Storage Media

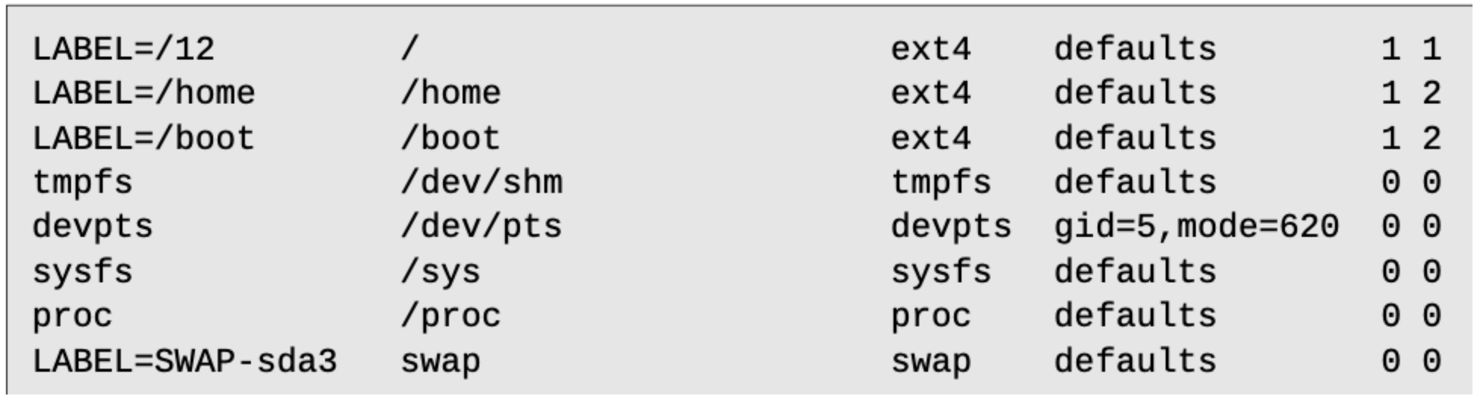
To carry out the exercises in this chapter, we will use a USB flash drive, a USB hard disk drive, and a CD-RW disc (for systems equipped with a CD-ROM burner).

We will look at the following commands:

- mount – Mount a file system
- umount – Unmount a file system
- parted – Partition manipulation program
- mkfs – Create a file system
- fsck – Check and repair a file system
- dd – Convert and copy a file
- genisoimage – Create an ISO 9660 image file
- wodim – Write data to optical storage media
- sha256sum – Compute and check SHA256 checksums

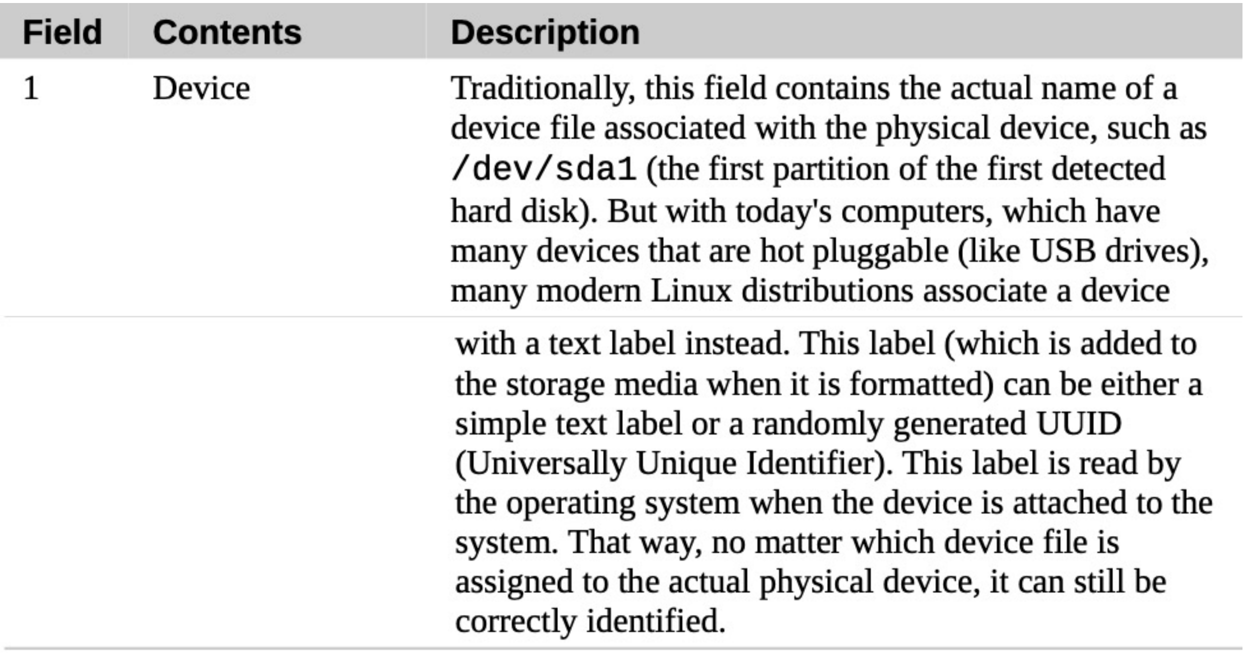
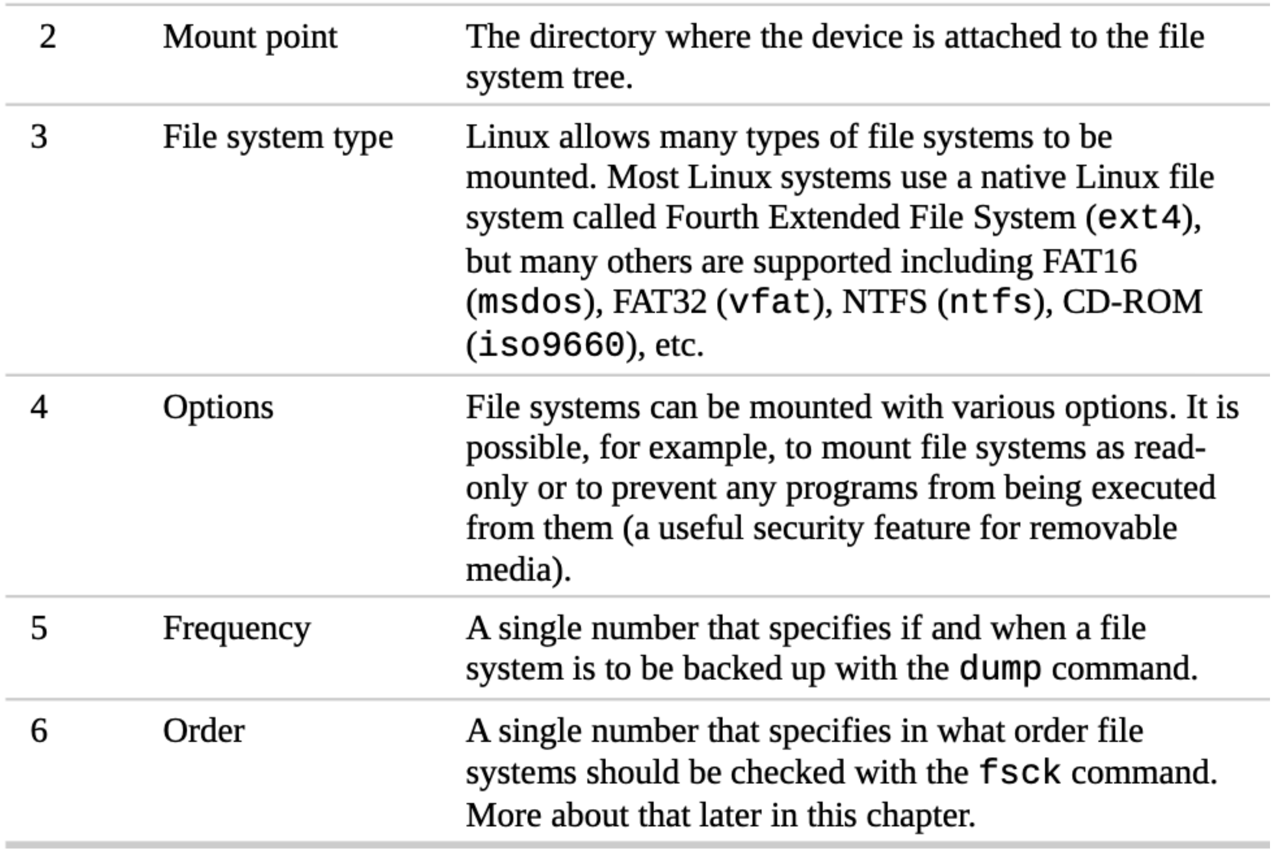
Mounting and Unmounting Storage Devices

