COMP 350
Introduction to DevOps

Lecture 5
Storage Media & File Systems

Hakan Ayral

Storage Media

- In previous lectures we’ve looked at manipulating data at the **file level**.
- In this lecture, we will consider data at the **device level**.
- Linux has amazing capabilities for handling storage devices, whether physical storage such as hard disks, network storage, or virtual storage devices like RAID (redundant array of independent disks) and LVM (logical volume manager).

Storage Media

We will look at the following commands:

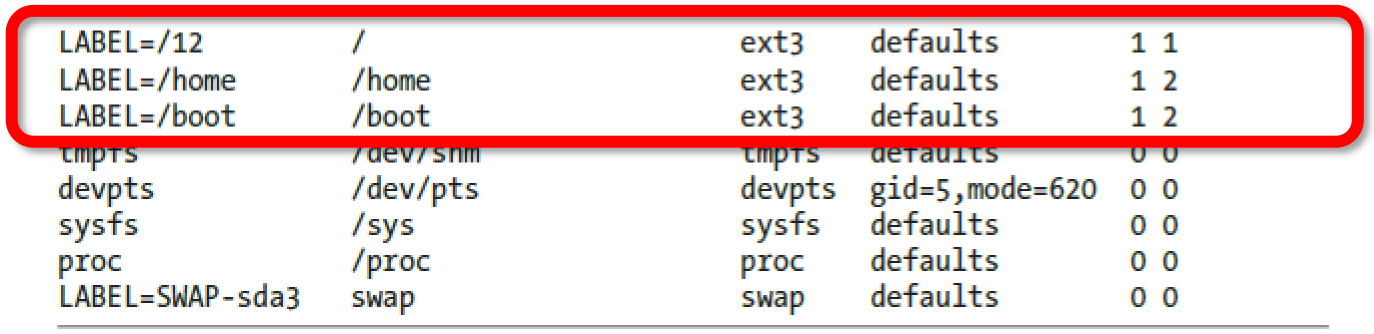
- **mount**—Mount a filesystem.
- **umount**—Unmount a filesystem.
- **fdisk**—Partition table manipulator.
- **fsck**—Check and repair a filesystem.
- **fdformat**—Format a floppy disk.
- **mkfs**—Create a filesystem.
- **dd**—Write block-oriented data directly to a device.
- **genisoimage** (**mkisofs**)—Create an ISO 9660 image file.
- **wodim** (cdrecord)—Write data to optical storage media.
- **md5sum**—Calculate an MD5 checksum.

Mounting and Unmounting Storage Devices

- Recent advances in the Linux desktop have made storage device management extremely easy for desktop users.
  - For the most part, we attach a device to our system and it just works.
  - Back in the old days (say, 2004), this stuff had to be done manually.
- On non-desktop systems (i.e., servers) this is still a largely manual procedure, because servers often have extreme storage needs and complex configuration requirements.
- The first step in managing a storage device is attaching the device to the filesystem tree.
- This process, called **mounting**, allows the device to participate with the operating system.
- Unix-like operating systems, like Linux, maintain a single filesystem tree with devices attached at various points.
  - This contrasts with other operating systems such as MS-DOS and Windows that maintain separate trees for each device (for example C:\, D:\, etc.).

Mounting and Unmounting Storage Devices

- A file named **/etc/fstab** lists the devices (typically hard disk partitions) that are to be mounted at boot time.
- Here is an example **/etc/fstab** file from a Fedora 7 system:
  - Most of the filesystems listed in this example file are virtual and are not applicable to our discussion.
  - For our purposes, the interesting ones are the first three.



```
LABEL=/12         /                ext3     defaults          1 1
LABEL=/home       /home            ext3     defaults          1 2
LABEL=/boot       /boot            ext3     defaults          1 2
tmpfs             /dev/shm         tmpfs    defaults          0 0
devpts            /dev/pts         devpts   gid=5,mode=620    0 0
sysfs             /sys             sysfs    defaults          0 0
proc              /proc            proc     defaults          0 0
LABEL=SWAP-sda3   swap             swap     defaults          0 0
```

Mounting and Unmounting Storage Devices

```
LABEL=/12          /                          ext3    defaults        1 1
LABEL=/home        /home                      ext3    defaults        1 2
LABEL=/boot        /boot                      ext3    defaults        1 2
```

- These are the hard disk partitions.
- Each line of the file consists of six fields, as shown in Table 15-1.

Table 15-1: */etc/fstab* Fields

Field	Contents	Description
1	Device	Traditionally, this field contains the actual name of a device file associated with the physical device, such as */dev/hda1* (the first partition of the master device on the first IDE channel). But with today's computers, which have many devices that are hot pluggable (like USB drives), many modern Linux distributions associate a device with a text label instead. This label (which is added to the storage medium when it is formatted) is read by the operating system when the device is attached to the system. That way, no matter which device file is assigned to the actual physical device, it can still be correctly identified.
2	Mount point	The directory where the device is attached to the filesystem tree
3	Filesystem type	Linux allows many filesystem types to be mounted. Most native Linux filesystems are ext3, but many others are supported, such as FAT16 (`msdos`), FAT32 (`vfat`), NTFS (`ntfs`), CD-ROM (`iso9660`), etc.
4	Options	Filesystems can be mounted with various options. It is possible, for example, to mount filesystems as read only or to prevent any programs from being executed from them (a useful security feature for removable media).
5	Frequency	A single number that specifies if and when a filesystem is to be backed up with the `dump` command
6	Order	A single number that specifies in what order filesystems should be checked with the `fsck` command

Viewing a List of Mounted Filesystems

- The mount command is used to mount filesystems.
  - Entering the command without arguments will display a list of the filesystems currently mounted:
- The format of the listing is

  device on *mount_point* type *filesystem_type* (*options*)

- For example, the first line shows that device */dev/sda2* is mounted as the root filesystem, is of type ext3, and is both readable and writable (the option rw).
- This listing also has two interesting entries at the bottom:
  - The next-to-last entry shows a 2-gigabyte SD memory card in a card reader mounted at */media/disk*,
  - and the last entry is a network drive mounted at */misc/musicbox*.

```
[me@linuxbox ~]$ mount
/dev/sda2 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/sda5 on /home type ext3 (rw)
/dev/sda1 on /boot type ext3 (rw)
tmpfs on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
fusectl on /sys/fs/fuse/connections type fusectl (rw)
/dev/sdd1 on /media/disk type vfat (rw,nosuid,nodev,noatime,
uhelper=hal,uid=500,utf8,shortname=lower)
twin4:/musicbox on /misc/musicbox type nfs4 (rw,addr=192.168.1.4)
```

Mounting a File System

- For our first experiment, we will work with a CD-ROM.
  - First, let’s look at a system before a CD-ROM is inserted:

```
[me@linuxbox ~]$ mount
/dev/mapper/VolGroup00-LogVol00 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/hda1 on /boot type ext3 (rw)
tmpfs on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
```

- This listing is from a CentOS 5 system that is using LVM to create its root filesystem.
- Like many modern Linux distributions, this system will attempt to automatically mount the CD-ROM after insertion.
- After we insert the disc, we see the following:
  - At the end of the listing, we see that the CD-ROM (which is device */dev/hdc* on this system) has been mounted on */media/live-1.0.10-8* and is type iso9660 (a CDROM).

```
[me@linuxbox ~]$ mount
/dev/mapper/VolGroup00-LogVol00 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/hda1 on /boot type ext3 (rw)
tmpfs on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
/dev/hdc on /media/live-1.0.10-8 type iso9660 (ro,noexec,nosuid,nodev,uid=500)
```

Unmounting a File System

- Now that we have the device name of the CD-ROM drive, let’s unmount the disc and remount it at another location in the filesystem tree.
- To do this, we become the superuser and unmount the disc with the **umount** command:

```
[me@linuxbox ~]$ su -
Password:
[root@linuxbox ~]# umount /dev/hdc
```

- The next step is to create a new **mount point** for the disc.
    - A ***mount point*** is simply a directory somewhere on the filesystem tree. Nothing special about it.

