

File system

From Wikipedia, the free encyclopedia

In computing, a **file system** (often also written as **filesystem**) is a method for storing and organizing computer files and the data they contain to make it easy to find and access them. File systems may use a data storage device such as a hard disk or CD-ROM and involve maintaining the physical location of the files, they might provide access to data on a file server by acting as clients for a network protocol (e.g., NFS, SMB, or 9P clients), or they may be virtual and exist only as an access method for virtual data (e.g., procfs).

More formally, a file system is a special-purpose database for the storage, hierarchical organization, manipulation, navigation, access, and retrieval of data.

Contents

- 1 Aspects of file systems
 - 1.1 File names
 - 1.2 Meta data
 - 1.3 Hierarchical file systems
 - 1.4 Facilities
 - 1.5 Secure access
- 2 Types of file systems
 - 2.1 Disk file systems
 - 2.2 Flash file systems
 - 2.3 Database file systems
 - 2.4 Transactional file systems
 - 2.5 Network file systems
 - 2.6 Special purpose file systems
- 3 File systems and operating systems
 - 3.1 Flat file systems
 - 3.2 File systems under Unix-like operating systems
 - 3.2.1 File systems under Linux
 - 3.2.2 File systems under Mac OS X
 - 3.3 File systems under Plan 9 from Bell Labs
 - 3.4 File systems under Microsoft Windows
 - 3.4.1 Fat Overview
 - 3.4.2 HPFS Overview
 - 3.4.3 Data retrieval process
 - 3.5 File systems under OpenVMS
 - 3.6 File systems under MVS [IBM Mainframe]
- 4 See also
- 5 References
 - 5.1 Cited references
 - 5.2 General references
 - 5.3 Further reading
- 6 External links

Aspects of file systems

The most familiar file systems make use of an underlying data storage device that offers access to an array of fixed-size blocks, sometimes called *sectors*, generally a power of 2 in size (512 bytes or 1, 2, or 4 kib are most common). The file system software is responsible for organizing these sectors into files and directories, and keeping track of which sectors belong to which file and which are not being used. Most file systems address data in fixed-sized units called "clusters" or "blocks" which contain a certain number of disk sectors (usually 1-64). This is the smallest logical amount of disk space that can be allocated to hold a file.

However, file systems need not make use of a storage device at all. A file system can be used to organize and represent access to any data, whether it be stored or dynamically generated (e.g., procs).

File names

Whether the file system has an underlying storage device or not, file systems typically have directories which associate **file names** with files, usually by connecting the file name to an index into a file allocation table of some sort, such as the FAT in a DOS file system, or an inode in a Unix-like file system. Directory structures may be flat, or allow hierarchies where directories may contain subdirectories. In some file systems, file names are structured, with special syntax for filename extensions and version numbers. In others, file names are simple strings, and per-file metadata is stored elsewhere.

Meta data

Other bookkeeping information is typically associated with each file within a file system. The length of the data contained in a file may be stored as the number of blocks allocated for the file or as an exact byte count. The time that the file was last modified may be stored as the file's timestamp. Some file systems also store the file creation time, the time it was last accessed, and the time that the file's meta-data was changed. (Note that many early PC operating systems did not keep track of file times.) Other information can include the file's device type (e.g., block, character, socket, subdirectory, etc.), its owner user-ID and group-ID, and its access permission settings (e.g., whether the file is read-only, executable, etc.).

Arbitrary attributes can be associated on advanced file systems, such as XFS, ext2/ext3, some versions of UFS, and HFS+, using extended file attributes. This feature is implemented in the kernels of Linux, FreeBSD and Mac OS X operating systems, and allows metadata to be associated with the file at the *file system* level. This, for example, could be the author of a document, the character encoding of a plain-text document, or a checksum.

Hierarchical file systems

The hierarchical file system was an early research interest of Dennis Ritchie of Unix fame; previous implementations were restricted to only a few levels, notably the IBM implementations, even of their early databases like IMS. After the success of Unix, Ritchie extended the file system concept to every object in his later operating system developments, such as Plan 9 and Inferno.

Facilities

Traditional file systems offer facilities to create, move and delete both files and directories. They lack facilities to create additional links to a directory (hard links in Unix), rename parent links (".." in Unix-like OS), and create bidirectional links to files.

