HOWTO recover deleted files on an ext3 file system

Carlo Wood, Mar 2008

Introduction

It happens to everyone sooner or later: a split second after you hit Enter you realize your mistake, but it's too late; you just deleted a valuable file or directory for which no backup exists. Or maybe you have a backup, but it's a month old... and in shock you see the past month flash before your eyes as you realize in pain what you'll have to do all over again...

Fortunately, you remember that files are never *really* deleted, at most overwritten by new content. So, you remount the disk read-only as fast as possible. But now?

If you Google for "undelete ext3", almost every article you find will be users asking if it's possible and the answer is every time: no.

The most frequently quoted passage comes from the ext3 FAQ itself:

> Q: How can I recover (undelete) deleted files from my ext3 partition?
>
> Actually, you can't! This is what one of the developers, Andreas Dilger, said about it:
>
> *In order to ensure that ext3 can safely resume an unlink after a crash, it actually zeros out the block pointers in the inode, whereas ext2 just marks these blocks as unused in the block bitmaps and marks the inode as "deleted" and leaves the block pointers alone.*
>
> *Your only hope is to "grep" for parts of your files that have been deleted and hope for the best.*

However, this is utter nonsense. *All* information is still there, also the block pointers. It is just slightly less likely that those are still there (than on ext2), since they have to be recovered from the journal. On top of that, the meta data is less coherently related to the real data so that heuristic algorithms are needed to find things back.

On February 7th, 2008, I accidently deleted my whole home directory: over 3 GB of data, deleted with `rm -rf`. The only backup that I had was from June 2007. Not being able to undelete was *unacceptable*. So, I ignored what everyone tried to tell me and started to learn how an ext3 file system really works, and what *exactly* happens when files are deleted...

Three weeks and nearly 5000 lines of code later, I had recovered every file on my disk.

What You Should Know Before You Begin

The tool that I wrote assumes a spike of *recently* deleted files (shortly before the last unmount). It does NOT deal with a corrupted file system, only with accidently but *cleanly* deleted files.

Also, the program is in a beta stage: as soon as I recovered my own data *I stopped the development* of the program. Therefore, it is likely that things might not work entirely out-of-the-box for you. I stashed the program with asserts, which makes it likely that if something doesn't work for you then the program will abort instead of trying to gracefully recover. In that case you will have to dig deeper, and finish the program yourself, so to say.

The program only needs read access to the file system with the deleted files: it does not attempt to recover the files. Instead, it allows you to make a copy of deleted files and writes those to a newly created directory tree in the *current directory* (which obviously should be a different file system). All paths are relative to the root of the partion, thus— if you are analysing a partition /dev/md5 which was mounted under /home, then /home is unknown to the partition and to the program and therefore not part of the path(s). Instead, a path will be something like for example "carlo/c++/foo.cc", without leading slash. The parition root (/home in the example) is the *empty* string, not '/'.

The name of the program, ext3grep was chosen because I was planning to write a highly intelligent program that would be able to reconstruct files by searching for blocks that looked *similar* to expected blocks (based on an old backup, or other rules). The *grep* in the name was in anticipation that the quote from the ext3 developer was true: I was preparing for the need to work with sets of blocks, each set corresponding to a search pattern and weighted with a likeliness, upon which then one would have to work with set operators in order to reduce the number of blocks and assign them to files, and give them their order. However, nothing like it turned out to be needed. Nevertheless, I kept the name ext3grep because someone might want to add a true grep-like functionality to the program (at this moment it's grep-functionality is limited to fixed strings, printing matching block numbers to standard output).

How Does EXT3 Store Files?

Block sizes

The content of files is stored in contiguous blocks of 4096 bytes (the actual size depends on command line parameters passed to mke2fs when the file system was first created and can be 1024, 2048 or 4096 bytes). A harddisk is a "block device", meaning that every I/O is done in terms those blocks; one can only read/write an integral number of a blocks at a time. This doesn't necessarily mean that the minimum size of a contiguous file fragment is of the same size (although it can only be smaller), but in practise it is. In fact, the program will not work if the fragment size is unequal the block size.

The actual block size, as well as the actual fragment size, are stored in the superblock and can be retrieved with the option `--superblock`. For example,

```
$ ext3grep $IMAGE --superblock | grep 'size:'
Block size: 4096
Fragment size: 4096
```

Here IMAGE is an environment variable that was set to the name of the device (or a copy thereof made with dd) of the partition holding the file system. For example /dev/sdd2 (in general, any of the device names as returned by the command df under the heading "Filesystem"). Normally only root can read devices directly, but you can (temporarily) make them readable by you, of course, or make a backup image with dd. Note that, for example, /dev/sdd is NOT a partition (note the missing digit) and will not contain usable data for our purpose.

