Configuring Hardware

Objectives:

- ✓ 101.1 Determine and configure hardware settings
- ✓ 102.1 Design hard disk layout
- ✓ 104.1 Create partitions and filesystems
- ✓ 104.2 Maintain the integrity of filesystems
- ✓ 104.3 Control mounting and unmounting of filesystems

CONFIGURING CORE HARDWARE

Before we look at the individual hardware cards available, let's first look at how the core hardware operates.

This section discusses what happens when you hit the power button on your Linux workstation or server.

Device Interfaces

- Each device you connect to your Linux system uses some type of standard protocol to communicate with the system hardware.
- The Linux kernel software must know how to send data to and receive data from the hardware device using those protocols.
- There are currently three popular standards used to connect devices.
  - ✓ Peripheral Component Interconnect (PCI) standard
  - ✓ Universal Serial Bus (USB) interface
  - ✓ General Purpose Input/Output (GPIO) interface

PCI Boards

- The Peripheral Component Interconnect (PCI) standard was developed in 1993 as a method for connecting hardware boards to PC motherboards.
- The standard has been updated a few times to accommodate faster interface speeds, as well as increasing data bus sizes on motherboards.
- The PCI Express (PCIe) standard is currently used on most server and desktop workstations to provide a common interface for external hardware devices.

PCI Boards

- Lots of different client devices use PCI boards to connect to a server or desktop workstation:
  - ✓Internal hard drives
  - ✓External hard drives
  - ✓Network interface cards
  - ✓Wireless cards
  - ✓Bluetooth devices
  - ✓Video accelerators
  - ✓Audio cards

PCI Boards

- Most PCI boards utilize the Plug-and-Play (PnP) standard, which automatically determines the configuration settings for the boards so no two boards conflict with each other.
- If you do run into conflicts, you can use the `setpci` utility to view and manually change settings for an individual PCI board.

The USB Interface

- The Universal Serial Bus (USB) interface has become increasingly popular due to its ease of use and its increasing support for high-speed data communication.
- Since the USB interface uses serial communications, it requires fewer connectors with the motherboard, allowing for smaller interface plugs.
- The USB standard has evolved over the years.
  - ✓ The original version, 1.0, supported data transfer speeds only up to 12 Mbps.
  - ✓ The 2.0 standard increased the data transfer speed to 480 Mbps.
  - ✓ The current USB standard, 3.2, allows for data transfer speeds up to 20 Gbps, making it useful for high-speed connections to external storage devices.

The USB Interface

- There are two steps to get Linux to interact with USB devices:

1. **The Linux kernel must have the proper module installed to recognize the USB controller that is installed on your server, workstation, or laptops.**
   - ✓ The controller provides communication between the Linux kernel and the USB bus on the system.
   - ✓ When the Linux kernel can communicate with the USB bus, any device you plug into a USB port on the system will be recognized by the kernel, but may not necessarily be useful.
2. **The Linux system must then also have a kernel module installed for the individual device type plugged into the USB bus.**

The GPIO Interface

- The General Purpose Input/Output (GPIO) interface has become popular with small utility Linux systems, designed for controlling external devices for automation projects.
- This includes popular hobbyist Linux systems such as the Raspberry Pi and BeagleBone kits.
- The GPIO interface provides multiple digital input and output lines that you can control individually, down to the single-bit level.
- The GPIO function is normally handled by a specialty integrated circuit (IC) chip, which is mapped into memory on the Linux system.

The GPIO Interface

- The GPIO interface is ideal for supporting communications to external devices such as relays, lights, sensors, and motors.
- Applications can read individual GPIO lines to determine the status of switches, turn relays on or off, or read digital values returned from any type of analog-to-digital sensors such as temperature or pressure sensors.
- The GPIO interface provides a wealth of possibilities for using Linux to control objects and environments.
- You can write programs that control the temperature in a room, sense when doors or windows are opened or closed, sense motion in a room, or even control the operation of a robot.

The /dev Directory

- After the Linux kernel communicates with a device on an interface, it must be able to transfer data to and from the device.
  - ✓ This is done using device files.
- Device files are files that the Linux kernel creates in the special `/dev` directory to interface with hardware devices.
- To retrieve data from a specific device, a program just needs to read the Linux device file associated with that device.
- To send data to the device, the program just needs to write to the Linux device file.
- This is a lot easier than requiring each application to know how to directly interact with a device.

Types of Device Files in Linux

- **Character device files**
  - ✓ Transfer data one character at a time. This method is often used for serial devices such as terminals and USB devices.
- **Block device files**
  - ✓ Transfer data in large blocks of data. This method is often used for high-speed data transfer devices such as hard drives and network cards.

Types of Device Files in Linux

```
$ ls -al sd* tty*
brw-rw---- 1 root disk 8, 0 Feb 16 17:49 sda
brw-rw---- 1 root disk 8, 1 Feb 16 17:49 sda1
crw-rw-rw- 1 root tty 5, 0 Feb 16 17:49 tty
crw--w---- 1 root tty 4, 0 Feb 16 17:49 tty0
crw--w---- 1 gdm tty 4, 1 Feb 16 17:49 tty1
```

Device Mapper

- **Besides device files, Linux also provides a system called the device mapper.**
- **The device mapper function is performed by the Linux kernel.**
  - ✓ It maps physical block devices to virtual block devices.
  - ✓ These virtual block devices allow the system to intercept the data written to or read from the physical device and perform some type of operation on them.
  - ✓ Mapped devices are used by
    - Logical Volume Manager (LVM) for creating logical drives
    - Linux Unified Key Setup (LUKS) for encrypting data on hard drives
- **The device mapper creates virtual devices in the /dev/mapper directory.**
- **These files are links to the physical block device files in the /dev directory**

