Digital Forensics

Federico Conti

2024/25

Contents

The Ext File System Family

The Ext (Extended) file system family—Ext2, Ext3, and Ext4—is a series of filesystems used in Linux. These are based on the traditional Unix File System (UFS), but add several modern capabilities over time.

Key Modern Features:

- Extended Attributes / POSIX ACLs: Allow more detailed permissions than traditional Unix read/write/execute.
- Journaling (from Ext3 onwards): Keeps a log (journal) of changes to help recover quickly after crashes.
- Encryption: Protects file contents and filenames (introduced in Ext4).
- 64-bit Support: Allows management of much larger files and filesystems.
- More…

The Ext file systems are modular, with features grouped by their compatibility level:

Feature Type	What It Means	Example
Compatible	If not supported, OS can still mount the FS normally	has_journal
Incompatible	If not supported, OS must not mount the FS	encrypt
Read-Only Compatible	If not supported, OS can mount in read-only mode	dir_index

Support for Very Large Volumes

- Ext3: Limited to volumes up to 16 TB (terabytes), calculated as $16 \cdot 10^{12}/2^{44}$.
- Ext4: Extends this limit to 1 exabyte (EB), or $10^{18}/2^{60}$. This makes Ext4 suitable for modern systems with very high storage demands.

Maximum File Size

- Ext3: The maximum size of a single file is 2 TB.
- Ext4: Increases this limit to 16 TB, making it more suitable for applications requiring the management of large files.

Backward Compatibility

- Ext4: Can mount Ext3 file systems and treat them as Ext4. This process is called “on-the-fly conversion”: existing files remain in the Ext3 format, but new files will use Ext4’s advanced features.
- Ext3: Cannot mount Ext4 file systems, as Ext3 does not support the new features introduced in Ext4.

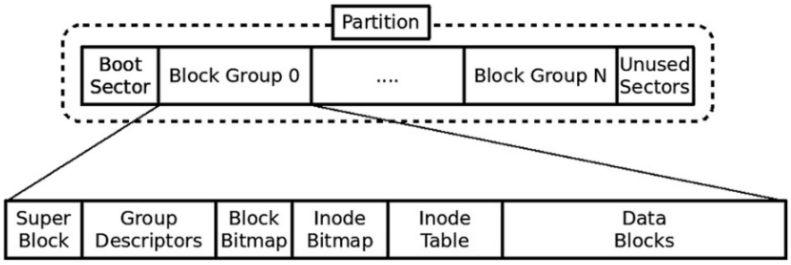
Extents

Ext4 uses a structure called extents to map data blocks more efficiently. Extents represent a continuous range of blocks, reducing fragmentation and improving performance. This approach is similar to the one used by NTFS (the Windows file system).

Layout

In the Ext file system family (Ext2, Ext3, Ext4):

- Sectors (the smallest physical units on a disk, typically 512 bytes) are grouped into blocks of 1, 2, or 4 KB.
- These blocks are conceptually similar to "clusters" in FAT or NTFS.
- Unlike the Unix File System (UFS), Ext does not assemble blocks from smaller "fragments." in Ext, the terms "block" and "fragment" are essentially synonymous (the term "block" is predominantly used).
- The superblock is located at an offset of 1024 bytes from the start of the volume (not at the very beginning). It contains all the fundamental parameters of the file system.
- The size of the superblock is 1024 bytes. An optional reserved area may follow the superblock.
- Boot code, if present, resides in the Master Boot Record (MBR) or Volume Boot Record (VBR), not within the file system itself.
- The rest of the file system is divided into block groups.



A key detail: when the first block group (BG0) starts at offset 0:

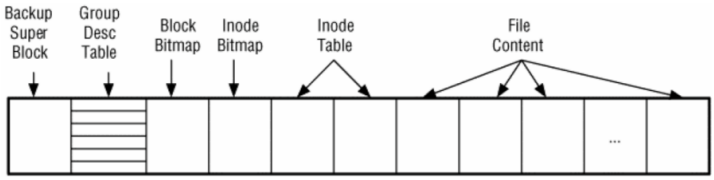
- The primary superblock is located at an offset of 1024 bytes from the volume start (or within block 0).
- Backup copies of the superblock (if present) are located at the beginning of other block groups (BG1, BG2, etc.).

Each "copy" of the superblock has its own specific metadata.

- This backup structure, with superblock copies in multiple locations, is a critical safety feature that allows recovery even if parts of the file system are damaged.

Each block group

1. contains its data bitmap, inode bitmap, inode table, data blocks
2. may contain the backup of the superblock and the GD table



Each group descriptor defines where to find the components of the block group.

- The main group descriptor table is located in the block immediately following the superblock.
- This table can be copied, after the superblock backup, to all or some groups, depending on the formatting options (see `sparse_super[2]` in the `ext4(5)` man page).