The entire partition is divided into an integral number of blocks, starting to count at 0. Thus, if you ever want to make a copy of block number N, you could do:

```
$ dd if=$IMAGE bs=4096 count=1 skip=$N of=block.$N
```

Where N runs from 0 till (but not including) the number of blocks as stored in the superblock. For example,

```
$ ext3grep $IMAGE --superblock | grep 'Blocks count:'
Blocks count: 2441824
```

Having any block number, one can print information about it by using the command line option `--block`. For example,

```
$ ext3grep $IMAGE --ls --block 600
[...]
Group: 0
Block 600 is Allocated. It's inside the inode table of group 0 (inodes [1 - 33>).

$ ext3grep $IMAGE --ls --block 1109
[...]
Group: 0

Block 1109 is a directory. The block is Allocated

          .--- File type in dir_entry (r=regular file, d=directory, l=symlink)
          |          .--- D: Deleted ; R: Reallocated
Indx Next |  Inode   | Deletion time                       Mode        File name
=========+==========+----------------data-from-inode------+-----------+=========
   0    1 d       2                                      drwxr-xr-x  .
   1  end d       2                                      drwxr-xr-x  ..
   2    3 d      11  D 1202351093 Thu Feb  7 03:24:53 2008  drwxr-xr-x  lost+found
   3  end d  195457  D 1202352103 Thu Feb  7 03:41:43 2008  drwxr-xr-x  carlo
```

The superblock

The superblock isn't really a block. It's size is always 1024 bytes and the first superblock starts at offset 1024. Thus, if the block size is 1024 then the superblock is block 1, but if the block size is 2048 or 4096, then the superblock is part of block 0. There are multiple backup copies elsewhere on the disk. ext3grep assumes that the first superblock is not corrupted and does not attempt to find or read the backup copies.

One could read the contents of the first superblock with dd as follows:

```
$ dd if=$IMAGE bs=1024 skip=1 count=1 of=superblock
```

The meaning of each byte in the superblock is given in table 1.

Bytes	type	Description
0 .. 3	__le32	Inodes count
4 .. 7	__le32	Blocks count
8 .. 11	__le32	Reserved blocks count
12 .. 15	__le32	Free blocks count
16 .. 19	__le32	Free inodes count
20 .. 23	__le32	First data block
24 .. 27	__le32	Block size
28 .. 31	__le32	Fragment size
32 .. 35	__le32	Number of blocks per group
36 .. 39	__le32	Number of fragments per group
40 .. 43	__le32	Number of inodes per group
44 .. 47	__le32	Mount time
48 .. 51	__le32	Write time
52 .. 53	__le16	Mount count
54 .. 55	__le16	Maximal mount count
56 .. 57	__le16	Magic signature
58 .. 59	__le16	File system state
60 .. 61	__le16	Behaviour when detecting errors
62 .. 63	__le16	minor revision level
64 .. 67	__le32	Time of last check
68 .. 71	__le32	Max. time between checks
72 .. 75	__le32	OS
76 .. 79	__le32	Revision level
80 .. 81	__le16	Default uid for reserved blocks
82 .. 83	__le16	Default gid for reserved blocks
84 .. 87	__le32	First non-reserved inode
88 .. 89	__le16	Size of inode structure
90 .. 91	__le16	Block group number of this superblock
92 .. 95	__le32	Compatible feature set
96 .. 99	__le32	Incompatible feature set
100 .. 103	__le32	Readonly-compatible feature set
104 .. 119	__u8[16]	128-bit uuid for volume
120 .. 135	char[16]	Volume name
136 .. 199	char[64]	Directory where last mounted
200 .. 203	__le32	For compression
204	__u8	Number of blocks to try to preallocate
205	__u8	Number to preallocate for dirs
206 .. 207	__le16	Per group descriptors for online growth
208 .. 223	__u8[16]	uuid of journal superblock
224 .. 227	__le32	Inode number of journal file
228 .. 231	__le32	Device number of journal file
232 .. 235	__le32	Start of list of inodes to delete
236 .. 251	__le32[4]	HTREE hash seed
252	__u8	Default hash version to use
253 .. 255		Reserved
256 .. 259	__le32	Default mount options
260 .. 263	__le32	First metablock block group
264 .. 1023		Reserved

The C-struct for the superblock is given in the header file `/usr/include/linux/ext3_fs.h` and was used to create table 1. The data of the unsigned integers is stored on disk in Little Endian format. On linux that means that `__le32` is in fact an `uint32_t` and `__le16` is equal to `uint16_t`.

Groups

Each ext3 file system is devided into groups, with a fixed number of blocks per group, except the last group which contains the remaining blocks. The number of blocks per group is given in the superblock, ie

```
$ ext3grep $IMAGE --superblock | grep 'Blocks per group'
# Blocks per group: 32768
```

Each group uses one block as a bitmap to keep track of which block inside that group is allocated (used); thus, there can be at most 4096 * 8 = 32768 normal blocks per group.