The /proc Directory

- It's not a physical directory on the filesystem, but instead a virtual directory that the kernel dynamically populates to provide access to information about the system hardware settings and status.
- The Linux kernel changes the files and data in the /proc directory as it monitors the status of hardware on the system.
- Various /proc files are available for different system features:
  - ✓ interrupt requests (IRQs)
  - ✓ input/output (I/O) ports
  - ✓ direct memory access (DMA)

Interrupt Requests

- Interrupt requests (IRQs) allow hardware devices to indicate when they have data to send to the CPU.
- The PnP system must assign each hardware device installed on the system a unique IRQ address.
- You can view the current IRQs in use on your Linux system by:

```
$ cat /proc/interrupts
```

I/O Ports

- The system I/O ports are locations in memory where the CPU can send data to and receive data from the hardware device.
- As with IRQs, the system must assign each device a unique I/O port.
- This is yet another feature handled by the PnP system.
- You can monitor the I/O ports assigned to the hardware devices on your system by:

```
$ sudo cat /proc/ioports
```

Direct Memory Access

- Using I/O ports to send data to the CPU can be somewhat slow.
  - ✓ To speed things up, many devices use direct memory access (DMA) channels.
- DMA channels send data from a hardware device directly to memory on the system, without having to wait for the CPU.
- The CPU can then read those memory locations to access the data when it's ready.
- As with I/O ports, each hardware device that uses DMA must be assigned a unique channel number.
- To view the DMA channels currently in use on the system:

```
$ cat /proc/dma
```

The /sys Directory

- The `/sys` directory is another virtual directory, similar to the `/proc` directory.
- It is created by the kernel in the **sysfs** filesystem format, and it provides additional information about hardware devices that any user on the system can access.
- Many different information files are available within the `/sys` directory.
- They are broken down into subdirectories based on the device and function in the system.

```
$ sudo ls -al /sys
```

Finding Devices

- **lsdev** - display information about installed hardware

  Lsdev

  - ✓ displays information about the hardware devices installed on the Linux system.
  - ✓ It retrieves information from the `/proc/interrupts`, `/proc/ioports`, and `/proc/dma` virtual files and combines them together in one output

Finding Blocks

- **lsblk** - list block devices

```
lsblk [options] [device...]
```

  - ✓ Displays information about the block devices installed on the Linux system.
  - ✓ By default, the **lsblk** command displays all block devices
  - ✓ **-S** displays information only about SCSI block devices on the system

Working with PCI Cards

lspci - list all PCI devices

```
lspci [options]
```

- ✓ **-A** Define the method to access the PCI information
- ✓ **-b** Display connection information from the card point-of-view
- ✓ **-k** Display the kernel driver modules for each installed PCI card
- ✓ **-m** Display information in machine-readable format
- ✓ **-n** Display vendor and device information as numbers instead of text
- ✓ **-q** Query the centralized PCI database for information about the installed PCI cards
- ✓ **-t** Display a tree diagram that shows the connections between cards and buses
- ✓ **-v** Display additional information (verbose) about the cards
- ✓ **-x** Display a hexadecimal output dump of the card information

Working with USB Devices

lsusb - list USB devices

```
lsusb [ options ]
```

- ✓ **-d** Display only devices from the specified vendor ID
- ✓ **-D** Display information only from devices with the specified device file
- ✓ **-s** Display information only from devices using the specified bus
- ✓ **-t** Display information in a tree format, showing related devices
- ✓ **-v** Display additional information about the devices (verbose mode)
- ✓ **-V** Display the version of the `lsusb` program

Hardware Modules

- The Linux kernel needs device drivers to communicate with the hardware devices installed on your Linux system.
- However, compiling device drivers for all known hardware devices into the kernel would make for an extremely large kernel binary file.
- To avoid that situation, the Linux kernel uses kernel **modules**, which are individual hardware driver files that can be linked into the kernel at runtime.
- That way, the system can link only the modules needed for the hardware present on your system.
- If the kernel is configured to load hardware device modules, the individual module files must be available on the system as well.
- If you're compiling a new Linux kernel, you'll also need to compile any hardware modules along with the new kernel.

Hardware Modules

- **Module files may be distributed either as source code that needs to be compiled or as binary object files on the Linux system that are ready to be dynamically linked to the main kernel binary program.**
  - ✓ If the module files are distributed as sourse code files, you must compile them to create the binary object file.
  - ✓ The `.ko` file extension is used to identify the module object files.
  - ✓ The standard location for storing module object files is in the `/lib/modules` directory.
    - This is where the Linux module utilities (such as `insmod` and `modprobe`) look for module object library files by default.

Modules Common Files

- Each kernel has its own directory for its own modules (such as `/lib/modules/4.3.3`), allowing you to create separate modules for each kernel version on the system if needed.
- **/etc/modules** The modules the kernel will load at boot time
- **/etc/modules.conf** The kernel module configurations
- **/lib/modules/*version*/modules.dep** determines the module dependencies

Listing Installed Modules

- **lsmod** - Show the status of modules in the Linux Kernel

```
lsmod
```

- **modinfo** - Show information about a Linux Kernel module

```
modinfo [-0] [-F field] [-k kernel] [modulename|filename...]
```

```
$ modinfo bluetooth
```

Installing New Modules

- **insmod** - Simple program to insert a module into the Linux Kernel

```
insmod [filename] [module options...]
```

  - ✓ The `insmod` command is the most basic, requiring you to specify the exact module file to load.
  - ✓ The downside to using the `insmod` program is that you may run into modules that depend on other modules, and the `insmod` program will fail if those other modules aren't already installed.

