xr-x. 2 root root 6 Oct 9 2021 srv

dr-xr-xr-x. 13 root root 0 Oct 6 13:20 sys

drwxrwxrwt. 3 root root 85 Oct 6 13:53 tmp

drwxr-xr-x. 12 root root 144 Aug 31 13:09 usr

drwxr-xr-x. 20 root root 278 Aug 31 13:16 var

[lsauser@almalinux ~]$

Examining the results, we can notice that:

* **/bin**, **/sbin**, **/lib**, and **/lib64** folders are symbolic links to folders in **/usr**. This varies between distributions
* **/proc** and **/sys** folders are with size of zero bytes, which is strange especially compared with other empty folders like **/opt** and **/srv**

Let's continue with the exploration. By listing a special folder, we can list all **SCSI**/**SAS**/**SATA** hard drives in the system and their partitions:

[lsauser@almalinux ~]$ **ls -al /dev/sd\***

brw-rw----. 1 root disk 8, 0 Oct 6 13:20 /dev/sda

brw-rw----. 1 root disk 8, 1 Oct 6 13:20 /dev/sda1

brw-rw----. 1 root disk 8, 2 Oct 6 13:20 /dev/sda2

[lsauser@almalinux ~]$

In a similar way, we can work with another special folder in order to retrieve information about the current system configuration and running processes. For example, we can extract information about the **CPU**:

[lsauser@almalinux ~]$ **cat /proc/cpuinfo**

processor : 0

vendor\_id : GenuineIntel

...

Usually, this command is combined with **grep** to check if one or another **CPU** instruction is supported.

We can continue our journey, but let's first ask for processes in our session:

[lsauser@almalinux ~]$ **ps**

PID TTY TIME CMD

1104 pts/0 00:00:00 bash

10223 pts/0 00:00:00 ps

[lsauser@almalinux ~]$

Now, we can take the **PID** **1104**, and use the pseudo filesystem at **/proc** to get some more information:

[lsauser@almalinux ~]$ **cat /proc/1104/environ**

USER=lsauserLOGNAME=lsauserHOME=/home/lsauserPATH=/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbinSHELL=/bin/bashTERM=xterm-256colorSELINUX\_ROLE\_REQUESTED=SELINUX\_LEVEL\_REQUESTED=SELINUX\_USE\_CURRENT\_RANGE=XDG\_SESSION\_ID=1XDG\_RUNTIME\_DIR=/run/user/1000DBUS\_SESSION\_BUS\_ADDRESS=unix:path=/run/user/1000/busSSH\_CLIENT=10.0.2.2 64913 22SSH\_CONNECTION=10.0.2.2 64913 10.0.2.15 22SSH\_TTY=/dev/pts/0

[lsauser@almalinux ~]$

[lsauser@almalinux ~]$ **cat /proc/1104/cmdline**

-bash

[lsauser@almalinux ~]$

Those were just two files out of many. The full listing is:

[lsauser@almalinux ~]$ **ls -Al /proc/1104**

total 0

dr-xr-xr-x. 2 lsauser lsauser 0 Oct 6 13:58 attr

-rw-r--r--. 1 lsauser lsauser 0 Oct 6 13:58 autogroup

-r--------. 1 lsauser lsauser 0 Oct 6 13:58 auxv

-r--r--r--. 1 lsauser lsauser 0 Oct 6 13:58 cgroup

--w-------. 1 lsauser lsauser 0 Oct 6 13:58 clear\_refs

-r--r--r--. 1 lsauser lsauser 0 Oct 6 13:27 cmdline

...

Of course, you must change the **1104** to a particular **PID** in your system.

In a similar fashion, by exploring the contents of the **/proc** folder, we can see the parameters given to the **Kernel**:

[lsauser@almalinux ~]$ **cat /proc/cmdline**

BOOT\_IMAGE=(hd0,msdos1)/vmlinuz-4.18.0-372.9.1.el8.x86\_64 root=/dev/mapper/almalinux-root ro resume=/dev/mapper/almalinux-swap rd.lvm.lv=almalinux/root rd.lvm.lv=almalinux/swap

[lsauser@almalinux ~]$

There are many other interesting files in the **/proc** folder, but one definitely stands out:

[lsauser@almalinux ~]$ **ls -alh /proc/kcore**

-r--------. 1 root root 128T Oct 6 13:20 /proc/kcore

[lsauser@almalinux ~]$

The size of this file shows the absolute limit of how much memory can be allocated by the **Kernel** running on a 64-bit system.