Ext4 introduces advanced options for managing metadata and resizing:

- flexible block groups (`flex_bg`): allows block group metadata to be placed anywhere, not necessarily within the same group.

- reserved GDT blocks: enables online resizing of the file system without requiring it to be unmounted.

Standard command `dumpe2fs` prints superblock and group information.

- All groups, except for the last, contain the same number of blocks n
- by default, n is equal to the # of bits in a block: per blocchi di 4 KB (4096 byte), ogni blocco contiene 32.768 bit (4096 × 8), quindi ogni gruppo può gestire 32.768 blocchi
  - the block bitmap is exactly 1 block
- the # of inodes can be chosen at format time, usually less than n

---

Example

Use `dumpe2fs/fsstat` to list

- block size = 2048 byte
- blocks and inodes per group = 888;296
- number of groups = 18

in the image linus-ext2.dd (SHA256: b6b1836ff1efef3a70…)

- Are there unused sector after the FS? no total size of fs == 16384*2048 == stat -c %s linus-ext2.dd
- All are group of the same size? no, the end group have a size of 400 blocks

---

Unused space

- The first 1024 bytes are not technically used
- There are unused bytes in the superblock
- There can be unused entries in the GD-table
  - Or they backup copies
- There are the reserved GDT blocks

Those are possible places where to hide data

Superblock details

Byte Range	Description	Essential
0–3	Number of inodes in file system	**Yes**
4–7	Number of blocks in file system	**Yes**
8–11	Number of blocks reserved to prevent file system from filling up	No
12–15	Number of unallocated blocks	No
16–19	Number of unallocated inodes	No
20–23	Block where block group 0 starts	**Yes**
24–27	Block size (saved as the number of places to shift 1,024 to the left)	**Yes**
28–31	Fragment size (saved as the number of bits to shift 1,024 to the left)	**Yes**
32–35	Number of blocks in each block group	**Yes**
36–39	Number of fragments in each block group	**Yes**
40–43	Number of inodes in each block group	**Yes**
44–47	Last mount time	No
48–51	Last written time	No
52–53	Current mount count	No
54–55	Maximum mount count	No
56–57	Signature (0xef53)	No

Byte Range	Description	Essential
58–59	File system state (see Table 15.2)	No
60–61	Error handling method (see Table 15.3)	No
62–63	Minor version	No
64–67	Last consistency check time	No
68–71	Interval between forced consistency checks	No
72–75	Creator OS (see Table 15.4)	No
76–79	Major version (see Table 15.5)	**Yes**
80–81	UID that can use reserved blocks	No
82–83	GID that can use reserved blocks	No
84–87	First non-reserved inode in file system	No
88–89	Size of each inode structure	**Yes**
90–91	Block group that this superblock is part of (if backup copy)	No
92–95	Compatible feature flags (see Table 15.6)	No
96–99	Incompatible feature flags (see Table 15.7)	**Yes**
100–103	Read only feature flags (see Table 15.8)	No
104–119	File system ID	No
120–135	Volume name	No
136–199	Path where last mounted on	No
200–203	Algorithm usage bitmap	No
204–204	Number of blocks to preallocate for files	No
205–205	Number of blocks to preallocate for directories	No
206–207	Unused	No
208–223	Journal ID	No
224–227	Journal inode	No
228–231	Journal device	No
232–235	Head of orphan inode list	No
236–1023	Unused	No

Example

1. use `dumpe2fs/fsstat` to check where the superblock copies are
2. dump the first three, and compare them: are they equal?
    - note: the main SB starts at offset 1024, the others at 0
    - are the 2nd and 3rd equal?

```
dumpe2fs linus-ext2.dd | grep "superblock"
    ##OUT##
    Primary superblock at 0, Group descriptors at 1-1
    Backup superblock at 888, Group descriptors at 889-889
    Backup superblock at 2664, Group descriptors at 2665-2665
    Backup superblock at 4440, Group descriptors at 4441-4441
    Backup superblock at 6216, Group descriptors at 6217-6217
    Backup superblock at 7992, Group descriptors at 7993-7993
    ###
#The primary superblock is located after the boot block
# the superblock, at volume offset 1024, contains the FS parameters
# the superblock is 1024 bytes in size
dd if=linus-ext2.dd bs=1k skip=1 count=1 of=sb1
```

```
#To convert from filesystem block (2048) numbers to 1KB blocks for dd, you need to multiply by 2:
dd if=linus-ext2.dd bs=1k skip=$((888*2)) count=1 of=sb2
dd if=linus-ext2.dd bs=1k skip=$((2664*2)) count=1 of=sb3

md5sum sb1 sb2 sb3
    ##OUT##
    922c7935bd45a7b36fc938f3b000d16c  sb1
    06469d508b7da81f55e090addf908598  sb2
    5858141cac833da5d7a0195dfd968119  sb3
    ###


vbindiff sb2 sb3

    ###
    change the bit that keeps track of the superblock group membership
    ###

vbindiff sb1 sb3
    ###
    more diversity because the primary superblock also tines the last path where the fs was mounted (no
    ###
```

