Linux File System - GeeksforGeeks

geeksforgeeks.org/linux-file-system/

Suggest changes

Like Article

Like

Share

Report

Follow

Operating systems, the software that powers your computer, rely on a crucial element known as the file system. Think of it as a virtual organizational tool that manages, stores, and retrieves your data efficiently. In the Linux world, a diverse range of file systems has emerged, each crafted to address specific needs and preferences. This article aims to simplify the intricacies of Linux file systems, guiding beginners through their layers, characteristics, and implementations. By shedding light on these nuances, we empower users to make informed choices in navigating the dynamic landscape of Linux operating systems.

Table of Content

What is the Linux File System

The Linux file system is a multifaceted structure comprised of three essential layers. At its foundation, the Logical File System serves as the interface between user applications and the file system, managing operations like opening, reading, and closing files. Above this, the Virtual File System facilitates the concurrent operation of multiple physical file systems, providing a standardized interface for compatibility. Finally, the Physical File System is responsible for the tangible management and storage of physical memory blocks on the disk, ensuring efficient data allocation and retrieval. Together, these layers form a cohesive architecture, orchestrating the organized and efficient handling of data in the Linux operating system.

In this article, we will be focusing on the **file system for hard disks on a Linux OS**and discuss which type of file system is suitable. The architecture of a file system comprises three layers mentioned below.

Linux File System Structure

A file system mainly consists of 3 layers. From top to bottom:

1. Logical File System:

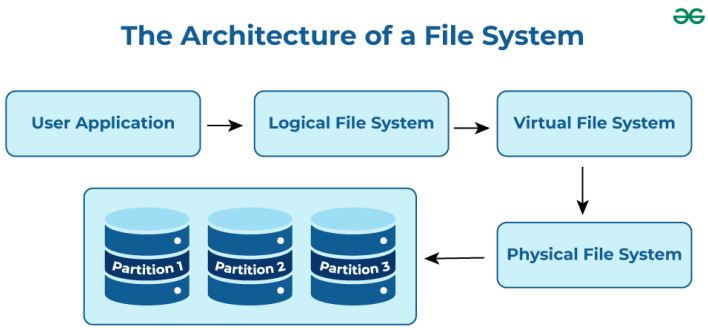
The Logical File System acts as the interface between the user applications and the file system itself. It facilitates essential operations such as opening, reading, and closing files. Essentially, it serves as the user-friendly front-end, ensuring that applications can interact with the file system in a way that aligns with user expectations.

2. Virtual File System:

The Virtual File System (VFS) is a crucial layer that enables the concurrent operation of multiple instances of physical file systems. It provides a standardized interface, allowing different file systems to coexist and operate simultaneously. This layer abstracts the underlying complexities, ensuring compatibility and cohesion between various file system implementations.

3. Physical File System:

The Physical File System is responsible for the tangible management and storage of physical memory blocks on the disk. It handles the low-level details of storing and retrieving data, interacting directly with the hardware components. This layer ensures the efficient allocation and utilization of physical storage resources, contributing to the overall performance and reliability of the file system.



Architecture Of a File System

Characteristics of a File System

- **Space Management**: how the data is stored on a storage device. Pertaining to the memory blocks and fragmentation practices applied in it.
- **Filename**: a file system may have certain restrictions to file names such as the name length, the use of special characters, and case sensitive-ness.
- **Directory**: the directories/folders may store files in a linear or hierarchical manner while maintaining an index table of all the files contained in that directory or subdirectory.
- **Metadata**: for each file stored, the file system stores various information about that file's existence such as its data length, its access permissions, device type, modified date-time, and other attributes. This is called metadata.
- **Utilities**: file systems provide features for initializing, deleting, renaming, moving, copying, backup, recovery, and control access of files and folders.
- **Design**: due to their implementations, file systems have limitations on the amount of data they can store.

Some important terms:

1) Journaling:

Journaling file systems keep a log called the journal, that keeps track of the changes made to a file but not yet permanently committed to the disk so that in case of a system failure the lost changes can be brought back.

