ext4 General Information

Ext4 is an advanced level of the ext3 filesystem which incorporates scalability and reliability enhancements for supporting large filesystems (64 bit) in keeping with increasing disk capacities and state-of-the-art feature requirements.

Mailing list: linux-ext4@vger.kernel.org Web site: http://ext4.wiki.kernel.org

Quick usage instructions

Note: More extensive information for getting started with ext4 can be found at the ext4 wiki site at the URL: http://ext4.wiki.kernel.org/index.php/Ext4_Howto

- The latest version of e2fsprogs can be found at:

  https://www.kernel.org/pub/linux/kernel/people/tytso/e2fsprogs/

  or

  http://sourceforge.net/project/showfiles.php?group_id=2406

  or grab the latest git repository from:

  https://git.kernel.org/pub/scm/fs/ext2/e2fsprogs.git

- Create a new filesystem using the ext4 filesystem type:

  ```
  # mke2fs -t ext4 /dev/hda1
  ```

  Or to configure an existing ext3 filesystem to support extents:

  ```
  # tune2fs -O extents /dev/hda1
  ```

  If the filesystem was created with 128 byte inodes, it can be converted to use 256 byte for greater efficiency via:

  ```
  # tune2fs -I 256 /dev/hda1
  ```

- Mounting:

  ```
  # mount -t ext4 /dev/hda1 /wherever
  ```

- When comparing performance with other filesystems, it's always important to try multiple workloads; very often a subtle change in a workload parameter can completely change the ranking of which filesystems do well compared to others. When comparing versus ext3, note that ext4 enables write barriers by default, while ext3 does not enable write barriers by default. So it is useful to use explicitly specify whether barriers are enabled or not when via the '-o barriers=[0|1]' mount option for both ext3 and ext4 filesystems for a fair comparison. When tuning ext3 for best benchmark numbers, it is often worthwhile to try changing the data journaling mode; '-o data=writeback' can be faster for some workloads. (Note however that running mounted with data=writeback can potentially leave stale data exposed in recently written files in case of an unclean shutdown, which could be a security exposure in some situations.) Configuring the filesystem with a large journal can also be helpful for metadata-intensive workloads.

Features

Currently Available

- ability to use filesystems > 16TB (e2fsprogs support not available yet)
- extent format reduces metadata overhead (RAM, IO for access, transactions)
- extent format more robust in face of on-disk corruption due to magics,
- internal redundancy in tree
- improved file allocation (multi-block alloc)
- lift 32000 subdirectory limit imposed by i_links_count[1]
- nsec timestamps for mtime, atime, ctime, create time
- inode version field on disk (NFSv4, Lustre)
- reduced e2fsck time via uninit_bg feature
- journal checksumming for robustness, performance
- persistent file preallocation (e.g for streaming media, databases)
- ability to pack bitmaps and inode tables into larger virtual groups via the flex_bg feature
- large file support

- inode allocation using large virtual block groups via flex_bg
- delayed allocation
- large block (up to pagesize) support
- efficient new ordered mode in JBD2 and ext4 (avoid using buffer head to force the ordering)
- Case-insensitive file name lookups
- file-based encryption support (fscrypt)
- file-based verity support (fsverity)

[1] Filesystems with a block size of 1k may see a limit imposed by the directory hash tree having a maximum depth of two.

case-insensitive file name lookups

The case-insensitive file name lookup feature is supported on a per-directory basis, allowing the user to mix case-insensitive and case-sensitive directories in the same filesystem. It is enabled by flipping the +F inode attribute of an empty directory. The case-insensitive string match operation is only defined when we know how text in encoded in a byte sequence. For that reason, in order to enable case-insensitive directories, the filesystem must have the casefold feature, which stores the filesystem-wide encoding model used. By default, the charset adopted is the latest version of Unicode (12.1.0, by the time of this writing), encoded in the UTF-8 form. The comparison algorithm is implemented by normalizing the strings to the Canonical decomposition form, as defined by Unicode, followed by a byte per byte comparison.

The case-awareness is name-preserving on the disk, meaning that the file name provided by userspace is a byte-per-byte match to what is actually written in the disk. The Unicode normalization format used by the kernel is thus an internal representation, and not exposed to the userspace nor to the disk, with the important exception of disk hashes, used on large case-insensitive directories with DX feature. On DX directories, the hash must be calculated using the casefolded version of the filename, meaning that the normalization format used actually has an impact on where the directory entry is stored.