Installing New Modules

modprobe - Add and remove modules from the Linux Kernel

```
modprobe [OPTION...] [modulename] [module parameters...]
```

- ✓ **-a** Insert all modules listed on the command line
- ✓ **-C** Specify a different configuration file other than the default
- ✓ **-f** Force the module installation even if there are version issues.
- ✓ **-i** Ignore the install and remove commands specified in the configuration file for the module.
- ✓ **-n** Perform a dry run of the module install to see if it will work, without actually installing it.
- ✓ **-q** Quiet mode—doesn't display any error messages if the module installation or removal fails.
- ✓ **-r** Remove the module listed.
- ✓ **-v** Provide additional information (verbose) as the module is processed.

Removing Modules

- **rmmod** - Simple program to remove a module from the Linux Kernel

```
rmmod [OPTION...] [modulename]
```

  - ✓ Equal to

```
$ modprobe -r [modulename]
```

STORAGE BASICS

Linux handles both HDD and SSD storage devices the same way. It mostly depends on the connection method used to connect the drives to the Linux system.

Types of Drives

- **Parallel Advanced Technology Attachment (PATA)**
  - ✓ connects drives using a parallel interface, which requires a wide cable. PATA supports two devices per adapter.
- **Serial Advanced Technology Attachment (SATA)**
  - ✓ connects drives using a serial interface, but at a much faster speed than PATA. SATA supports up to four devices per adapter.
- **Small Computer System Interface (SCSI)**
  - ✓ connects drives using a parallel interface, but with the speed of SATA. SCSI supports up to eight devices per adapter.

Types of Drives

Types of Drives

- When you connect a drive to a Linux system, the Linux kernel assigns the drive device a file in the `/dev` directory.
- That file is called a raw device, as it provides a path directly to the drive from the Linux system.
- Any data written to the file is written to the drive, and reading the file reads data directly from the drive.
- For PATA devices, this file is named `/dev/hda, /dev/hdb ...`
- For SATA and SCSI devices, Linux uses `/dev/sda, /dev/sdb ...`

Drive Partitions

- Most operating systems, including Linux, allow you to partition a drive into multiple sections.
- A partition is a self-contained section within the drive that the operating system treats as a separate storage space.
- Partitioning drives can help you better organize your data, such as segmenting operating system data from user data.
- If a rogue user fills up the disk space with data, the operating system will still have room to operate on the separate partition.

Drive Partitions

- Partitions must be tracked by some type of indexing system on the drive.
- Master Boot Record (MBR)
  - ✓ Use for the old BIOS boot loader
  - ✓ This method supports only up to four primary partitions on a drive.
  - ✓ Each primary partition itself, however, can be split into multiple extended partitions.
- GUID Partition Table (GPT)
  - ✓ Use for the UEFI boot loader
  - ✓ Supports up to 128 partitions on a drive.

Drive Partitions

- Linux assigns the partition numbers in the order that the partition appears on the drive, starting with number 1
- Linux creates `/dev` files for each separate disk partition.
- It attaches the partition number to the end of the device name and numbers the primary partitions starting at 1.
  - ✓So the first primary partition on the first SATA drive would be /dev/sda1.
- MBR extended partitions are numbered starting at 5
  - ✓So the first extended partition is assigned the file /dev/sda5.

Automatic Drive Detection

- Linux systems detect drives and partitions at boot time and assign each one a unique device filename.
- However, with the invention of removable USB drives (such as memory sticks), which can be added and removed at will while the system is running, that method needed to be modified.
- Most Linux systems now use the `udev` application.
- The `udev` program runs in the background at all times and automatically detects new hardware connected to the running Linux system.
  - ✓ As you connect new drives, `udev` will detect them and assign each one a unique device filename in the **/dev** directory.

Automatic Drive Detection

- Another feature of the `udev` application is that it also creates persistent device files for storage devices.
- When you add or remove a removable storage device, the `/dev` name assigned to it may change, depending on what devices are connected at any given time.
- That can make it difficult for applications to find the same storage device each time.
- To solve that problem, the `udev` application uses the `/dev/disk` directory to create links to the `/dev` storage device files based on unique attributes of the drive.

Automatic Drive Detection

- udev creates four separate directories for storing links:
  - /dev/disk/by-id
    - Links storage devices by their manufacturer make, model, and serial number
  - /dev/disk/by-label
    - Links storage devices by the label assigned to them
  - /dev/disk/by-path
    - Links storage devices by the physical hardware port they are connected to
  - /dev/disk/by-uuid
    - Links storage devices by the 128-bit universally unique identifier (UUID) assigned to the device

STORAGE ALTERNATIVES

Standard partition layouts on storage devices do have their limitations. After you create and format a partition, it’s not easy making it larger or smaller. Individual partitions are also susceptible to disk failures, in which case all data stored in the partition will be lost.

To accommodate more dynamic storage options, as well as fault-tolerance features, Linux has incorporated a few advanced storage management techniques.