- It doesn’t even have to be an empty directory, though if you mount a device on a non-empty directory, you will not be able to see the directory’s previous contents until you unmount the device.

```
[root@linuxbox ~]# mkdir /mnt/cdrom
```

Mounting a File System

- Finally, we mount the CD-ROM at the new mount point.
  - The **-t** option is used to specify the filesystem type:

```
[root@linuxbox ~]# mount -t iso9660 /dev/hdc /mnt/cdrom
```

- Afterward, we can examine the contents of the CD-ROM via the new mount point:

```
[root@linuxbox ~]# cd /mnt/cdrom
[root@linuxbox cdrom]# ls
```

- Notice what happens when we try to unmount the CD-ROM:

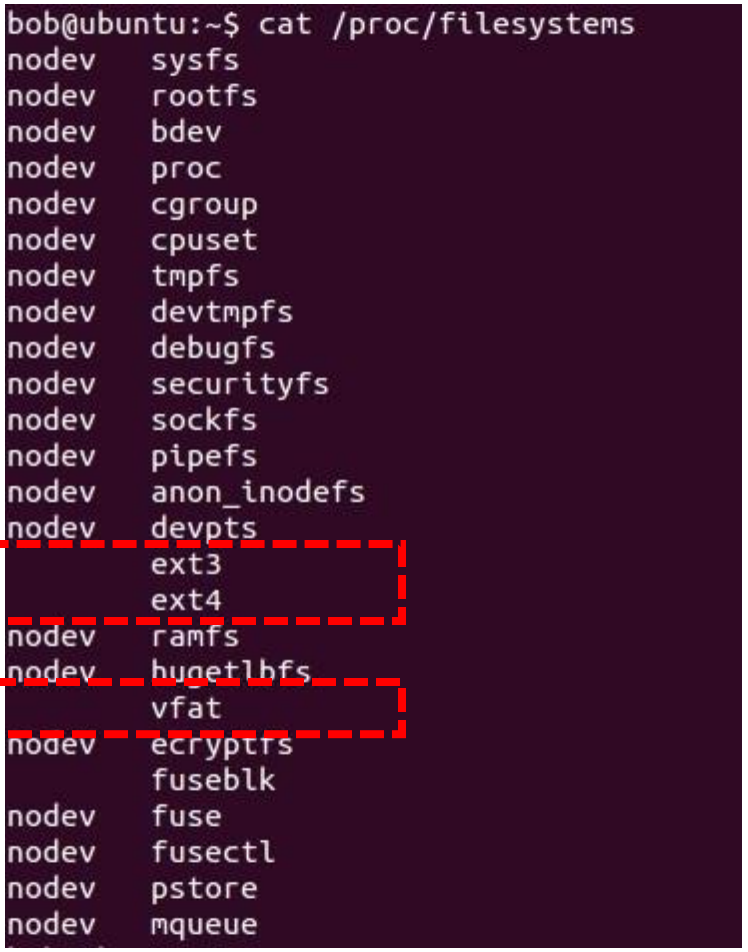
```
[root@linuxbox cdrom]# umount /dev/hdc
umount: /mnt/cdrom: device is busy
```

- Why is this? We cannot unmount a device if the device is being used by someone or some process.
  - In this case, we changed our working directory to the mount point for the CD-ROM, which causes the device to be busy.
  - We can easily remedy the issue by changing the working directory to something other than the mount point:
  - Now the device unmounts successfully.

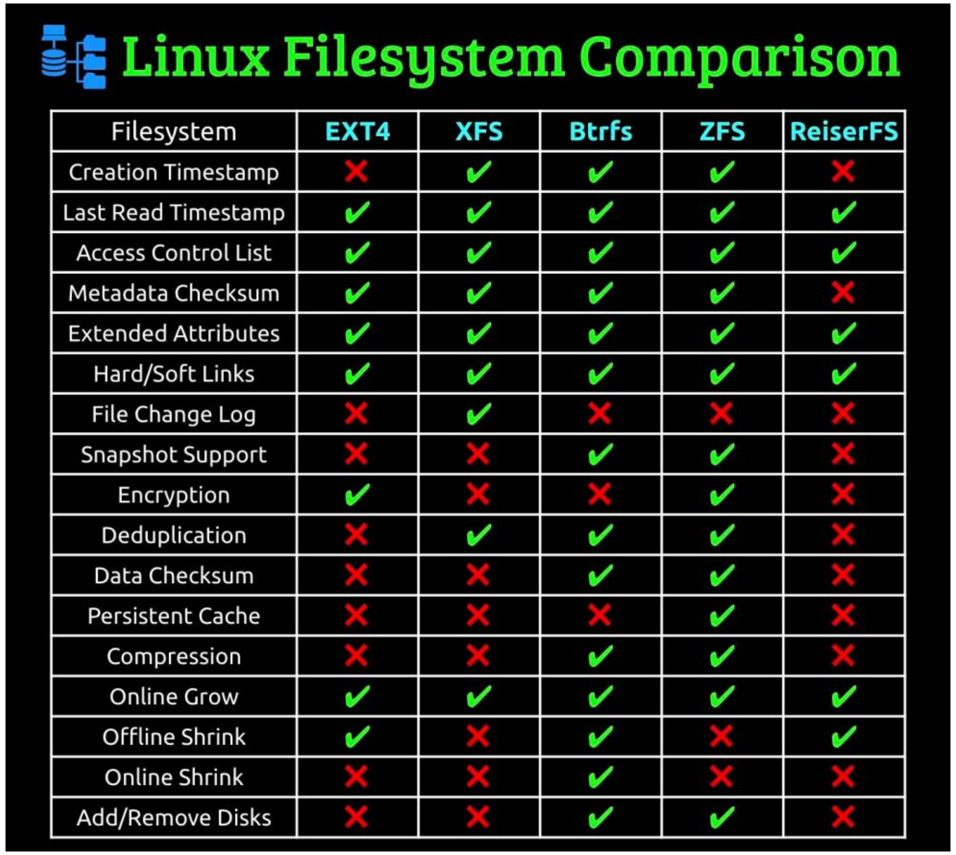
```
[root@linuxbox cdrom]# cd
[root@linuxbox ~]# umount /dev/hdc
```

Listing the supported filesystems

- The virtual file **/proc/filesystems** shows the file system drivers currently loaded by the kernel.
- Even though it is not a real file, you can look at its contents as if it is a regular text file.
- You can install new file system drivers to support other file systems too.



```
bob@ubuntu:~$ cat /proc/filesystems
nodev   sysfs
nodev   rootfs
nodev   bdev
nodev   proc
nodev   cgroup
nodev   cpuset
nodev   tmpfs
nodev   devtmpfs
nodev   debugfs
nodev   securityfs
nodev   sockfs
nodev   pipefs
nodev   anon_inodefs
nodev   devpts
        ext3
        ext4
nodev   ramfs
nodev   hugetlbfs
        vfat
nodev   ecryptfs
        fuseblk
nodev   fuse
nodev   fusectl
nodev   pstore
nodev   mqueue
```



