NAME

ext2 – the second extended file system
ext3 – the third extended file system
ext4 – the fourth extended file system

DESCRIPTION

The second, third, and fourth extended file systems, or ext2, ext3, and ext4 as they are commonly known, are Linux file systems that have historically been the default file system for many Linux distributions. They are general purpose file systems that have been designed for extensibility and backwards compatibility. In particular, file systems previously intended for use with the ext2 and ext3 file systems can be mounted using the ext4 file system driver, and indeed in many modern Linux distributions, the ext4 file system driver has been configured to handle mount requests for ext2 and ext3 file systems.

FILE SYSTEM FEATURES

A file system formatted for ext2, ext3, or ext4 can have some collection of the following file system feature flags enabled. Some of these features are not supported by all implementations of the ext2, ext3, and ext4 file system drivers, depending on Linux kernel version in use. On other operating systems, such as the GNU/HURD or FreeBSD, only a very restrictive set of file system features may be supported in their implementations of ext2.

64bit

Enables the file system to be larger than 2^32 blocks. This feature is set automatically, as needed, but it can be useful to specify this feature explicitly if the file system might need to be resized larger than 2^32 blocks, even if it was smaller than that threshold when it was originally created. Note that some older kernels and older versions of e2fsprogs will not support file systems with this ext4 feature enabled.

bigalloc

This ext4 feature enables clustered block allocation, so that the unit of allocation is a power of two number of blocks. That is, each bit in the what had traditionally been known as the block allocation bitmap now indicates whether a cluster is in use or not, where a cluster is by default composed of 16 blocks. This feature can decrease the time spent on doing block allocation and brings smaller fragmentation, especially for large files. The size can be specified using the **mke2fs -C** option.

Warning: The bigalloc feature is still under development, and may not be fully supported with your kernel or may have various bugs. Please see the web page http://ext4.wiki.kernel.org/index.php/Bigalloc for details. May clash with delayed allocation (see **nodelalloc** mount option).

This feature requires that the **extent** feature be enabled.

dir_index

Use hashed b-trees to speed up name lookups in large directories. This feature is supported by ext3 and ext4 file systems, and is ignored by ext2 file systems.

dir_nlink

Normally, ext4 allows an inode to have no more than 65,000 hard links. This applies to regular files as well as directories, which means that there can be no more than 64,998 subdirectories in a directory (because each of the '.' and '..' entries, as well as the directory entry for the directory in its parent directory counts as a hard link). This feature lifts this limit by causing ext4 to use a link count of 1 to indicate that the number of hard links to a directory is not known when the link count might exceed the maximum count limit.

ea_inode

Normally, a file's extended attributes and associated metadata must fit within the inode or the inode's associated extended attribute block. This feature allows the value of each extended attribute to be placed in the data blocks of a separate inode if necessary, increasing the limit on the size and number of extended attributes per file.

encrypt
This ext4 feature provides file-system level encryption of data blocks and file names. The inode metadata (timestamps, file size, user/group ownership, etc.) is *not* encrypted.

This feature is most useful on file systems with multiple users, or where not all files should be encrypted. In many use cases, especially on single-user systems, encryption at the block device layer using dm-crypt may provide much better security.

ext_attr
This feature enables the use of extended attributes. This feature is supported by ext2, ext3, and ext4.

extent
This ext4 feature allows the mapping of logical block numbers for a particular inode to physical blocks on the storage device to be stored using an extent tree, which is a more efficient data structure than the traditional indirect block scheme used by the ext2 and ext3 file systems. The use of the extent tree decreases metadata block overhead, improves file system performance, and decreases the needed to run e2fsck(8) on the file system. (Note: both **extent** and **extents** are accepted as valid names for this feature for historical/backwards compatibility reasons.)

extra_isize
This ext4 feature reserves a specific amount of space in each inode for extended metadata such as nanosecond timestamps and file creation time, even if the current kernel does not currently need to reserve this much space. Without this feature, the kernel will reserve the amount of space for features it currently needs, and the rest may be consumed by extended attributes.

For this feature to be useful the inode size must be 256 bytes in size or larger.

filetype
This feature enables the storage of file type information in directory entries. This feature is supported by ext2, ext3, and ext4.

flex_bg
This ext4 feature allows the per-block group metadata (allocation bitmaps and inode tables) to be placed anywhere on the storage media. In addition, **mke2fs** will place the per-block group metadata together starting at the first block group of each "flex_bg group". The size of the flex_bg group can be specified using the **−G** option.

has_journal
Create a journal to ensure filesystem consistency even across unclean shutdowns. Setting the filesystem feature is equivalent to using the **−j** option with **mke2fs** or **tune2fs.** This feature is supported by ext3 and ext4, and ignored by the ext2 file system driver.

huge_file
This ext4 feature allows files to be larger than 2 terabytes in size.

inline_data
Allow data to be stored in the inode and extended attribute area.

journal_dev
This feature is enabled on the superblock found on an external journal device. The block size for the external journal must be the same as the file system which uses it.