- The first step in managing a storage device is attaching the device to the file system tree. This process, called mounting, allows the device to interact with the operating system.
- Unix-like operating systems, such as Linux, maintain a single file system tree with devices attached at various points. This contrasts with other operating systems such as MS-DOS and Windows that maintain separate file system trees for each device (for example C:\, D:\, etc.).
- A file named `/etc/fstab` (short for “file system table”) lists the devices (typically hard disk partitions) that are to be mounted at boot time. Here is an example `/etc/fstab` file from an early Fedora system:
- For our purposes, the interesting ones are the first three



```
LABEL=/12         /                        ext4    defaults         1 1
LABEL=/home       /home                    ext4    defaults         1 2
LABEL=/boot       /boot                    ext4    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
```

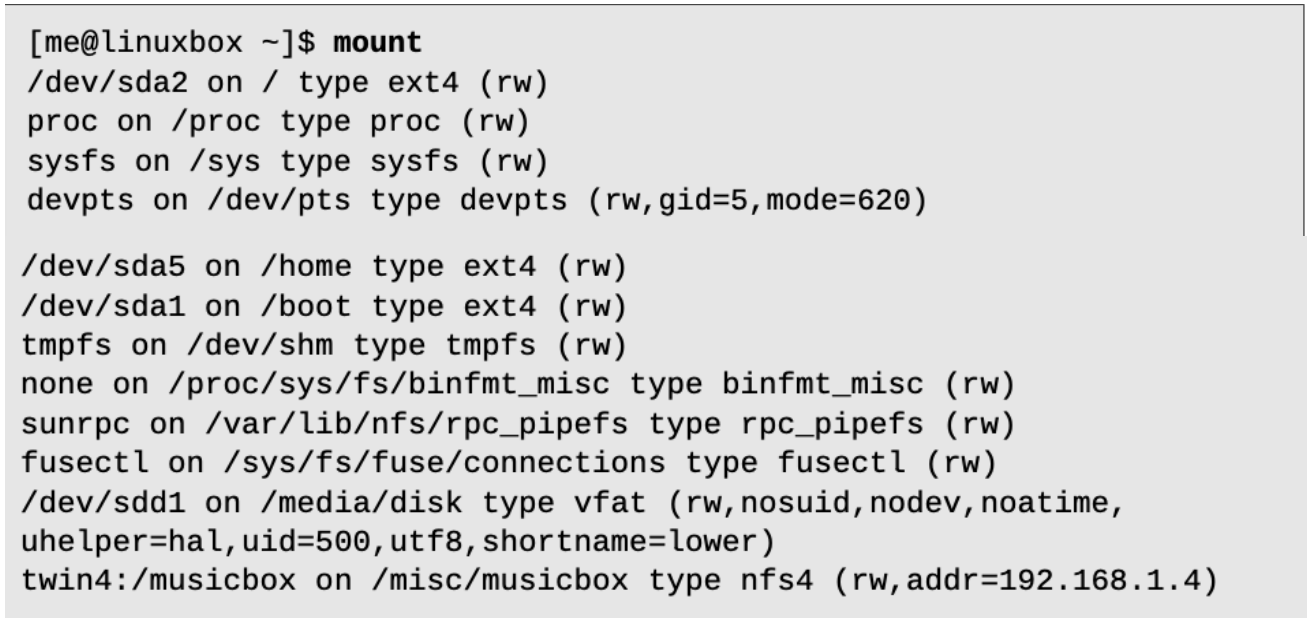
Each line of the file consists of six fields, as described in the table

Field	Contents	Description
1	Device	Traditionally, this field contains the actual name of a device file associated with the physical device, such as `/dev/sda1` (the first partition of the first detected hard disk). 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 media when it is formatted) can be either a simple text label or a randomly generated UUID (Universally Unique Identifier). This label 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 file system tree.
3	File system type	Linux allows many types of file systems to be mounted. Most Linux systems use a native Linux file system called Fourth Extended File System (`ext4`), but many others are supported including FAT16 (`msdos`), FAT32 (`vfat`), NTFS (`ntfs`), CD-ROM (`iso9660`), etc.
4	Options	File systems can be mounted with various options. It is possible, for example, to mount file systems 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 file system is to be backed up with the `dump` command.
6	Order	A single number that specifies in what order file systems should be checked with the `fsck` command. More about that later in this chapter.

Viewing a List of Mounted File Systems

- The `mount` command is used to mount file systems. Entering the command without arguments will display a list of the file systems currently mounted:
- The format of the listing is as follows: device on mount_point type file_system_type (options). For example, the first line shows that device `/dev/sda2` is mounted as the root file system, is of type ext4, and is both readable and writable (the option "rw").
- This listing also has two interesting entries at the bottom of the list. The next-to-last entry shows a 2GB 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 ext4 (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 ext4 (rw)
/dev/sda1 on /boot type ext4 (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)
```

- For our first experiment, we will work with a 4 GB flash drive. First, let's look at Ubuntu 22.04 system before the drive is inserted:

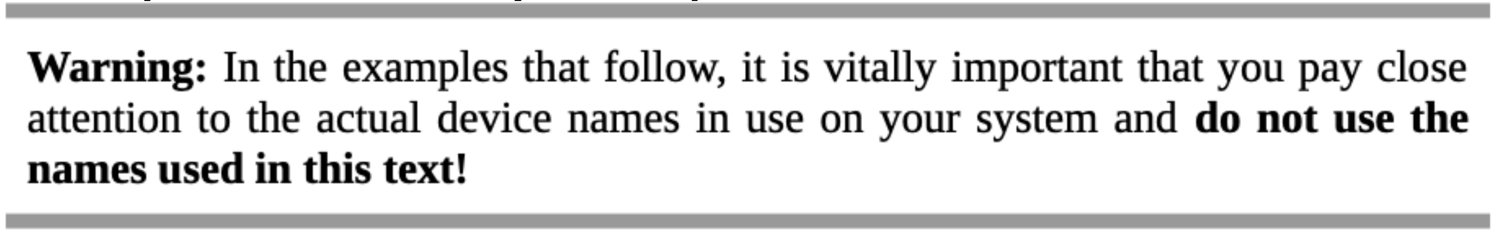
```
[me@linuxbox ~]$ mount | grep /dev/sd
/dev/sda2 on / type ext4 (rw,relatime,errors=remount-ro)
/dev/sda1 on /boot/efi type vfat (rw,relatime,fmask=0077,dmask=0077,
codepage=437,iocharset=iso8859-1,shortname=mixed,errors=remount-ro)
/dev/sdb1 on /home type ext4 (rw,relatime)
```

- We piped the output through grep to only list the `/dev/sd*` devices in the list for clarity. We can see that this system has two hard disks, `/dev/sda` and `/dev/sdb`. There are two partitions on `/dev/sda`. They are `/dev/sda1` and `/dev/sda2` while `/dev/sdb` has a single partition, `/dev/sdb1`.
- Like many modern Linux distributions, this system will attempt to automatically mount the flash drive after insertion. After we insert the drive, we see the following:

```
[me@linuxbox ~]$ mount | grep /dev/sd
```

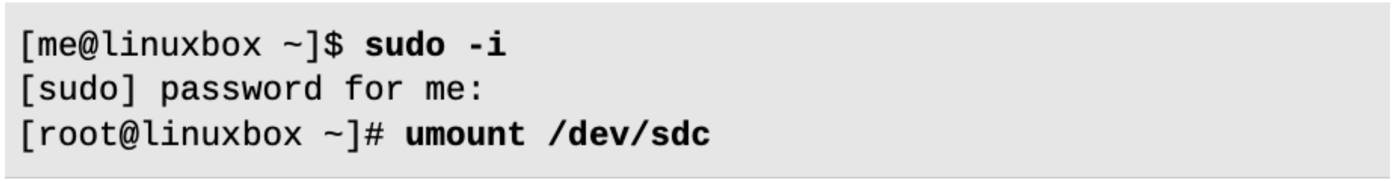
```
dev/sda2 on / type ext4 (rw,relatime,errors=remount-ro)
/dev/sda1 on /boot/efi type vfat
(rw,relatime,fmask=0077,dmask=0077,codepage=437,iocharset=iso8859-
1,shortname=mixed,errors=remount-ro)
/dev/sdb1 on /home type ext4 (rw,relatime)
/dev/sdc on /media/me/C911-C314 type vfat (rw,nosuid,nodev,relatime,
uid=1000,gid=1000,fmask=0022,dmask=0022,codepage=437,iocharset=iso885
9-1,shortname=mixed,showexec,utf8,flush,errors=remount-ro)
```

- After we insert the drive, we see the same listing as before with one additional entry.
- At the end of the listing we see that the flash drive (which is device `/dev/sdc` on this system) has been mounted on `/media/me/C911-C314`, and is type vfat (also known as FAT32, a Windows compatible file system).



Warning: In the examples that follow, it is vitally important that you pay close attention to the actual device names in use on your system and **do not use the names used in this text!**

- Now that we have the device name of the flash drive, let's unmount the drive and remount it at another location in the file system tree.
- To do this, we become the superuser (using the command appropriate for our system) and unmount the drive with the umount (notice the spelling) command.



```
[me@linuxbox ~]$ sudo -i
[sudo] password for me:
[root@linuxbox ~]# umount /dev/sdc
```

- The next step is to create a new mount point for the disk. A mount point is simply a directory somewhere on the file system tree.
- 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.
- For our purposes, we will create a new directory.

```
mkdir /mnt/flash
```

- Finally, we mount the flash drive at the new mount point. The `-t` option is used to specify the file system type.

```
mount -t vfat /dev/sdc /mnt/flash
```

- we can examine the contents of the flash drive via the new mount point.

```
 cd /mnt/flash
  ls
```

- Notice what happens when we try to unmount the drive.

```
umount /dev/sdc
umount: /mnt/flash: device is busy
```

- The reason is that we cannot unmount a device if the device is being used bysomeone or some process.
- We can easily remedy the issue by changing the working directory to something other than the mount point:

```
cd
umount /dev/sdc
```

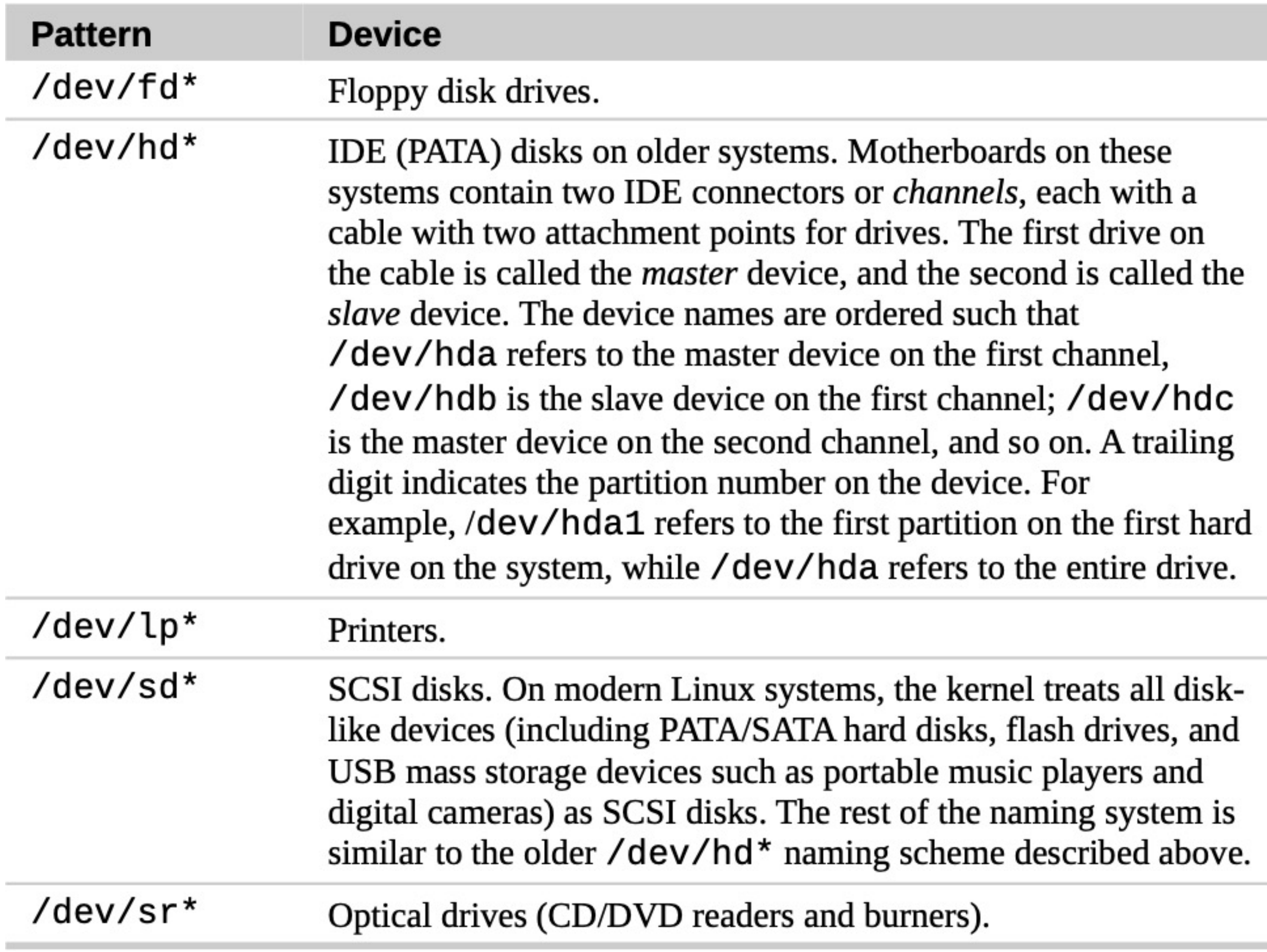
- Now the device unmounts successfully.

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 and lots of devices.