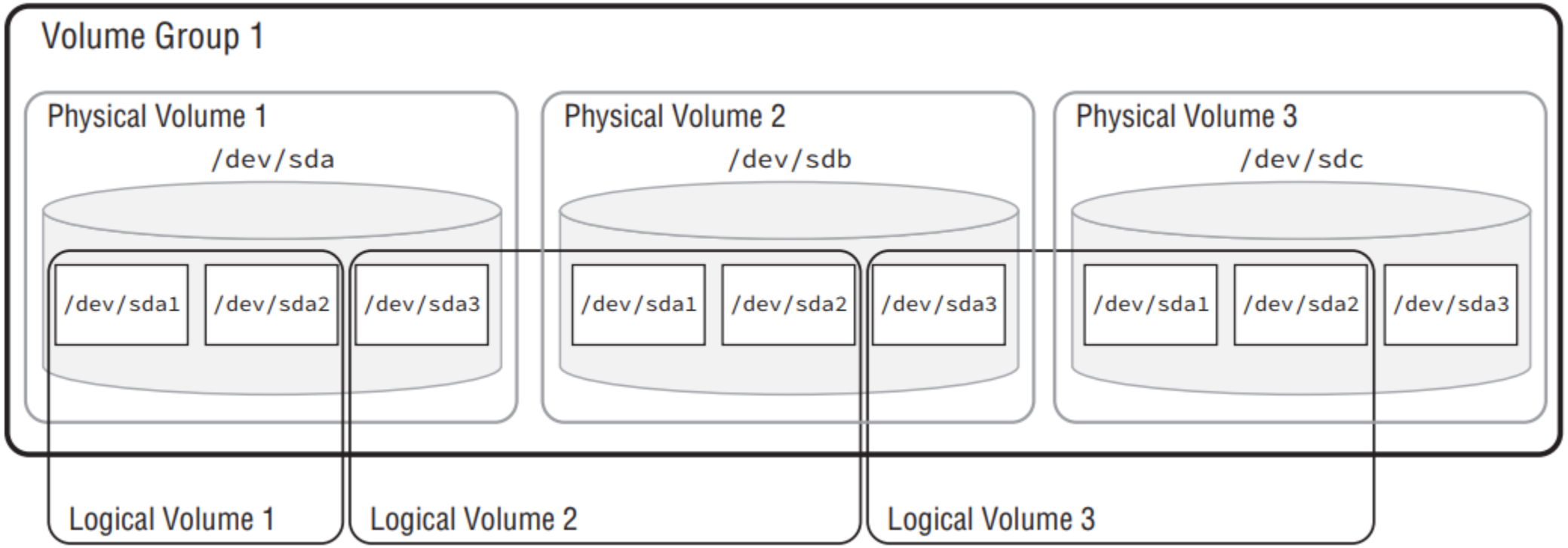
Multipath

- **The Linux kernel now supports Device Mapper (DM) multipathing.**
  - ✓ configure multiple paths between the Linux system and network storage devices.
- **Multipathing aggregates the paths providing for increased throughout while all paths are active, or fault tolerance if one of the paths becomes inactive.**
- **The Linux DM multipathing tools include:**
  - ✓ `dm-multipath:` The kernel module that provides multipath support
  - ✓ `multipath:` A command-line command for viewing multipath devices
  - ✓ `multipathd:` A background process for monitoring paths and activating/deactivating paths
  - ✓ `kpartx`: A command-line tool for creating device entries for multipath storage devices
- **/dev/mapper/mpathN**
  - ✓ where N is the number of the multipath drive.
  - ✓ Acts as a normal device file to the Linux system, allowing you to create partitions and filesystems on the multipath device

Logical Volume Manager (LVM)

- The Linux Logical Volume Manager (LVM) also utilizes the `/dev/mapper` dynamic device directory to allow you to create virtual drive devices.
- You can aggregate multiple physical drive partitions into virtual volumes, which you then treat as a single partition on your system.
- The benefit of `LVM` is that you can add and remove physical partitions as needed to a logical volume, expanding and shrinking the logical volume as needed.
- Using `LVM` is somewhat complicated.

Logical Volume Manager (LVM)



Logical Volume Manager (LVM)

- For each physical partition, you must mark the partition type as the Linux LVM filesystem type in fdisk or gdisk.
- Then, you must use several LVM tools to create and manage the logical volumes:
  - ✓ `pvcreate`: Creates a physical volume
  - ✓ `vgcreate`: Groups physical volumes into a volume group
  - ✓ `lvcreate`: Creates a logical volume from partitions in each physical volume
- The logical volumes create entries in the `/dev/mapper` directory, which represent the LVM device you can format with a filesystem and use like a normal partition.

Using RAID Technology

- **Redundant Array of Inexpensive Disks (RAID)**
- **RAID technology allows you to**
  - ✓ Improve data access performance and reliability,
  - ✓ Implement data redundancy for fault tolerance by combining multiple drives into one virtual drive.
- **The downside is that hardware RAID storage devices can be somewhat expensive (despite what the I stands for), and they are often impractical for most home uses.**
  - ✓ Because of that, Linux has implemented a software RAID system that can implement RAID features on any disk system.

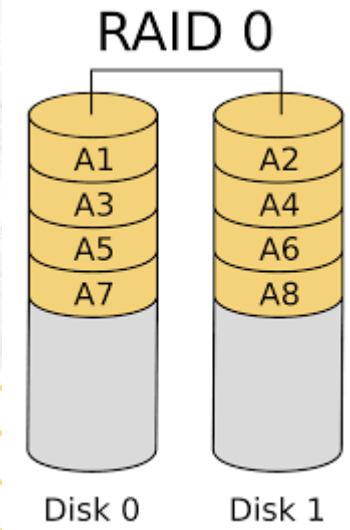
Using RAID Technology

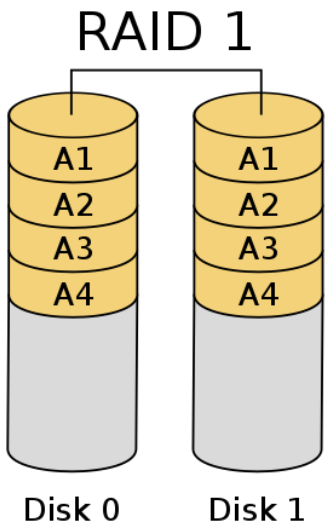
- The `mdadm` utility allows you to specify multiple partitions to be used in any type of RAID environment.
- The RAID device appears as a single device in the `/dev/mapper` directory, which you can then partition and format to a specific filesystem.