When we change from viewing filenames as opaque byte sequences to seeing them as encoded strings we need to address what happens when a program tries to create a file with an invalid name. The Unicode subsystem within the kernel leaves the decision of what to do in this case to the filesystem, which select its preferred behavior by enabling/disabling the strict mode. When Ext4 encounters one of those strings and the filesystem did not require strict mode, it falls back to considering the entire string as an opaque byte sequence, which still allows the user to operate on that file, but the case-insensitive lookups won't work.

Options

When mounting an ext4 filesystem, the following option are accepted: (*) == default

ro
: Mount filesystem read only. Note that ext4 will replay the journal (and thus write to the partition) even when mounted "read only". The mount options "ro,noload" can be used to prevent writes to the filesystem.

journal_checksum
: Enable 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.

journal_path=path, journal_dev=devnum
: 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 through either 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
: All data are committed into the journal prior to being written into the main file system. Enabling this mode will disable delayed allocation and O_DIRECT support.

data=ordered (*)
: All data are forced directly out to the main file system prior to its metadata being committed to the journal.

data=writeback
: Data ordering is not preserved, data may be written into the main file system after its metadata has been committed to the journal.

commit=nrsec (*)
: This setting limits the maximum age of the running transaction to 'nrsec' seconds. The default value is 5 seconds. This means that if you lose your power, you will lose as much as the latest 5 seconds of metadata changes (your filesystem will not be damaged though, thanks to the journaling). This default value (or any low value) will hurt performance, but it's good for data-safety. Setting it to 0 will have the same effect as leaving it at the default (5 seconds). Setting it to very large values will improve performance. Note that due to delayed allocation even older data can be lost on power failure since writeback of those data begins only after time set in

/proc/sys/vm/dirty_expire_centisecs.

barrier=<0|1(*)>, barrier(*), nobarrier
: This enables/disables the use of write barriers in the jbd code. barrier=0 disables, barrier=1 enables. This also requires an IO stack which can support barriers, and if jbd gets an error on a barrier write, it will disable 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. The mount options "barrier" and "nobarrier" can also be used to enable or disable barriers, for consistency with other ext4 mount options.

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 default value is 32 blocks.

nouser_xattr
: Disables Extended User Attributes. See the attr(5) manual page for more information about extended attributes.

noacl
: This option disables POSIX Access Control List support. If ACL support is enabled in the kernel configuration (CONFIG_EXT4_FS_POSIX_ACL), ACL is enabled by default on mount. See the acl(5) manual page for more information about acl.

bsddf(*)
: Make 'df' act like BSD.

minixdf
: Make 'df' act like Minix.

debug
: Extra debugging information is sent to syslog.

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

errors=remount-ro
: Remount the filesystem read-only on an error.

errors=continue
: Keep going on a filesystem error.

errors=panic
: Panic and halt the machine if an error occurs. (These mount options override the errors behavior specified in the superblock, which can be configured using tune2fs)

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.

grpid | bsdgroups
: New objects have the group ID of their parent.

nogrpid (*) | sysvgroups
: New objects have the group ID of their creator.

resgid=n
: The group ID which may use the reserved blocks.

resuid=n
: The user ID which may use the reserved blocks.

sb=
: Use alternate superblock at this location.

quota, noquota, grpquota, usrquota
: These options are ignored by the filesystem. They are used only by quota tools to recognize volumes where quota should be turned on. See documentation in the quota-tools package for more details (http://sourceforge.net/projects/linuxquota).

jqfmt=<quota type>, usrjquota=<file>, grpjquota=<file>
: These options tell filesystem details about quota so that quota information can be properly updated during journal replay. They replace the above quota options. See documentation in the quota-tools package for more details (http://sourceforge.net/projects/linuxquota).

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 file system blocks.

delalloc (*)
: Defer block allocation until just before ext4 writes out the block(s) in question. This allows ext4 to better allocation decisions more efficiently.

nodelalloc
: Disable delayed allocation. Blocks are allocated when the data is copied from userspace to the page cache, either via the write(2) system call or when an mmap'ed page which was previously unallocated is written for the first time.

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 15000us (15ms). 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.

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 file system is unmounted.

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 the system performance while file system'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.