## Part 2

One way to get information about the hard drives available in the system is by:

[lsauser@almalinux ~]$ **lsblk**

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

sda 8:0 0 16G 0 disk

├─sda1 8:1 0 1G 0 part /boot

└─sda2 8:2 0 15G 0 part

├─almalinux-root 253:0 0 13.4G 0 lvm /

└─almalinux-swap 253:1 0 1.6G 0 lvm [SWAP]

sr0 11:0 1 1024M 0 rom

[lsauser@almalinux ~]$

Now, let's turn off the **VM** and add two additional hard disks - one with size of **1GB** and another one with size of **2GB**. Then power on the **VM** again and check the situation:

[lsauser@almalinux ~]$ **lsblk**

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

sda 8:0 0 16G 0 disk

├─sda1 8:1 0 1G 0 part /boot

└─sda2 8:2 0 15G 0 part

├─almalinux-root 253:0 0 13.4G 0 lvm /

└─almalinux-swap 253:1 0 1.6G 0 lvm [SWAP]

**sdb 8:16 0 1G 0 disk**

**sdc 8:32 0 2G 0 disk**

sr0 11:0 1 1024M 0 rom

[lsauser@almalinux ~]$

Our new disks are listed and reachable via **sdb** and **sdc** identifiers.

### MBR Partitioning Scheme

Before we can use a disk, we must define one or more partitions. For this, we will use the **fdisk** utility against the **sdb** drive:

[lsauser@almalinux ~]$ **sudo fdisk /dev/sdb**

...

Command (m for help): **n**

Partition type:

p primary (0 primary, 0 extended, 4 free)

e extended

Select (default p): **p**

Partition number (1-4, default 1): **1**

First sector (2048-2097151, default 2048): **2048**

Last sector, +sectors or +size{K,M,G} (2048-2097151, default 2097151): **+200M**

Partition 1 of type Linux and of size 200 MiB is set

Command (m for help): **n**

Partition type:

p primary (1 primary, 0 extended, 3 free)

e extended

Select (default p): **p**

Partition number (2-4, default 2): **2**

First sector (411648-2097151, default 411648): **411648**

Last sector, +sectors or +size{K,M,G} (411648-2097151, default 2097151): **+300M**

Partition 2 of type Linux and of size 300 MiB is set

We defined two partitions so far. Let's do one final check before writing the changes:

Command (m for help): **p**

Disk /dev/sdb: 1073 MB, 1073741824 bytes, 2097152 sectors

Units = sectors of 1 \* 512 = 512 bytes

Sector size (logical/physical): 512 bytes / 512 bytes

I/O size (minimum/optimal): 512 bytes / 512 bytes

Disk label type: dos

Disk identifier: 0xaa128700

Device Boot Start End Blocks Id System

/dev/sdb1 2048 411647 204800 83 Linux

/dev/sdb2 411648 1026047 307200 83 Linux

Command (m for help): **w**

The partition table has been altered!

Calling ioctl() to re-read partition table.

Syncing disks.

[lsauser@almalinux ~]$

If we execute the **lslkb** command again, but this time with **/dev/sdb** as parameter, we will receive:

[lsauser@almalinux ~]$ **lsblk /dev/sdb**

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

sdb 8:16 0 1G 0 disk

├─sdb1 8:17 0 200M 0 part

└─sdb2 8:18 0 300M 0 part

[lsauser@almalinux ~]$

We can create a backup of the partition table:

[lsauser@almalinux ~]$ **sudo dd if=/dev/sdb of=512.mbr bs=512 count=1**

1+0 records in

1+0 records out

512 bytes copied, 0.000848409 s, 603 kB/s

[lsauser@almalinux ~]$

Now, that we have am archive of the partition table, we can do something destructive. We will use the same command, but this time as a result we will end up with damaged partition table:

[lsauser@almalinux ~]$ **sudo dd if=/dev/zero of=/dev/sdb bs=512 count=1**

1+0 records in

1+0 records out

512 bytes copied, 0.0535238 s, 9.6 kB/s

[lsauser@almalinux ~]$

We must pay attention to each and every argument in order to avoid damage to our system.