The external journal device can be used by a file system by specifying the **−J device=**<external-device> option to mke2fs(8) or tune2fs(8)

large_dir
This feature increases the limit on the number of files per directory by raising the maximum size of directories and, for hashed b-tree directories (see **dir_index**), the maximum height of the hashed b-tree used to store the directory entries.

large_file
This feature flag is set automatically by modern kernels when a file larger than 2 gigabytes is created. Very old kernels could not handle large files, so this feature flag was used to prohibit those kernels from mounting file systems that they could not understand.

metadata_csum
This ext4 feature enables metadata checksumming. This feature stores checksums for all of the filesystem metadata (superblock, group descriptor blocks, inode and block bitmaps, directories, and extent tree blocks). The checksum algorithm used for the metadata blocks is different than the one used for group descriptors with the **uninit_bg** feature. These two features are incompatible and **metadata_csum** will be used preferentially instead of **uninit_bg**.

metadata_csum_seed
This feature allows the filesystem to store the metadata checksum seed in the superblock, which allows the administrator to change the UUID of a filesystem using the **metadata_csum** feature while it is mounted.

meta_bg
This ext4 feature allows file systems to be resized on-line without explicitly needing to reserve space for growth in the size of the block group descriptors. This scheme is also used to resize file systems which are larger than 2^32 blocks. It is not recommended that this feature be set when a file system is created, since this alternate method of storing the block group descriptors will slow down the time needed to mount the file system, and newer kernels can automatically set this feature as necessary when doing an online resize and no more reserved space is available in the resize inode.

mmp
This ext4 feature provides multiple mount protection (MMP). MMP helps to protect the filesystem from being multiply mounted and is useful in shared storage environments.

project
This ext4 feature provides project quota support. With this feature, the project ID of inode will be managed when the filesystem is mounted.

quota
Create quota inodes (inode #3 for userquota and inode #4 for group quota) and set them in the superblock. With this feature, the quotas will be enabled automatically when the filesystem is mounted.

Causes the quota files (i.e., user.quota and group.quota which existed in the older quota design) to be hidden inodes.

resize_inode
This file system feature indicates that space has been reserved so that the block group descriptor table can be extended while resizing a mounted file system. The online resize operation is carried out by the kernel, triggered by resize2fs(8). By default **mke2fs** will attempt to reserve enough space so that the filesystem may grow to 1024 times its initial size. This can be changed using the **resize** extended option.

This feature requires that the **sparse_super** or **sparse_super2** feature be enabled.

sparse_super
This file system feature is set on all modern ext2, ext3, and ext4 file systems. It indicates that backup copies of the superblock and block group descriptors are present only in a few block groups, not all of them.

sparse_super2
This feature indicates that there will only be at most two backup superblocks and block group descriptors. The block groups used to store the backup superblock(s) and blockgroup descriptor(s) are stored in the superblock, but typically, one will be located at the beginning of block group #1, and one in the last block group in the file system. This feature is essentially a more extreme

version of sparse_super and is designed to allow a much larger percentage of the disk to have contiguous blocks available for data files.

uninit_bg
This ext4 file system feature indicates that the block group descriptors will be protected using checksums, making it safe for mke2fs(8) to create a file system without initializing all of the block groups. The kernel will keep a high watermark of unused inodes, and initialize inode tables and blocks lazily. This feature speeds up the time to check the file system using e2fsck(8), and it also speeds up the time required for mke2fs(8) to create the file system.

MOUNT OPTIONS

This section describes mount options which are specific to ext2, ext3, and ext4. Other generic mount options may be used as well; see mount(8) for details.