Traditional file systems also offer facilities to truncate, append to, create, move, delete and in-place modify files. They do not offer facilities to prepend to or truncate from the beginning of a file, let alone arbitrary insertion into or deletion from a file. The operations provided are highly asymmetric and lack the generality to be useful in unexpected contexts. For example, interprocess pipes in Unix have to be implemented outside of the file system because the pipes concept does not offer truncation from the beginning of files.

Secure access

Secure access to basic file system operations can be based on a scheme of access control lists or capabilities. Research has shown access control lists to be difficult to secure properly, which is why research operating systems tend to use capabilities. Commercial file systems still use access control lists. *see: secure computing*

Types of file systems

File system types can be classified into disk file systems, network file systems and special purpose file systems.

Disk file systems

A *disk file system* is a file system designed for the storage of files on a data storage device, most commonly a disk drive, which might be directly or indirectly connected to the computer. Examples of disk file systems include FAT (FAT12, FAT16, FAT32), NTFS, HFS and HFS+, ext2, ext3, ISO 9660, ODS-5, and UDF. Some disk file systems are journaling file systems or versioning file systems.

Flash file systems

A *flash file system* is a file system designed for storing files on flash memory devices. These are becoming more prevalent as the number of mobile devices is increasing, and the capacity of flash memories catches up with hard drives.

While a block device layer can emulate a disk drive so that a disk file system can be used on a flash device, this is suboptimal for several reasons:

- Erasing blocks: Flash memory blocks have to be explicitly erased before they can be written to. The time taken to erase blocks can be significant, thus it is beneficial to erase unused blocks while the device is idle.
- Random access: Disk file systems are optimized to avoid disk seeks whenever possible, due to the high cost of seeking. Flash memory devices impose no seek latency.
- Wear levelling: Flash memory devices tend to wear out when a single block is repeatedly overwritten; flash file systems are designed to spread out writes evenly.

Log-structured file systems have all the desirable properties for a flash file system. Such file systems include JFFS2 and YAFFS.

Database file systems

A new concept for file management is the concept of a database-based file system. Instead of, or in addition to, hierarchical structured management, files are identified by their characteristics, like type of file, topic, author, or similar metadata.

Transactional file systems

Each disk operation may involve changes to a number of different files and disk structures. In many cases, these changes are related, meaning that it is important that they all be executed at the same time. Take for example a bank sending another bank some money electronically. The bank's computer will "send" the transfer instruction to the other bank and also update its own records to indicate the transfer has occurred. If for some reason the computer crashes before it has had a chance to update its own records, then on reset, there will be no record of the transfer but the bank will be missing some money.

Transaction processing introduces the guarantee that at any point while it is running, a transaction can either be finished completely or reverted completely (though not necessarily both at any given point). This means that if there is a crash or

power failure, after recovery, the stored state will be consistent. (Either the money will be transferred or it will not be transferred, but it won't ever go missing "in transit".)

This type of file system is designed to be fault tolerant, but may incur additional overhead to do so.

Journaling file systems are one technique used to introduce transaction-level consistency to filesystem structures.

Network file systems

A network file system is a file system that acts as a client for a remote file access protocol, providing access to files on a server. Examples of network file systems include clients for the NFS, AFS, SMB protocols, and file-system-like clients for FTP and WebDAV.

Special purpose file systems

A special purpose file system is basically any file system that is not a disk file system or network file system. This includes systems where the files are arranged dynamically by software, intended for such purposes as communication between computer processes or temporary file space.

Special purpose file systems are most commonly used by file-centric operating systems such as Unix. Examples include the procfs (/proc) file system used by some Unix variants, which grants access to information about processes and other operating system features.

Deep space science exploration craft, like Voyager I & II used digital tape based special file systems. Most modern space exploration craft like Cassini-Huygens used Real-time operating system file systems or RTOS influenced file systems. The Mars Rovers are one such example of an RTOS file system, important in this case because they are implemented in flash memory.

