Preparing storage

Storage is a fundamental component of working with computers and we will dig a little bit into a few basics right now.

Before you can store files, you need a filesystem. A filesystem is a mechanism for organizing data. You get different types of filesystems, each with their own special properties. The most common filesystem is called FAT which originated from Microsoft and is generally understood by all Operating Systems.

Let's have a look at a flash drive which you could purchase at a store like Best Buy. If you were to store files on it, you'd need a filesystem and to simplify matters, the flash drive was pre-allocated the FAT filesystem so that you can start using it immediately. Without a filesystem you would never be able to store data on a storage device.

What if you didn't have a filesystem? Well you'd have to allocate one to your partition and that is a completely different element.

Partitions are logical divisions of your storage. In order for you to have a filesystem, you need a partition which is capable of receiving a filesystem. Not all partitions can be used for storage though.

Each disk has enough entries for 4 partitions in the partition table. The partition table itself is an element of the Master Boot Record which is responsible for loading an operating system off a disk.

partition 1	partition 2	partition 3	partition 4

Once you've exhausted the entries in the main partition table, then you cannot create any more partitions.

You do however get 3 different types of partitions. Each partition type has special attributes and use cases. When planning your partitioning layout, you need to be very well aware of these attributes.

Here's a quick breakdown of the different partition types and their purposes.

Primary partitions:

These are the most overrated (and therefore the most overused) partition types.

Advantages	Disadvantages
Can be allocated a filesystem.	Occupies an entry in the main partition table.
Can be used to boot an operating system .	Limited to 4 primary partitions per disk provided that an extended partition is used (in which case you are limited to 3 primary partitions).

Extended partitions:

Every disk making use of the standard partitioning scheme should use this.

Advantages	Disadvantages
Can host 56 logical partitions (they do not occupy an entry in the main partition table).	Cannot be allocated a filesystem.
	Cannot be used to load an operating system.
	Limited to 1 extended partition per partition table.

Logical partitions:

Advantages	Disadvantages
Can create 56 logical partitions	Cannot be used to load an operating system.
Can be allocated a file system	

So far our order of operations are:

1) Acquire storage
2) Partition storage
3) Allocate filesystem to partition

But how do you access your filesystem then if you've done this?

Unlike Microsoft Windows, we do not allocate drive letters to our partitions.

In Windows it would work like this:

Disk 1:

- Has 1 primary partition.

Disk 2:

- Has 3 partitions - 1 primary, 1 extended with 1 logical partition inside it.

Disk 3:

- Has 5 partitions - 2 primary, 1 extended with 2 logical partitions inside it.

Disk 1, partition 1 (Primary)	C: drive
Disk 2, partition 1 (Primary)	D: drive
Disk 2, partition 2 (Extended)	Extended partitions don't get allocated drive letters
Disk 2, partition 3 (Logical)	E: drive
Disk 3, partition 1 (Primary)	F: drive
Disk 3, partition 2 (Primary)	G: drive
Disk 3, partition 3 (Extended)	Extended partitions don't get allocated drive letters
Disk 3, partition 4 (Logical)	H: drive
Disk 3, partition 5 (Logical)	I: Drive

This is complicated further when you add a CDROM Drive and then later connect other disks to it.

The same situation would work like this in Linux:

Linux uses device files to represent currently connected hardware like disks. These device files reside below the /dev directory and do not occupy any additional space.

If you are using SCSI, SATA or USB connected storage, the device files will have names beginning with sd.

The first disk would be called sda, second sdb, third sdc, etc.

Partitions are given numbers with the partitions in the main partition table given numbers 1 to 4 and logical partitions given numbers starting at 5.

So in Linux, the above scenario would result in the following:

Disk 1:
- Has 1 primary partition.

Disk 2:
- Has 3 partitions - 1 primary, 1 extended with 1 logical partition inside it.

Disk 3:
- Has 5 partitions - 2 primary, 1 extended with 2 logical partitions inside it.

Disk 1, partition 1 (Primary)	/dev/sda1
Disk 2, partition 1 (Primary)	/dev/sdb1
Disk 2, partition 2 (Extended)	/dev/sdb2
Disk 2, partition 3 (Logical)	/dev/sdb5
Disk 3, partition 1 (Primary)	/dev/sdc1
Disk 3, partition 2 (Primary)	/dev/sdc2
Disk 3, partition 3 (Extended)	/dev/sdc3
Disk 3, partition 4 (Logical)	/dev/sdc6
Disk 3, partition 5 (Logical)	/dev/sdc6

This is much better and easier to follow.

Without the additional complication which Microsoft has when it comes to CDROM drives, we have a predefined way in which CDROM drives are recognized.