```
ls /dev
```

- The contents of this listing reveal some patterns of device naming. The table outlines a few of these patterns.
- In addition, we often see symbolic links such as `/dev/cdrom`, `/dev/dvd`, and `/dev/floppy`, which point to the actual device files, provided as a convenience.



Pattern	Device
`/dev/fd*`	Floppy disk drives.
`/dev/hd*`	IDE (PATA) disks on older systems. Motherboards on these systems 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` is 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 modern 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).

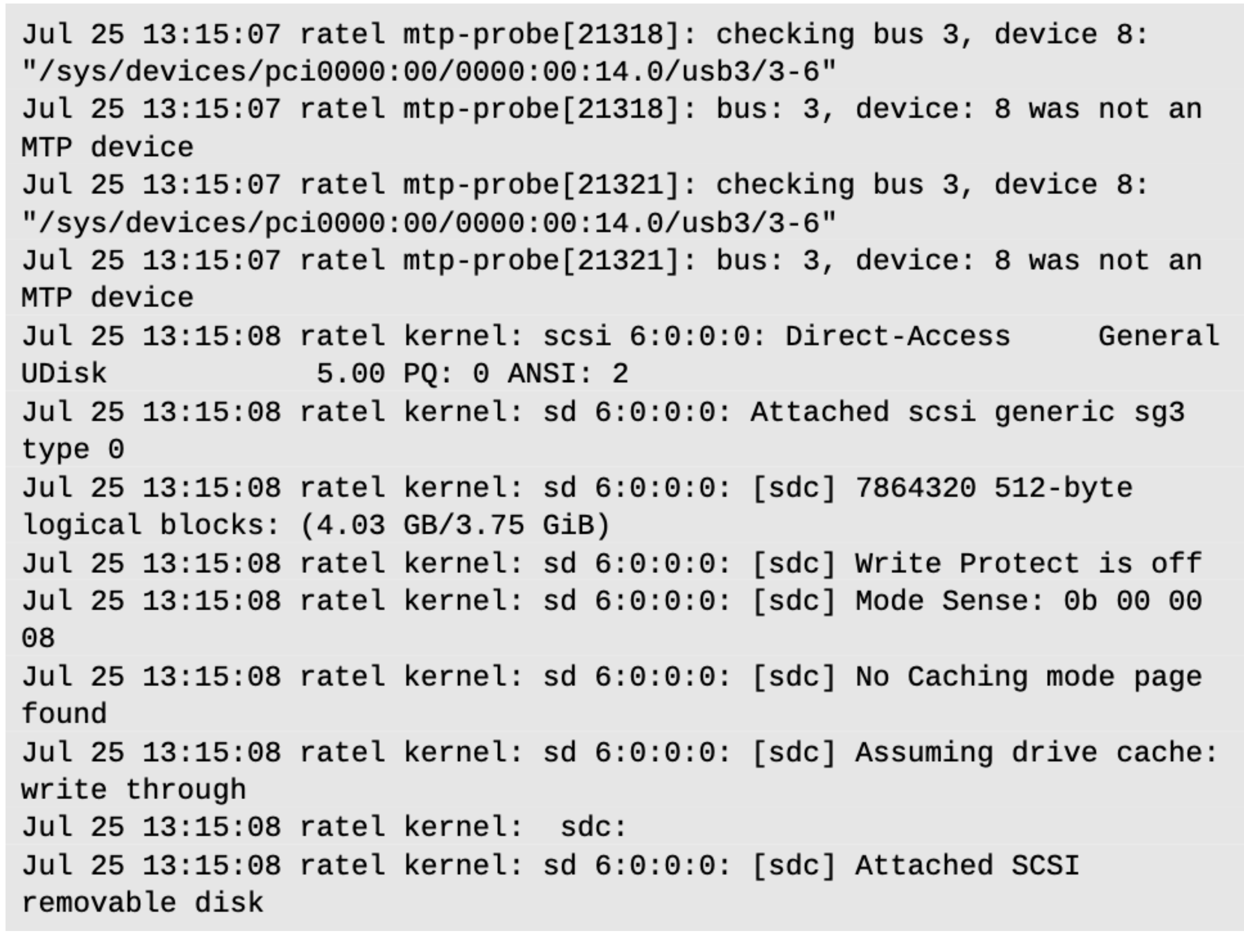
- If we are working on a system that does not automatically mount removable devices, we can use the following technique to determine how the removable device is named when it is attached. First, start a real-time view of the `/var/log/messages` or `/var/log/syslog` file (you may require superuser privileges for this).