File systems and operating systems

Most operating systems provide a file system, as a file system is an integral part of any modern operating system. Early microcomputer operating systems' only real task was file management — a fact reflected in their names (see DOS). Some early operating systems had a separate component for handling file systems which was called a disk operating system. On some microcomputers, the disk operating system was loaded separately from the rest of the operating system. On early operating systems, there was usually support for only one, native, unnamed file system; for example, CP/M supports only its own file system, which might be called "CP/M file system" if needed, but which didn't bear any official name at all.

Because of this, there needs to be an interface provided by the operating system software between the user and the file system. This interface can be textual (such as provided by a command line interface, such as the Unix shell, or OpenVMS DCL) or graphical (such as provided by a graphical user interface, such as file browsers). If graphical, the metaphor of the *folder*, containing documents, other files, and nested folders is often used (see also: directory and folder).

Flat file systems

In a flat file system, there are no subdirectories—everything is stored at the same (root) level on the media, be it a hard disk, floppy disk, etc. While simple, this system rapidly becomes inefficient as the number of files grows, and makes it difficult for users to organize data into related groups.

Like many small systems before it, the original Apple Macintosh featured a flat file system, called Macintosh File System. Its version of Mac OS was unusual in that the file management software (Macintosh Finder) created the illusion of a partially hierarchical filing system on top of MFS. This structure meant that every file on a disk had to have a

unique name, even if it appeared to be in a separate folder. MFS was quickly replaced with Hierarchical File System, which supported real directories.

File systems under Unix-like operating systems

Unix-like operating systems create a virtual file system, which makes all the files on all the devices appear to exist in a single hierarchy. This means, in those systems, there is one root directory, and every file existing on the system is located under it somewhere. Furthermore, the root directory does not have to be in any physical place. It might not be on your first hard drive - it might not even be on your computer. Unix-like systems can use a network shared resource as its root directory.

Unix-like systems assign a device name to each device, but this is not how the files on that device are accessed. Instead, to gain access to files on another device, you must first inform the operating system where in the directory tree you would like those files to appear. This process is called mounting a file system. For example, to access the files on a CD-ROM, one must tell the operating system "Take the file system from this CD-ROM and make it appear under such-and-such directory". The directory given to the operating system is called the *mount point* - it might, for example, be */media*. The */media* directory exists on many Unix systems (as specified in the Filesystem Hierarchy Standard) and is intended specifically for use as a mount point for removable media such as CDs, DVDs and like floppy disks. It may be empty, or it may contain subdirectories for mounting individual devices. Generally, only the administrator (i.e. root user) may authorize the mounting of file systems.

Unix-like operating systems often include software and tools that assist in the mounting process and provide it new functionality. Some of these strategies have been coined "auto-mounting" as a reflection of their purpose.

1. In many situations, file systems other than the root need to be available as soon as the operating system has booted. All Unix-like systems therefore provide a facility for mounting file systems at boot time. System administrators define these file systems in the configuration file *fstab*, which also indicates options and mount points.
2. In some situations, there is no need to mount certain file systems at boot time, although their use may be desired thereafter. There are some utilities for Unix-like systems that allow the mounting of predefined file systems upon demand.
3. Removable media have become very common with microcomputer platforms. They allow programs and data to be transferred between machines without a physical connection. Common examples include USB flash drives, CD-ROMs, and DVDs. Utilities have therefore been developed to detect the presence and availability of a medium and then mount that medium without any user intervention.
4. Progressive Unix-like systems have also introduced a concept called **supermounting**; see, for example, the Linux **supermount-ng** project (<https://web.archive.org/web/20080625213724/http://sourceforge.net/projects/supermount-ng>) . For example, a floppy disk that has been supermounted can be physically removed from the system. Under normal circumstances, the disk should have been synchronized and then unmounted before its removal. Provided synchronization has occurred, a different disk can be inserted into the drive. The system automatically notices that the disk has changed and updates the mount point contents to reflect the new medium. Similar functionality is found on standard Windows machines.
5. A similar innovation preferred by some users is the use of **autofs** (<https://web.archive.org/web/20080625213724/http://freshmeat.net/projects/autofs/>) , a system that, like supermounting, eliminates the need for manual mounting commands. The difference from supermount, other than compatibility in an apparent greater range of applications such as access to file systems on network servers, is that devices are mounted transparently when requests to their file systems are made, as would be appropriate for file systems on network servers, rather than relying on events such as the insertion of media, as would be appropriate for removable media.