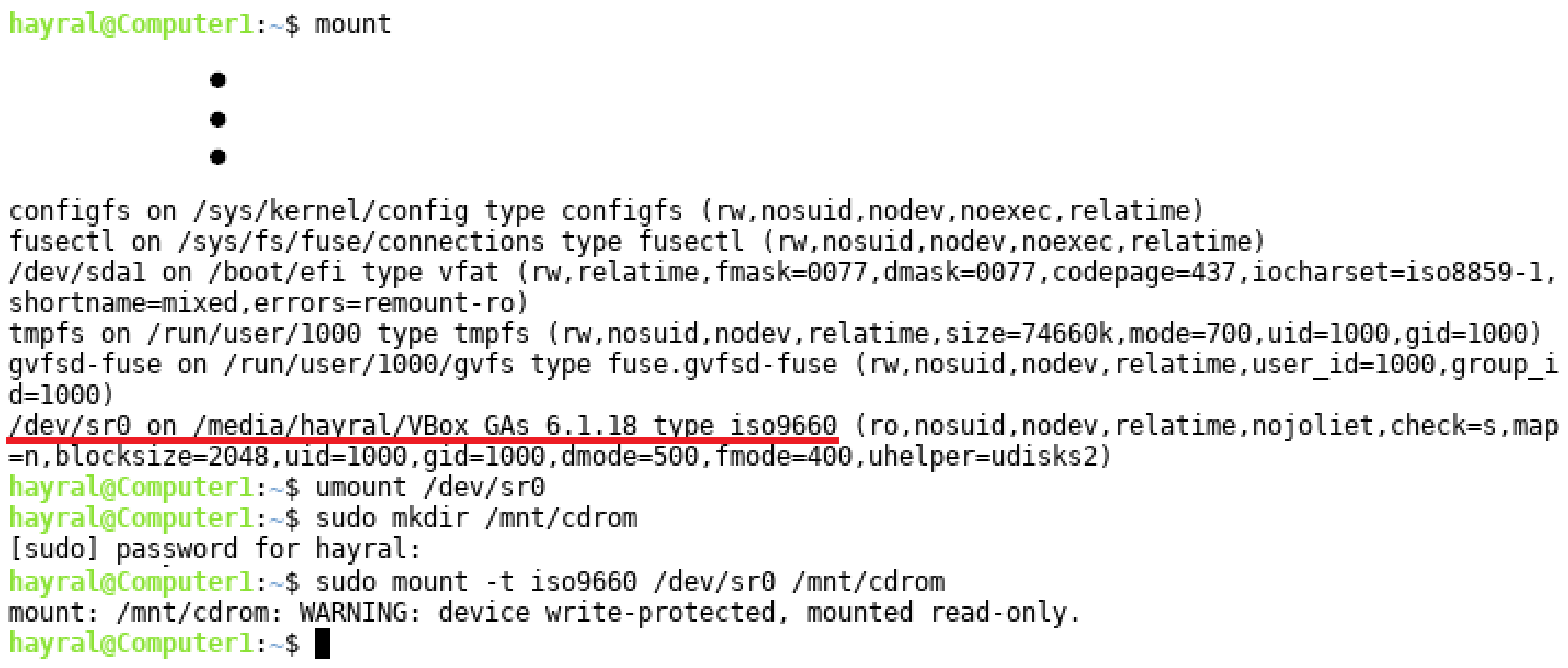
Linux Filesystem Comparison

Filesystem	EXT4	XFS	Btrfs	ZFS	ReiserFS
Creation Timestamp	✗	✓	✓	✓	✗
Last Read Timestamp	✓	✓	✓	✓	✓
Access Control List	✓	✓	✓	✓	✓
Metadata Checksum	✓	✓	✓	✓	✗
Extended Attributes	✓	✓	✓	✓	✓
Hard/Soft Links	✓	✓	✓	✓	✓
File Change Log	✗	✓	✗	✗	✗
Snapshot Support	✗	✗	✓	✓	✗
Encryption	✓	✗	✗	✓	✗
Deduplication	✗	✓	✓	✓	✗
Data Checksum	✗	✗	✓	✓	✗
Persistent Cache	✗	✗	✗	✓	✗
Compression	✗	✗	✓	✓	✗
Online Grow	✓	✓	✓	✓	✓
Offline Shrink	✓	✗	✓	✗	✓
Online Shrink	✗	✗	✓	✗	✗
Add/Remove Disks	✗	✗	✓	✓	✗

also see: https://en.wikipedia.org/wiki/Comparison_of_file_systems

Credit: Linux file system comparison table created by @dan_nanni on Instagram

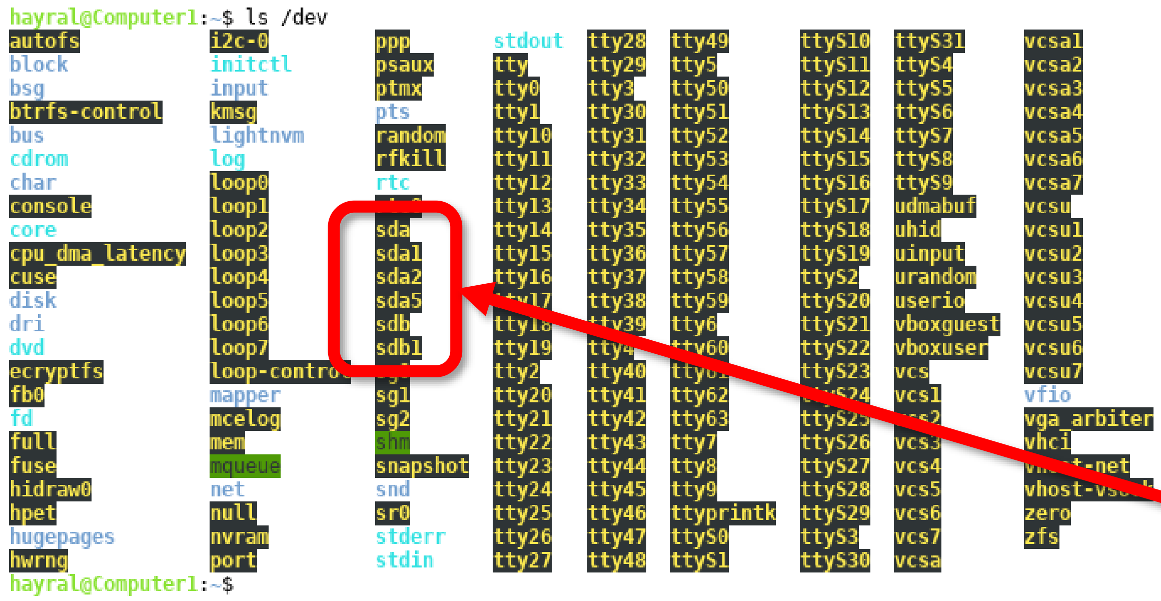
Unmounting/Remounting Example



```
hayral@Computer1:~$ mount
            •
            •
            •
configfs on /sys/kernel/config type configfs (rw,nosuid,nodev,noexec,relatime)
fusectl on /sys/fs/fuse/connections type fusectl (rw,nosuid,nodev,noexec,relatime)
/dev/sda1 on /boot/efi type vfat (rw,relatime,fmask=0077,dmask=0077,codepage=437,iocharset=iso8859-1,
shortname=mixed,errors=remount-ro)
tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,relatime,size=74660k,mode=700,uid=1000,gid=1000)
gvfsd-fuse on /run/user/1000/gvfs type fuse.gvfsd-fuse (rw,nosuid,nodev,relatime,user_id=1000,group_i
d=1000)
/dev/sr0 on /media/hayral/VBox_GAs_6.1.18 type iso9660 (ro,nosuid,nodev,relatime,nojoliet,check=s,map
=n,blocksize=2048,uid=1000,gid=1000,dmode=500,fmode=400,uhelper=udisks2)
hayral@Computer1:~$ umount /dev/sr0
hayral@Computer1:~$ sudo mkdir /mnt/cdrom
[sudo] password for hayral:
hayral@Computer1:~$ sudo mount -t iso9660 /dev/sr0 /mnt/cdrom
mount: /mnt/cdrom: WARNING: device write-protected, mounted read-only.
hayral@Computer1:~$ █
```