Another block is used as bitmap for the number of allocated inodes. Inodes are data structures of 128 bytes (they can be extended in theory; the real size is given in the superblock once again) that are stored in a table, (4096 / 128 = 32 inodes per block) in each group. Having at most 32768 bits in the bitmap, we can conclude that there will be at most 32768 inodes per group, and thus 32768 / 32 = 1024 blocks in the inode table of each group. The actual size of the inode table is given by the actual number of inodes per group, which is also stored in the superblock.

```
$ ext3grep $IMAGE --superblock | egrep 'Size of inode|inodes per group'
Number of inodes per group: 16288
Size of inode structure: 128
```

The block numbers of both bitmaps and the start of the inode table is given in the "group descriptor table", which resides in the block following the superblock; thus, block 1 or block 2 depending on the size of a block. This group descriptor table exists of a series of consecutive ext3_group_desc structs, also defined in /usr/include/linux/ext3_fs.h, see table 2.

Table 2. A group descriptor

Bytes	type	Description
0 .. 3	__le32	Blocks bitmap block
4 .. 7	__le32	Inodes bitmap block
8 .. 11	__le32	Inodes table block
12 .. 13	__le16	Free blocks count
14 .. 15	__le16	Free inodes count
16 .. 17	__le16	Directories count
18 .. 31		Reserved

Since the size of this struct is padded to a power of 2, 32 bytes, there fit precisely an integral number of descriptors in a block. Therefore the table is contiguous even when spanning multiple blocks. Note that one block of 4096 bytes is already capable of holding 128 group descriptors, each of which can store 32768 blocks— thus only a partition larger than 16 GB will use more than one block for the group descriptor table.

The content of the table is printed by ext3grep if no action or group is specified on the command line. For example,

```
$ ext3grep $IMAGE
No action specified; implying --superblock.
[...]
Number of groups: 75
 Group  0: block bitmap at 598, inodes bitmap at 599, inode table at 600
           4 free blocks, 16278 free inodes, 1 used directory
 Group  1: block bitmap at 33366, inodes bitmap at 33367, inode table at 33368
           30510 free blocks, 16288 free inodes, 0 used directory
[...]
 Group  74: block bitmap at 2424832, inodes bitmap at 2424833, inode table at 2424834
           16481 free blocks, 16288 free inodes, 0 used directory
[...]
```

Inodes

The inodes in the inode table of each group contain meta data for each type of data that the file system can store. This type might be a symbolic link, in which case only the inode is sufficient, it might be a directory, a file, a FIFO, a UNIX socket and so on. In the case of files and directories the real data is stored in file system blocks outside the inode. The first 12 block numbers are stored in the inode, if more blocks are needed, then then the inode points to an indirect block: a block with more block numbers that contain data. Subsequently the inode can store a double indirect block and a triple indirect block. The structure of an inode is given in table 3.

Bytes	type	Description
0 .. 1	__le16	File mode
2 .. 3	__le16	Low 16 bits of Owner uid
4 .. 7	__le32	Size in bytes
8 .. 11	__le32	Access time
12 .. 15	__le32	Creation time
16 .. 19	__le32	Modification time
20 .. 23	__le32	Deletion Time
24 .. 25	__le16	Low 16 bits of Group Id
26 .. 27	__le16	Links count
28 .. 31	__le32	Blocks count
32 .. 35	__le32	File flags
36 .. 39	linux1	OS dependent 1
40 .. 99	__le32[15]	Pointers to blocks
100 .. 103	__le32	File version (for NFS)
104 .. 107	__le32	File ACL
108 .. 111	__le32	Directory ACL
112 .. 115	__le32	Fragment address
116 .. 127	linux2	OS dependent 2

The C-struct for the inode, `struct ext3_inode`, is given in the header file `/usr/include/linux/ext3_fs.h` and was used to create table 3. That same header file also defines a number of constants in the form of macros that should be used to access the data. For example, the struct member that is stored in bytes 40 to 99 is `i_block`, it's size is `EXT3_N_BLOCKS` 32-bit block numbers. `i_block[EXT3_IND_BLOCK]` points to (contains the block number of) an indirect block if one exists. `i_block[EXT3_DIND_BLOCK]` to a double indirect block and `i_block[EXT3_TIND_BLOCK]` to a tripple indirect block. Basically, every constant has it's macro, see the header file for more details. ext3grep uses `i_reserved1` to store the inode number, so that printing an `ext3_inode` struct in gdb shows which inode it really is.