  ```
  sudo tail -f /var/log/syslog
  ```

- On modern systemd-based systems use this command to follow the systemd journal:

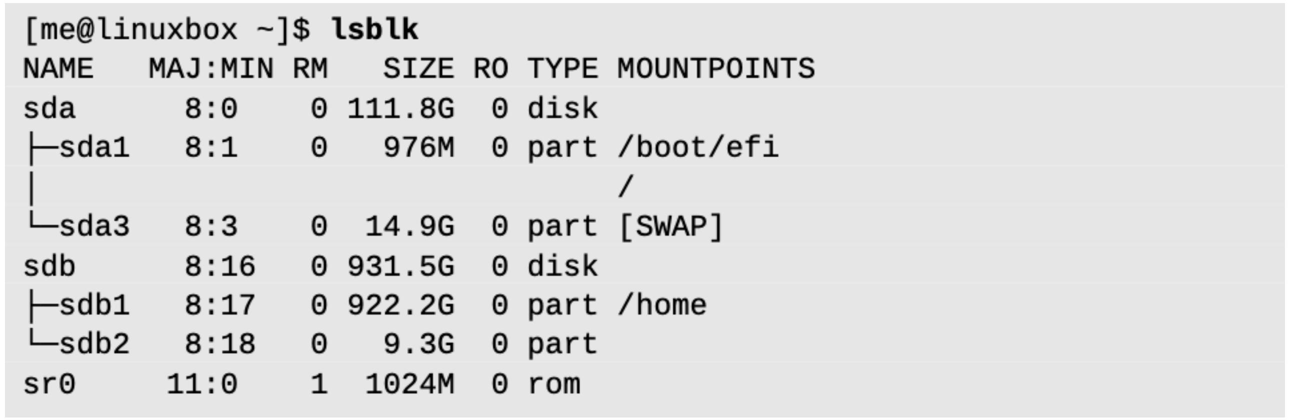
  ```
  sudo journalctl -f
  ```

- The last few lines of the listing will be displayed and then will pause. Next, plug in the removable device. In this example, we will unmount and remove our flash drive and then reinsert it. Almost immediately, the kernel will notice the device and probe it.
- After the display pauses again, press Ctrl-c to get the prompt back



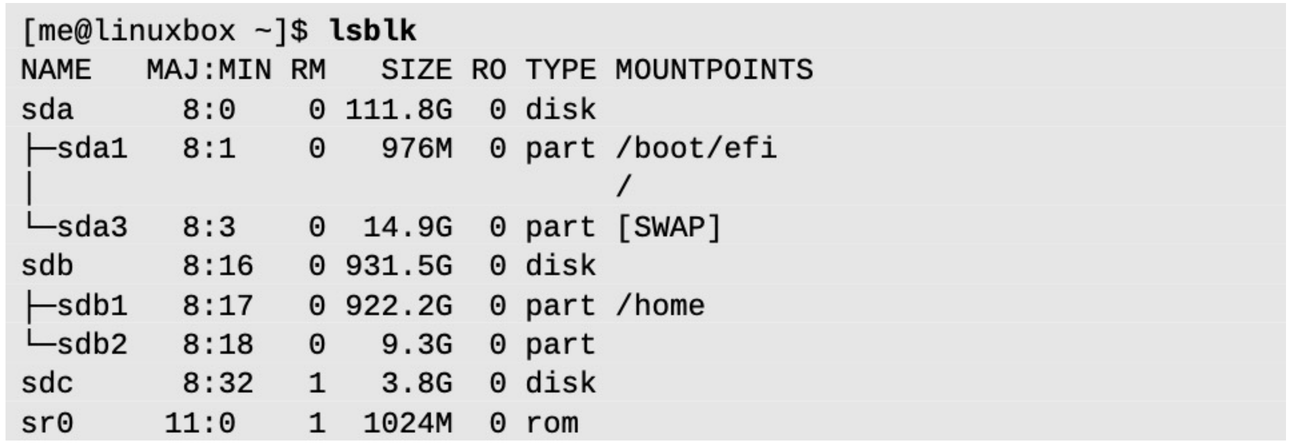
```
Jul 25 13:15:07 ratel mtp-probe[21318]: checking bus 3, device 8:
"/sys/devices/pci0000:00/0000:00:14.0/usb3/3-6"
Jul 25 13:15:07 ratel mtp-probe[21318]: bus: 3, device: 8 was not an
MTP device
Jul 25 13:15:07 ratel mtp-probe[21321]: checking bus 3, device 8:
"/sys/devices/pci0000:00/0000:00:14.0/usb3/3-6"
Jul 25 13:15:07 ratel mtp-probe[21321]: bus: 3, device: 8 was not an
MTP device
Jul 25 13:15:08 ratel kernel: scsi 6:0:0:0: Direct-Access     General
UDisk            5.00 PQ: 0 ANSI: 2
Jul 25 13:15:08 ratel kernel: sd 6:0:0:0: Attached scsi generic sg3
type 0
Jul 25 13:15:08 ratel kernel: sd 6:0:0:0: [sdc] 7864320 512-byte
logical blocks: (4.03 GB/3.75 GiB)
Jul 25 13:15:08 ratel kernel: sd 6:0:0:0: [sdc] Write Protect is off
Jul 25 13:15:08 ratel kernel: sd 6:0:0:0: [sdc] Mode Sense: 0b 00 00
08
Jul 25 13:15:08 ratel kernel: sd 6:0:0:0: [sdc] No Caching mode page
found
Jul 25 13:15:08 ratel kernel: sd 6:0:0:0: [sdc] Assuming drive cache:
write through
Jul 25 13:15:08 ratel kernel:  sdc:
Jul 25 13:15:08 ratel kernel: sd 6:0:0:0: [sdc] Attached SCSI
removable disk
```

- There is a another way we can determine a device name (there's always more than one way to do things in Linux!). We can use the lsblk command.
- This command lists all of the block devices attached to the system regardless if they are mounted or not.
- Assuming we have unmounted and removed our flash drive, we can look at the list of attached block devices:



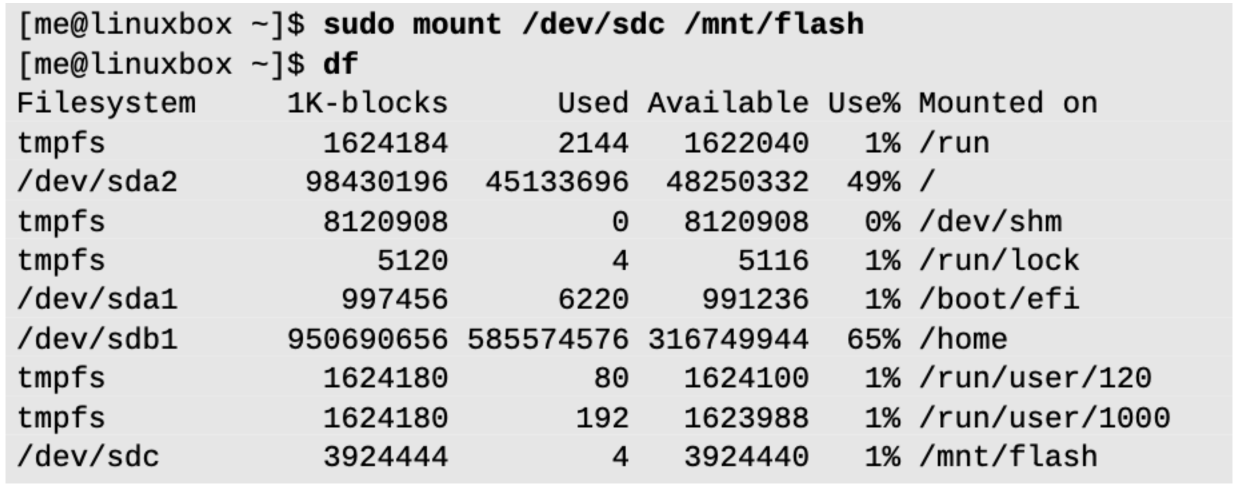
```
[me@linuxbox ~]$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda      8:0    0 111.8G  0 disk
├─sda1   8:1    0   976M  0 part /boot/efi
│                                /
└─sda3   8:3    0  14.9G  0 part [SWAP]
sdb      8:16   0 931.5G  0 disk
├─sdb1   8:17   0 922.2G  0 part /home
└─sdb2   8:18   0   9.3G  0 part
sr0     11:0    1  1024M  0 rom
```

- Next, we’ll reinsert the flash drive and run `lsblk` again:



```
[me@linuxbox ~]$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda      8:0    0 111.8G  0 disk
├─sda1   8:1    0   976M  0 part /boot/efi
│                                /
└─sda3   8:3    0  14.9G  0 part [SWAP]
sdb      8:16   0 931.5G  0 disk
├─sdb1   8:17   0 922.2G  0 part /home
└─sdb2   8:18   0   9.3G  0 part
sdc      8:32   1   3.8G  0 disk
sr0     11:0    1  1024M  0 rom
```

- We now see a new device (sdc) has been added to the list. The device name will remain the same as long as it remains physically attached to the computer and the computer is not rebooted. With our device name in hand, we can mount the flash drive.



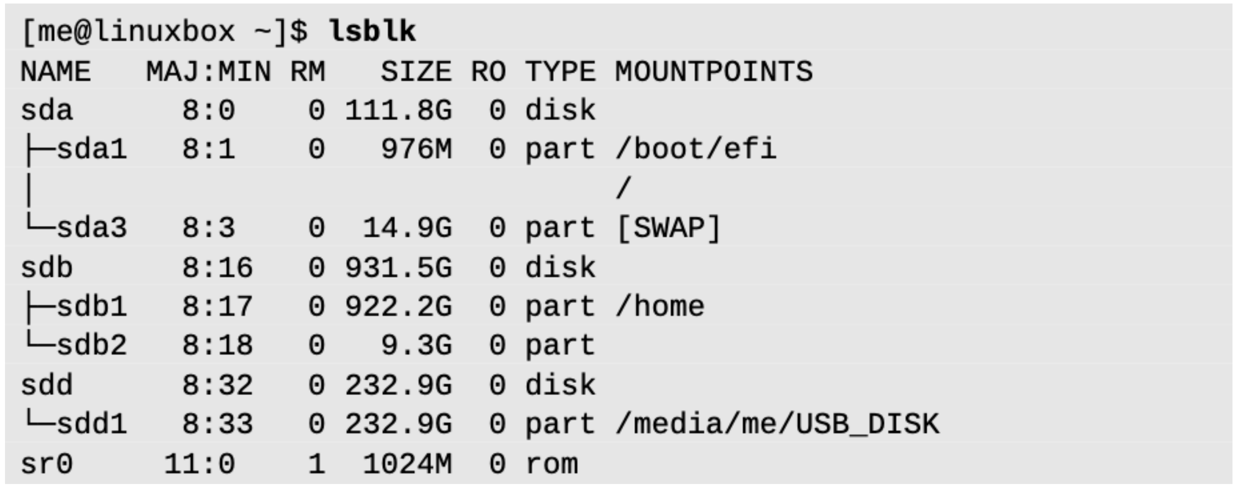
```
[me@linuxbox ~]$ sudo mount /dev/sdc /mnt/flash
[me@linuxbox ~]$ df
Filesystem     1K-blocks      Used Available Use% Mounted on
tmpfs            1624184      2144   1622040   1% /run
/dev/sda2       98430196  45133696  48250332  49% /
tmpfs            8120908         0   8120908   0% /dev/shm
tmpfs               5120         4      5116   1% /run/lock
/dev/sda1         997456      6220    991236   1% /boot/efi
/dev/sdb1      950690656 585574576 316749944  65% /home
tmpfs            1624180        80   1624100   1% /run/user/120
tmpfs            1624180       192   1623988   1% /run/user/1000
/dev/sdc         3924444         4   3924440   1% /mnt/flash
```

Creating New File Systems

- Let's say that we have an external USB hard disk with a single FAT32 file system and would like to split the drive into two partitions, a Linux-native file system (ext4) and a second one formatted as NTFS for use with a Windows system. This involves two steps.
    1. Create a new partition layout.
    2. Create new, empty file systems on the drive.
- **Warning!** In the following exercise, we are going to format an external hard drive. Use a drive that contains nothing you care about because it will be erased! Again, make absolutely sure you are specifying the correct device name for your system, not the one shown in the text. Failure to heed this warning could result in you formatting (i.e., erasing) the wrong drive!

Manipulating Partitions with parted

- `parted` is one of a host of programs (both command line and graphical) that allow us to interact directly with disk-like devices (such as hard disk drives and flash drives) at a very low level. We'll attach the drive and use `lsblk` to get its name:



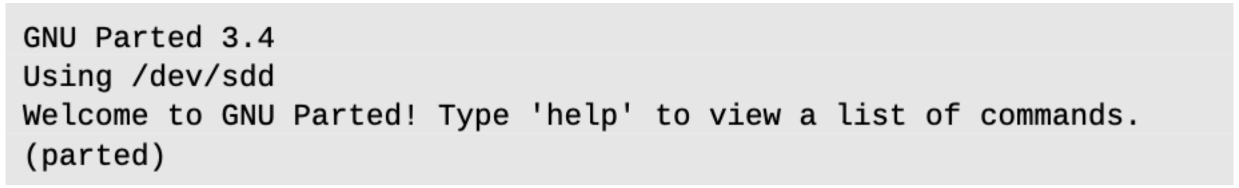
```
[me@linuxbox ~]$ lsblk
NAME    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda       8:0    0 111.8G  0 disk
├─sda1    8:1    0   976M  0 part /boot/efi
│                                 /
└─sda3    8:3    0  14.9G  0 part [SWAP]
sdb       8:16   0 931.5G  0 disk
├─sdb1    8:17   0 922.2G  0 part /home
└─sdb2    8:18   0   9.3G  0 part
sdd       8:32   0 232.9G  0 disk
└─sdd1    8:33   0 232.9G  0 part /media/me/USB_DISK
sr0      11:0    1  1024M  0 rom
```

- As we can see, the external USB hard disk is attached and it is named `/dev/sdd` and it contains one partition named `/dev/sdd1`.

- With parted, we can edit, delete, and create partitions on the device. To work with our hard drive, we must first unmount it (if needed) and then invoke the parted program as follows:

```
sudo umount /dev/sdd1
sudo parted /dev/sdd
```

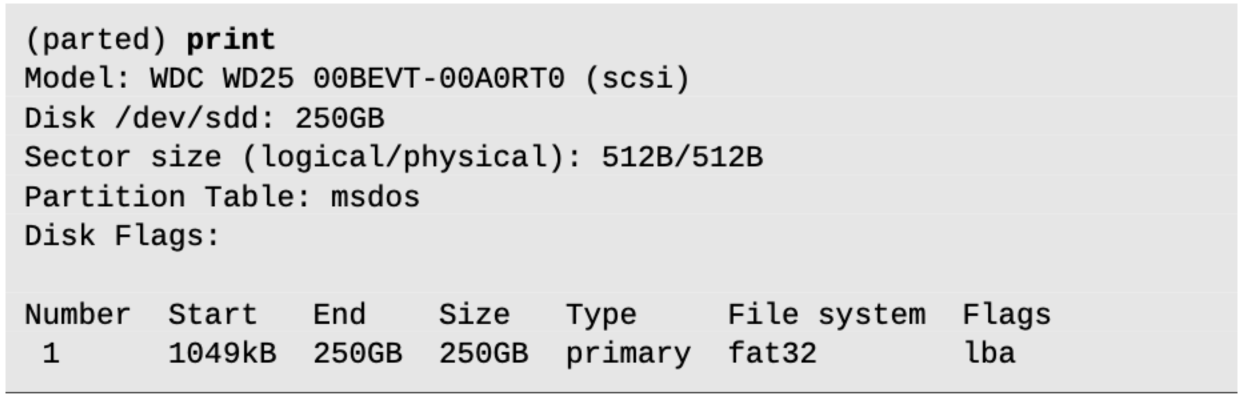
- Notice that we must specify the device in terms of the entire device, not by partition number. After the program starts up, we will see the following prompt:



```
GNU Parted 3.4
Using /dev/sdd
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted)
```

- Entering “help” will display a list of available commands.

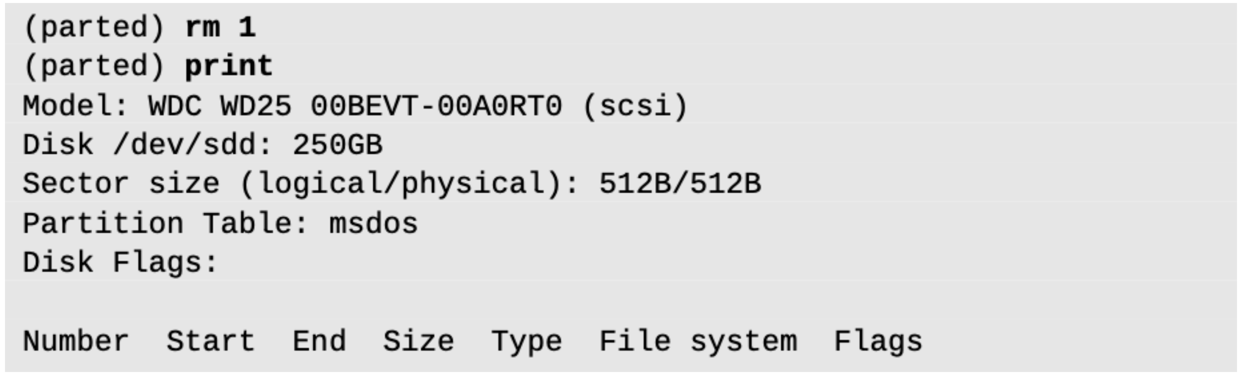
- The first thing we want to do is examine the existing partition layout. We do this by entering the “print” command to print the partition table for the device.



```
(parted) print
Model: WDC WD25 00BEVT-00A0RT0 (scsi)
Disk /dev/sdd: 250GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End    Size   Type     File system  Flags
 1      1049kB  250GB  250GB  primary  fat32        lba
```

- We can see that our disk has 250 GB of space in a single FAT32 partition.
- To create our new partitions, we must first delete the current partition. This is easily done with the `rm` command.



```
(parted) rm 1
(parted) print
Model: WDC WD25 00BEVT-00A0RT0 (scsi)
Disk /dev/sdd: 250GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start  End  Size  Type  File system  Flags
```

- We specify the partition number we want to remove then look at the partition table again to see that the partition is gone.
- Next, we'll create our new partitions. We do this with the `mkpart` command.

```
(parted) mkpart
Partition type?  primary/extended? primary
File system type?  [ext2]? ext4
Start? 1
End? 120000
```

- When creating a partition on this disk (which has a MBR, master boot record style partition table) we specify either a primary or extended partition followed by the start and end of the partition in the default one megabyte increments. Using an end value of 120,000 MB consumes roughly half of the available space on the drive.
- Next, we'll create the second partition using the same technique.

```
(parted) mkpart
Partition type?  primary/extended? primary
File system type?  [ext2]? ntfs
Start? 120001
End? 240000
```

- we look at the partition table to see our results.