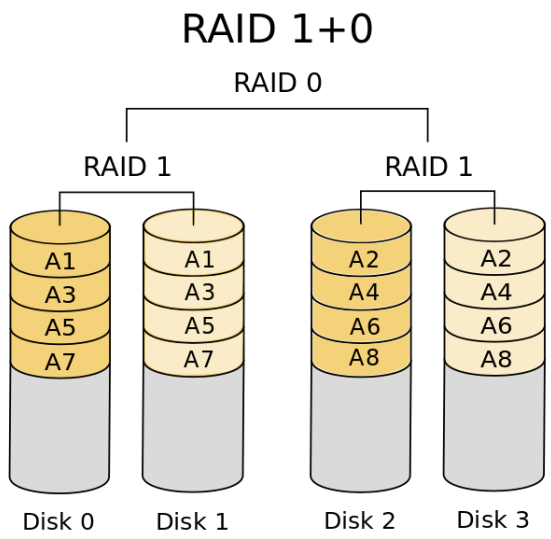
Versions of RAID

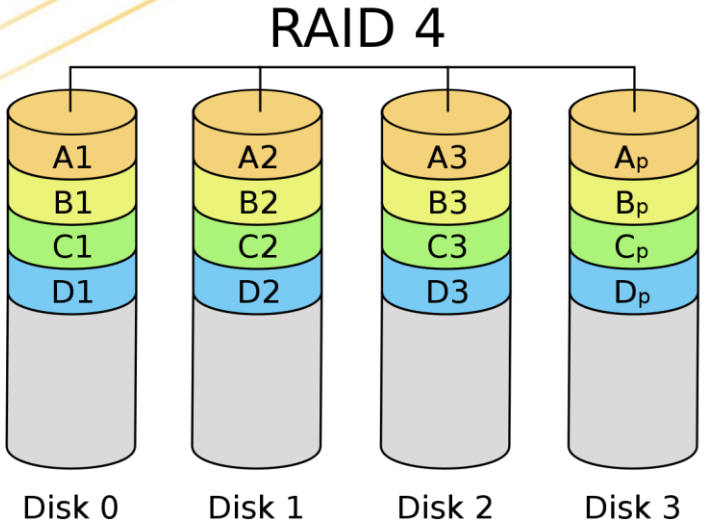
- **RAID 0:** Disk striping, which spreads data across multiple disks for faster access
- **RAID 1:** Disk mirroring, which duplicates data across two drives
- **RAID 10:** Disk mirroring and striping, which provides striping for performance and mirroring for fault tolerance
- **RAID 4:** Disk striping with parity, which adds a parity bit stored on a separate disk so hat data on a failed data disk can be recovered
- **RAID 5:** Disk striping with distributed parity, which adds a parity bit to the data stripe so that it appears on all disks so that any failed disk can be recovered
- **RAID 6:** Disk striping with double parity, which stripes both the data and the parity bit so that two failed drives can be recovered

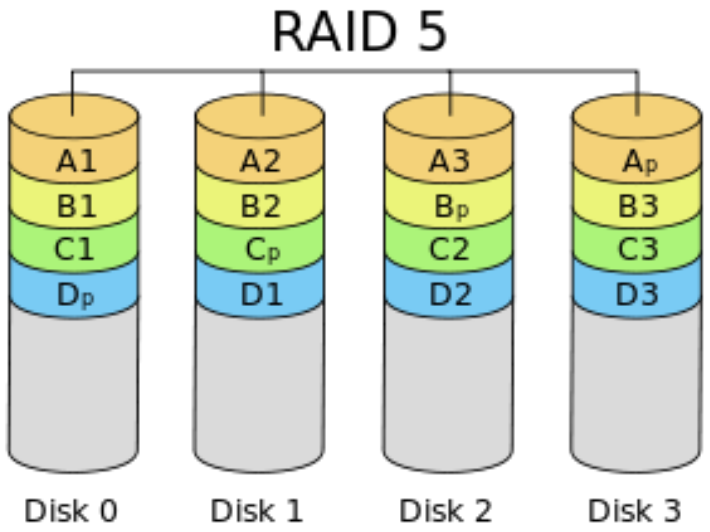
Versions of RAID

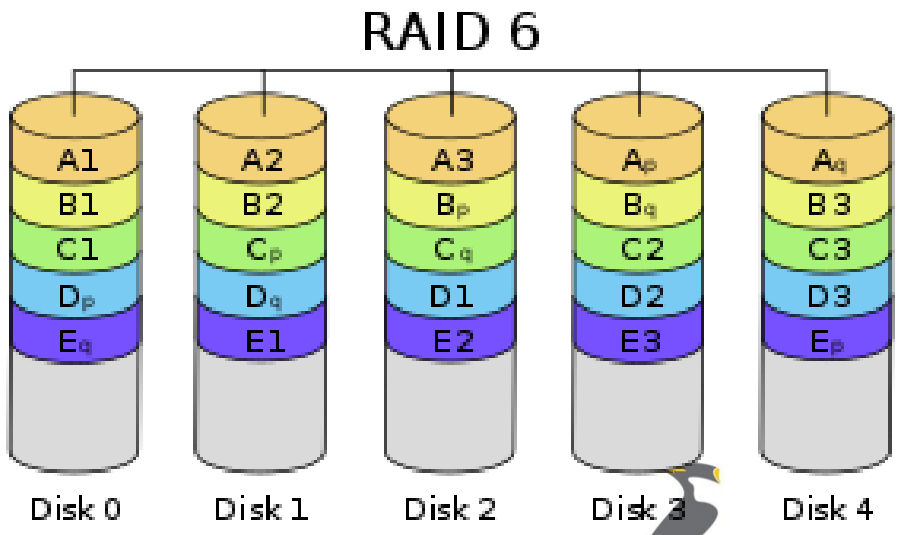












PARTITIONING TOOLS

After you connect a drive to your Linux system you'll need to create partitions on it (even if there's only one partition).

Linux provides several tools for working with raw storage devices to create partitions.

Working with fdisk

- **fdisk** - manipulate disk partition table

```
fdisk [options] device
```

  - ✓ The **fdisk** program allows you to create, view, delete, and modify partitions on any drive that uses the MBR method of indexing partitions.
  - ✓ The **fdisk** command is somewhat rudimentary in that it doesn't allow you to alter the size of an existing partition
    - All you can do is delete the existing partition and rebuild it from scratch.

