The Ext File System Family

ext2/ext3 and ext4

Giovanni Lagorio

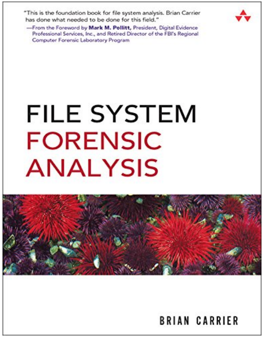
giovanni.lagorio@unige.it
https://csec.it/people/giovanni_lagorio
X/GitHub/...: zxgio

DIBRIS - Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi
University of Genova, Italy

Outline

File System Forensics Analysis

The “bible” for this part of the course is Brian Carrier’s “File System Forensics Analsysis” [Car05], where you can find more details

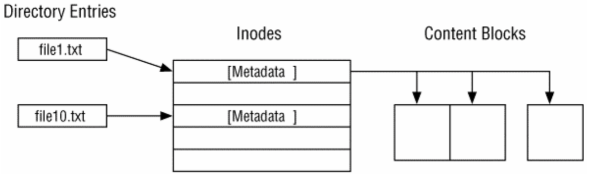


However, for the latest additions, see `ext4(5)` and [Fai12]

Introduction

Linux Ext2/3/4

- based on the traditional Unix FS (UFS)



- with modern features
  - extended-attributes/POSIX ACLs
  - journaling
  - encryption
  - 64-bit support
  - ...

Extensibility

Ext-family designed to be extensible via optional features, organized into:

compatible if not supported, the OS can still mount the FS

- E.g., `has_journal`

incompatible if not supported, the FS should not be mounted

- E.g., `encrypt` (only filenames and data blocks; dm-crypt is usually better for single-user PCs)

r.o. compatible if not supported, the OS can still r.o. mount the FS

- E.g., `dir_index`

Ext4

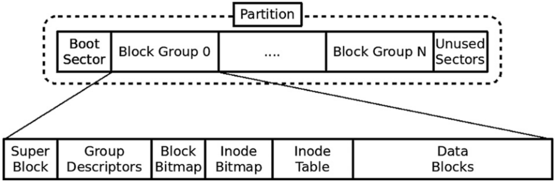
- support for very large volumes: ext3 limited to 16TB $\approx 16 \cdot 10^{12}/2^{44}$, ext4 to 1 exabyte $\approx 10^{18}/2^{60}$
  - max file size from 2 TB to 16 TB
- backward compatibility: ext4 can mount ext3 file-systems, as ext4 ("converted on-the-fly": new files will use new features)
  - but ext3 cannot mount ext4
- extents for more efficient data-block mapping, like NTFS
- ...

see [Fai12]

Outline

Blocks

- sectors are grouped into 1/2/4 KB blocks (=FAT/NTFS clusters)
  - ext*N*, differently from UFS, do not split blocks into "fragments" (although there are traces of them in some fields)
- the superblock, at volume offset 1024, contains the FS parameters
  - the superblock is 1024 bytes in size
  - then there is an optional reserved area
- the boot-code, if any, is contained in the MBR/VBR
- the remainder of the FS is divided into block groups



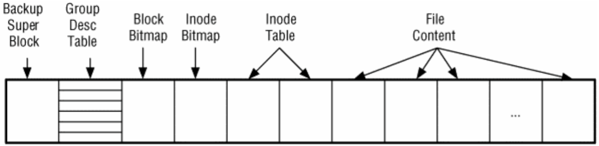
from [Fai12]

the SB at offset 1024 of the volume, while its backup copies (when present) are at the beginning of block-groups 1, 2, . . . note that each "copy" has its own metadata

Block groups

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 (1)
- the main group descriptor table is located in the block after the SB
  - can be copied, after the superblock backup, in all/some groups (depending on format options; see `sparse_super`[2] in `ext4(5)`)
  - ext4 also offers `flex_bg` option, which allows per-block group metadata to be placed anywhere
- there also are reserved GDT blocks to allow *on-line* resizing of the FS

Non-TSK commands

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
  - the block bitmap is exactly 1 block
- the # of inodes can be chosen at format time, usually less than n

Demo/exercise

Use `dumpe2fs`/`fsstat` to list

- block size
- blocks and inodes per group
- number of groups

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

Are there unused sector after the FS? All are group of the same size?

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 (1/3)

Table 15.1. Data structure for the ExtX superblock.

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

Superblock: details (2/3)

52–53	Current mount count	No
54–55	Maximum mount count	No
56–57	Signature (0xef53)	No
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

Superblock: details (3/3)

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

Exercise

Demo/exercise

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?

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

Outline

Inodes

Inodes are the primary metadata structure

- fixed size, defined in the superblock
  - minimum 128 bytes
  - 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 an "address", starting with 1 (at index 0)
- the inodes of each block-group are stored in a table
  - whose location is specified in the block descriptor
- allocation status determined using the inode bitmap
  - whose location is specified in the block descriptor

Demo/exercise

In our example, which is the size of an inode?