Determining Device Names

- First, let's look at how the system names devices. If we list the contents of the /dev directory (where all devices live), we can see that there are lots of devices:



```
hayral@Computer1:~$ ls /dev
autofs           i2c-0         ppp       stdout  tty28  tty49      ttyS10  ttyS31     vcsa1
block            initctl       psaux     tty     tty29  tty5       ttyS11  ttyS4      vcsa2
bsg              input         ptmx      tty0    tty3   tty50      ttyS12  ttyS5      vcsa3
btrfs-control    kmsg          pts       tty1    tty30  tty51      ttyS13  ttyS6      vcsa4
bus              lightnvm      random    tty10   tty31  tty52      ttyS14  ttyS7      vcsa5
cdrom            log           rfkill    tty11   tty32  tty53      ttyS15  ttyS8      vcsa6
char             loop0         rtc       tty12   tty33  tty54      ttyS16  ttyS9      vcsa7
console          loop1         rtc0      tty13   tty34  tty55      ttyS17  udmabuf    vcsu
core             loop2         sda       tty14   tty35  tty56      ttyS18  uhid       vcsu1
cpu_dma_latency  loop3         sda1      tty15   tty36  tty57      ttyS19  uinput     vcsu2
cuse             loop4         sda2      tty16   tty37  tty58      ttyS2   urandom    vcsu3
disk             loop5         sda5      tty17   tty38  tty59      ttyS20  userio     vcsu4
dri              loop6         sdb       tty18   tty39  tty6       ttyS21  vboxguest  vcsu5
dvd              loop7         sdb1      tty19   tty4   tty60      ttyS22  vboxuser   vcsu6
ecryptfs         loop-control  sg0       tty2    tty40  tty61      ttyS23  vcs        vcsu7
fb0              mapper        sg1       tty20   tty41  tty62      ttyS24  vcs1       vfio
fd               mcelog        sg2       tty21   tty42  tty63      ttyS25  vcs2       vga_arbiter
full             mem           shm       tty22   tty43  tty7       ttyS26  vcs3       vhci
fuse             mqueue        snapshot  tty23   tty44  tty8       ttyS27  vcs4       vhost-net
hidraw0          net           snd       tty24   tty45  tty9       ttyS28  vcs5       vhost-vsock
hpet             null          sr0       tty25   tty46  ttyprintk  ttyS29  vcs6       zero
hugepages        nvram         stderr    tty26   tty47  ttyS0      ttyS3   vcs7       zfs
hwrng            port          stdin     tty27   tty48  ttyS1      ttyS30  vcsa
hayral@Computer1:~$
```

- The contents of this listing reveal some patterns of device naming. Table 15-2 lists a few:

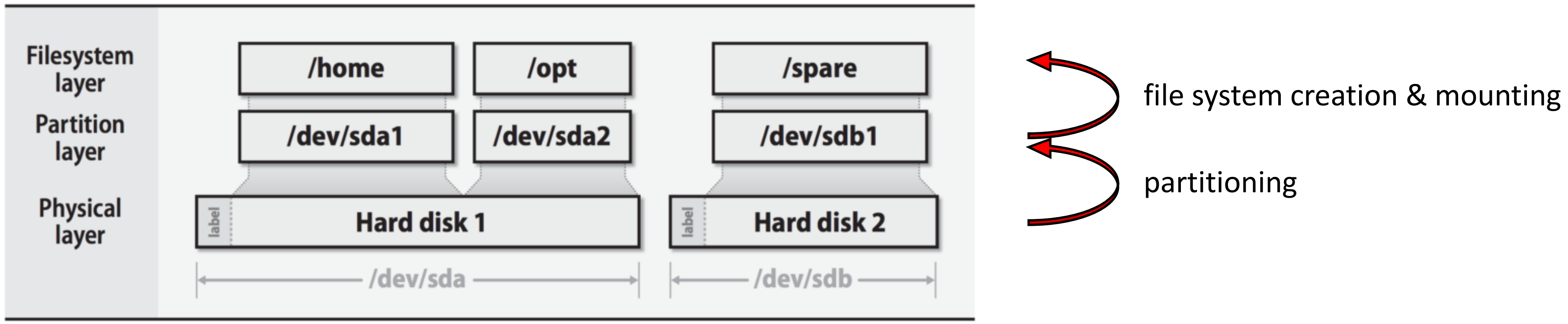
Table 15-2: Linux Storage Device Names

Pattern	Device
*/dev/fd**	Floppy disk drives
*/dev/hd**	IDE (PATA) disks on older systems. Typical motherboards contain two IDE connectors, or *channels*, each with a cable with two attachment points for drives. The first drive on the cable is called the *master* device and the second is called the *slave* device. The device names are ordered such that */dev/hda* refers to the master device on the first channel, */dev/hdb* is the slave device on the first channel; */dev/hdc*, the master device on the second channel, and so on. A trailing digit indicates the partition number on the device. For example, */dev/hda1* refers to the first partition on the first hard drive on the system while */dev/hda* refers to the entire drive.
*/dev/lp**	Printers
*/dev/sd**	SCSI disks. On recent Linux systems, the kernel treats all disk-like devices (including PATA/SATA hard disks, flash drives, and USB mass storage devices such as portable music players and digital cameras) as SCSI disks. The rest of the naming system is similar to the older */dev/hd** naming scheme described above.
*/dev/sr**	Optical drives (CD/DVD readers and burners)

Storage Device Names in Linux

- In Linux storage devices like mechanical hard drives and solid state drives are represented as a file named as **/dev/sda**, **/dev/sdb**, **/dev/sdc**, … in order. These are also called block storage devices (unlike the character devices like modems, tty, keyboard and screen)
- Then each partition on each of these devices are further represented as files with the same name suffixed with a number for partition no. (i.e. **/dev/sda1**, **/dev/sda2**, …, **/dev/sdb1**, **/dev/sdb2**, …)
  - These partitions are also block storage devices.
  - You can create a file system on and mount the whole device or a partition of it in the very same way. (Creating a partition, even if there is only one, is the more common way)
  - You can mount a block device to anywhere you want on your root file system.

Traditional data disk partitioning scheme (Linux device names)



Creating New Filesystems

- Let’s say that we want to reformat the flash drive with a Linux native filesystem, rather than the FAT32 system it has now.
- This involves two steps:
  1. (optional) create a new partition layout if the existing one is not to your liking
  2. creating a new, empty filesystem on the drive.