The superblock shows how many inodes exist in total, and how many inodes there are per group. This allows one to calculate the number of groups. Because the inodes are stored in their respective inode tables per group, one first has to determine the group that an inode number belongs to. Because inodes start to count at 1, the formula to convert an inode number to the group it belongs to is:

```
group = (inode_number - 1) / inodes_per_group
```

This gives the correct inode table. To find the index of the inode in this table we subtract the inode number of the first inode in the table from our inode number:

```
index = inode_number - (group * inodes_per_group + 1)
```

Note that this index also determines the corresponding bit in the inodes bitmap.

As such, groups have been made transparent: every inode can be addressed with a number in the contiguous range `[1, number_of_inodes]`, where the number of inodes is given by:

```
$ ext3grep $IMAGE --superblock | grep 'Inodes count'
Inodes count: 1221600
```

In some case you might want to know which block in the file system belongs to the inode table that stores a particular inode. This can be retrieved with the command line option `--inode-to-block`, for example:

```
$ ext3grep $IMAGE --inode-to-block 2
[...]
Inode 2 resides in block 600 at offset 0x80.
```

Inode number 2 (the macro `EXT3_ROOT_INO` in `ext3_fs.h`) is always used for the root of the partition: it's type is a directory. Of all other special inodes we only use `EXT3_JOURNAL_INO` (number 8).

Having the inode number, one can print it's contents with ext3grep, for example:

```
$ ext3grep $IMAGE --inode 2 --print
Number of groups: 75
Loading group metadata... done
[...]

Hex dump of inode 2:
0000 | ed 41 00 00 00 10 00 00 97 6f aa 47 08 6c aa 47 | .A.......o.G.l.G
0010 | 08 6c aa 47 00 00 00 00 00 00 02 00 08 00 00 00 | .l.G............
0020 | 00 00 00 00 02 00 00 00 55 04 00 00 00 00 00 00 | ........U.......
0030 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
0040 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
0050 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
0060 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
0070 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................

Inode is Allocated
Group: 0
Generation Id: 0
uid / gid: 0 / 0
```

```
size: 4096
num of links: 2
sectors: 8 (--> 0 indirect blocks).

Inode Times:
Accessed:       1202352023 = Thu Feb  7 03:40:23 2008
File Modified:  1202351112 = Thu Feb  7 03:25:12 2008
Inode Modified: 1202351112 = Thu Feb  7 03:25:12 2008
Deletion time:  0

Direct Blocks: 1109
[...]
Inode 2 is directory "".
Directory block 1109:
          .-- File type in dir_entry (r=regular file, d=directory, l=symlink)
          |          .-- D: Deleted ; R: Reallocated
Indx Next |  Inode   | Deletion time                        Mode        File name
==========+==========+----------------data-from-inode------+-----------+=========
   0    1 d       2                                       drwxr-xr-x  .
   1  end d       2                                       drwxr-xr-x  ..
   2    3 d      11  D 1202351093 Thu Feb  7 03:24:53 2008  drwxr-xr-x  lost+found
   3  end d  195457  D 1202352103 Thu Feb  7 03:41:43 2008  drwxr-xr-x  carlo
```

As you see, ext3grep first dumps the hexadecimal content of the inode table; then interprets it and prints the struct members, ending with the line `Direct Blocks: 1109`. It then detects that this block is a directory (which can also be seen in the mode field of the inode) and therefore continuous with listing this block as directory.

Regular Files

If an inode represents a regular file, then the blocks it refers to simply contain the data of the file. If the size of a file is not an integral number of times the block size, than the excess bytes in the last block will be zeroed out (at least, on linux).

Symbolic links

The value of a symbolic link is a string: the pathname to it's target. The length of the string is given in `i_size`. If `i_blocks` is zero, then `i_block` does *not* contain block numbers, but is used to store the string directly. However, if the name of the target is longer than fits in `i_block`, then `i_blocks` will be non-zero and `i_block[0]` will point to a block containing the target name.

Directories

If an inode represents a directory then its blocks are (singly) linked lists of `ext3_dir_entry_2` data structures. Each block is self-contained: no dir entry points outside the block. The first block will always start with the dir entries for "." and "..".

Table 4. A Directory Entry

Bytes	type	Description
0 .. 3	__le32	Inode number
4 .. 5	__le16	Directory entry length
6	__u8	Name length
7	__u8	File Type
8	char[]	File-, symlink- or directory name