Standard commands to inspect the FS

Standard commands can be used to inspect the FS:

- `-i` in `ls`, shows inode-numbers
- `stat` shows some metadata
- `debugfs` is a FS “debugger” that can display *a lot* of information

Demo/Exercise: Hard-links vs Soft/Sym-links

Check the inode-numbers when creating hard vs soft links

Reserved inodes

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

Inode: details (1/2)

Byte Range	Description	Essential
0–1	File mode (type and permissions) (see Tables 15.11, 15.12, and 15.13)	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 (see Table 15.14)	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

Inode: details (2/2)

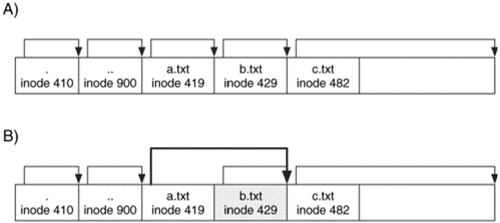
100–103	Generation number (NFS)	No
104–107	Extended attribute block (File ACL)	No
108–111	Upper 32 bits of size / Directory ACL Yes /	No
112–115	Block address of fragment	No
116–116	Fragment index in block	No
117–117	Fragment size	No
118–119	Unused	No
120–121	Upper 16 bits of user ID	No
122–123	Upper 16 bits of group ID	No
124–127	Unused	No

Outline

Directories

The old directory entry was a simple data structure containing

- the file name (variable length, 1-255 chars)
  - 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. This could be used to detect when an inode has been reallocated

Directories (new version): details

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 (see Table 15.24)	No
8+	Name in ASCII	Yes

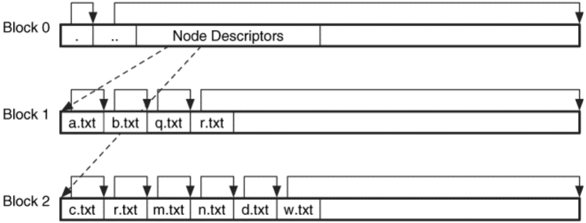
file type:

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

Hash trees

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

- similar to NTFS
- a ro-compatible feature flag will be set in the superblock
- still the same directory-entry structures, but in sorted order
  - node descriptors are “hidden” like deleted files



Exercises (1/2)

Demo/exercise

If your root FS is ext4, verify the inode number of the root directory of your system

Demo/exercise

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
  - 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
  - ...

Exercises (2/2)

Demo/exercise

List the data block for `/pics/linus/linus_2018.jpg`, by using

1. TSK
   - output can be confusing: "Direct Blocks" are all blocks, while "Indirect Blocks" are the blocks containing the indirect pointers and so on
2. ImHex; *beware:* the inode number for `/pics` is 5033
   i.e., it's the first inode of Group 18 (index 17); since, $17 * 296 = 5032$

Outline

Extended attributes

- extended attributes are a list of name and value pairs
- to get/set attributes: commands `getfattr`/`setfattr`
- the attributes are stored in a block allocated to the file
  - If more files have the same attributes, the files share the same block
- One use is implementing POSIX ACLs, see `setfacl`

Extended attributes: details

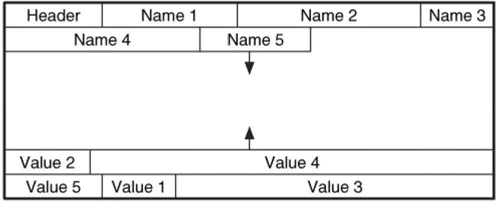


Table 15.15. Data structure for the extended attribute header.

Byte Range	Description	Essential
0–3	Signature (0xEA020000)	No
4–7	Reference count	No
8–11	Number of blocks	Yes
12–15	Hash	No
16–31	Reserved	No

Table 15.16. Data structure for the extended attribute name entries.

Byte Range	Description	Essential
0–0	Length of name	Yes
1–1	Attribute type (see Table 15.17)	Yes
2–3	Offset to value	Yes
4–7	Block location of value	Yes
8–11	Size of value	Yes
12–15	Hash of value	No
16+	Name in ASCII	Yes

Demo/exercise

Extended attributes

- List the extended attributes by ro-mounting the image and `getfattr`
- Try to decode them using ImHex
  - hint: one is "easy", but others are tricky; why?

References

[Car05] Brian Carrier.
File System Forensic Analysis.
Addison-Wesley Professional, 2005.

[Fai12] Kevin D. Fairbanks.
An analysis of ext4 for digital forensics.
Digital Investigation, 9:S118–S130, 2012.
The Proceedings of the Twelfth Annual DFRWS Conference.