```
(parted) print
Model: WDC WD25 00BEVT-00A0RT0 (scsi)
Disk /dev/sdd: 250GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number Start   End    Size   Type     File system  Flags
1      1049kB  120GB  120GB  primary  ext4         lba
2      120GB   240GB  120GB  primary  ntfs         lba
```

- Now that our partitioning is complete, we can exit the parted program.

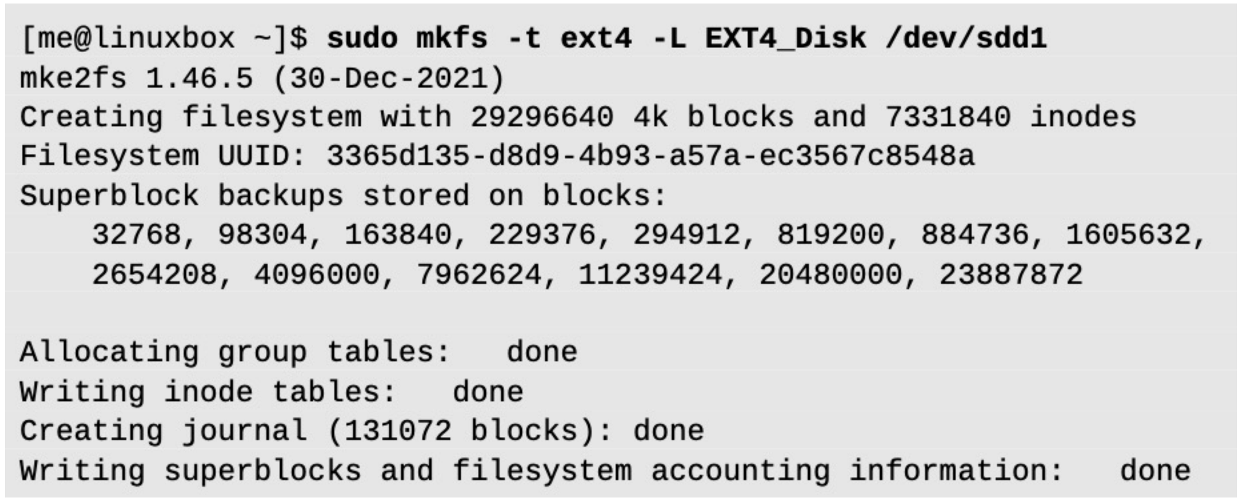
  `(parted) quit`

- If we run `lsblk` again we can see our drive and its two partitions.

```
[me@linuxbox ~]$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda      8:0    0 111.8G  0 disk
├─sda1   8:1    0   976M  0 part /boot/efi
│                                /
└─sda3   8:3    0  14.9G  0 part [SWAP]
sdb      8:16   0 931.5G  0 disk
├─sdb1   8:17   0 922.2G  0 part /home
└─sdb2   8:18   0   9.3G  0 part
sdd      8:48   0 232.9G  0 disk
├─sdd1   8:49   0 111.8G  0 part
└─sdd2   8:50   0 111.8G  0 part
sr0     11:0    1  1024M  0 rom
```

Creating a New File System with `mkfs`

- With our partition editing done, it’s time to create (i.e., format) the new file systems on our drive. To do this, we will use `mkfs` (short for “make file system”), which can create file systems in a variety of formats.
- To create the ext4 file system on the drive, we use the `-t` option to specify the ext4 file system type, followed by a descriptive volume label, and the name of the device containing the partition we want to format.



```
[me@linuxbox ~]$ sudo mkfs -t ext4 -L EXT4_Disk /dev/sdd1
mke2fs 1.46.5 (30-Dec-2021)
Creating filesystem with 29296640 4k blocks and 7331840 inodes
Filesystem UUID: 3365d135-d8d9-4b93-a57a-ec3567c8548a
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632,
    2654208, 4096000, 7962624, 11239424, 20480000, 23887872

Allocating group tables:   done
Writing inode tables:   done
Creating journal (131072 blocks): done
Writing superblocks and filesystem accounting information:   done
```

- Next, we’ll do the NTFS partition

```
[me@linuxbox ~]$ sudo mkfs -t ntfs --quick -L NTFS_Disk /dev/sdd2
Cluster size has been automatically set to 4096 bytes.
Creating NTFS volume structures.
mkntfs completed successfully. Have a nice day.
```

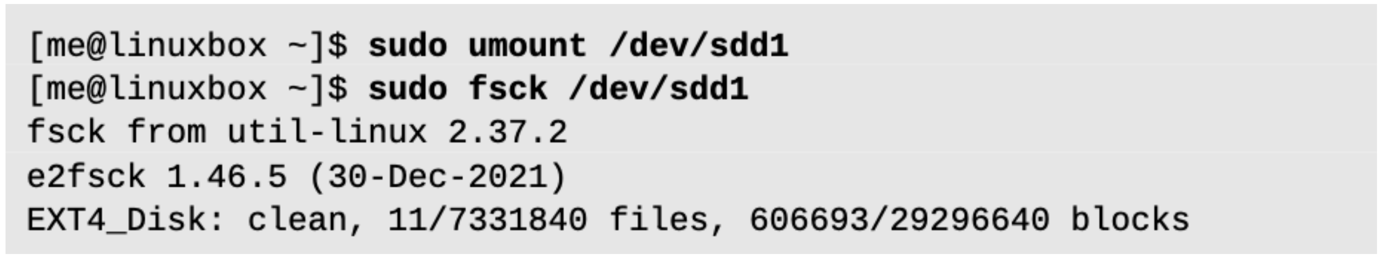
- Again we specify a file system type, a descriptive volume label, and a device. We include the `--quick` option to skip the bad block checking because that takes a long time to perform.
- Note that different file system types support different options. For details, see the `mkfs` man page.
- With our work completed, let’s unplug the drive and reattach it to cause the system to automatically mount the drive. Running `lsblk` once again reveals the results.
- As we can see, the volume labels are used to create the mount points names in the `/media` directory where removable storage devices are automatically mounted.

```
[me@linuxbox ~]$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda      8:0    0 111.8G  0 disk
├─sda1   8:1    0   976M  0 part /boot/efi
│                                /
└─sda3   8:3    0  14.9G  0 part [SWAP]
sdb      8:16   0 931.5G  0 disk
├─sdb1   8:17   0 922.2G  0 part /home
└─sdb2   8:18   0   9.3G  0 part
sdd      8:48   0 232.9G  0 disk
├─sdd1   8:49   0 111.8G  0 part /media/me/EXT4_Disk
```

Testing and Repairing File Systems

- 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 file systems before mounting them. This is done by the `fsck` program (short for “file system check”).
- The last number in each fstab entry specifies the order in which the devices are to be checked. In our previous example, we see that the root file system is checked first, followed by the home and boot file systems. Devices with a zero as the last digit are not routinely checked.
- In addition to checking the integrity of file systems, `fsck` can also repair corrupt file systems with varying degrees of success, depending on the amount of damage. On Unix-like file systems, recovered portions of files are placed in the `lost+found` directory, located in the root of each file system.

- To check our EXT4_Disk partition (which should be unmounted first), we could do the following:



```
[me@linuxbox ~]$ sudo umount /dev/sdd1
[me@linuxbox ~]$ sudo fsck /dev/sdd1
fsck from util-linux 2.37.2
e2fsck 1.46.5 (30-Dec-2021)
EXT4_Disk: clean, 11/7331840 files, 606693/29296640 blocks
```

- On most systems, file system corruption detected at boot time will cause the system to stop and direct you to run fsck before continuing.

Moving Data Directly to and from Devices

- While we usually think of data on our computers as being organized into files, 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. However, if we could treat a disk drive as simply a large collection of data blocks, we could perform useful tasks, such as cloning devices.
- The dd program performs this task. It copies blocks of data from one place to another. It uses a unique syntax (for historical reasons) and is usually used this way:

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

- **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 specification. Always double-check your input and output specifications before pressing enter!

- Let’s say we had two USB flash drives of the same size and we wanted to exactly copy the first drive to the second. If we attached both drives to the computer and they are assigned to devices `/dev/sdb` and `/dev/sdc` respectively, we could copy everything on the first drive to the second drive with the following:

```
sudo dd if=/dev/sdb of=/dev/sdc
```

- Alternately, if only the first device were attached to the computer, we could copy its contents to an ordinary file for later restoration or copying.