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

block_validity(*), noblock_validity
: These options enable or disable the in-kernel facility for tracking filesystem metadata blocks within internal data structures. This allows multi- block allocator and other routines to notice bugs or corrupted allocation bitmaps which cause blocks to be allocated which overlap with filesystem metadata blocks.

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 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 512mb memory available, a 176mb directory may seriously cramp the system's style.)

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

dax
: Use direct access (no page cache). See Documentation/filesystems/dax.rst. Note that this option is incompatible with data=journal.

inlinecrypt
: When possible, encrypt/decrypt the contents of encrypted files using the blk-crypto framework rather than

filesystem-layer encryption. This allows the use of inline encryption hardware. The on-disk format is unaffected. For more details, see Documentation/block/inline-encryption.rst.

Data Mode

There are 3 different data modes:

- writeback mode

  In data=writeback mode, ext4 does not journal data at all. This mode provides a similar level of journaling as that of XFS, JFS, and ReiserFS in its default mode - metadata journaling. A crash+recovery can cause incorrect data to appear in files which were written shortly before the crash. This mode will typically provide the best ext4 performance.

- ordered mode

  In data=ordered mode, ext4 only officially journals metadata, but it logically groups metadata information related to data changes with the data blocks into a single unit called a transaction. When it's time to write the new metadata out to disk, the associated data blocks are written first. In general, this mode performs slightly slower than writeback but significantly faster than journal mode.

- journal mode

  data=journal mode provides full data and metadata journaling. All new data is written to the journal first, and then to its final location. In the event of a crash, the journal can be replayed, bringing both data and metadata into a consistent state. This mode is the slowest except when data needs to be read from and written to disk at the same time where it outperforms all others modes. Enabling this mode will disable delayed allocation and O_DIRECT support.

/proc entries

Information about mounted ext4 file systems can be found in /proc/fs/ext4. Each mounted filesystem will have a directory in /proc/fs/ext4 based on its device name (i.e., /proc/fs/ext4/hdc or /proc/fs/ext4/dm-0). The files in each per-device directory are shown in table below.

Files in /proc/fs/ext4/<devname>

mb_groups
: details of multiblock allocator buddy cache of free blocks

/sys entries

Information about mounted ext4 file systems can be found in /sys/fs/ext4. Each mounted filesystem will have a directory in /sys/fs/ext4 based on its device name (i.e., /sys/fs/ext4/hdc or /sys/fs/ext4/dm-0). The files in each per-device directory are shown in table below.

Files in /sys/fs/ext4/<devname>:

(see also Documentation/ABI/testing/sysfs-fs-ext4)

delayed_allocation_blocks
: This file is read-only and shows the number of blocks that are dirty in the page cache, but which do not have their location in the filesystem allocated yet.

inode_goal
: Tuning parameter which (if non-zero) controls the goal inode used by the inode allocator in preference to all other allocation heuristics. This is intended for debugging use only, and should be 0 on production systems.

inode_readahead_blks
: Tuning parameter which controls the maximum number of inode table blocks that ext4's inode table readahead algorithm will pre-read into the buffer cache.

lifetime_write_kbytes
: This file is read-only and shows the number of kilobytes of data that have been written to this filesystem since it was created.

max_writeback_mb_bump
: The maximum number of megabytes the writeback code will try to write out before move on to another inode.

mb_group_prealloc
: The multiblock allocator will round up allocation requests to a multiple of this tuning parameter if the stripe size is not set in the ext4 superblock

mb_max_inode_prealloc
: The maximum length of per-inode ext4_prealloc_space list.

mb_max_to_scan
: The maximum number of extents the multiblock allocator will search to find the best extent.

mb_min_to_scan
: The minimum number of extents the multiblock allocator will search to find the best extent.

mb_order2_req
: Tuning parameter which controls the minimum size for requests (as a power of 2) where the buddy cache is used.

mb_stats
: Controls whether the multiblock allocator should collect statistics, which are shown during the unmount. 1 means to collect statistics, 0 means not to collect statistics.

mb_stream_req
: Files which have fewer blocks than this tunable parameter will have their blocks allocated out of a block group specific preallocation pool, so that small files are packed closely together. Each large file will have its blocks allocated out of its own unique preallocation pool.

session_write_kbytes
: This file is read-only and shows the number of kilobytes of data that have been written to this filesystem since it was mounted.

reserved_clusters
: This is RW file and contains number of reserved clusters in the file system which will be used in the specific situations to avoid costly zeroout, unexpected ENOSPC, or possible data loss. The default is 2% or 4096 clusters, whichever is smaller and this can be changed however it can never exceed number of clusters in the file system. If there is not enough space for the reserved space when mounting the file mount will _not_ fail.