Now, if we ask for the partitions on the drive, the result will be similar to this:

[lsauser@almalinux ~]$ **lsblk /dev/sdb**

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

sdb 8:16 0 1G 0 disk

[lsauser@almalinux ~]$

It appears that there are no partitions at all. The reason for this is that we zeroed the first 512 bytes of the disk, which damaged the partition table.

We are prepared for situations like this, because we have a backup. Let's use it to restore the partition table:

[lsauser@almalinux ~]$ **sudo dd if=512.mbr of=/dev/sdb bs=512 count=1**

1+0 records in

1+0 records out

512 bytes copied, 0.00202351 s, 253 kB/s

[lsauser@almalinux ~]$

Now, we can check again with:

[lsauser@almalinux ~]$ **lsblk /dev/sdb**

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

sdb 8:16 0 1G 0 disk

├─sdb1 8:17 0 200M 0 part

└─sdb2 8:18 0 300M 0 part

[lsauser@almalinux ~]$

Or with:

[lsauser@almalinux ~]$ **sudo fdisk -l /dev/sdb**

Disk /dev/sdb: 1073 MB, 1073741824 bytes, 2097152 sectors

Units = sectors of 1 \* 512 = 512 bytes

Sector size (logical/physical): 512 bytes / 512 bytes

I/O size (minimum/optimal): 512 bytes / 512 bytes

Disk label type: dos

Disk identifier: 0xaa128700

Device Boot Start End Blocks Id System

/dev/sdb1 2048 411647 204800 83 Linux

/dev/sdb2 411648 1026047 307200 83 Linux

[lsauser@almalinux ~]$

It appears that everything is as expected – we have two partitions defined.

### GPT Partitioning Scheme

Now, we will try the **GPT** partitioning scheme. We will use another tool – **gdisk**. It is not installed always, so we must install it if missing:

[lsauser@almalinux ~]$ **sudo dnf install -y gdisk**

...

*Use the package manager for your distribution.*

Let's clear the partition table again:

[lsauser@almalinux ~]$ **sudo dd if=/dev/zero of=/dev/sdb bs=512 count=1**

1+0 records in

1+0 records out

512 bytes copied, 0.0544046 s, 9.4 kB/s

[lsauser@almalinux ~]$

Now, we are ready to start using the tool:

[lsauser@almalinux ~]$ **sudo gdisk /dev/sdb**

GPT fdisk (gdisk) version 1.0.3

Partition table scan:

MBR: not present

BSD: not present

APM: not present

GPT: not present

Creating new GPT entries.

User experience with both **fdisk** and **gdisk** is similar. In fact, most commands are the same. Let's define two partitions:

Command (? for help): **n**

Partition number (1-128, default 1): **1**

First sector (34-2097118, default = 2048) or {+-}size{KMGTP}: **2048**

Last sector (2048-2097118, default = 2097118) or {+-}size{KMGTP}: **+200M**

Current type is 'Linux filesystem'

Hex code or GUID (L to show codes, Enter = 8300): **8300**

Changed type of partition to 'Linux filesystem'

Command (? for help): **n**

Partition number (2-128, default 2): **2**

First sector (34-2097118, default = 411648) or {+-}size{KMGTP}: **411648**

Last sector (411648-2097118, default = 2097118) or {+-}size{KMGTP}: **+300M**

Current type is 'Linux filesystem'

Hex code or GUID (L to show codes, Enter = 8300): **8300**

Changed type of partition to 'Linux filesystem'

Command (? for help): **p**

Disk /dev/sdb: 2097152 sectors, 1024.0 MiB

Logical sector size: 512 bytes

Disk identifier (GUID): E9AD6253-4D7D-430E-8099-A64025136C6C

Partition table holds up to 128 entries

First usable sector is 34, last usable sector is 2097118

Partitions will be aligned on 2048-sector boundaries

Total free space is 1073085 sectors (524.0 MiB)

Number Start (sector) End (sector) Size Code Name

1 2048 411647 200.0 MiB 8300 Linux filesystem

2 411648 1026047 300.0 MiB 8300 Linux filesystem

Okay, it is time to save the changes to the partition table:

Command (? for help): **w**

Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING PARTITIONS!!