```
sudo dd if=/dev/sdb of=flash_drive.img
```

Creating CD-ROM Images

Writing a recordable CD-ROM (either a CD-R or CD-RW) consists of two steps.

1. Constructing an ISO image file that is the exact file system image of the CD-ROM
2. Writing the image file onto the CD-ROM media

Creating an Image Copy of a CD-ROM

- If we want to make an ISO image of an existing CD-ROM, we can use dd to read all the data blocks off the CD-ROM and copy them to a local file. Say we had an Ubuntu CD and we wanted to make an ISO file that we could later use to make more copies. After inserting the CD and determining its device name (we'll assume `/dev/cdrom`), we can make the ISO file like so:

  ```
  dd if=/dev/cdrom of=ubuntu.iso
  ```

- This technique works for data DVDs as well but will not work for audio CDs, as they do not use a file system for storage. For audio CDs, look at the `cdrdao` command.

Creating an Image From a Collection of Files

- To create an ISO image file containing the contents of a directory, we use the `genisoimage` program.
- To do this, we first create a directory containing all the files we want to include in the image, and then execute the `genisoimage` command to create the image file.
- For example, if we had created a directory called `~/cd-rom-files` and filled it with files for our CD-ROM, we could create an image file named `cd-rom.iso` with the following command:

```
genisoimage -o cd-rom.iso -R -J ~/cd-rom-files
```

- The -R option adds metadata for the Rock Ridge extensions, which allows the use of long filenames and POSIX-style file permissions. Likewise, the `-J` option enables the Joliet extensions, which permit long filenames for Windows.

Writing CD-ROM Images: Mounting an ISO Image Directly

- There is a trick that we can use to mount an ISO image while it is still on our hard disk and treat it as though it were already on optical media. By adding the "`-o loop`" option to mount (along with the required "`-t iso9660`" file system type), we can mount the image file as though it were a device and attach it to the file system tree.

```
mkdir /mnt/iso_image
mount -t iso9660 -o loop image.iso /mnt/iso_image
```

- In the example above, we created a mount point named `/mnt/iso_image` and then mounted the image file `image.iso` at that mount point.
- After the image is mounted, it can be treated just as though it were a real CD-ROM or DVD. Remember to unmount the image when it is no longer needed.

Blanking a Rewritable CD-ROM

- Rewritable CD-RW media needs to be erased or blanked before it can be reused.
- To do this, we can use `wodim`, specifying the device name for the CD writer and the type of blanking to be performed. The `wodim` program offers several types. The most minimal (and fastest) is the “fast” type.

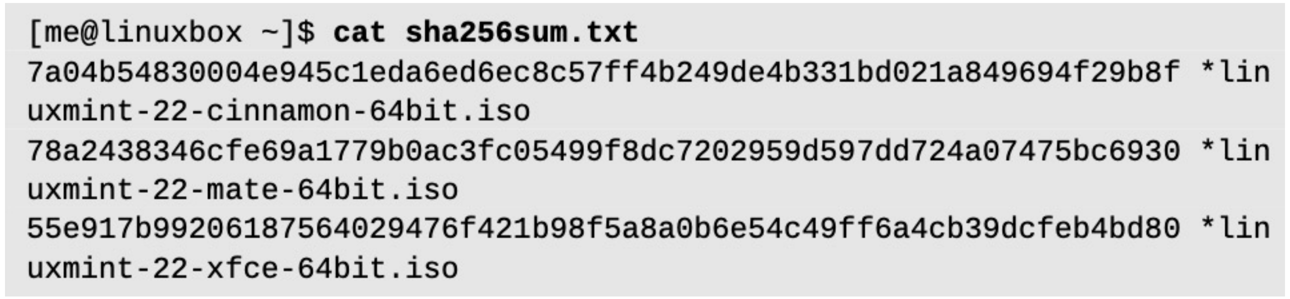
```
wodim dev=/dev/cdrw blank=fast
```

Writing an Image

- To write an image, we again use `wodim`, specifying the name of the optical media writer device and the name of the image file.
- `wodim dev=/dev/cdrw image.iso`
- In addition to the device name and image file, `wodim` supports a large set of options.
- Two common ones are “`-v`” for verbose output, and “`-dao`”, which writes the disc in *disc-at-once* mode. This mode should be used if we are preparing a disc for commercial reproduction.
- The default mode for `wodim` is *track-at-once*, which is useful for recording music tracks.

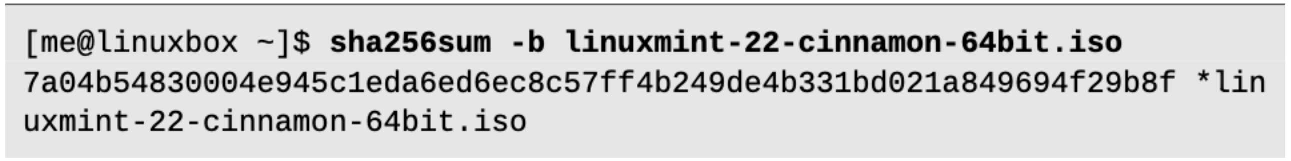
Verifying Data

- While we’re on the subject of ISO files, let’s look at how to verify the integrity of files we download. Often when we download a large file such as a new Linux installation image, we will want to make sure that image file we downloaded is complete and not corrupted. One tool we can use for this is `sha256sum`, a modern replacement for an earlier program called `md5sum`.
- There are two common ways this can be done. Let’s try them out. Say we want to download an installation image of Linux Mint 22.
- We go to the Linux Mint website and get the `linuxmint-22-cinnamon-64bit.iso` file. While we are there we also download a checksum file called `sha256sum.txt`. When we look at its contents we see the following:



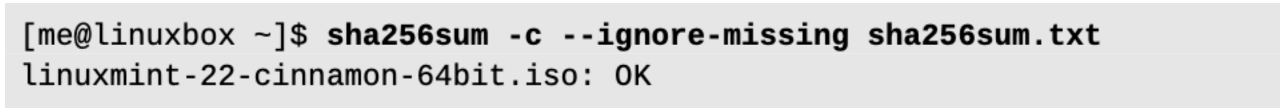
```
[me@linuxbox ~]$ cat sha256sum.txt
7a04b54830004e945c1eda6ed6ec8c57ff4b249de4b331bd021a849694f29b8f *lin
uxmint-22-cinnamon-64bit.iso
78a2438346cfe69a1779b0ac3fc05499f8dc7202959d597dd724a07475bc6930 *lin
uxmint-22-mate-64bit.iso
55e917b99206187564029476f421b98f5a8a0b6e54c49ff6a4cb39dcfeb4bd80 *lin
uxmint-22-xfce-64bit.iso
```

- Three lines of data with a long strings of hexadecimal numbers and the names of files available for downloading including the ISO file we downloaded.
- The strings of hex digits are checksums, a precise mathematical representation of the ISO files. These numbers are special in that if the downloaded ISO file is altered by even one bit from the original file, the checksum will be significantly different. We'll test the ISO file to see if the checksums match.



```
[me@linuxbox ~]$ sha256sum -b linuxmint-22-cinnamon-64bit.iso
7a04b54830004e945c1eda6ed6ec8c57ff4b249de4b331bd021a849694f29b8f *lin
uxmint-22-cinnamon-64bit.iso
```

- The -b option tells sha256sum that we are looking at a binary file rather than text. As we can see, the checksum matches the one we saw in sha256sum.txt. However looking at the checksum this way is a little tedious. Another way we can check the file is to have sha256sum process the sha256sum.txt file to compare its checksums against ones it generates from the files on the list. We can do this by running the sha256sum program this way:



```
[me@linuxbox ~]$ sha256sum -c --ignore-missing sha256sum.txt
linuxmint-22-cinnamon-64bit.iso: OK
```

- The -c option (short for "check") invokes this mode while the --ignore-missing option tells sha256sum not to complain that the files we didn't download, linuxmint-22-mate-64bit.iso and linuxmint-22-xfce-64bit.iso aren't present.