Using the options `--ls --inode $N`, ext3grep lists the contents of each directory block of inode N. For example, to list the root directory of a partition:

```
$ ext3grep $IMAGE --ls --inode 2
Number of groups: 75
Loading group metadata... done
Minimum / maximum journal block: 1115 / 35026
Loading journal descriptors... done
Journal transaction 4381435 wraps around, some data blocks might have been lost of this transaction.
Number of descriptors in journal: 30258; min / max sequence numbers: 4379495 / 4382264
Inode is Allocated
Loading md5.ext3grep.stage2... done
The first block of the directory is 1109.
Inode 2 is directory "".
Directory block 1109:
          .-- File type in dir_entry (r=regular file, d=directory, l=symlink)
          |          .-- D: Deleted ; R: Reallocated
Indx Next |  Inode   | Deletion time                        Mode        File name
==========+==========+----------------data-from-inode------+-----------+=========
   0    1 d       2                                       drwxr-xr-x  .
   1  end d       2                                       drwxr-xr-x  ..
   2    3 d      11  D 1202351093 Thu Feb  7 03:24:53 2008  drwxr-xr-x  lost+found
   3  end d  195457  D 1202352103 Thu Feb  7 03:41:43 2008  drwxr-xr-x  carlo
```

Subsequently, one could use `ext3grep --ls --inode 195457` to list directory carlo, and so on.

Note that ext3grep prints all directory entries, deleted not. There are two ways that one can see that a directory is deleted: firstly, it's inode will have a non-zero Deletion Time, secondly the dir entry might be taken out of the linked list by skipping it; the "Directory Entry Length", bytes 4 and 5 of each directory entry, basically 'point' to the next entry, or to the byte directly following the block if there are no other dir entries. In the listing of ext2grep the address of the dir entries has has been replaced by an artificial index (in the first column) and the "Directory Entry Length" is replaced with the column called Next, which either points to the next entry or contains end when there are no other dir entries. In the above example, 0 is the first entry, 1 is the next and last entry. The entries with index 2 and 3 are skipped. However, it is still visible that entry 2 used to point to entry 3. In fact, entries 2 and 3 are deleted at the same time by changing the "Directory Entry Length" of entry 1 such that it did not 'point' to entry 2 anymore, but to the end of the block.

Because ext3grep prints also deleted entries, it is very well possible that the SAME entry occurs multiple times. In particular, if a file is moved, a duplicate remains that will still be visible. Ie,

```
$ ext3grep $IMAGE --ls --inode 195457 | grep '\.viminfo$'
   7    8 r  201434  D 1202351096 Thu Feb  7 03:24:56 2008  rrw-r--r--  .viminfo
  18   19 r  195995  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195995  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195995  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195995  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195995  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195995  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195995  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195995  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195995  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195995  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195995  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195995  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195995  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  197221  D 1202351110 Thu Feb  7 03:25:10 2008  rrw-r--r--  .viminfo
  18   19 r  197221  D 1202351110 Thu Feb  7 03:25:10 2008  rrw-r--r--  .viminfo
  18   19 r  197221  D 1202351110 Thu Feb  7 03:25:10 2008  rrw-r--r--  .viminfo
  18   19 r  197221  D 1202351110 Thu Feb  7 03:25:10 2008  rrw-r--r--  .viminfo
  17   19 r  197221  D 1202351110 Thu Feb  7 03:25:10 2008  rrw-r--r--  .viminfo
  17   19 r  197221  D 1202351110 Thu Feb  7 03:25:10 2008  rrw-r--r--  .viminfo
  17   19 r  197221  D 1202351110 Thu Feb  7 03:25:10 2008  rrw-r--r--  .viminfo
  18   19 r  195993  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195981  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-r--r--  .viminfo
  18   19 r  195993  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195981  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-r--r--  .viminfo
  18   19 r  195993  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195987  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195993  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195993  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195993  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195993  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195993  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195993  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195993  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195995  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195995  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195994  D 1202351111 Thu Feb  7 03:25:11 2008  rrw-r--r--  .viminfo
  18   19 r  195995  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  18   19 r  195993  D 1202351097 Thu Feb  7 03:24:57 2008  rrw-------  .viminfo
  17   19 r  197221  D 1202351110 Thu Feb  7 03:25:10 2008  rrw-r--r--  .viminfo
```

In order to understand this, the following remarks.

Firstly, these duplicated entries come mostly from duplicated directory blocks, which is already apparent from the index number of the entries: if they were all from the same block then all index numbers would have been different. Of course, without piping the output to grep it would be clear that each entry belongs to a different directory block, but that output is too much to show here.

Secondly, you have to realize that only the inode number, the file type in the third column and the file name is data from the dir entry itself. The Deletion time, and the Mode column are extracted from the current data in the corresponding inode. However, that inode could have been reused long ago by another file and the data it contains would not be related anymore to this dir entry. This is clearly the case in the above example because it is certain that all those .viminfo files weren't deleted on the same day! In a *few* case it can be detected that an inode has been reallocated (reused): if it is still in use (that can't be by this *deleted* dir entry), or when the file type in the inode differs from the file type in the dir entry. In those cases the fifth column shows an 'R' instead of a 'D', and the content of the inode is not shown. However, because such entries show little information of use, they are normally suppressed. If you want to see entries with known reallocated inodes, you have to add the command line option `--reallocated`. Moreover, sometimes the inode number in the dir entry itself is zeroed. Such entries are obviously useless as well and therefore also suppressed. In order to show them use the command line option `--zeroed-inodes`.