Warning:

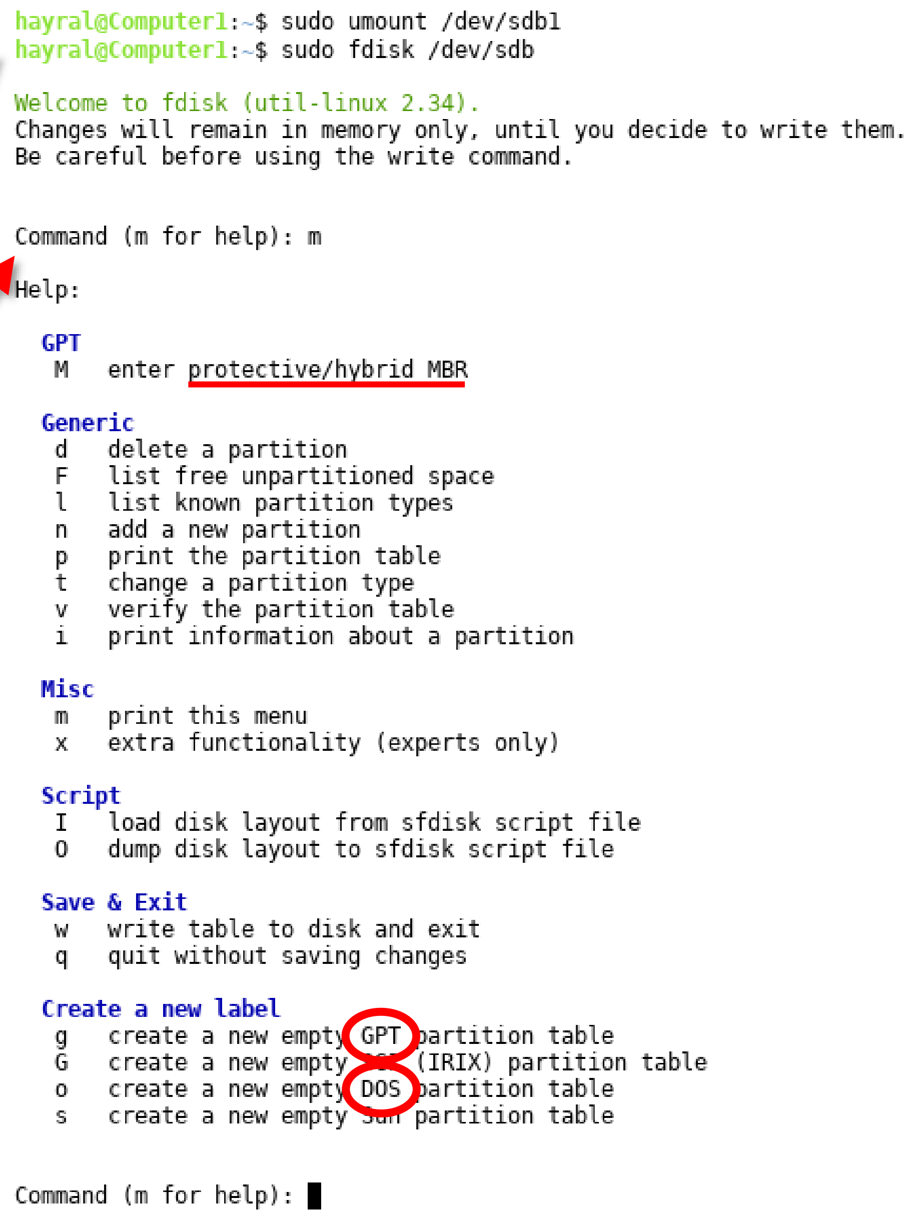
Creating a new file system on a device or partition

is equivalent to formatting it,

your data will be unretrievably lost!

Manipulating Partitions with fdisk

- The **fdisk** program allows us to interact directly with disk-like devices (such as hard disk drives and flash drives) at a very low level.
- With this tool we can edit, delete, and create partitions on the device.
- To work with our drive, we must first **unmount** it (if needed) and then invoke the **fdisk** program as follows:
    - You can also check available commands by pressing **m** and ENTER.
- GPT is the most commonly used partition table type
    - DOS is an older partition type but still used on older systems.



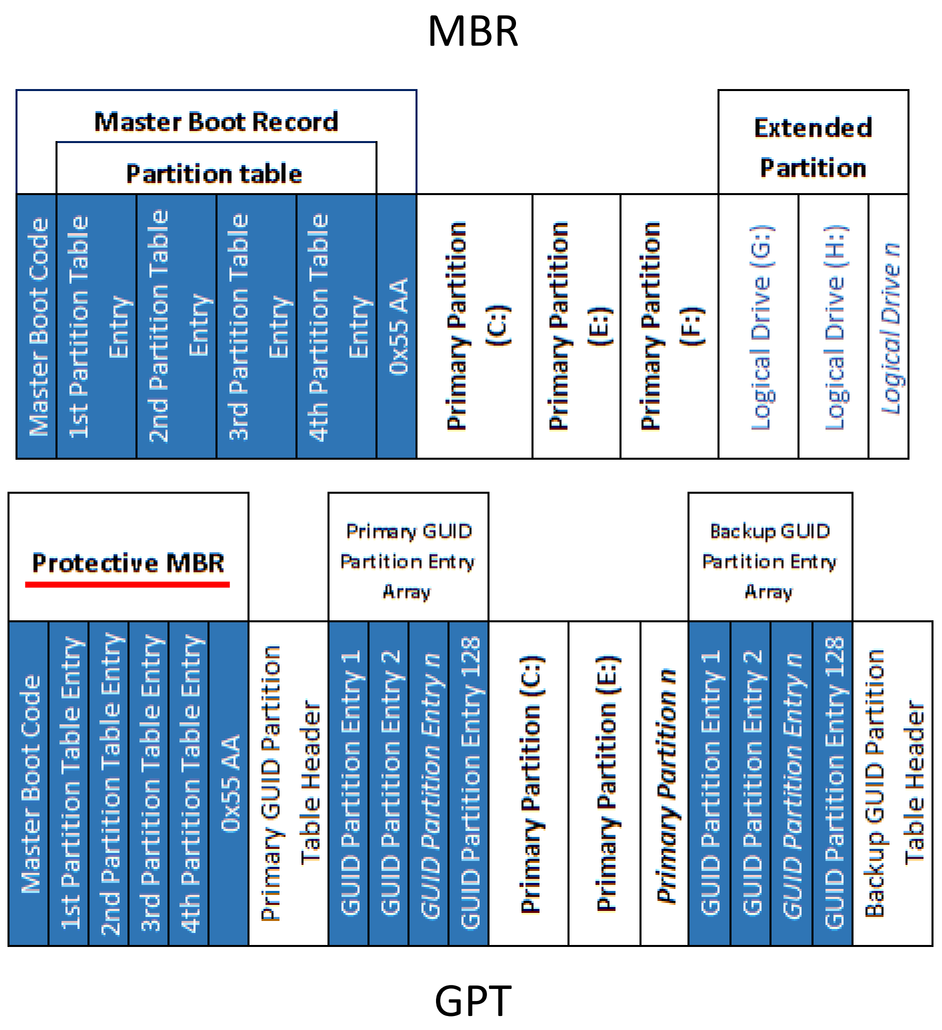
```
hayral@Computer1:~$ sudo umount /dev/sdb1
hayral@Computer1:~$ sudo fdisk /dev/sdb

Welcome to fdisk (util-linux 2.34).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.


Command (m for help): m

Help:

  GPT
   M   enter protective/hybrid MBR

  Generic
   d   delete a partition
   F   list free unpartitioned space
   l   list known partition types
   n   add a new partition
   p   print the partition table
   t   change a partition type
   v   verify the partition table
   i   print information about a partition

  Misc
   m   print this menu
   x   extra functionality (experts only)

  Script
   I   load disk layout from sfdisk script file
   O   dump disk layout to sfdisk script file

  Save & Exit
   w   write table to disk and exit
   q   quit without saving changes

  Create a new label
   g   create a new empty GPT partition table
   G   create a new empty SGI (IRIX) partition table
   o   create a new empty DOS partition table
   s   create a new empty Sun partition table


Command (m for help):
```