Do you want to proceed? (Y/N): **y**

OK; writing new GUID partition table (GPT) to /dev/sdb.

The operation has completed successfully.

[lsauser@almalinux ~]$

Let's use the usual command and check:

[lsauser@almalinux ~]$ **lsblk /dev/sdb**

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

sdb 8:16 0 1G 0 disk

├─sdb1 8:17 0 200M 0 part

└─sdb2 8:18 0 300M 0 part

[lsauser@almalinux ~]$

Let's apply our destructive measures again:

[lsauser@almalinux ~]$ **sudo dd if=/dev/zero of=/dev/sdb bs=17K count=1**

1+0 records in

1+0 records out

17408 bytes (17 kB, 17 KiB) copied, 0.00191301 s, 9.1 MB/s

[lsauser@almalinux ~]$

Indeed, we succeeded:

[lsauser@almalinux ~]$ **lsblk /dev/sdb**

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

sdb 8:16 0 1G 0 disk

[lsauser@almalinux ~]$

We know that there is a second copy of the partition table. Let's try to put it in good use:

[lsauser@almalinux ~]$ **sudo gdisk /dev/sdb**

GPT fdisk (gdisk) version 1.0.3

Caution: invalid main GPT header, but valid backup; regenerating main header

from backup!

Caution! After loading partitions, the CRC doesn't check out!

Warning! Main partition table CRC mismatch! Loaded backup partition table

instead of main partition table!

Warning! One or more CRCs don't match. You should repair the disk!

Partition table scan:

MBR: not present

BSD: not present

APM: not present

GPT: damaged

Found invalid MBR and corrupt GPT. What do you want to do? (Using the

GPT MAY permit recovery of GPT data.)

1 - Use current GPT

2 - Create blank GPT

**gdisk** detected that there is a problem with our partition table and asked us what we want to do. Let's invoke the copy of the table, then print it and save the changes:

Your answer: **1**

Command (? for help): **p**

Disk /dev/sdb: 2097152 sectors, 1024.0 MiB

Model: VBOX HARDDISK

Sector size (logical/physical): 512/512 bytes

Disk identifier (GUID): 10507A4E-82FD-4049-9FC2-D0446DE18B12

Partition table holds up to 128 entries

Main partition table begins at sector 2 and ends at sector 33

First usable sector is 34, last usable sector is 2097118

Partitions will be aligned on 2048-sector boundaries

Total free space is 1073085 sectors (524.0 MiB)

Number Start (sector) End (sector) Size Code Name

1 2048 411647 200.0 MiB 8300 Linux filesystem

2 411648 1026047 300.0 MiB 8300 Linux filesystem

Command (? for help): **w**

Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING

PARTITIONS!!

Do you want to proceed? (Y/N): **y**

OK; writing new GUID partition table (GPT) to /dev/sdb.

The operation has completed successfully.

[lsauser@almalinux ~]$

Of course, we can check again with:

[lsauser@almalinux ~]$ **lsblk /dev/sdb**

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

sdb 8:16 0 1G 0 disk

├─sdb1 8:17 0 200M 0 part

└─sdb2 8:18 0 300M 0 part

[lsauser@almalinux ~]$

All partitions seem to be there.

## Part 3

All tasks in this section rely on the fact that we did everything in the previous two sections (at least in section #2).

### Filesystems

So far, we have two partitions on our second drive. In order to be able to store data on them, we must create filesystem on each one of the partitions. For the first we will use the **ext4** filesystem, and for the second – **xfs**.