They also get device files and are allocated as follows:

First CDROM drive = /dev/scd0
Second CDROM drive = /dev/scd1

You can use various tools to prepare your disk for storage to create partitions.

fdisk

This is the most basic of tools which you could use to create partitions and is available on all enterprise Linux operating systems.

Use fdisk with the -cu options to insure that DOS compatibility mode is disabled (c) and that the sizes are displayed in sectors (must better to determine sizes) instead of by cylinder (u).

Examples:

```
# fdisk -cu
```

This enters the fdisk tool on the only disk installed on this system.

```
# fdisk -cul
```

This displays the partition table of all disks connected to the system. To conduct the transaction against a specific disk, refer to the disk in the last argument as fdisk -cul /dev/sdb

```
# fdisk -cu /dev/sdb
```

This enters the fdisk tool for partitioning the disk represented by the device file /dev/sdb

Once in the fdisk tool you may use the m command to get additional help.

```
# fdisk -cu /dev/sdb

Command (m for help): n
Command action
e   extended
p   primary partition (1-4)
e
Partition number (1-4): 4
First sector (2048-2097151, default 2048): <hit enter>
Using default value 2048
Last sector, +sectors or +size{K,M,G} (2048-2097151, default 2097151): <hit enter>
Using default value 2097151

Command (m for help):
```

!!! NOTE !!!
Selecting <enter> when prompted for a value for the first sector and for the last sector uses the next available sectors as specified in brackets. By doing this we're selecting all the available space to allocate to the extended partition which is GOOD practise. Typically we do not create extended partitions of a fixed size.

In the above example (the commands issued are in bold and highlighted in red) we have executed n create a new partition of the type e for extended and allocated it slot number 4 in the partition table. After which, the prompt queried which sector I'd like to allocate as the beginning of the partition in which case we just hit enter and then for the last sector we hit enter once more to use the default values which are in brackets. What this essentially has done is dedicate all space on the disk /dev/sdb to create an extended partition.

!!! NOTE !!!
No changes have been committed to the disk at this stage. This only happens when we save the changes using the w command.

Now we will create a logical partition of a fixed size of 100M inside of the extended partition and is continued from the block above:

```
Command (m for help): n
Command action
l   logical (5 or over)
p   primary partition (1-4)
l
First sector (4096-2097151, default 4096): <hit enter>
Using default value 4096
Last sector, +sectors or +size{K,M,G} (4096-2097151, default 2097151): +100M
```

So here we are creating a new partition using the n command of the type l for logical partition. When we're prompted for a value for the first sector we merely hit enter to select the next available sector (as opposed to specifying a specific sector). With the value for the last sector we specify a value here as we want to create a fixed size of 100M using the format +numberunit (as opposed to using all available space inside of the extended partition to allocate to this logical partition) and the command +100M achieves this.

The block below is a continuation from the one above:

```
Command (m for help): w
The partition table has been altered!
```

```
Calling ioctl() to re-read partition table.
Syncing disks.
```

We are able to repeat this process of creating logical partitions as long as we have enough sectors. Once we are done we can commit the changes using the w command

!!! NOTE !!!
If you conduct a partition transaction against the disk that contains your currently running kernel then you will have to reboot in order to generate the device files representing those newly created partitions. You could bypass this process by using the command partx -a /dev/<device file of disk> but the reboot command is what is officially supported.

cfdisk

This is an interface driving variant of disk using the ncurses library.

parted

This command driven partitioning tool supports the IBM partitioning framework as well as GPT based partitions. GPT based partitions are supported by many operating systems and use Globally Unique Identifiers (GUID's) to identify partitions and allow us to overcome the limitations imposed by the IBM partitioning framework of 64 partitions. GPT uses the 64 bit disk pointers, which allow for a maximum disk partition size of 9.4 Zetabytes, or 9.4 billion TeraBytes. With GPT, we support 128 primary partitions.

Now why are we mentioning all of this GPT stuff? That's because parted supports GPT partitions

Examples:

```
# parted /dev/sdb print
```

This command displays the partition table of the disk identified by the device file /dev/sdb.

```
(parted) mklabel gpt
```

Here we specify that we're not using the standard partitioning type (awkwardly referred to as msdos), but instead will be using GPT.

```
(parted) mkpart foo 0 1G
```

This command creates a partition with the name foo starting at the first available sector and ending at 1 Gigabyte.

```
(parted) mkpart bar 1G 4G
```

Now we're just creating another partition starting at the first available sector after the previously created partition and ending at 4 Gigabytes.

Just so that we're clear that we're making use of GPT, we're going to list the partition table of /dev/sdb using the command fdisk -cul /dev/sdb

```
# fdisk -cul /dev/sdb

WARNING: GPT (GUID Partition Table) detected on '/dev/sdb'! The util fdisk doesn't support GPT. Use GNU Parted.
---truncated---
```

Now that we have our partitions created, we're able to allocate filesystems. Remember, you cannot allocate a filesystem to an extended partition. So this means that we'll be only allocating filesystems to Primary and Logical partitions for now.