Working with fdisk

- `a` Toggle a bootable flag
- `b` Edit BSD disk label
- `c` Toggle the DOS compatibility flag
- `d` Delete a partition
- `g` Create a new empty GPT partition table
- `G` Create an IRIX (SGI) partition table
- `l` List known partition types
- `m` Print this menu
- `n` Add a new partition
- `o` Create a new empty DOS partition table
- `p` Print the partition table
- `q` Quit without saving changes
- `s` Create a new empty Sun disk label
- `t` Change a partition's system ID
- `u` Change display/entry units
- `v` Verify the partition table
- `w` Write table to disk and exit
- `x` Extra functionality (experts only)

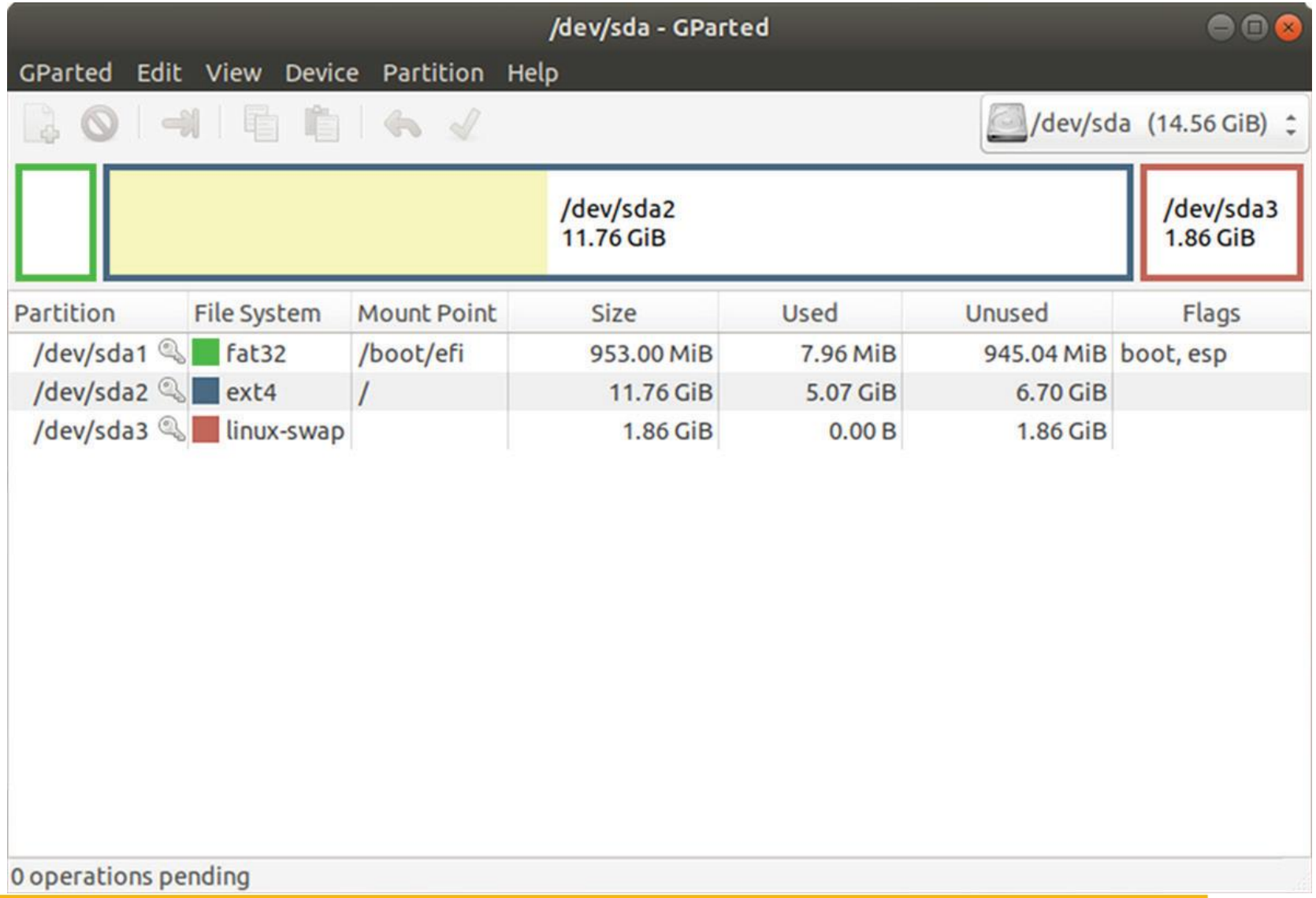
The GNU parted Command

- **GNU Parted** - a partition manipulation program

```
parted [options] [device [command [options...]...]]
```

  - ✓ It allows you to modify existing partition sizes
    - So you can easily shrink or grow partitions on the drive.
  - ✓ `$ sudo parted`

Graphical Tools



UNDERSTANDING FILESYSTEMS

Just like storing stuff in a closet, storing data in a Linux system requires some method of organization to be efficient. Linux utilizes filesystems to manage data stored on storage devices. A filesystem maintains a map to locate each file placed in the storage device.

Understanding Filesystems

- Windows assigns drive letters to each storage device you connect to the system.

```
C:\Users\rich\Documents\test.docx
```

  - ✓The Windows path tells you exactly what physical device the file is stored on.

- Linux, however, **doesn't** use this method to reference files.

  - ✓It uses a virtual directory structure.