Ioctls

Ext4 implements various ioctls which can be used by applications to access ext4-specific functionality. An incomplete list of these ioctls is shown in the table below. This list includes truly ext4-specific ioctls (`EXT4_IOC_*`) as well as ioctls that may have been ext4-specific originally but are now supported by some other filesystem(s) too (`FS_IOC_*`).

Table of Ext4 ioctls

FS_IOC_GETFLAGS
: Get additional attributes associated with inode. The ioctl argument is an integer bitfield, with bit values described in ext4.h.

FS_IOC_SETFLAGS
: Set additional attributes associated with inode. The ioctl argument is an integer bitfield, with bit values described in ext4.h.

EXT4_IOC_GETVERSION, EXT4_IOC_GETVERSION_OLD
: Get the inode i_generation number stored for each inode. The i_generation number is normally changed only when new inode is created and it is particularly useful for network filesystems. The '_OLD' version of this ioctl is an alias for FS_IOC_GETVERSION.

EXT4_IOC_SETVERSION, EXT4_IOC_SETVERSION_OLD
: Set the inode i_generation number stored for each inode. The '_OLD' version of this ioctl is an alias for FS_IOC_SETVERSION.

EXT4_IOC_GROUP_EXTEND
: This ioctl has the same purpose as the resize mount option. It allows to resize filesystem to the end of the last existing block group, further resize has to be done with resize2fs, either online, or offline. The argument points to the unsigned logn number representing the filesystem new block count.

EXT4_IOC_MOVE_EXT
: Move the block extents from orig_fd (the one this ioctl is pointing to) to the donor_fd (the one specified in move_extent structure passed as an argument to this ioctl). Then, exchange inode metadata between orig_fd and donor_fd. This is especially useful for online defragmentation, because the allocator has the opportunity to allocate moved blocks better, ideally into one contiguous extent.

EXT4_IOC_GROUP_ADD
: Add a new group descriptor to an existing or new group descriptor block. The new group descriptor is described by ext4_new_group_input structure, which is passed as an argument to this ioctl. This is especially useful in conjunction with EXT4_IOC_GROUP_EXTEND, which allows online resize of the filesystem to the end of the last existing block group. Those two ioctls combined is used in userspace online resize tool (e.g. resize2fs).

EXT4_IOC_MIGRATE
: This ioctl operates on the filesystem itself. It converts (migrates) ext3 indirect block mapped inode to ext4 extent mapped inode by walking through indirect block mapping of the original inode and converting contiguous block ranges into ext4 extents of the temporary inode. Then, inodes are swapped. This ioctl might help, when migrating from ext3 to ext4 filesystem, however suggestion is to create fresh ext4 filesystem and copy data from the backup. Note, that filesystem has to support extents for this ioctl to work.

EXT4_IOC_ALLOC_DA_BLKS
: Force all of the delay allocated blocks to be allocated to preserve application-expected ext3 behaviour. Note that this will also start triggering a write of the data blocks, but this behaviour may change in the future as it is not necessary and has been done this way only for sake of simplicity.

EXT4_IOC_RESIZE_FS
: Resize the filesystem to a new size. The number of blocks of resized filesystem is passed in via 64 bit integer

argument. The kernel allocates bitmaps and inode table, the userspace tool thus just passes the new number of blocks.

EXT4_IOC_SWAP_BOOT
: Swap i_blocks and associated attributes (like i_blocks, i_size, i_flags, ...) from the specified inode with inode EXT4_BOOT_LOADER_INO (#5). This is typically used to store a boot loader in a secure part of the filesystem, where it can't be changed by a normal user by accident. The data blocks of the previous boot loader will be associated with the given inode.

References

kernel source: <file:fs/ext4/>
<file:fs/jbd2/>

programs: http://e2fsprogs.sourceforge.net/

useful links: https://fedoraproject.org/wiki/ext3-devel
http://www.bullopensource.org/ext4/ http://ext4.wiki.kernel.org/index.php/Main_Page
https://fedoraproject.org/wiki/Features/Ext4