We can prepare our playground by creating a few new folders, which we will use to mount our new filesystems:

[lsauser@almalinux ~]$ **sudo mkdir -p /disks/{ext4,xfs,lvm}**

We can create a volume with **ext4** filesystem by executing the following command:

[lsauser@almalinux ~]$ **sudo mkfs.ext4 -L EXT4-VOLUME /dev/sdb1**

mke2fs 1.45.6 (20-Mar-2020)

Creating filesystem with 204800 1k blocks and 51200 inodes

Filesystem UUID: fec03d08-5bf6-45f3-b7d1-568155c98ab0

Superblock backups stored on blocks:

8193, 24577, 40961, 57345, 73729

Allocating group tables: done

Writing inode tables: done

Creating journal (4096 blocks): done

Writing superblocks and filesystem accounting information: done

[lsauser@almalinux ~]$

At this stage, we can mount temporary (just for the session) our new volume with:

[lsauser@almalinux ~]$ **sudo mount /dev/sdb1 /disks/ext4**

Now, let's create **xfs** filesystem on the second partition on the same drive:

[lsauser@almalinux ~]$ **sudo mkfs.xfs -L VOLUME-XFS /dev/sdb2**

meta-data=/dev/sdb2 isize=512 agcount=4, agsize=19200 blks

= sectsz=512 attr=2, projid32bit=1

= crc=1 finobt=1, sparse=1, rmapbt=0

= reflink=1 bigtime=0 inobtcount=0

data = bsize=4096 blocks=76800, imaxpct=25

= sunit=0 swidth=0 blks

naming =version 2 bsize=4096 ascii-ci=0, ftype=1

log =internal log bsize=4096 blocks=1368, version=2

= sectsz=512 sunit=0 blks, lazy-count=1

realtime =none extsz=4096 blocks=0, rtextents=0

[lsauser@almalinux ~]$

*Please note that the* ***XFS*** *utilities may not be installed by default on* ***Debian****. In order to handle this, we must execute:*

*lsauser@debian:~$* ***sudo apt-get update && sudo apt-get upgrade -y***

*...*

*lsauser@debian:~$* ***sudo apt-get install xfsprogs***

*...*

*Then we are good to go and can re-execute the previous command for filesystem creation.*

Mount it in the same way, as we did with the first:

[lsauser@almalinux ~]$ **sudo mount /dev/sdb2 /disks/xfs**

Of course, we can force our new filesystems to be mounted automatically each time the system is booted. For this purpose, we must get the unique identifier of both volumes:

[lsauser@almalinux ~]$ **sudo blkid /dev/sdb\***

/dev/sdb: PTUUID="10507a4e-82fd-4049-9fc2-d0446de18b12" PTTYPE="gpt"

/dev/sdb1: LABEL="EXT4-VOLUME" **UUID="fec03d08-5bf6-45f3-b7d1-568155c98ab0"** BLOCK\_SIZE="1024" TYPE="ext4" PARTLABEL="Linux filesystem" PARTUUID="72be33f8-1044-4fc0-ab96-d7a5dfde6fc4"

/dev/sdb2: LABEL="VOLUME-XFS" **UUID="a6bb16de-da66-467c-a907-62b49593dbf8"** BLOCK\_SIZE="512" TYPE="xfs" PARTLABEL="Linux filesystem" PARTUUID="8ac8bed8-03fe-4aa8-b2ee-a8a43905a433"

[lsauser@almalinux ~]$

We are interested in the **UUID** fields.

As we know **/etc/fstab** file contains instructions for volumes mounted during boot. First, we must make a backup copy and open the original file for editing:

[lsauser@almalinux ~]$ **sudo cp /etc/fstab /etc/fstab.bak**

[lsauser@almalinux ~]$ **sudo vi /etc/fstab**

Then we must append the following two lines at the end:

**UUID=fec03d08-5bf6-45f3-b7d1-568155c98ab0 /disks/ext4 ext4 defaults 0 0**

**UUID=a6bb16de-da66-467c-a907-62b49593dbf8 /disks/xfs xfs defaults 0 0**

Before we reboot the system, we can test that everything with our **/etc/fstab** file is okay. For this we can execute the following two commands:

[lsauser@almalinux ~]$ **sudo mount -a**

[lsauser@almalinux ~]$ **mount | grep sdb**

/dev/sdb1 on /disks/ext4 type ext4 (rw,relatime,seclabel)

/dev/sdb2 on /disks/xfs type xfs (rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,noquota)

[lsauser@almalinux ~]$

If we see output like the shown above, we are safe to proceed with system reboot.