- The virtual directory contains file paths from all the storage devices installed on the system, consolidated into a single directory structure

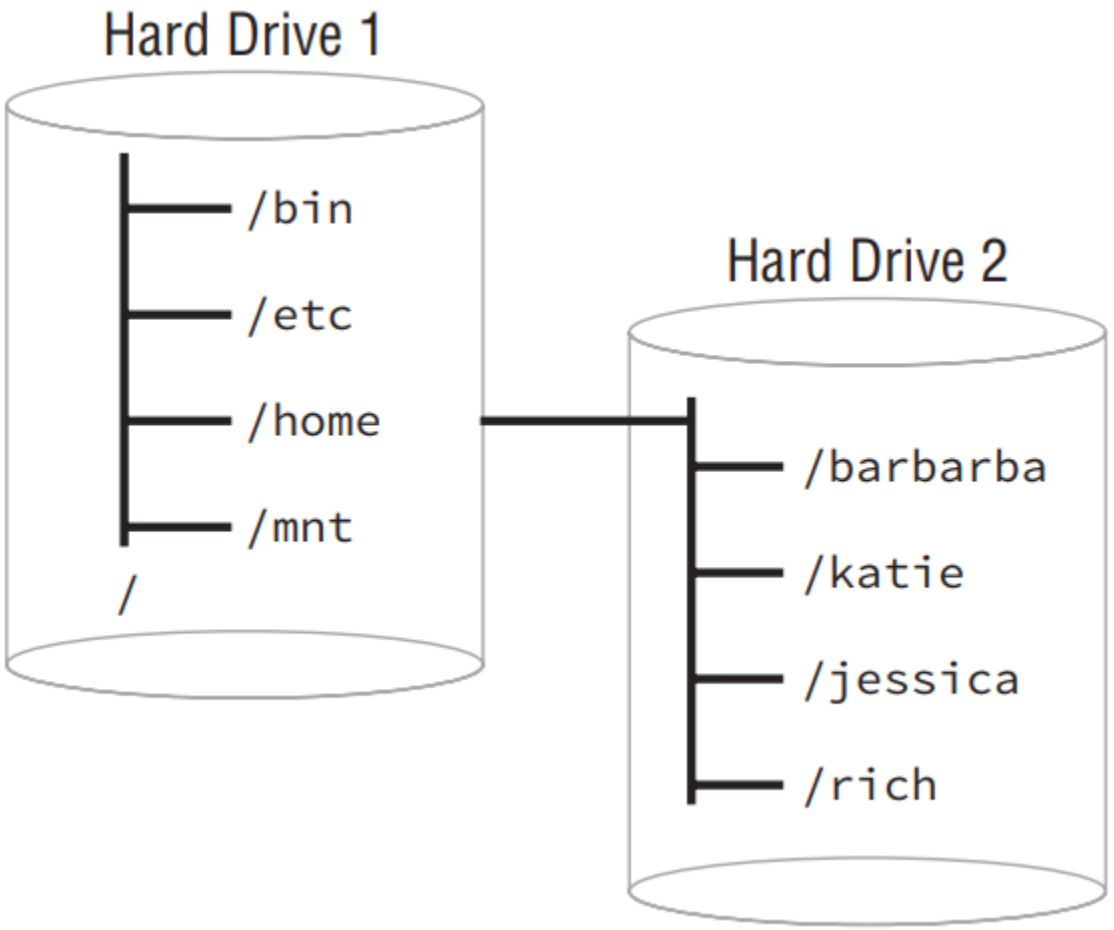
The Virtual Directory

- The Linux virtual directory structure contains a single base directory, called the root directory.
- The root directory lists files and directories beneath it based on the directory path used to get to them.

```
/home/rich/Documents/test.doc
```

- It doesn't give you any clues as to which physical device contains the file.
- Linux places physical devices in the virtual directory using **mount points**.
  - ✓ A mount point is a directory placeholder within the virtual directory that points to a specific physical device.

The Virtual Directory and Physical Devices



Filesystem Hierarchy Standard (FHS)

- A standard format has been defined for the Linux virtual directory called the Linux **filesystem hierarchy standard (FHS)**.
- The FHS defines core directory names and locations that should be present on every Linux system, as well as what type of data they should contain.

Common Linux FHS directories

- **/boot** Contains boot loader files used to boot the system
- **/etc** Contains system and application configuration files
- **/home** Contains user data files
- **/media** Used as a mount point for removable devices
- **/mnt** Also used as a mount point for removable devices
- **/opt** Contains data for optional third-party programs
- **/tmp** Contains temporary files created by system users
- **/usr** Contains data for standard Linux programs
- **/usr/bin** Contains local user programs and data
- **/usr/local** Contains data for programs unique to the local installation
- **/usr/sbin** Contains data for system programs and data
- **/var** Contains variable data files, including system and application logs

Maneuvering Around the Filesystem

- Using the virtual directory makes it a breeze to move files from one physical device to another.
  - ✓ You don't need to worry about drive letters—just the locations within the virtual directory:

```
$ cp /home/rich/Documents/myfile.txt /media/usb
```

FORMATTING FILESYSTEMS

Before you can assign a drive partition to a mount point in the virtual directory, you must format it using a filesystem.

There are numerous filesystem types that Linux supports, each with different features and capabilities.

Linux Filesystems

❖ btrfs:

- ✓ Supports files up to 16 exbibytes (EiB) in size, and a total filesystem size of 16 EiB.
- ✓ Perform its own form of RAID as well as LVM and subvolumes.
- ✓ Built-in snapshots for backup
- ✓ Improved fault tolerance
- ✓ data compression on the fly

❖ Ecryptfs (Enterprise Cryptographic Filesystem):

- ✓ Applies a Portable Operating System Interface (POSIX)–compliant encryption protocol to data before storing it on the device.
- ✓ This provides a layer of protection for data stored on the device.
- ✓ Only the operating system that created the filesystem can read data from it.