It is possible to apply filters to the output of `--ls`. An overview of the available filters is given in the output of the `--help` option:

```
$ ext3grep $IMAGE --help
[...]
Filters:
  --group grp            Only process group 'grp'.
  --directory            Only process directory inodes.
  --after dtime          Only entries deleted on or after 'dtime'.
  --before dtime         Only entries deleted before 'dtime'.
  --deleted              Only show/process deleted entries.
  --allocated            Only show/process allocated inodes/blocks.
  --unallocated          Only show/process unallocated inodes/blocks.
  --reallocated          Do not suppress entries with reallocated inodes.
                         Inodes are considered 'reallocated' if the entry
                         is deleted but the inode is allocated, but also when
                         the file type in the dir entry and the inode are
                         different.
  --zeroed-inodes        Do not suppress entries with zeroed inodes. Linked
                         entries are always shown, regardless of this option.
  --depth depth          Process directories recursively up till a depth
                         of 'depth'.
[...]
```

In order to easily determine sensible values for `--after` and `--before` the action `--histogram=dtime` was added. This command line option causes ext3grep to print a histogram of time versus number of deleted inodes. If you delete a large number of files at once, for example with `rm -rf`, then it should be easy to determine a time window within which the deletion took place. For example, here I zoomed in on my personal disaster where I deleted a little over fifty thousand files from my home directory:

```
$ ext3grep $IMAGE --histogram=dtime --after=1202351086 --before=1202351129
Only show/process deleted entries if they are deleted on or after Thu Feb  7 03:24:46 2008 and before Thu Feb  7 03:25:29 2008.

Number of groups: 75
Minimum / maximum journal block: 1115 / 35026
Loading journal descriptors... done
Journal transaction 4381435 wraps around, some data blocks might have been lost of this transaction.
Number of descriptors in journal: 30258; min / max sequence numbers: 4379495 / 4382264

Only show/process deleted entries if they are deleted on or after 1202351086 and before 1202351129.
Only showing deleted entries.
Thu Feb  7 03:24:46 2008  1202351086        0
Thu Feb  7 03:24:47 2008  1202351087        1
Thu Feb  7 03:24:48 2008  1202351088        0
Thu Feb  7 03:24:49 2008  1202351089        0
Thu Feb  7 03:24:50 2008  1202351090        0
Thu Feb  7 03:24:51 2008  1202351091        0
Thu Feb  7 03:24:52 2008  1202351092        0
Thu Feb  7 03:24:53 2008  1202351093      705 ==============
Thu Feb  7 03:24:54 2008  1202351094     1698 ==================================
Thu Feb  7 03:24:55 2008  1202351095     2320 ===============================================
Thu Feb  7 03:24:56 2008  1202351096     3652 ============================================================================
Thu Feb  7 03:24:57 2008  1202351097     3332 ===================================================================
Thu Feb  7 03:24:58 2008  1202351098     2014 ========================================
Thu Feb  7 03:24:59 2008  1202351099     1160 ========================
Thu Feb  7 03:25:00 2008  1202351100     4188 ========================================================================================
Thu Feb  7 03:25:01 2008  1202351101     2480 ==================================================
Thu Feb  7 03:25:02 2008  1202351102     1945 =======================================
Thu Feb  7 03:25:03 2008  1202351103     1471 ==============================
Thu Feb  7 03:25:04 2008  1202351104     2724 =======================================================
Thu Feb  7 03:25:05 2008  1202351105     3090 ================================================================
Thu Feb  7 03:25:06 2008  1202351106     3360 ====================================================================
Thu Feb  7 03:25:07 2008  1202351107     4902 ================================================================================================================
Thu Feb  7 03:25:08 2008  1202351108      698 ==============
Thu Feb  7 03:25:09 2008  1202351109     1612 =================================
Thu Feb  7 03:25:10 2008  1202351110     4547 ============================================================================================
Thu Feb  7 03:25:11 2008  1202351111     2651 =====================================================
Thu Feb  7 03:25:12 2008  1202351112     1513 ==============================
Thu Feb  7 03:25:13 2008  1202351113        0
Thu Feb  7 03:25:14 2008  1202351114        0
Thu Feb  7 03:25:15 2008  1202351115        0
Thu Feb  7 03:25:16 2008  1202351116        0
Thu Feb  7 03:25:17 2008  1202351117        1
Thu Feb  7 03:25:18 2008  1202351118        0
Thu Feb  7 03:25:19 2008  1202351119        0
Thu Feb  7 03:25:20 2008  1202351120        0
Thu Feb  7 03:25:21 2008  1202351121        0
Thu Feb  7 03:25:22 2008  1202351122        0
Thu Feb  7 03:25:23 2008  1202351123        0
Thu Feb  7 03:25:24 2008  1202351124        0
Thu Feb  7 03:25:25 2008  1202351125        0
Thu Feb  7 03:25:26 2008  1202351126        0
Thu Feb  7 03:25:27 2008  1202351127        0
Thu Feb  7 03:25:28 2008  1202351128        0
Thu Feb  7 03:25:29 2008  1202351129
Totals:
1202351086 - 1202351128    50064
```

It is important to set a good value for `--after` before recovering all files, or way too many files will be "recovered".