For this purpose, we use the tool mkfs.

At this stage we haven't created file systems on either partitions created with the utilities fdisk or parted so we have no mechanism to organize files. To create a filesystem on a partition we make use of the mkfs command. With Linux we always have choice, and we support a number of filesystems but for the demonstration we're going to focus on a great performing, journaling file system called ext4.

```
# mkfs -t ext4 /dev/sdb1
```

This command creates a filesystem of the type ext4 using the -t option on the partition represented by the device file /dev/sdb1

Once we have a filesystem allocated to the partitions we wish to use then we can access it by mounting it. Think of the command mount as being a verb (in other words, a doing word) which means 'to make available'. So the mount command 'makes a filesystem available' But where?

This is where another term comes in called a mountpoint which you should see as a noun (something you can see, feel and touch). A mountpoint is essentially a directory where a filesystem is accessed.

So given that we want to access our newly created ext4 filesystem (which was created on a partition represented by the device file /dev/sdb1) at the mountpoint /data.

We should create the directory /data using the command:

```
# mkdir /data
```

Then we do a non-persistent (it won't survive a reboot) mount:

```
# mount /dev/sdb1 /data
```

You're able to verify that the mount of the filesystem was successful by simply typing:

```
# mount

---truncated---
/dev/sdb1 on /data type ext4 (rw)
```

!!! NOTE !!!
The opposite of mount is umount - NOT unmount.

Persistency in Linux is defined in configuration files and file responsible for the persistent mounting of filesystems is /etc/fstab

This is a whitespace (1 of more space characters) separated file consisting of 6 fields.

```
|The device file/label/UUID which contains the filesystem you want to mount
|
|
|               |The directory where the filesystem gets mounted to
|               |
|               |
|               |                 |The filesystem type
|               |                 |
|               |                 |
|               |                 |                 |Options used for mounting
|               |                 |                 |
```

```
|              |               |              |
|              |               |              |              |Do we backup this filesystem using
|              |               |              |              |
|              |               |              |              |
|              |               |              |              |              |The order in which we run fsck
|              |               |              |              |              |
|              |               |              |              |              |
|              |               |              |              |              |
|              |               |              |              |              |
/dev/sdb1    /dev/data     ext4        defaults          0              0
```

!!! TIP !!!

To see some of the possible options which could be used in column 4 see man mount

The dump program isn't really used, so 0 is an acceptable value for column 5

3 Possible values exist for the 6th column:

0 = do not automatically run fsck on suspect filesystems
1 = check the root filesystem
2 = check this filesystem after the root filesystem

!!! NOTE !!!

fsck is automatically called for filesystems which were not unmounted cleanly.

Using UUID's

If you're using local disks then it is acceptable to refer to the device file in /etc/fstab, however if you're using iSCSI it is recommended to use the UUID of a filesystem.

To determine the UUID of the filesystem on /dev/sdb1 use the command:

```
# blkid /dev/sdb1

/dev/sdb1: UUID="b16241e1-df59-4750-95c8-fdb8a74a2f05" TYPE="ext4"
```

The UUID is a randomly generated alphanumeric string which is more accurately identifies a filesystem than a device file.

The line in /etc/fstab using the UUID would look as follows:

```
UUID="b16241e1-df59-4750-95c8-fdb8a74a2f05"    /dev/data    ext4    defaults    0    0
```

Using Labels instead of UUID's

Labels fail to accurately reference a filesystem uniquely as they may be duplicated and as such should not be used.

Setting a label can be done during filesystem creation time:

```
# mkfs -L MYLABEL /dev/sdb1
```

Or post filesystem creation time:

```
# tune2fs -L MYLABEL /dev/sdb1
```

Should you really wish to mount a filesystem by its label in /etc/fstab then devise an entry similar to this:

```
LABEL=DATA    /dev/data    ext4          defaults      0      0
```

Lab activity

1) Choose all applicable answers: Which of the following can be formatted with a filesystem?

A. Primary partitions
B. Extended partitions
C. Logical partitions

Answer:

2) Answer True or False: You can only create 4 extended partitions in the main partition table.

Answer:

3) Choose all applicable answers: Which filesystems does Enterprise Linux understand?

A. fat
B. msdos
C. ntfs

D. ext2
E. ext3
F. ext4

Using a flash drive or other form of storage, clear the partition table and create the following new partitions:

1 Extended partition
2 Logical partitions inside of the extended partition

Choose your own sizes based on what you have.

Now allocate the ext2 filesystem to the first logical partition and the ext4 partition to the second one. Make sure that these partitions are automatically mounted on startup to the directories /elp/flash1 and /elp/flash2 respectively.