Group descriptors details

Byte Range	Description	Essential
0–3	Starting block address of block bitmap	**Yes**
4–7	Starting block address of inode bitmap	**Yes**
8–11	Starting block address of inode table	**Yes**
12–13	Number of unallocated blocks in group	No
14–15	Number of unallocated inodes in group	No
16–17	Number of directories in group	No
18–31	Unused	No

Inode

Inodes are the primary metadata structure.

- fixed size, defined in the superblock, minimum 128 bytes (`dumpe2fs -h linus-ext2.dd | grep "Inode size"`)
- extra-space can be used to store extended attributes (otherwise, data blocks are used)
- small non-user content can be stored inside direct-block pointer array; e.g., symlink values
- each inode has a unique 'address' or number, starting with 1 (index 0 is not used)
- The inodes of each block group are stored in a table and the position of this table is specified in the block descriptor
- The allocation status of each inode is determined via the inode bitmap nd the position of this table is specified in the block descriptor

Standard commands can be used to inspect the FS:ù

- `-i`,`-lai` in `ls`, shows inode-numbers

- `stat` shows some metadata

debugfs is a FS "debugger" that can display a lot of information.

```
ln pippo.txt pippo-hardlink.txt
ln -s pippo.txt pippo-hardlink.txt
ls -li
    ##OUT##
    1445743 -rw-r--r--  2 root    root       0 Apr  3 10:20 pippo-hardlink.txt
    1445746 lrwxrwxrwx  1 root    root       9 Apr  3 10:42 pippo-symblink.txt -> pippo.txt
    1445743 -rw-r--r--  2 root    root       0 Apr  3 10:20 pippo.txt
    ###

    ###
    for symb link we get differen inode number beacsue is
    the newfile and fs must allocated a new inode and
    content for symb link. The contet of ippo-symblink.txt is the path " -> pippo.txt",
    that just a string that is resolved we we tryb to open the pippo-symblink.txt.
    ###
```

Inodes 1 to 10 are reserved

- 1 is (was?) used for keeping track of bad blocks
- 2 is used for the root directory
- 8 is typically for the journal

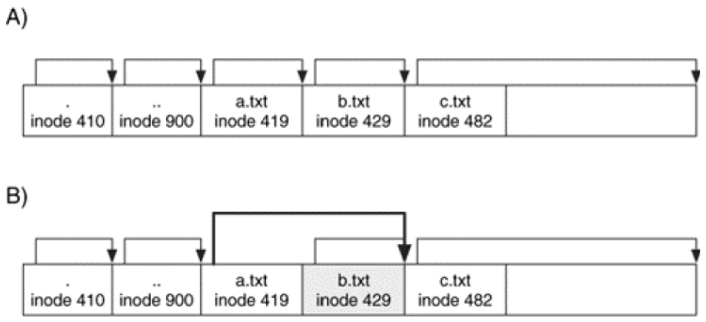
Byte Range	Description	Essential
0–1	File mode (type and permissions)	**Yes**
2–3	Lower 16 bits of user ID	No
4–7	Lower 32 bits of size in bytes	**Yes**
8–11	Access Time	No
12–15	Change Time	No
16–19	Modification time	No
20–23	Deletion time	No
24–25	Lower 16 bits of group ID	No
26–27	Link count	No
28–31	Sector count	No
32–35	Flags	No
36–39	Unused	No
40–87	12 direct block pointers	**Yes**
88–91	1 single indirect block pointer	**Yes**
92–95	1 double indirect block pointer	**Yes**
96–99	1 triple indirect block pointer	**Yes**
100–103	Generation number (NFS)	No
104–107	Extended attribute block (File ACL)	No
108–111	Upper 32 bits of size / Directory ACL	No
112–115	Block address of fragment	No
116–116	Fragment index in block	No
117–117	Fragment size	No
118–119	Unused	No
120–121	Upper 16 bits of user ID	No
122–123	Upper 16 bits of group ID	No

Byte Range	Description	Essential
124–127	Unused	No

Directories

The old directory entry was a simple data structure containing

- the file name (variable length, 1-255 chars) where the length of an entry is rounded up to a multiple of four
- the inode number (AKA index+1)



The newer one uses one byte in the filename-length to store the file type.

Allows detecting when an inode has been reassigned (reallocated)!