GPT vs MBR Partition Table

	MBR (Master Boot Record)	GPT (GUID Partition Table)
Maximum Supported Disk Size	2TB	9.4ZB (1ZB=10^9 TB)
Maximum Number of Partitions	4 primary (or 3 primary + many logical)	128 primary
Firmware Interface	BIOS	UEFI
Released	Early 1980s	Specified in 1990, wide use in 2000s

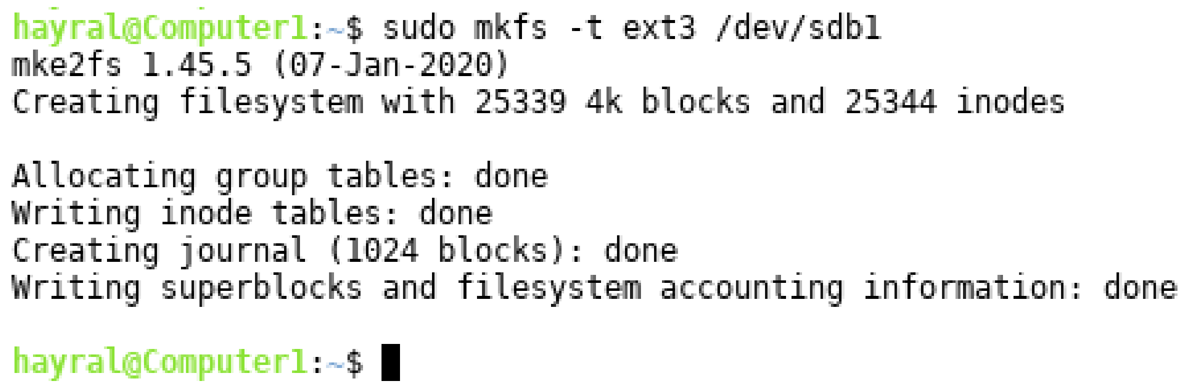
Demo Here:

1. Create partition table
2. Create partition
3. Create file system
4. Mount



Creating a New Filesystem with mkfs

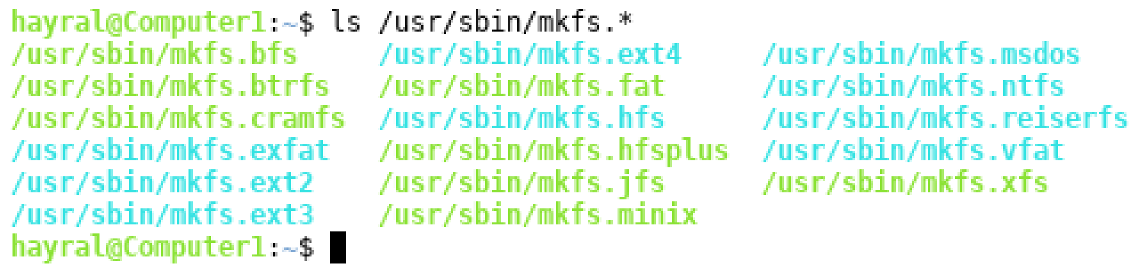
- With our partition editing done, it’s time to create a new filesystem on our drive.
- To do this, we will use **mkfs** (short for make filesystem), which can create filesystems in a variety of formats.
- To create an **ext3** filesystem on the device, we use the **-t** option to specify the **ext3** system type, followed by the name of the device containing the partition we wish to format:



```
hayral@Computer1:~$ sudo mkfs -t ext3 /dev/sdb1
mke2fs 1.45.5 (07-Jan-2020)
Creating filesystem with 25339 4k blocks and 25344 inodes

Allocating group tables: done
Writing inode tables: done
Creating journal (1024 blocks): done
Writing superblocks and filesystem accounting information: done

hayral@Computer1:~$ 
```

mkfs

- As a matter of fact **mkfs** itself does not contain any code for file system creation.
- Based on the file system type you specify it searches the system for the actual binary which will create the file system.
- You can check which file systems **mkfs** can create for you.



```
hayral@Computer1:~$ ls /usr/sbin/mkfs.*
/usr/sbin/mkfs.bfs     /usr/sbin/mkfs.ext4     /usr/sbin/mkfs.msdos
/usr/sbin/mkfs.btrfs   /usr/sbin/mkfs.fat      /usr/sbin/mkfs.ntfs
/usr/sbin/mkfs.cramfs  /usr/sbin/mkfs.hfs      /usr/sbin/mkfs.reiserfs
/usr/sbin/mkfs.exfat   /usr/sbin/mkfs.hfsplus  /usr/sbin/mkfs.vfat
/usr/sbin/mkfs.ext2    /usr/sbin/mkfs.jfs      /usr/sbin/mkfs.xfs
/usr/sbin/mkfs.ext3    /usr/sbin/mkfs.minix
hayral@Computer1:~$ 
```

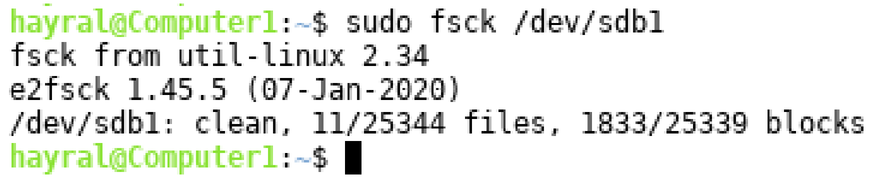
- In order to use a new file system, you must also have the necessary file system driver on your kernel.
- When you install a new file system to linux, it installs the necessary kernel driver and also creates the necessary mkfs.*** file system creation tool.

Testing and Repairing Filesystems

- In our earlier discussion of the ***/etc/fstab*** file, we saw some mysterious digits at the end of each line.
- Each time the system boots, it routinely checks the integrity of the filesystems before mounting them.
- This is done by the **fsck** program (short for *filesystem check*).
- The last number in each ***fstab*** entry specifies the order in which the devices are to be checked.
  - The root filesystem is checked first, followed by the *home* and *boot* filesystems.
  - Devices with a zero as the last digit are not routinely checked.
- In addition to checking the integrity of filesystems, **fsck** can also repair corrupt filesystems with varying degrees of success, depending on the amount of damage.
- On Unix-like filesystems, recovered portions of files are placed in the ***lost+found*** directory, located in the root of each filesystem.

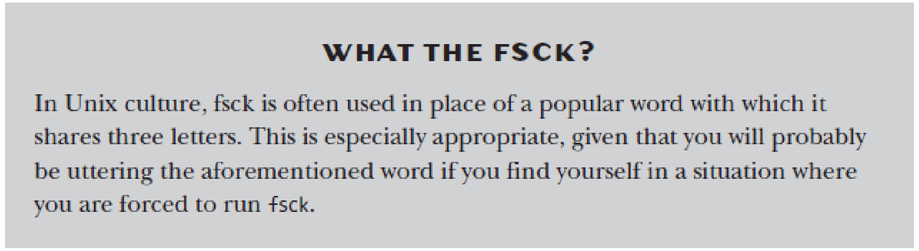
Testing and Repairing Filesystems

- To check our drive (which should be unmounted first), we can do the following:



```
hayral@Computer1:~$ sudo fsck /dev/sdb1
fsck from util-linux 2.34
e2fsck 1.45.5 (07-Jan-2020)
/dev/sdb1: clean, 11/25344 files, 1833/25339 blocks
hayral@Computer1:~$ 
```

- The command works directly on data structures stored on disk, which are internal and specific to the particular file system in use - so an fsck command tailored to the file system is required.
- **The file system is normally checked while unmounted, mounted read-only, or with the system in a special maintenance mode. (see next slide)**
- Full copy-on-write file systems such as ZFS and Btrfs are designed to avoid most causes of corruption and have no traditional "fsck" repair tool.
  - Both have a "scrub" utility which examines and repairs any problems; in the background and on a mounted file system.