File systems under Linux

Linux supports many different file systems, but common choices for the system disk include the ext* family (such as ext2 and ext3), XFS, and JFS.

File systems under Mac OS X

Mac OS X uses a file system that it inherited from classic Mac OS called HFS Plus. HFS Plus is a metadata-rich and case preserving file system. Due to the Unix roots of Mac OS X, Unix permissions were added to HFS Plus. Later versions of HFS Plus added journaling to prevent corruption of the file system structure and introduced a number of optimizations to the allocation algorithms in an attempt to defragment files automatically without requiring an external defragmenter.

Filenames can be up to 255 characters. HFS Plus uses Unicode to store filenames. On Mac OS X, the filetype can come from the type code, stored in file's metadata, or the filename.

HFS Plus has three kinds of links: Unix-style hard links, Unix-style symbolic links and aliases. Aliases are designed to maintain a link to their original file even if they are moved or renamed; they are not interpreted by the file system itself, but by the File Manager code in userland.

Mac OS X also supports the UFS file system, derived from the BSD Unix Fast File System via NeXTSTEP. However, as of Mac OS X 10.5 (Leopard), Mac OS X can no longer be installed on a UFS volume, nor can a pre-Leopard system installed on a UFS volume be upgraded to Leopard. [1]

(<https://web.archive.org/web/20080625213724/http://docs.info.apple.com/article.html?artnum=306516>)

File systems under Plan 9 from Bell Labs

Plan 9 from Bell Labs was originally designed to extend some of Unix's good points, and to introduce some new ideas of its own while fixing the shortcomings of Unix.

With respect to file systems, the Unix system of treating things as files was continued, but in Plan 9, *everything* is treated as a file, and accessed as a file would be (i.e., no ioctl or mmap). Perhaps surprisingly, while the file interface is made universal it is also simplified considerably, for example symlinks, hard links and suid are made obsolete, and an atomic create/open operation is introduced. More importantly the set of file operations becomes well defined and subversions of this like ioctl are eliminated.

Secondly, the underlying 9P protocol was used to remove the difference between local and remote files (except for a possible difference in latency). This has the advantage that a device or devices, represented by files, on a remote computer could be used as though it were the local computer's own device(s). This means that under Plan 9, multiple file servers provide access to devices, classing them as file systems. Servers for "synthetic" file systems can also run in user space bringing many of the advantages of micro kernel systems while maintaining the simplicity of the system.

Everything on a Plan 9 system has an abstraction as a file; networking, graphics, debugging, authentication, capabilities, encryption, and other services are accessed via I-O operations on file descriptors. For example, this allows the use of the IP stack of a gateway machine without need of NAT, or provides a network-transparent window system without the need of any extra code.

Another example: a Plan-9 application receives FTP service by opening an FTP site. The ftpfs server handles the open by essentially mounting the remote FTP site as part of the local file system. With ftpfs as an intermediary, the application can now use the usual file-system operations to access the FTP site as if it were part of the local file system. A further example is the mail system which uses file servers that synthesize virtual files and directories to represent a user mailbox as /mail/fs/mbox. The wikifs provides a file system interface to a wiki.

These file systems are organized with the help of private, per-process namespaces, allowing each process to have a different view of the many file systems that provide resources in a distributed system.

The Inferno operating system shares these concepts with Plan 9.

File systems under Microsoft Windows

Windows makes use of the FAT, NTFS (New Technology File System) and HPFS (High Performance File System) file systems .

FAT - File Allocation Table:

Is supported by all versions of Microsoft Windows; it is an evolution of Microsoft's earlier operating system (MS-DOS, which, in turn, was based on 86-DOS). FAT can ultimately trace its roots back to CP/M. Over the years various features have been added to it, inspired by similar features found on other file systems used by other operating systems such as Unix.

There are several versions of FAT used in Windows, these are: FAT12, FAT16 and FAT32; WinFS was supposed to have been included with Windows Vista.