The Journal

The journal is a file existing of a fixed number of blocks. It's inode is `EXT3_JOURNAL_INO`, which is usually 8. The actual inode can also be found in the superblock:

```
$ ext3grep $IMAGE --superblock | grep 'Inode number of journal file'
Inode number of journal file: 8
```

```
$ ext3grep $IMAGE --print --inode 8
Number of groups: 75
Loading group metadata... done
Minimum / maximum journal block: 1115 / 35026
Loading journal descriptors... done
Journal transaction 4381435 wraps around, some data blocks might have been lost of this transaction.
Number of descriptors in journal: 30258; min / max sequence numbers: 4379495 / 4382264

Hex dump of inode 8:
0000 | 80 81 00 00 00 00 00 08 00 00 00 00 62 07 57 46 | ............b.WF
0010 | 62 07 57 46 00 00 00 00 00 00 01 00 10 01 04 00 | b.WF............
0020 | 00 00 00 00 08 00 00 00 5b 04 00 00 5c 04 00 00 | ........[...\...
0030 | 5d 04 00 00 5e 04 00 00 5f 04 00 00 60 04 00 00 | ]...^..._...`...
0040 | 61 04 00 00 62 04 00 00 63 04 00 00 64 04 00 00 | a...b...c...d...
0050 | 65 04 00 00 66 04 00 00 67 04 00 00 68 08 00 00 | e...f...g...h...
0060 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
0070 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................

Inode is Allocated
Group: 0
Generation Id: 0
uid / gid: 0 / 0
mode: rrw-------
size: 134217728
num of links: 1
sectors: 262416 (--> 34 indirect blocks).

Inode Times:
Accessed:       0
File Modified:  1180108642 = Fri May 25 17:57:22 2007
Inode Modified: 1180108642 = Fri May 25 17:57:22 2007
Deletion time:  0

Direct Blocks: 1115 1116 1117 1118 1119 1120 1121 1122 1123 1124 1125 1126
Indirect Block: 1127
Double Indirect Block: 2152
```

where you can see that the size of my journal is 134217728 bytes, or 32768 blocks. The first 12 blocks are listed directly in the inode: blocks 1115 - 1126. Then an indirect block is placed in 1127. This indirect block can contain 1024 block numbers each of which follow the indirect block directly (1128 - 2151). Then the inode refers to a double indirect block containing 31 block numbers of additional indirect blocks. The total number of (double/tripple) indirect blocks is calculated to be 34 (using the fact that a sector is 512 byte). Therefore, if everything would be stored contiguously, the last block of the journal would be 1115 + 32768 + 34 - 1 = 33916. However, the journal didn't fit entirely in group 0, so the last blocks are in group 1 and the header of group 1 (most notably it's inode table) is inserted somewhere between the journal blocks, causing the last block to be 35025. On top of this, there could be bad blocks anywhere in between as well. Therefore, the correct way to approach the journal is in terms of 'journal block numbers'.

The first block of the journal file (block 1115 in the above example) contains the 'journal superblock'. It's structure is defined in `/usr/include/linux/jbd.h` as `journal_superblock_t`. It can be printed with:

```
$ ext3grep $IMAGE --journal --superblock
Journal Super Block:

Signature: 0x3225106840
Block type: Superblock version 2
Sequence Number: 0
Journal block size: 4096
Number of journal blocks: 32768
Journal block where the journal actually starts: 1
Sequence number of first transaction: 4382265
Journal block of first transaction: 0
Error number: 0
Compatible Features: 0
Incompatible features: 1
Read only compatible features: 0
Journal UUID: 0xe3 0x88 0xd9 0x09 0x94 0xca 0x43 0x95 0x9b 0x53 0xac 0x2c 0xd8 0xe0 0x3d 0x25
Number of file systems using journal: 1
Location of superblock copy: 0
Max journal blocks per transaction: 0
Max file system blocks per transaction: 0
IDs of all file systems using the journal:
1. 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

Minimum / maximum journal block: 1115 / 35026
Loading journal descriptors... done
Journal transaction 4381435 wraps around, some data blocks might have been lost of this transaction.
Number of descriptors in journal: 30258; min / max sequence numbers: 4379495 / 4382264
```

Here you can see that the journal actually starts in Journal Block Number 1, and the last block is Journal Block Number 32768. These are thus not the same as the file system block numbers. One can find the real block number with, for example,

```
$ ext3grep $IMAGE --journal --journal-block 1
[...]
Group: 0
Block 1116 belongs to the journal.
[...]
```

which reveals that Journal Block Number 1 is file system block 1116.

end of the journal is reached, writing continuous at the start, wrapping around. However, if a file system is cleanly unmounted then the next mount writing always starts at the beginning (I think).