Linux Filesystems

ext3 (ext3fs):

- ✓ Descendant of the original Linux `ext` filesystem.
- ✓ Supports files up to 2 tebibytes (TiB), with a total filesystem size of 16 TiB.
- ✓ Supports journaling, as well as faster startup and recovery.

ext4 (ext4fs):

- ✓ The current version of the original Linux filesystem.
- ✓ Supports files up to 16 TiB, with a total filesystem size of 1 EiB.
- ✓ Supports journaling and utilizes improved performance features.
- ✓ Default filesystem used by most Linux distributions.

Linux Filesystems

reiserFS:

- ✓ Created before the Linux ext3fs filesystem and commonly used on older Linux systems
- ✓ Provides features now found in ext3fs and ext4fs.
- ✓ Linux has dropped support for the most recent version, reiser4fs.

swap:

- ✓ The swap filesystem allows you to create virtual memory for your system using space on a physical drive.
- ✓ The system can then swap data out of normal memory into the swap space, providing a method of adding additional memory to your system.
- ✓ This is not intended for storing persistent data

journaling

- `journaling` is a method of tracking data not yet written to the drive in a log file, called the journal.
- If the system fails before the data can be written to the drive, the journal data can be recovered and stored upon the next system boot.

Non-Linux Filesystems

- **CIFS** Common Internet Filesystem (CIFS)
- **exFAT** The Extended File Allocation Table
- **HFS** The Hierarchical Filesystem (HFS)
- **ISO-9660**
- **NFS** The Network Filesystem (NFS)
- **NTFS** The New Technology Filesystem (NTFS)
- **SMB** The Server Message Block (SMB)
- **UDF** The Universal Disk Format (UDF)
- **VFAT** The Virtual File Allocation Table (VFAT)
- **XFS** The X Filesystem (XFS)
- **ZFS** The Zettabyte Filesystem (ZFS)

Creating Filesystems

- **mkfs** - build a Linux filesystem

```
mkfs [options] [-t type] [fs-options] device [size]
```

```
$ sudo mkfs -t ext4 /dev/sdb1
```

MOUNTING FILESYSTEMS

After you've formatted a drive partition with a filesystem, you can add it to the virtual directory on your Linux system. This process is called mounting the filesystem.

You can either manually mount the partition within the virtual directory structure from the command line or allow Linux to automatically mount the partition at boot time.

Manually Mounting Devices

- **mount** - mount a filesystem

```
mount -t fstype device mountpoint
```

```
$ sudo mount -t ext4 /dev/sdb1 /media/usb1
```

- ✓ The downside to the mount command is that it only temporarily mounts the device in the virtual directory.
- ✓ When you reboot the system, you have to manually mount the devices again.

- **umount** - unmount a filesystem

```
umount [device] [mountpoint]
```

Automatically Mounting Devices

- `/etc/fstab`
  - ✓ Indicate which drive devices should be mounted to the virtual directory at boot time.
  - ✓ Is a table that indicates
    - The drive device file (either the raw file or one of its permanent `udev` filenames)
    - The mount point location
    - The filesystem type, ...
  - ✓ Universally Unique Identifier (**UUID**)

```
$ cat /etc/fstab
```

MANAGING FILESYSTEMS

After you've created a filesystem and mounted it to the virtual directory, you may have to manage and maintain it to keep things running smoothly.

Retrieving Filesystem Stats

- **df** - report file system disk space usage

```
df [OPTION]... [FILE]...
```

- ✓ **-h, --human-readable** print sizes in powers of 1024
- ✓ **-H, --si** print sizes in powers of 1000
- ✓ **-T, --print-type** print file system type
- ✓ **-i, --inodes** list inode information instead of block usage

Retrieving Filesystem Stats

du - estimate file space usage

```
du [OPTION]... [FILE]...
```

- ✓ Produces text usage report (in kilobytes) by directory
- ✓ Lists size of every file in all sub-directories by default

✓ **-a, --all**	write counts for all files, not just directories
✓ **-c, --total**	produce a grand total
✓ **-h, --human-readable**	print sizes in human readable format
✓ **-s, --summarize**	display only a total for each argument

Retrieving Filesystem Stats

- **iostat**: Displays a real-time chart of disk statistics by partition
- **lsblk**: Displays current partition sizes and mount points
- `/proc/partitions`
- `/proc/mounts`

Filesystem Tools

- **e2fsprogs (ext Filesystem Tools):**
  - ✓ `blkid`: Display information about block devices, such as storage drives
  - ✓ `chattr`: Change file attributes on the filesystem
  - ✓ `debugfs`: Manually view and modify the filesystem structure, such as undeleting a file or extracting a corrupted file
  - ✓ `dumpe2fs`: Display block and superblock group information
  - ✓ `e2label`: Change the label on the filesystem
  - ✓ `resize2fs`: Expand or shrink a filesystem
  - ✓ `tune2fs`: Modify filesystem parameters

Filesystem Tools

XFS Filesystem Tools

- `xfs_admin`: Display or change filesystem parameters such as the label or UUID assigned
- `xfs_db`: Examine and debug an XFS filesystem
- `xfs_fsr`: Improve the organization of mounted filesystems
- `xfs_info`: Display information about a mounted filesystem, including the block sizes and sector sizes, as well as label and UUID information

`xfs_repair`: Repair corrupted or damaged XFS filesystems

Filesystem Tools

fsck - check and repair a Linux filesystem

- The fsck program is a front end to several different programs that check the various filesystems to match the index against the actual files stored in the filesystem.
  - If any discrepancies occur, run the fsck program in repair mode, and it will attempt to reconcile the discrepancies and fix the filesystem.
  - If the fsck program is unable to repair the drive on the first run, try running it again a few times to fix any broken files and directory links.
  - If running the fsck program multiple times doesn't repair the drive, you may have to resort to reformatting the drive and losing any data on it.

```
$ sudo fsck -f /dev/sdb1
```