Fat Overview

FAT is by far the most simplistic of the file systems supported by Windows. The FAT file system is characterized by the file allocation table (FAT), which is really a table that resides at the very "top" of the volume. To protect the volume, two copies of the FAT are kept in case one becomes damaged. In addition, the FAT tables and the root directory must be stored in a fixed location so that the system's boot files can be correctly located.

A disk formatted with FAT is allocated in clusters, whose size are determined by the size of the volume. When a file is created, an entry is created in the directory and the first cluster number containing data is established. This entry in the FAT table either indicates that this is the last cluster of the file, or points to the next cluster.

Updating the FAT table is very important as well as time consuming. If the FAT table is not regularly updated, it can lead to data loss. It is time consuming because the disk read heads must be repositioned to the drive's logical track zero each time the FAT table is updated.

There is no organization to the FAT directory structure, and files are given the first open location on the drive. In addition, FAT supports only a few file attributes: read-only, hidden, system, and archive.

Older versions of the FAT file system (FAT12 and FAT16) had file name length limits, a limit on the number of entries in the root directory of the file system and had restrictions on the maximum size of FAT-formatted disks or partitions. Specifically, FAT12 and FAT16 had a limit of 8 characters for the file name, and 3 characters for the extension. This is commonly referred to as the 8.3 filename limit. VFAT, which was an extension to FAT12 and FAT16 introduced in Windows NT 3.5 and subsequently included in Windows 95, allowed long file names (LFN). FAT32 also addressed many of the limits in FAT12 and FAT16, but remains limited compared to NTFS.

HPFS - High Performance File System:

HPFS is only supported under Windows NT versions 3.1, 3.5, and 3.51. Windows NT 4.0 cannot access HPFS partitions. It was also used by IBM's now defunct OS/2.

HPFS Overview

The HPFS file system was first introduced with OS/2 1.2 to allow for greater access to the larger hard drives that were then appearing on the market. Additionally, it was necessary for a new file system to extend the naming system, organization, and security for the growing demands of the network server market. HPFS maintains the directory organization of FAT, but adds automatic sorting of the directory based on filenames. Filenames are extended to up to 254 double byte characters. HPFS also allows a file to be composed of "data" and special attributes to allow for increased flexibility in terms of supporting other naming conventions and security. In addition, the unit of allocation is changed from clusters to physical sectors (512 bytes), which reduces lost disk space.

Under HPFS, directory entries hold more information than under FAT. As well as the attribute file, this includes information about the modification, creation, and access date and times. Instead of pointing to the first cluster of the file, the directory entries under HPFS point to the FNODE. FNODE (pronounced "eff node") occupies a single sector and contains control and access history information used internally by the file system extended attributes and Access control list the length and the first 15 characters of the name of the associated file or directory and an allocation structure. The FNODE can contain the file's data, or pointers that may point to the file's data or to other structures that will eventually point to the file's data.

HPFS attempts to allocate as much of a file in contiguous sectors as possible. This is done in order to increase speed when doing sequential processing of a file.

HPFS organizes a drive into a series of 8 MB bands, and whenever possible a file is contained within one of these bands. Between each of these bands are 2K allocation bitmaps, which keep track of which sectors within a band have and have not been allocated. Banding increases performance because the drive head does not have to return to the logical top (typically cylinder 0) of the disk, but to the nearest band allocation bitmap to determine where a file is to be stored.

Additionally, HPFS includes a couple of unique special data objects:

Super Block

The Super Block is located in logical sector 16 and contains a pointer to the FNODE of the root directory. One of the biggest dangers of using HPFS is that if the Super Block is lost or corrupted due to a bad sector, so are the contents of the partition, even if the rest of the drive is fine. It would be possible to recover the data on the drive by copying everything to another drive with a good sector 16 and rebuilding the Super Block. However, this is a very complex task.

Spare Block

The Spare Block is located in logical sector 17 and contains a table of "hot fixes" and the Spare Directory Block. Under HPFS, when a bad sector is detected, the "hot fixes" entry is used to logically point to an existing good sector in place of the bad sector. This technique for handling write errors is known as hot fixing.