2) Versioning:

Versioning file systems store previously saved versions of a file, i.e., the copies of a file are stored based on previous commits to the disk in a minutely or hourly manner to create a backup.

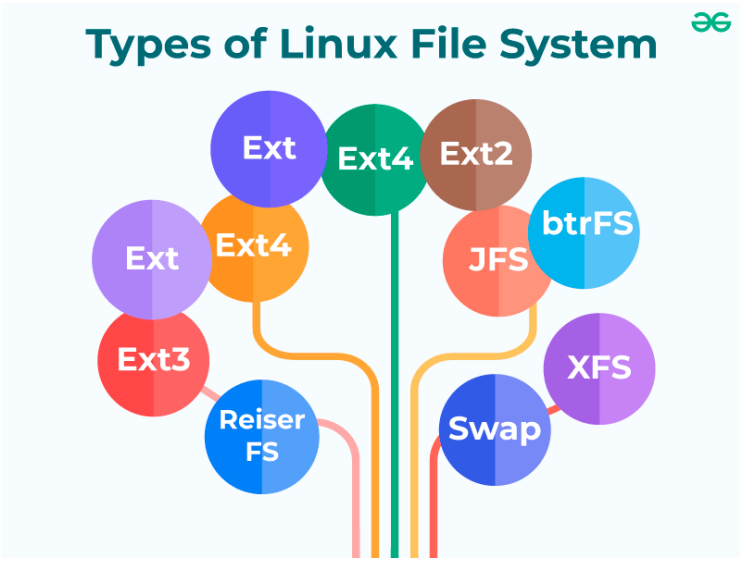
3) Inode:

The index node is the representation of any file or directory based on the parameters – size, permission, ownership, and location of the file and directory.

Now, we come to part where we discuss the various implementations of the file system in Linux for disk storage devices.

Linux File Systems:

Note: *Cluster and distributed file systems will not be included for simplicity.*



1) ext (Extended File System):

Implemented in 1992, it is the first file system specifically designed for Linux. It is the first member of the ext family of file systems.

2) ext2:

The second ext was developed in 1993. It is a non-journaling file system that is preferred to be used with flash drives and SSDs. It solved the problems of separate timestamp for access, inode modification and data modification. Due to not being journaled, it is slow to load at boot time.

3) Xiafs:

Also developed in 1993, this file system was less powerful and functional than ext2 and is no longer in use anywhere.

4) ext3:

The third ext developed in 1999 is a journaling file system. It is reliable and unlike ext2, it prevents long delays at system boot if the file system is in an inconsistent state after an unclean shutdown. Other factors that make it better and different than ext2 are online file system growth and HTree indexing for large directories.

5) JFS (Journaled File System):

First created by IBM in 1990, the original JFS was taken to open source to be implemented for Linux in 1999. JFS performs well under different kinds of load but is not commonly used anymore due to the release of ext4 in 2006 which gives better performance.

6) ReiserFS:

It is a journal file system developed in 2001. Despite its earlier issues, it has tail packing as a scheme to reduce internal fragmentation. It uses a B+ Tree that gives less than linear time in directory lookups and updates. It was the default file system in SUSE Linux till version 6.4, until switching to ext3 in 2006 for version 10.2.

7) XFS:

XFS is a 64-bit journaling file system and was ported to Linux in 2001. It now acts as the default file system for many Linux distributions. It provides features like snapshots, online defragmentation, sparse files, variable block sizes, and excellent capacity. It also excels at parallel I/O operations.

8) SquashFS:

Developed in 2002, this file system is read-only and is used only with embedded systems where low overhead is needed.

9) Reiser4:

It is an incremental model to ReiserFS. It was developed in 2004. However, it is not widely adapted or supported on many Linux distributions.

10) ext4:

The fourth ext developed in 2006, is a journaling file system. It has backward compatibility with ext3 and ext2 and it provides several other features, some of which are persistent pre-allocation, unlimited number of subdirectories, metadata checksumming and large file size. ext4 is the default file system for many Linux distributions and also has compatibility with Windows and Macintosh.

