Mounting a filesystem means making the data on a storage device (like a hard drive partition, USB stick, or network share) accessible to the operating system as part of its main directory tree.

Think of it like putting a new book on an empty shelf in your library. Before you do, the book is a self-contained object. By placing it on a specific shelf, you integrate it into your library's organization, and you can now find it at a known location.

The Detailed Process from A to Z

A: The Unmounted State

Imagine you've just partitioned a new hard drive. This partition (/dev/sdb1) has been formatted with a filesystem, like EXT4. At this point, the filesystem exists on the device, complete with its own internal structure (inodes, data blocks, and a superblock), but the main operating system has no idea how to access it. It's like a library that has received a new box of books but hasn't unpacked it yet. The books are there, but they aren't on the shelves.

B: The Mount Point

To mount the filesystem, you need a **mount point**. A mount point is simply an existing, empty directory on your main filesystem. This directory acts as the entry point or "doorway" to the new filesystem's contents. For example, you might create a directory like /mnt/data.

C: The mount Command

This is where the magic happens. A user with sufficient privileges runs the mount command, telling the kernel to attach the device's filesystem to the mount point.

Bash

```
sudo mount /dev/sdb1 /mnt/data
```

Here's what the kernel does in response:

1. **Identify Filesystem Type**: The kernel examines the device (/dev/sdb1) to determine its filesystem type (e.g., EXT4, XFS, NTFS). It does this by reading the initial blocks of the partition, which contain a "magic number" identifying the filesystem.
2. **Load Filesystem Driver**: The kernel loads the appropriate driver for that filesystem type into memory if it isn't already loaded. This driver is the software that knows how to read and write the specific on-disk structure of that filesystem.
3. **Read the Superblock**: The driver reads the **superblock** from the device. The superblock is a critical part of the filesystem that contains its metadata, such as the block size, total number of inodes, and whether the filesystem is "clean" (was unmounted properly last time). If the superblock is corrupted, the mount will likely fail.
4. **Attach to the VFS**: The Linux kernel uses a **Virtual File System (VFS)** layer. The VFS provides a single, uniform interface for all filesystems. The mount operation tells the VFS to attach the root directory of the new filesystem (/dev/sdb1) to the mount point directory (/mnt/data).

D: The Mounted State

After the mount command succeeds:

- The original contents of /mnt/data (if any) are temporarily hidden.
- Any request to access /mnt/data is now seamlessly redirected by the VFS to the root of the filesystem on /dev/sdb1.
- You can navigate the new filesystem just like any other part of your system (e.g., ls /mnt/data, cd /mnt/data/photos, touch /mnt/data/newfile.txt). The VFS and the EXT4 driver handle all the translation between your commands and the physical blocks on the disk.

E: Automating with /etc/fstab

To avoid manually mounting devices every time you boot, you use the /etc/fstab (file system table) file. This file contains a list of devices, their mount points, and options, instructing the

system to automatically mount them during the boot process.

A typical entry looks like this:
/dev/sdb1 /mnt/data ext4 defaults 0 2

Z: Unmounting

When you're finished with the device, you should **unmount** it. This is the process of safely detaching the filesystem from the VFS.

Bash

```
sudo umount /mnt/data
```

The kernel ensures all pending data is written to the device (flushing the cache), updates the superblock to mark the filesystem as "clean," and then detaches it from the mount point. The /mnt/data directory becomes an empty directory again. Properly unmounting is crucial to prevent data corruption.