Hot fixing is a technique where if an error occurs because of a bad sector, the file system moves the information to a different sector and marks the original sector as bad. This is all done transparent to any applications that are performing disk I/O (that is, the application never knows that there were any problems with the hard drive). Using a file system that supports hot fixing will eliminate error messages such as the FAT "Abort, Retry, or Fail?" error message that occurs when a bad sector is encountered.

Note: The version of HPFS that is included with Windows NT does not support hot fixing.

NTFS - New Technology File System:

NTFS, introduced with the Windows NT operating system, allowed ACL-based permission control. Hard links, multiple file streams, attribute indexing, quota tracking, compression and mount-points for other file systems (called "junctions") are also supported, though not all these features are well-documented.

Unlike many other operating systems, Windows uses a *drive letter* abstraction at the user level to distinguish one disk or partition from another. For example, the path C:\WINDOWS represents a directory WINDOWS on the partition represented by the letter C. The C drive is most commonly used for the primary hard disk partition, on which Windows is usually installed and from which it boots. This "tradition" has become so firmly ingrained that bugs came about in older versions of Windows which made assumptions that the drive that the operating system was installed on was C. The tradition of using "C" for the drive letter can be traced to MS-DOS, where the letters A and B were reserved for up to two floppy disk drives. Network drives may also be mapped to drive letters.

WinFS - Windows Future Storage:

This was intended to be released with Windows Vista but it was canceled with no further information provided, except for that they are still working on WinFS.

Data retrieval process

When a computer retrieves any file from a disk, the program in which the file is opened first calls on an API (Application Programming Interface) in order to build a dialogue box for viewing the file or folder. The operating system calls on the IFS (installable file system) manager. The IFS calls on the correct FSD (file system driver) in order to open the selected file from a choice of FSD's that work with different storage systems.

The FSD gets the location on the disk for the first cluster of the file from the FAT (File Allocation Table) or, in the case of NTFS, the MFT (Master File Table). In short, the whole point of these are to map out all the files on the disk and record where they are located (which track and sector of the disk).

Once the FSD knows where the file is located, it passes the read command to the I/O subsystem (input/output subsystem). This is generally an assistant to FSDs, doing chores such as routing messages back and forth between FSDs and lower, device-specific drivers. The VTD (Volume Tracking driver) might come into action if the file is not located on a drive. The VTD would only come into action if the file was stored on a floppy, CD, DVD, or other removable drive. The only thing the VTD does is determine if the correct disk is in the correct drive. Once this is done, the operation is passed off to the TSD (Type Specific Driver) which passes the read signal to the correct drive adapter. There is a separate TSD for hard drives, floppy drives, etc. Finally, the adapter takes over, moving the read/write head to the correct series of disk clusters to read the file into RAM so that it can be viewed and edited.

File systems under OpenVMS

File systems under MVS [IBM Mainframe]

See also

- List of file systems
- Comparison of file systems
- Virtual file system
- File system fragmentation

- Distributed file system
- Filesystem API
- Physical and logical storage
- List of Unix programs
- Filename extension
- Disk sharing
- File system driver
- Garbage Collected Filesystem

References

Cited references

General references

- Jonathan de Boyne Pollard (1996). Disc and volume size limits (<https://web.archive.org/web/20080625213724/http://homepages.tesco.net/~J.deBoynePollard/FGA/os2-disc-and-volume-size-limits.html>) . *Frequently Given Answers*. Retrieved on February 9, 2005.
- IBM. OS/2 corrective service fix JR09427 (<https://web.archive.org/web/20080625213724/ftp://service.boulder.ibm.com/ps/products/os2/fixes/v4warp/english-us/jr09427/JR09427.TXT>) . Retrieved on February 9, 2005.
- Attribute - \$EA_INFORMATION (0xD0) (https://web.archive.org/web/20080625213724/http://linux-ntfs.sourceforge.net/ntfs/attributes/ea_information.html) . *NTFS Information, Linux-NTFS Project*. Retrieved on February 9, 2005.
- Attribute - \$EA (0xE0) (<https://web.archive.org/web/20080625213724/http://linux-ntfs.sourceforge.net/ntfs/attributes/ea.html>) . *NTFS Information, Linux-NTFS Project*. Retrieved on February 9, 2005.
- Attribute - \$STANDARD_INFORMATION (0x10) (https://web.archive.org/web/20080625213724/http://linux-ntfs.sourceforge.net/ntfs/attributes/standard_information.html) . *NTFS Information, Linux-NTFS Project*. Retrieved on February 21, 2005.
- Apple Computer Inc. Technical Note TN1150: HFS Plus Volume Format (<https://web.archive.org/web/20080625213724/http://developer.apple.com/technotes/tn/tn1150.html>) . *Detailed HFS Plus and HFSX description*. Retrieved on May 2, 2006.
- File System Forensic Analysis (<https://web.archive.org/web/20080625213724/http://www.digital-evidence.org/fsfa/>) , Brian Carrier, Addison Wesley, 2005.