11) btrfs (Better/Butter/B-tree FS):

It was developed in 2007. It provides many features such as snapshotting, drive pooling, data scrubbing, self-healing and online defragmentation. It is the default file system for Fedora Workstation.

12) bcachefs:

This is a copy-on-write file system that was first announced in 2015 with the goal of performing better than btrfs and ext4. Its features include full filesystem encryption, native compression, snapshots, and 64-bit check summing.

13) Others:

Linux also has support for file systems of operating systems such as NTFS and exFAT, but these do not support standard Unix permission settings. They are mostly used for interoperability with other operating systems.

Below is a table, listing out the criteria on which filesystems can be compared:

Please note that there are more criteria than the ones listed in the table. This table is supposed to give you an idea of how file systems have evolved.

	File Systems									
Parameters	**ext**	**ext2**	**Xiafs**	**ext3**	**JFS**	**ReiserFS**	**XFS**	**Reiser4**	**ext4**	**btrfs**
Max. filename length (bytes)	255	255	248	255	255	4032 255 characters	255	3976	255	255
Allowable characters in directory entries (Any byte)	except NUL	except NUL, /	except NUL	except NUL or /	Any Unicode except NUL	except NUL or /	except NUL	except NUL, /	except NUL, /	except NUL, /
Max. pathname length	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefi
Max. file size	2 GB	16GB – 2TB	64MB	16GB – 2TB	4PB	8TB	8EB	8TB (on x86)	16GB – 16TB	16EB
Max. volume size	2 GB	2TB – 32TB	2GB	2TB – 32TB	32PB	16TB	8EB	–	1EB	16EB
Max. no. of files	–	–	–	–	–	–	–	–	2^32	2^64
Metadata only journaling	No	No	No	Yes	Yes	Yes	Yes	No	Yes	No
Compression	No	No	No	No	No	No	No	Yes	No	Yes
Block sub-allocation	No	No	No	No	Yes	Yes	No	Yes	No	Yes
Online grow	No	No	–	Yes	No	Yes	Yes	Yes	Yes	Yes
Encryption	No	No	No	No	No	No	No	Yes	Yes (experimental)	No
Checksum	No	No	No	No	No	No	Partial	No	Partial	Yes

Observations:

We see that XFS, ext4 and btrfs perform the best of all the other file systems. In fact, btrfs looks as if it's almost the best. Despite that, the ext family of file systems has been the default for most Linux distributions for a long time. So, what is it that made the developers choose ext4 as the default rather than btrfs or XFS? Since ext4 is so important for this discussion, let's describe it a bit more.

ext4 in Linux File System

Ext4 was designed to be backward compatible with ext3 and ext2, its previous generations. It's better than the previous generations in the following ways:

- It provides a **large file system** as described in the table above.
- Utilizes **extents** that improve large file performance and reduces fragmentation.
- Provides **persistent pre-allocation** which guarantees space allocation and contiguous memory.
- **Delayed allocation** improves performance and reduces fragmentation by effectively allocating larger amounts of data at a time.
- It uses HTree indices to allow **unlimited number of subdirectories.**
- Performs **journal checksumming** which allows the file system to realize that some of its entries are invalid or out of order after a crash.
- Support for time-of-creation timestamps and **improved timestamps** to induce granularity.
- Transparent encryption.
- Allows cleaning of inode tables in background which in turn speeds initialization. The process is called **lazy initialization**.
- Enables **writing barriers** by default. Which ensures that file system metadata is correctly written and ordered on disk, even when write caches lose power.

There are still some features in the process of developing like metadata checksumming, first-class quota supports, and large allocation blocks.

However, ext4 has some limitations. Ext4 does not guarantee the integrity of your data, if the data is corrupted while already on disk then it has no way of detecting or repairing such corruption. The ext4 file system cannot secure deletion of files, which is supposed to cause overwriting of files upon deletion. It results in sensitive data ending up in the file-system journal.

XFS performs highly well for large filesystems and high degrees of concurrency. So XFS is stable, yet there's not a solid borderline that would make you choose it over ext4 since both work about the same. Unless you want a file system that directly solves a problem of ext4 like having capacity > 50TiB.