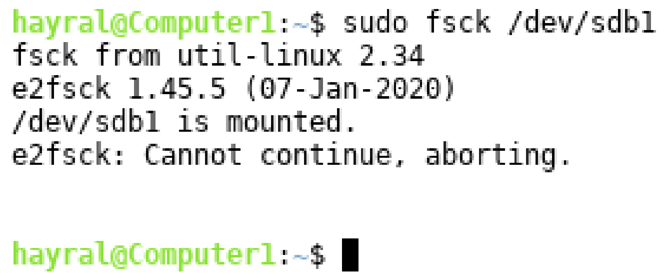
WHAT THE FSCK?

In Unix culture, fsck is often used in place of a popular word with which it shares three letters. This is especially appropriate, given that you will probably be uttering the aforementioned word if you find yourself in a situation where you are forced to run fsck.

fsck

Do not run fsck on a live or mounted file system !

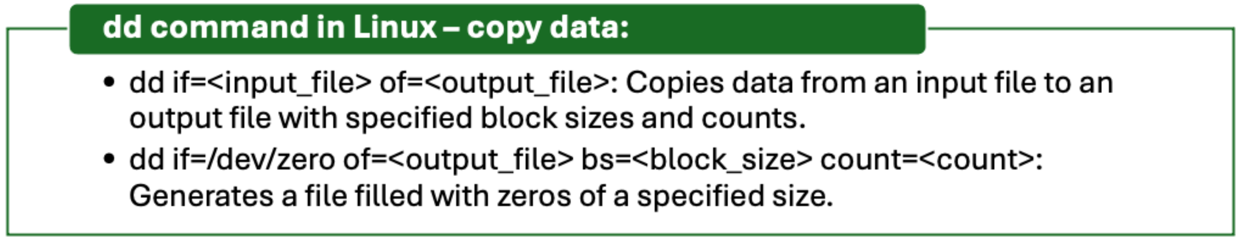
- Running **fsck** on a mounted filesystem can usually result in disk and/or data corruption.
- This behavior is different from Windows, where you can run disk check and repair file system errors on a live system.
- Luckily linux doesn't let you do fsck a mounted file system anyway:



```
hayral@Computer1:~$ sudo fsck /dev/sdb1
fsck from util-linux 2.34
e2fsck 1.45.5 (07-Jan-2020)
/dev/sdb1 is mounted.
e2fsck: Cannot continue, aborting.


hayral@Computer1:~$ 
```

Moving Data Directly to and from Devices

- Data on our computers as being organized into files, but it is also possible to think of the data in “raw” form.
- If we look at a disk drive, for example, we see that it consists of a large number of “blocks” of data that the operating system sees as directories and files.
- If we could treat a disk drive as simply a large collection of data blocks, we could perform useful tasks, such as cloning devices.

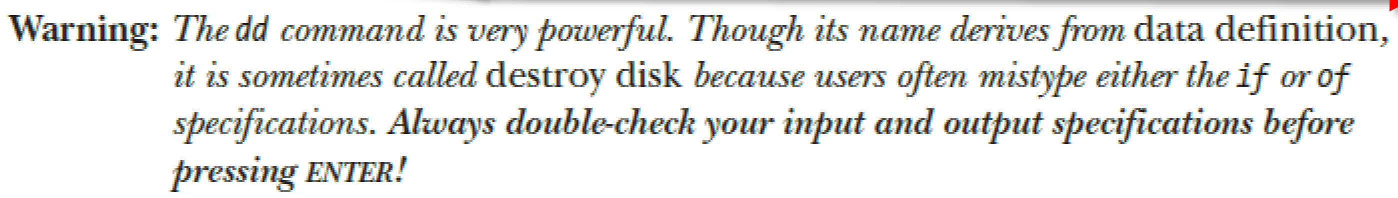


dd command in Linux – copy data:

- dd if=<input_file> of=<output_file>: Copies data from an input file to an output file with specified block sizes and counts.
- dd if=/dev/zero of=<output_file> bs=<block_size> count=<count>: Generates a file filled with zeros of a specified size.

```
dd if=input_file of=output_file [bs=block_size [count=blocks]]
```

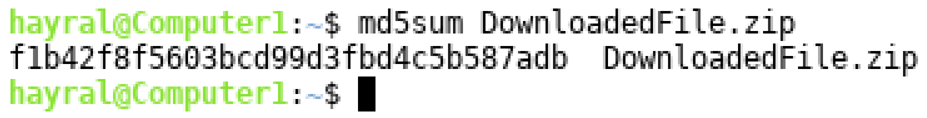
These can be real files or block devices



Warning: *The dd command is very powerful. Though its name derives from* data definition, *it is sometimes called* destroy disk *because users often mistype either the* `if` *or* `of` *specifications. Always double-check your input and output specifications before pressing* ENTER*!*

md5sum

- It's often useful to verify the integrity of a file that we have downloaded.
- In some cases, the distributor of a file will also supply a checksum value or a checksum file.
- A checksum is the result of a mathematical calculation resulting in a number that represents the content of the target file. (think like a finger print of the file)
- If the contents of the file change by even one bit, the resulting checksum will be totally different.
- The most common method of checksum generation uses the **md5sum** program.
- When you use md5sum, it produces a unique hexadecimal number:



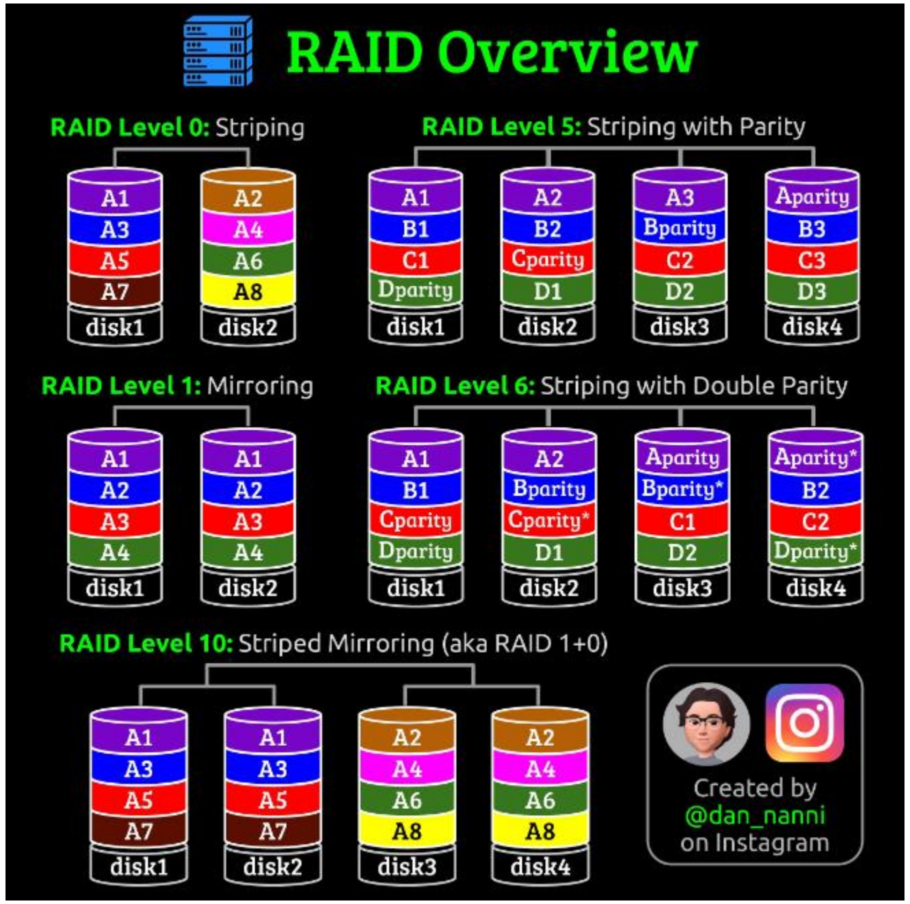
```
hayral@Computer1:~$ md5sum DownloadedFile.zip
f1b42f8f5603bcd99d3fbd4c5b587adb  DownloadedFile.zip
hayral@Computer1:~$ 
```