There is an alternative way of getting back information about the mounted volumes. We can explore the contents of a special file at **/proc/mounts**:

[lsauser@almalinux ~]$ **cat /proc/mounts | grep sdb**

/dev/sdb1 /disks/ext4 ext4 rw,seclabel,relatime 0 0

/dev/sdb2 /disks/xfs xfs rw,seclabel,relatime,attr2,inode64,logbufs=8,logbsize=32k,noquota 0 0

[lsauser@almalinux ~]$

### LVM

In addition to the standard way of handling hard disks and volumes we saw so far, **Linux** offers additional abstraction layer – **Logical Volume Management** or **LVM** for short. One of the many options it offers is to mix a set of disks and/or partitions, include them in a pool, and then create one or more logical volumes on top of it.

Currently, we have some unpartitioned space on our second drive. In addition, we have third drive with no partitions on it.

Let's first create one more partition on our second drive:

[lsauser@almalinux ~]$ **sudo gdisk /dev/sdb**

...

Command (? for help): **n**

Partition number (3-128, default 3): **3**

First sector (34-2097118, default = 1026048) or {+-}size{KMGTP}: **1026048**

Last sector (1026048-2097118, default = 2097118) or {+-}size{KMGTP}: **+300M**

Current type is 'Linux filesystem'

Hex code or GUID (L to show codes, Enter = 8300): **8300**

Changed type of partition to 'Linux filesystem'

Command (? for help): **w**

...

Do you want to proceed? (Y/N): **y**

...

[lsauser@almalinux ~]$

*If you see a warning that the new partition table will be available after reboot, then you may execute the following:*

*[lsauser@almalinux ~]$* ***sudo partprobe***

Now, let's create one **400MB** partition using the **MBR** partitioning scheme on the third drive. This time, we will change the type of the partition to **LVM**:

[lsauser@almalinux ~]$ **sudo fdisk /dev/sdc**

...

Command (m for help): **n**

Partition type:

p primary (0 primary, 0 extended, 4 free)

e extended

Select (default p): **p**

Partition number (1-4, default 1): **1**

First sector (2048-4194303, default 2048): **2048**

Last sector, +sectors or +size{K,M,G} (2048-4194303, default 4194303): **+400M**

Partition 1 of type Linux and of size 400 MiB is set

Command (m for help): **t**

Selected partition 1

Hex code (type L to list all codes): **8e**

Changed type of partition 'Linux' to 'Linux LVM'

Command (m for help): **w**

...

[lsauser@almalinux ~]$

Depending on the changes we made, we may need to inform the **Kernel** that something changed with the partitions:

[lsauser@almalinux ~]$ **sudo partprobe -s**

/dev/sda: msdos partitions 1 2

/dev/sdb: gpt partitions 1 2 3

/dev/sdc: msdos partitions 1

[lsauser@almalinux ~]$

Now, let's check the situation:

[lsauser@almalinux ~]$ **lsblk /dev/sd[bc]**

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

sdb 8:16 0 1G 0 disk

├─sdb1 8:17 0 200M 0 part /disks/ext4

├─sdb2 8:18 0 300M 0 part /disks/xfs

└─sdb3 8:19 0 300M 0 part

sdc 8:32 0 2G 0 disk

└─sdc1 8:33 0 400M 0 part

[lsauser@almalinux ~]$

We are ready to move on with the **LVM**. First, we must initialize the physical volumes:

[lsauser@almalinux ~]$ **sudo pvcreate /dev/sdb3**

Physical volume "/dev/sdb3" successfully created.

[lsauser@almalinux ~]$ **sudo pvcreate /dev/sdc1**

Physical volume "/dev/sdc1" successfully created.

[lsauser@almalinux ~]$ **sudo pvscan**

PV /dev/sda2 VG almalinux lvm2 [<15.00 GiB / 0 free]

PV /dev/sdb3 lvm2 [300.00 MiB]

PV /dev/sdc1 lvm2 [400.00 MiB]

Total: 3 [<15.68 GiB] / in use: 1 [<15.00 GiB] / in no VG: 2 [700.00 MiB]

[lsauser@almalinux ~]$

Let's create one new volume group named **vg1**:

[lsauser@almalinux ~]$ **sudo vgcreate vg1 /dev/sdb3 /dev/sdc1**

Volume group "vg1" successfully created

[lsauser@almalinux ~]$ **sudo vgscan**

Found volume group "vg1" using metadata type lvm2

Found volume group "almalinux" using metadata type lvm2

[lsauser@almalinux ~]$

Last step is to create a logical volume named **lv1**:

[lsauser@almalinux ~]$ **sudo lvcreate -n lv1 -L 600m vg1**

Logical volume "lv1" created.