Byte Range	Description	Essential
0–3	Inode value	**Yes**
4–5	Length of this entry	**Yes**
6–6	Name length	**Yes**
7–7	File type	No
8+	Name in ASCII	**Yes**

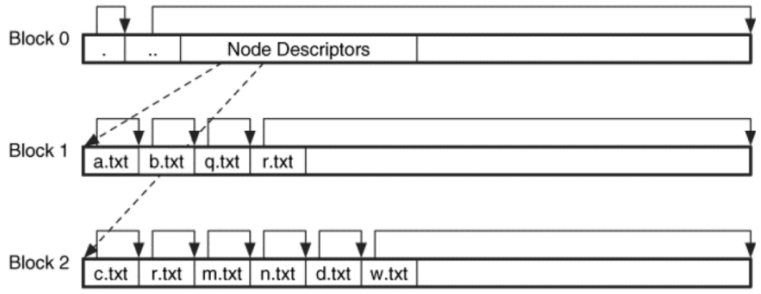
File Type	Description
1	Regular file
2	Directory
3	Character device
4	Block device
5	FIFO
6	Unix Socket
7	Symbolic link

In ext3 and ext4, option `dir_index` allows the FS to use hashed B-trees to speed up name lookups

- similar to NTFS: This approach is conceptually similar to that used by NTFS in Windows
- when this functionality is active, a `ro-compatible` (read-only compatible) flag is set in the superblock (This means that systems that do not support this function can still mount the file system, but only in read-only mode)

Still the same directory-entry structures, but in sorted order

- each entry continues to have the same format and contain the same information (Inode value, Length of this entry …) but instead of storing the entries in a linear manner (one after the other), the entries are organised in a B-tree structure



Example

List files in the root directory of linus-ext2.dd by

- (ro) mounting it
- using TSK's fls
- using ImHex
  - get the block-size from the SB == 1 (=1K)
  - the GD-table is in the block after the SB
  - decode the first GD and get the block for the (first) inode-table
  - the inode n.2 (index 1) is for the root directory
  - the first direct-block-pointer points to the beginning of the directory
  - decode the directory entries

```
mount -oro  linus-ext2.dd mnt/linus-ext2
ls -li
    ##OUT##
    total 28
    12 -rw-r----- 2 root root   773 Apr  8  2023 hard_link.txt
    12 -rw-r----- 2 root root   773 Apr  8  2023 linus.txt
    15 lrwxrwxrwx 1 root root   109 Apr  9  2023 long_symlink -> /media/someuser/whatever/subdir1/subdi
    11 drwx------ 2 root root 16384 Apr  8  2023 lost+found
    5033 drwxr-xr-x 4 root root  2048 Apr  8  2023 pics
    13 lrwxrwxrwx 1 root root     4 Apr  8  2023 pictures -> pics
    14 -rw-r--r-- 1 root root     8 Apr  9  2023 very_short_text.txt
    ###

fls -rp linus-ext2.dd
    ##OUT##
    d/d 11: lost+found
    d/d 5033:       pics
    d/d 5034:       pics/tux
    r/r 5040:       pics/tux/tux-lego.png
    r/r 5041:       pics/tux/tux-windows.png
    r/r 5042:       pics/tux/tux.png
    d/d 5035:       pics/linus
    r/r 5036:       pics/linus/linus_2018.jpg
    r/r 12: linus.txt
```

```
l/l 13: pictures
r/r 12: hard_link.txt
r/r 14: very_short_text.txt
l/l 15: long_symlink
V/V 5625:       $OrphanFiles
-/r * 5037:     $OrphanFiles/OrphanFile-5037
-/r * 5038:     $OrphanFiles/OrphanFile-5038
-/r * 5039:     $OrphanFiles/OrphanFile-5039
###
```

Extended

Extended attributes are an advanced feature of Ext file systems (Ext2/3/4) that allows additional metadata to be associated with files and directories. This metadata is stored as name-value pairs.

- Ogni attributo è una coppia composta da:
  - Name: specified in the `namespace.attribute-name format` (current namespaces are security, system, trusted, and user)
  - Value: An arbitrary datum associated with the name

Gli attributi estesi sono utilizzati per implementare le Access Control Lists (ACL), che consentono di definire permessi più dettagliati rispetto ai tradizionali permessi Unix.

```
setfacl -m u:username:rwx file.txt  # Set up an ACL
getfacl file.txt                    # View ACLs
```

kernel and filesystem may place limits on the maximum number and size of extended attributes that can be associated with a file

Extended attributes are stored in a separate block associated with the file. If several files share the same attributes, they can use the same block to optimise space.

```
getfattr -d file.txt # Display extended attributes

# set an extended attribute
setfattr -n user.comment -v "Questo è un file importante" file.txt
setfattr -x user.comment file.txt # setfattr -x user.comment file.txt
```

See xattr(7)