Btrfs on the other hand, despite offering features like multiple device management, per-block checksumming, asynchronous replication and inline compression, does not perform the best in many common use cases as compared to ext4 and XFS. Several of its features can be buggy and result in reduced performance and data loss.

Some HandsOn Example on Linux File System

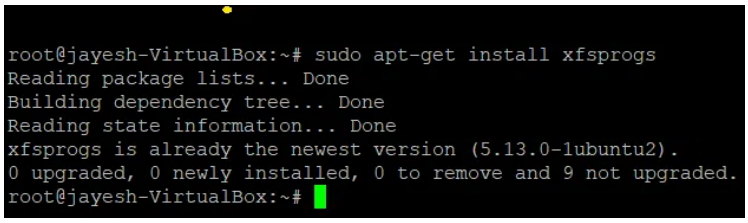
For example, if our use_case is to set up a server that will first store and serve large multimedia files (videos and audios). In that case we have to prioritize efficient speed and use of storage space.

According to this requirement the XFS file system would be a better choice. Because we know that XFS is optimized for large files and can work on high volumes of data transfer which in general makes it ideal for media servers.

Following steps to use it:

Step 1: Installing XFS utilities package on Linux system.

```
sudo apt-get install xfsprogs
```



Installing xfsprogs

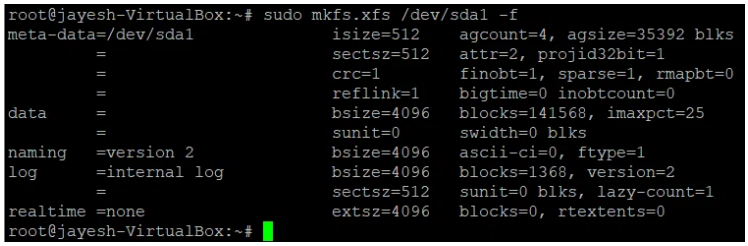
Step 2: Create a partition to format as XFS.

For example: `/dev/sda1`

This can be done using tool like `fdisk`.

Step 3: Format the partition as XFS.

```
sudo mkfs.xfs /dev/sda1 -f
```

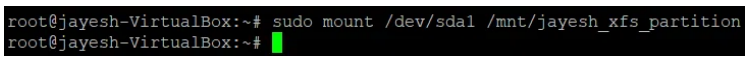


Format the partition

We have formatted partition using XFS filesystem. (Used -f for forcefully to avoid error or warning) **.**

Step 4: Mount the XFS partition to a directory we want.

```
sudo mount /dev/sda1 /mnt/jayesh_xfs_partition
```

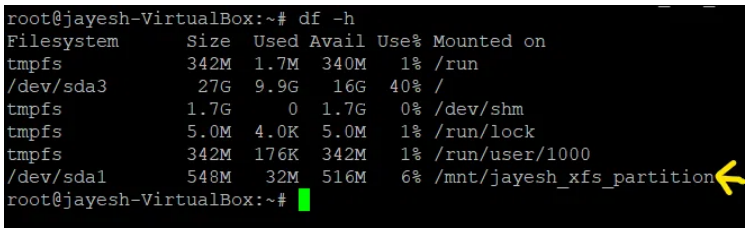


mounting of XFS partition

We have mounted XFS partition to a directory `/mnt/jayesh_xfs_partition`, (you can create your own directory.)

Step 5: To verify the mount.

```
df -h
```



Successful mount

Conclusion:

In this article we discussed Linux file system in operating systems, delving into its layers, characteristics, and the architecture of Linux file systems. It provides a thorough exploration of various options, from ext to contemporary choices like ext4, XFS, and btrfs. The comparison table highlights the superior performance of XFS, ext4, and btrfs, with ext4 standing out for its backward compatibility and design enhancements. The article wisely recommends ext4 as the default for general users unless specific needs dictate alternatives, citing instances where XFS excels for large media files. In essence, the article serves as a practical guide for users to navigate the complexities of file systems, emphasizing the reliable nature of ext4 for most use cases while acknowledging niche applications for other systems.