Mount options for ext2

The 'ext2' filesystem is the standard Linux filesystem. Since Linux 2.5.46, for most mount options the default is determined by the filesystem superblock. Set them with tune2fs(8).

acl|noacl
Support POSIX Access Control Lists (or not). See the acl(5) manual page.

bsddf|minixdf
Set the behavior for the *statfs* system call. The **minixdf** behavior is to return in the *f_blocks* field the total number of blocks of the filesystem, while the **bsddf** behavior (which is the default) is to subtract the overhead blocks used by the ext2 filesystem and not available for file storage. Thus

% mount /k −o minixdf; df /k; umount /k
Filesystem 1024-blocks Used Available Capacity Mounted on
/dev/sda6 2630655 86954 2412169 3% /k

% mount /k −o bsddf; df /k; umount /k
Filesystem 1024-blocks Used Available Capacity Mounted on
/dev/sda6 2543714 13 2412169 0% /k

(Note that this example shows that one can add command line options to the options given in */etc/fstab*.)

check=none or **nocheck**
No checking is done at mount time. This is the default. This is fast. It is wise to invoke e2fsck(8) every now and then, e.g. at boot time. The non-default behavior is unsupported (check=normal and check=strict options have been removed). Note that these mount options don't have to be supported if ext4 kernel driver is used for ext2 and ext3 filesystems.

debug Print debugging info upon each (re)mount.

errors={continue|remount-ro|panic}
Define the behavior when an error is encountered. (Either ignore errors and just mark the filesystem erroneous and continue, or remount the filesystem read-only, or panic and halt the system.) The default is set in the filesystem superblock, and can be changed using tune2fs(8).

grpid|bsdgroups and **nogrpid|sysvgroups**
These options define what group id a newly created file gets. When **grpid** is set, it takes the group id of the directory in which it is created; otherwise (the default) it takes the fsgid of the current process, unless the directory has the setgid bit set, in which case it takes the gid from the parent directory, and also gets the setgid bit set if it is a directory itself.

grpquota|noquota|quota|usrquota
The usrquota (same as quota) mount option enables user quota support on the filesystem. grpquota enables group quotas support. You need the quota utilities to actually enable and manage the quota system.

nouid32
Disables 32-bit UIDs and GIDs. This is for interoperability with older kernels which only store and expect 16-bit values.

oldalloc or **orlov**
Use old allocator or Orlov allocator for new inodes. Orlov is default.

resgid=*n* and **resuid=***n*
The ext2 filesystem reserves a certain percentage of the available space (by default 5%, see mke2fs(8) and tune2fs(8)). These options determine who can use the reserved blocks. (Roughly: whoever has the specified uid, or belongs to the specified group.)

sb=*n* Instead of using the normal superblock, use an alternative superblock specified by *n*. This option is normally used when the primary superblock has been corrupted. The location of backup superblocks is dependent on the filesystem's blocksize, the number of blocks per group, and features such as **sparse_super**.

Additional backup superblocks can be determined by using the **mke2fs** program using the **−n** option to print out where the superblocks exist, supposing **mke2fs** is supplied with arguments that are consistent with the filesystem's layout (e.g. blocksize, blocks per group, **sparse_super**, etc.).

The block number here uses 1 k units. Thus, if you want to use logical block 32768 on a filesystem with 4 k blocks, use "sb=131072".

user_xattr|nouser_xattr
Support "user." extended attributes (or not).

Mount options for ext3

The ext3 filesystem is a version of the ext2 filesystem which has been enhanced with journaling. It supports the same options as ext2 as well as the following additions:

journal_dev=devnum/journal_path=path
When the external journal device's major/minor numbers have changed, these options allow the user to specify the new journal location. The journal device is identified either through its new major/minor numbers encoded in devnum, or via a path to the device.

norecovery/noload
Don't load the journal on mounting. Note that if the filesystem was not unmounted cleanly, skipping the journal replay will lead to the filesystem containing inconsistencies that can lead to any number of problems.

data={journal|ordered|writeback}
Specifies the journaling mode for file data. Metadata is always journaled. To use modes other than **ordered** on the root filesystem, pass the mode to the kernel as boot parameter, e.g. *rootflags=data=journal*.

journal
All data is committed into the journal prior to being written into the main filesystem.

ordered
This is the default mode. All data is forced directly out to the main file system prior to its metadata being committed to the journal.

writeback
Data ordering is not preserved – data may be written into the main filesystem after its metadata has been committed to the journal. This is rumoured to be the highest-throughput option. It guarantees internal filesystem integrity, however it can allow old data to appear in files after a crash and journal recovery.

data_err=ignore
Just print an error message if an error occurs in a file data buffer in ordered mode.

data_err=abort
Abort the journal if an error occurs in a file data buffer in ordered mode.

barrier=0 / barrier=1
This disables / enables the use of write barriers in the jbd code. barrier=0 disables, barrier=1 enables (default). This also requires an IO stack which can support barriers, and if jbd gets an error on a barrier write, it will disable barriers again with a warning. Write barriers enforce proper on-disk ordering of journal commits, making volatile disk write caches safe to use, at some performance penalty. If your disks are battery-backed in one way or another, disabling barriers may safely improve performance.