<u>Note:</u> Today few people still use md5, most commonly used hash tool today is SHA256; for example Ubuntu stopped providing md5 values of their downloadable files for verification anymore.

RAID

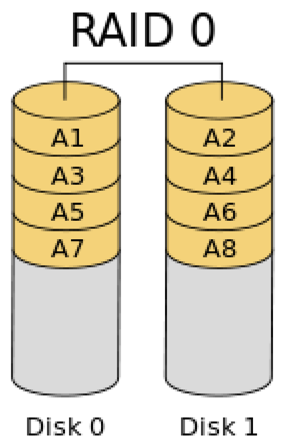
("Redundant Array of Inexpensive Disks" or "Redundant Array of Independent Disks")

- **RAID** is a data storage virtualization technology that **combines multiple physical disk drive components into one or more logical units** for the purposes of data redundancy, performance improvement, or both.
- This was in contrast to the previous concept of highly reliable mainframe disk drives referred to as "single large expensive disk" (**SLED**).
- Data is distributed across the drives in one of several ways, referred to as RAID levels, depending on the required level of redundancy and performance.
- The different schemes, or data distribution layouts, are named by the word "RAID" followed by a number, for example RAID **0** or RAID **1**.
- Each scheme, or RAID level, provides a different balance among the key goals: **reliability**, **availability**, **performance**, and **capacity**.
- RAID levels greater than RAID 0 provide **protection against unrecoverable sector read errors, as well as against failures of whole physical drives**.



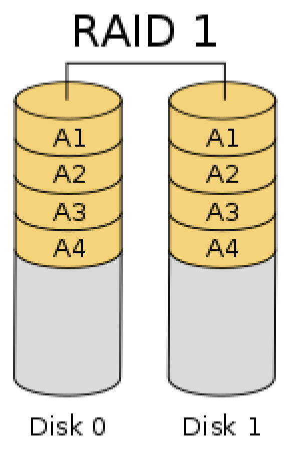
RAID 0 - Striping

- RAID 0 (also known as a stripe set or striped volume) splits ("stripes") data evenly across two or more disks, without parity information, redundancy, or fault tolerance.
- Since **RAID 0 provides no fault tolerance or redundancy**, the failure of one drive will cause the entire array to fail; as a result of having data striped across all disks, the failure will result in total data loss.
    - This configuration is typically implemented having speed as the intended goal.
- RAID 0 is normally used to increase performance, although it can also be used as a way to create a large logical volume out of two or more physical disks.
- A RAID 0 setup can be created with disks of differing sizes, but the storage space added to the array by each disk is limited to the size of the smallest disk.
    - For example, if a 120 GB disk is striped together with a 320 GB disk, the size of the array will be 120 GB × 2 = 240 GB. (However, some RAID implementations allow the remaining 200 GB to be used for other purposes.)
- The diagram in this section shows how the data is distributed into Ax stripes on two disks, with A1:A2 as the first stripe, A3:A4 as the second one, etc.
- Once the stripe size is defined during the creation of a RAID 0 array, it needs to be maintained at all times.
- Since the stripes are accessed in parallel, an n-drive RAID 0 array appears as a single large disk with a data rate n times higher than the single-disk rate.



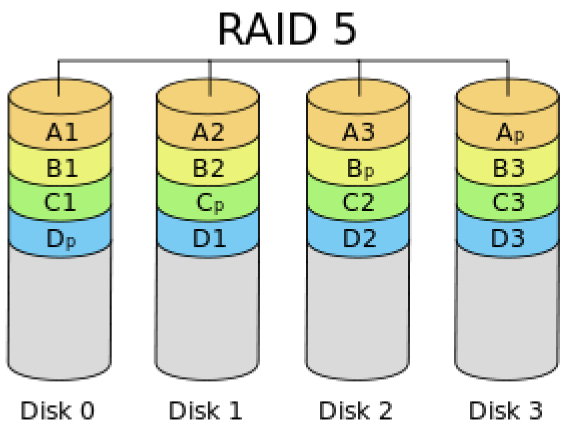
RAID 1 - Mirroring

- RAID 1 consists of an exact copy (or mirror) of a set of data on two or more disks; a classic mirrored pair contains two disks.
- This configuration offers no parity, striping, or spanning of disk space across multiple disks, since the data is mirrored on all disks belonging to the array, and the array can only be as big as the smallest member disk.
- This layout is useful when read performance or reliability is more important than write performance or the resulting data storage capacity.
- The array will continue to operate so long as at least one member drive is operational.
- **Performance**
  - Any read request can be serviced and handled by any drive in the array; thus, depending on the nature of I/O load, random read performance of a RAID 1 array may equal up to the sum of each member's performance, while the write performance remains at the level of a single disk. However, if disks with different speeds are used in a RAID 1 array, overall write performance is equal to the speed of the slowest disk.
  - Synthetic benchmarks show varying levels of performance improvements when multiple HDDs or SSDs are used in a RAID 1 setup, compared with single-drive performance.



RAID 5 – Striping+Parity

- RAID 5 consists of block-level striping with distributed parity. (parity information is distributed among the drives).
- It requires that all drives but one be present to operate.
- Upon failure of a single drive, subsequent reads can be calculated from the distributed parity such that no data is lost.
- RAID 5 requires at least three disks.
- There are many layouts of data and parity in a RAID 5 disk drive array depending upon the sequence of writing across the disks, that is:
  1. the sequence of data blocks written, left to right or right to left on the disk array, of disks 0 to N, and
  2. the location of the parity block at the beginning or end of the stripe, and
  3. the location of the first block of a stripe with respect to parity of the previous stripe.
- The figure to the right shows 1) data blocks written left to right, 2) the parity block at the end of the stripe and 3) the first block of the next stripe not on the same disk as the parity block of the previous stripe.
- Since parity calculation is performed on the full stripe, small changes to the array experience write amplification: in the worst case when a single, logical sector is to be written, the original sector and the according parity sector need to be read, the original data is removed from the parity, the new data calculated into the parity and both the new data sector and the new parity sector are written.


RAID 5
A1
B1
C1
Dp
A2
B2
Cp
D1
A3
Bp
C2
D2
Ap
B3
C3
D3
Disk 0
Disk 1
Disk 2
Disk 3