Further reading

- Local Filesystems for Windows (<https://web.archive.org/web/20080625213724/http://www.microsoft.com/whdc/device/storage/LocFileSys.msp>)
- Understanding File-Size Limits on NTFS and FAT (<https://web.archive.org/web/20080625213724/http://www.winnetmag.com/Article/ArticleID/38803/38803.html>)
- Benchmarking Filesystems Part II (<https://web.archive.org/web/20080625213724/http://linuxgazette.net/122/piszc.html>) using kernel 2.6, by Justin Piszcz, Linux Gazette 122, January 2006
- Linux File System Benchmarks (<https://web.archive.org/web/20080625213724/http://fsbench.netnation.com/>) v2.6 kernel with a stress on CPU usage
- Interview With the People Behind JFS, ReiserFS & XFS (https://web.archive.org/web/20080625213724/http://www.osnews.com/story.php?news_id=69)
- Large List of File System Summaries (<https://web.archive.org/web/20080625213724/http://www.osdata.com/system/logical/logical.htm>)
- Filesystems (ext3, ReiserFS, XFS, JFS) comparison on Debian Etch (<https://web.archive.org/web/20080625213724/http://www.debian-administration.org/articles/388>)
- Overview of some filesystems (outdated) (<https://web.archive.org/web/20080625213724/http://osdev.berlios.de/osd-fs.html>)

- Linux large file support (outdated) (https://web.archive.org/web/20080625213724/http://www.suse.de/~aj/linux_lfs.html)
- Sparse files support (outdated) (<https://web.archive.org/web/20080625213724/http://www.lrdev.com/lr/unix/sparsefile.html>)
- Benchmarking Filesystems (outdated) (<https://web.archive.org/web/20080625213724/http://linuxgazette.net/102/piszczy.html>) by Justin Piszcz, Linux Gazette 102, May 2004
- Journaled Filesystem Benchmarks (outdated) (<https://web.archive.org/web/20080625213724/http://staff.osuosl.org/~kveton/fs/>) : A comparison of ReiserFS, XFS, JFS, ext3 & ext2
- Journal File System Performance (outdated) (https://web.archive.org/web/20080625213724/http://www.open-mag.com/features/Vol_18/filesystems/filesystems.htm) : ReiserFS, JFS, and Ext3FS show their merits on a fast RAID appliance
- Linux Filesystem Benchmarks (<https://web.archive.org/web/20080625213724/http://www.techyblog.com/linux-news/linux-26-file-system-benchmarks-older.html>)
- Jeremy Reimer. "From BFS to ZFS: past, present, and future of file systems" (<https://web.archive.org/web/20080625213724/http://arstechnica.com/articles/paedia/past-present-future-file-systems.ars>) ", arstechnica.com, March 16, 2008. Retrieved on 2008-03-18.

External links

- Filesystems of Operating Systems (<https://web.archive.org/web/20080625213724/http://filesystems.palconit.com/>)
- Filesystem Specifications - Links & Whitepapers (<https://web.archive.org/web/20080625213724/http://www.forensics.nl/filesystems>)
- Interesting File System Projects (<https://web.archive.org/web/20080625213724/http://filesystems.org/all-projects.html>)

Retrieved from "http://en.wikipedia.org/wiki/File_system"

Categories: Computer file systems

- This page was last modified on 18 June 2008, at 23:32.
- All text is available under the terms of the GNU Free Documentation License. (See **Copyrights** for details.) Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a U.S. registered 501(c)(3) tax-deductible nonprofit charity.