[lsauser@almalinux ~]$ **sudo lvscan**

ACTIVE '/dev/vg1/lv1' [600.00 MiB] inherit

ACTIVE '/dev/almalinux/swap' [1.60 GiB] inherit

ACTIVE '/dev/almalinux/root' [13.39 GiB] inherit

[lsauser@almalinux ~]$

As with the regular partitions, we must create a filesystem in order to use the logical volume:

[lsauser@almalinux ~]$ **sudo mkfs.xfs -L LVM-XFS /dev/vg1/lv1**

meta-data=/dev/vg1/lv1 isize=512 agcount=4, agsize=38400 blks

= sectsz=512 attr=2, projid32bit=1

= crc=1 finobt=1, sparse=1, rmapbt=0

= reflink=1 bigtime=0 inobtcount=0

data = bsize=4096 blocks=153600, imaxpct=25

= sunit=0 swidth=0 blks

naming =version 2 bsize=4096 ascii-ci=0, ftype=1

log =internal log bsize=4096 blocks=1368, version=2

= sectsz=512 sunit=0 blks, lazy-count=1

realtime =none extsz=4096 blocks=0, rtextents=0

[lsauser@almalinux ~]$

Let's mount the volume:

[lsauser@almalinux ~]$ **sudo mount /dev/vg1/lv1 /disks/lvm**

Okay, so far, we have a **600MB** volume, based on **LVM**. In addition, we have almost **100MB** unallocated space. So, let's extend first the logical volume:

[lsauser@almalinux ~]$ **sudo lvextend -l +100%FREE /dev/vg1/lv1**

Size of logical volume vg1/lv1 changed from 600.00 MiB (150 extents) to 692.00 MiB (173 extents).

Logical volume vg1/lv1 successfully resized.

[lsauser@almalinux ~]$

Now, we are ready to extend the filesystem as well:

[lsauser@almalinux ~]$ **sudo xfs\_growfs /disks/lvm**

meta-data=/dev/mapper/vg1-lv1 isize=512 agcount=4, agsize=38400 blks

= sectsz=512 attr=2, projid32bit=1

= crc=1 finobt=1, sparse=1, rmapbt=0

= reflink=1 bigtime=0 inobtcount=0

data = bsize=4096 blocks=153600, imaxpct=25

= sunit=0 swidth=0 blks

naming =version 2 bsize=4096 ascii-ci=0, ftype=1

log =internal log bsize=4096 blocks=1368, version=2

= sectsz=512 sunit=0 blks, lazy-count=1

realtime =none extsz=4096 blocks=0, rtextents=0

data blocks changed from 153600 to 177152

[lsauser@almalinux ~]$

Finally, we can execute **lsblk** to check current status:

[lsauser@almalinux ~]$ **lsblk**

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

sda 8:0 0 16G 0 disk

├─sda1 8:1 0 1G 0 part /boot

└─sda2 8:2 0 15G 0 part

├─almalinux-root 253:0 0 13.4G 0 lvm /

└─almalinux-swap 253:1 0 1.6G 0 lvm [SWAP]

sdb 8:16 0 1G 0 disk

├─sdb1 8:17 0 200M 0 part /disks/ext4

├─sdb2 8:18 0 300M 0 part /disks/xfs

└─sdb3 8:19 0 300M 0 part

└─vg1-lv1 253:2 0 692M 0 lvm /disks/lvm

sdc 8:32 0 2G 0 disk

└─sdc1 8:33 0 400M 0 part

└─vg1-lv1 253:2 0 692M 0 lvm /disks/lvm

sr0 11:0 1 1024M 0 rom

[lsauser@almalinux ~]$

Should we want our new filesystem to be automatically mounted on system boot, we must add the appropriate line to our **/etc/fstab** file.