commit=*nrsec*
Start a journal commit every *nrsec* seconds. The default value is 5 seconds. Zero means default.

user_xattr
Enable Extended User Attributes. See the attr(5) manual page.

jqfmt={**vfsold**|**vfsv0**|**vfsv1**}
Apart from the old quota system (as in ext2, jqfmt=vfsold aka version 1 quota) ext3 also supports journaled quotas (version 2 quota). jqfmt=vfsv0 or jqfmt=vfsv1 enables journaled quotas. Journaled quotas have the advantage that even after a crash no quota check is required. When the **quota** filesystem feature is enabled, journaled quotas are used automatically, and this mount option is ignored.

usrjquota=aquota.user|grpjquota=aquota.group
For journaled quotas (jqfmt=vfsv0 or jqfmt=vfsv1), the mount options usrjquota=aquota.user and grpjquota=aquota.group are required to tell the quota system which quota database files to use. When the **quota** filesystem feature is enabled, journaled quotas are used automatically, and this mount option is ignored.

Mount options for ext4

The ext4 filesystem is an advanced level of the ext3 filesystem which incorporates scalability and reliability enhancements for supporting large filesystem.

The options **journal_dev, journal_path, norecovery, noload, data, commit, orlov, oldalloc, [no]user_xattr, [no]acl, bsddf, minixdf, debug, errors, data_err, grpid, bsdgroups, nogrpid, sysvgroups, resgid, resuid, sb, quota, noquota, nouid32, grpquota, usrquota, usrjquota, grpjquota, and jqfmt are backwardly compatible with ext3 or ext2.**

journal_checksum | nojournal_checksum
The journal_checksum option enables checksumming of the journal transactions. This will allow the recovery code in e2fsck and the kernel to detect corruption in the kernel. It is a compatible change and will be ignored by older kernels.

journal_async_commit
Commit block can be written to disk without waiting for descriptor blocks. If enabled older kernels cannot mount the device. This will enable 'journal_checksum' internally.

barrier=0 / barrier=1 / barrier / nobarrier
These mount options have the same effect as in ext3. The mount options "barrier" and "nobarrier" are added for consistency with other ext4 mount options.

The ext4 filesystem enables write barriers by default.

inode_readahead_blks=*n*
This tuning parameter controls the maximum number of inode table blocks that ext4's inode table readahead algorithm will pre-read into the buffer cache. The value must be a power of 2. The default value is 32 blocks.

stripe=*n*
Number of filesystem blocks that mballoc will try to use for allocation size and alignment. For RAID5/6 systems this should be the number of data disks * RAID chunk size in filesystem blocks.

delalloc
Deferring block allocation until write-out time.

nodelalloc
Disable delayed allocation. Blocks are allocated when data is copied from user to page cache.

max_batch_time=*usec*
Maximum amount of time ext4 should wait for additional filesystem operations to be batch together with a synchronous write operation. Since a synchronous write operation is going to force a commit and then a wait for the I/O complete, it doesn't cost much, and can be a huge throughput win, we wait for a small amount of time to see if any other transactions can piggyback on the synchronous write. The algorithm used is designed to automatically tune for the speed of the disk, by measuring the amount of time (on average) that it takes to finish committing a transaction. Call this time the "commit time". If the time that the transaction has been running is less than the commit time, ext4 will try sleeping for the commit time to see if other operations will join the transaction. The commit time is capped by the max_batch_time, which defaults to 15000 µs (15 ms). This optimization can be turned off entirely by setting max_batch_time to 0.

min_batch_time=*usec*
This parameter sets the commit time (as described above) to be at least min_batch_time. It defaults to zero microseconds. Increasing this parameter may improve the throughput of multi-threaded, synchronous workloads on very fast disks, at the cost of increasing latency.

journal_ioprio=*prio*
The I/O priority (from 0 to 7, where 0 is the highest priority) which should be used for I/O operations submitted by kjournald2 during a commit operation. This defaults to 3, which is a slightly higher priority than the default I/O priority.

abort Simulate the effects of calling ext4_abort() for debugging purposes. This is normally used while remounting a filesystem which is already mounted.

auto_da_alloc|noauto_da_alloc
Many broken applications don't use fsync() when replacing existing files via patterns such as

fd = open("foo.new")/write(fd,...)/close(fd)/ rename("foo.new", "foo")

or worse yet

fd = open("foo", O_TRUNC)/write(fd,...)/close(fd).

