Understanding Linux file systems

Linux supports a number of different file systems with different sets of features and intended use-cases.

Ext4: The All-rounder

Ext4 is the latest iteration of the "Extended file system" and is the default choice for most major Linux distributions. As the direct descendant of Ext2 and Ext3, it represents decades of battle-tested development, making it one of the most mature, reliable, and stable file systems for Linux.

Ext4 keeps a dedicated log (or journal) of files that are about to be written to the disk. Once the data is safely committed, it is removed from the journal. This improves file system integrity and prevents data corruption in the event of an unexpected power loss or system crash.

Instead of writing data to the drive immediately, Ext4 buffers the data in system memory and allocates the physical disk blocks just before writing them to storage. This feature improves overall write performance and helps maximize the lifespan of flash-based storage devices, like SSDs.

Ext4 actively optimizes how files are scattered across the drive to prevent file fragmentation, which benefits slower storage devices like HDDs by minimizing seek times of the drive head.

Since Ext4 is mature and battle-tested, tooling around it is as reliable as it gets. There exist a myriad of data recovery tools in the case something *does* go wrong. The file system is also very flexible in that it can be expanded or shrunk as needed.

When you want it: You just need a reliable, "set-it-and-forget-it" file system that works well without any manual tuning. Even with its old heritage, it's still a good choice for everyday desktop users, gamers, older computers, and external backup drives. Choose Ext4 if you need low system overhead and want the flexibility to safely shrink your partition later if your storage needs change.

When you don't want it: You need advanced, modern storage features. If you want to create instant system-wide snapshots before updating your software, pool multiple different-sized hard drives together, or use transparent data compression to save space on an SSD.

Btrfs: The new kid on the block

Btrfs is a new type of Linux file system that is designed differently from Ext4 in some respects. Where Ext4 is built around simplicity, btrfs shines with modern storage features.

Btrfs is a copy-on-write (CoW for short) file system, which means that copies of files are only "virtual" and do not occupy any additional storage space, and a copy only becomes "real" once it has been changed. Writes do not overwrite data in place; instead, a modified copy of the block is written to a new location, and metadata is updated to point at the new location.

Btrfs organizes its data in subvolumes, which can be mounted like partitions. Unlike partitions, subvolumes do not have a fixed size. Instead, subvolumes are merely an organizational unit on the same Btrfs partition, the size of which depends on the contents stored in them. Any number of subvolumes can be created for different mount points, e.g. `/` and `/home`. This allows, amongst other things, for multiple operating systems to be installed to the same disk on the same computer without interfering with each other.

Another feature of Btrfs is its ability to create snapshots of the file system. The state of a subvolume can be recorded in a snapshot, e.g. before a critical system update, in order to revert to a previous state of the file system if necessary. Thanks to CoW, snapshots require very little storage space compared to full-fledged backups (although they are no replacement for them!) and can be mounted and booted from like regular subvolumes. This makes it possible to "rewind" the state of the file system with comparatively little effort. Tools such as `snapper` or `timeshift` can simplify and automate the process of creating snapshots during system updates and restoring from snapshots from the commandline.

Btrfs also implements transparent compression of data blocks. Written data is automatically stored in compressed form if the appropriate mount options are set. There are a number of different compression algorithms to choose from, including lz4, gzip and Zstandard. This can also increase the life span of flash based storage devices, as less data is written to the disk and not as much wear-leveling is taking place.

Btrfs comes with RAID management for RAID 0, 1 and 10 built into the file system itself and makes an additional software or firmware RAID superfluous for these configurations. In addition, the integrated RAID functionality offers the advantage that it is aware of used and free data blocks in mirrored setups, which can considerably speed up the reconstruction of a RAID, as only the used blocks are reconstructed. Using the built-in RAID functionality in Btrfs also allows for more storage devices to be added to the RAID later on.

When you want it: You love to tinker with your system, run rolling-release Linux distributions (like Arch or openSUSE Tumbleweed), or want bulletproof peace of mind. Its instant snapshotting turns a system-breaking update into a ten-second rollback. It is also ideal if you want to squeeze more space out of your SSD via built-in compression or don't want to agonize over the correct partition sizes for your `/` and `/home` partitions and instead share storage dynamically between them.