If auto_da_alloc is enabled, ext4 will detect the replace-via-rename and replace-via-truncate patterns and force that any delayed allocation blocks are allocated such that at the next journal commit, in the default data=ordered mode, the data blocks of the new file are forced to disk before the rename() operation is committed. This provides roughly the same level of guarantees as ext3, and avoids the "zero-length" problem that can happen when a system crashes before the delayed allocation blocks are forced to disk.

noinit_itable
Do not initialize any uninitialized inode table blocks in the background. This feature may be used by installation CD's so that the install process can complete as quickly as possible; the inode table initialization process would then be deferred until the next time the filesystem is mounted.

init_itable=n
The lazy itable init code will wait n times the number of milliseconds it took to zero out the previous block group's inode table. This minimizes the impact on system performance while the filesystem's inode table is being initialized.

discard/nodiscard
Controls whether ext4 should issue discard/TRIM commands to the underlying block device when blocks are freed. This is useful for SSD devices and sparse/thinly-provisioned LUNs, but it is off by default until sufficient testing has been done.

block_validity/noblock_validity
This option enables/disables the in-kernel facility for tracking filesystem metadata blocks within internal data structures. This allows multi-block allocator and other routines to quickly locate extents which might overlap with filesystem metadata blocks. This option is intended for debugging purposes and since it negatively affects the performance, it is off by default.

dioread_lock/dioread_nolock
Controls whether or not ext4 should use the DIO read locking. If the dioread_nolock option is specified ext4 will allocate uninitialized extent before buffer write and convert the extent to initialized after IO completes. This approach allows ext4 code to avoid using inode mutex, which improves scalability on high speed storages. However this does not work with data journaling and dioread_nolock option will be ignored with kernel warning. Note that dioread_nolock code path is only used for extent-based files. Because of the restrictions this options comprises it is off by default (e.g. dioread_lock).

max_dir_size_kb=n
This limits the size of the directories so that any attempt to expand them beyond the specified limit in kilobytes will cause an ENOSPC error. This is useful in memory-constrained environments, where a very large directory can cause severe performance problems or even provoke the Out Of Memory killer. (For example, if there is only 512 MB memory available, a 176 MB directory may seriously cramp the system's style.)

i_version
Enable 64-bit inode version support. This option is off by default.

nombcache
This option disables use of mbcache for extended attribute deduplication. On systems where extended attributes are rarely or never shared between files, use of mbcache for deduplication adds unnecessary computational overhead.

prjquota
The prjquota mount option enables project quota support on the filesystem. You need the quota utilities to actually enable and manage the quota system. This mount option requires the **project** filesystem feature.

FILE ATTRIBUTES

The ext2, ext3, and ext4 filesystems support setting the following file attributes on Linux systems using the chattr(1) utility:

a - append only

A - no atime updates

d - no dump

D - synchronous directory updates

i - immutable

S - synchronous updates

u - undeletable

In addition, the ext3 and ext4 filesystems support the following flag:

j - data journaling

Finally, the ext4 filesystem also supports the following flag:

e - extents format

For descriptions of these attribute flags, please refer to the chattr(1) man page.

KERNEL SUPPORT

This section lists the file system driver (e.g., ext2, ext3, ext4) and upstream kernel version where a particular file system feature was supported. Note that in some cases the feature was present in earlier kernel versions, but there were known, serious bugs. In other cases the feature may still be considered in an experimental state. Finally, note that some distributions may have backported features into older kernels; in particular the kernel versions in certain "enterprise distributions" can be extremely misleading.

filetype	ext2, 2.2.0
sparse_super	ext2, 2.2.0
large_file	ext2, 2.2.0
has_journal	ext3, 2.4.15
ext_attr	ext2/ext3, 2.6.0
dir_index	ext3, 2.6.0
resize_inode	ext3, 2.6.10 (online resizing)
64bit	ext4, 2.6.28
dir_nlink	ext4, 2.6.28
extent	ext4, 2.6.28
extra_isize	ext4, 2.6.28
flex_bg	ext4, 2.6.28
huge_file	ext4, 2.6.28
meta_bg	ext4, 2.6.28
uninit_bg	ext4, 2.6.28
mmp	ext4, 3.0
bigalloc	ext4, 3.2
quota	ext4, 3.6
inline_data	ext4, 3.8
sparse_super2	ext4, 3.16
metadata_csum	ext4, 3.18
encrypt	ext4, 4.1
metadata_csum_seed	ext4, 4.4
project	ext4, 4.5
ea_inode	ext4, 4.13
large_dir	ext4, 4.13

SEE ALSO

mke2fs(8), mke2fs.conf(5), e2fsck(8), dumpe2fs(8), tune2fs(8), debugfs(8), mount(8), chattr(1)