When you don't want it: Your daily workflow involves heavy database management (like MySQL or PostgreSQL) or hosting several virtual machine images. Because of its Copy-on-Write mechanics, frequently modified files inside VMs and databases will heavily fragment the drive, dragging down performance. You should also avoid it if you plan on using RAID 5 or RAID 6 storage arrays, as the Btrfs implementation for those specific modes remains notoriously unstable.

XFS: The Enterprise Heavyweight

Originally developed by Silicon Graphics (SGI) in 1993 for their high-end IRIX operating system, XFS is a mature, 64-bit, high-performance file system. It was specifically engineered to handle massive media workstations and massive datasets long before modern storage sizes became common. Today, it is highly revered for its industrial-grade reliability and serves as the default file system for enterprise distributions like Red Hat Enterprise Linux (RHEL) and Fedora Server.

The most notable feature of XFS is its *Allocation Group* architecture. It divides the storage volume into discrete, independent regions. Each allocation group manages its own free space and inodes, allowing multiple CPU cores and application processes to perform read and write operations simultaneously without locking each other out. This results in exceptional parallel I/O performance.

Like Ext4, XFS protects data integrity using a metadata journal, ensuring rapid recovery after a sudden power loss. Its on-disk structure (advanced B+ trees) allows it to track block of free space to write files efficiently into contiguous blocks, preventing file fragmentation and keeping throughput high.

When you want it: You are building an enterprise-grade server, a database host, or a media production workstation. XFS shines brightest when handling massive files—like multi-gigabyte raw video footage, massive machine learning datasets, or monolithic backups. It is the go-to choice if you have a powerful multi-core CPU and multiple applications demanding high-speed, parallel access to the disk at the exact same time.

When you don't want it: You are setting up a standard consumer laptop or desktop where partition flexibility matters. Because XFS cannot be shrunk under any circumstances, it is a terrible choice if you ever plan to resize your partitions later or dual-boot with another operating system. It is also less efficient than Ext4 on smaller drives that are filled with millions of tiny text or code files rather than giant media blocks.

Swap: When RAM just isn't enough

A special type of file system on Linux is the Swap file system. As the name implies, it is used for "swapping" out data in RAM onto a storage device, like an SSD or HDD, to make room for actively used data the system needs to work with. It prevents the system from running out of RAM when lots of things are going on at the same time which your physical RAM can't hold.

Swap can either be its own *partition* or a *file*. Which one you choose comes down to what your use-case is and the administrative effort you're willing to put up with.

A Swap partition carves out a rigid portion of your drive just for swapped out memory pages. If you want to use hibernation on a laptop, a swap partition typically saves you a couple headaches in setting it up correctly—just point the system at the swap partition and you're done (`resume="PARTLABEL=swap"`). If you upgrade your RAM, however, you will also have to resize your swap partition to be at least as big as the amount of RAM in your system to continue using hibernation. This can entail its own set of administrative headaches, because resizing the swap partition usually means other partitions have to be shrunk (something that's impossible if you went with XFS for example).

A Swap file on the other hand is very flexible and can be created in seconds while the system is running. In case you need more swap space, you can simply create another swap file, turn it on, and delete it again after you're done. The downside with this approach with regards to hibernation is that you will have to tell the system the exact physical file offset of the swap file on disk (`resume="PARTLABEL=root" resume_offset=38912`) and need to update this value in case you re-create the swap file. You also won't be able to span multiple swap files for hibernation, as all contents in RAM must fit into a single file. There's also special considerations to take into account when using a swap file in conjunction with btrfs, since its CoW capabilities can cause considerable performance hits and file system fragmentation, unless CoW is specifically disabled for the swap file (`chattr +C`).

If you don't need hibernation and want the flexibility of easily expanding swap space, a **swap file** offers exactly what you need.

If you have a more complex storage setup (e.g. ZFS or btrfs RAID), or you're not bothered by its inflexibility, a **swap partition** is the more sensible choice.

Revision #5
Created 2023-12-17 12:36:35 UTC by Sebin
Updated 2026-06-23 16:01:07 UTC by Sebin