A single transaction exist of a one or more "Descriptors". The last descriptor of a transaction is a "Commit Block", signaling that the transaction has been closed succesfully and the data in the previous descriptors of that transaction has been written to disk. There are two other types of descriptors: revoke blocks and blocks containing "tags". A revoke block is filled with block numbers that should be (or are) unallocated by this transaction. A tag is a structure that assigns subsequent journal blocks (not file system blocks!) to file system blocks: the following journal blocks contain the data that should (have been) written to the given file system block.

That makes "tags" in particular interesting for us: they contain copies of data that was written to disk in the past, including old inodes.

Manual recovery example

In the following example we will manually recover a small file. Only partial output is given in order to save space and to make the example more readable.

Using `ext3grep $IMAGE --ls --inode` we find the name of the file that we want to recover:

```
$ ext3grep $IMAGE --ls --inode 2 | grep carlo
   3  end d  195457  D 1202352103 Thu Feb  7 03:41:43 2008  drwxr-xr-x  carlo

$ ext3grep $IMAGE --ls --inode 195457 | grep ' bin$' | head -n 1
  34   35 d  309540  D 1202352104 Thu Feb  7 03:41:44 2008  drwxr-xr-x  bin

$ ext3grep $IMAGE --ls --inode 309540 | grep start_azureus
   9   10 r  309631  D 1202351093 Thu Feb  7 03:24:53 2008  rrwxr-xr-x  start_azureus
```

Obviously, inode 309631 is erased and we have no block numbers for this file:

```
$ ext3grep $IMAGE --print --inode 309631
[...]
Inode is Unallocated
Group: 19
Generation Id: 2771183319
uid / gid: 1000 / 1000
mode: rrwxr-xr-x
size: 0
num of links: 0
sectors: 0 (--> 0 indirect blocks).

Inode Times:
Accessed:       1202350961 = Thu Feb  7 03:22:41 2008
File Modified:  1202351093 = Thu Feb  7 03:24:53 2008
Inode Modified: 1202351093 = Thu Feb  7 03:24:53 2008
Deletion time:  1202351093 = Thu Feb  7 03:24:53 2008

Direct Blocks:
```

Therefore, we will try to look for an older copy of it in the journal. First, we find the file system block that contains this inode:

```
$ ext3grep $IMAGE --inode-to-block 309631 | grep resides
Inode 309631 resides in block 622598 at offset 0xf00.
```

Then we find all journal descriptors referencing block 622598:

```
$ ext3grep $IMAGE --journal --block 622598
[...]
Journal descriptors referencing block 622598:
4381294 26582
4381311 28693
4381313 28809
4381314 28814
4381321 29308
4381348 30676
4381349 30986
4381350 31299
4381374 32718
4381707 1465
4381709 2132
4381755 2945
4381961 4606
4382098 6073
4382137 6672
4382138 7536
4382139 7984
4382140 8931
```

This means that the transaction with sequence number 4381294 has a copy of block 622598 in block 26582, and so on. The largest sequence number, at the bottom, should be the last data written to disk and thus block 8931 should be the same as the current block 622598. In order to find the last non-deleted copy, one should start at the bottom and work upwards.

If you try to print such a block, ext3grep recognizes that it's a block from an inode table and will print the contents of all 32 inodes in it. We only wish to see inode 309631 however; so we use a smart grep:

```
$ ext3grep $IMAGE --print --block 8931 | grep -A15 'Inode 309631'
--------------Inode 309631----------------------
Generation Id: 2771183319
uid / gid: 1000 / 1000
mode: rrwxr-xr-x
```

```
num of links: 0
sectors: 0 (--> 0 indirect blocks).

Inode Times:
Accessed:       1202350961 = Thu Feb  7 03:22:41 2008
File Modified:  1202351093 = Thu Feb  7 03:24:53 2008
Inode Modified: 1202351093 = Thu Feb  7 03:24:53 2008
Deletion time:  1202351093 = Thu Feb  7 03:24:53 2008

Direct Blocks:
```

This is indeed the same as we saw in block 622598. Next we look at smaller sequence numbers until we find one with a 0 Deletion time. The first one that we find (bottom up) is block 6703:

```
$ ext3grep $IMAGE --print --block 6073 | grep -A15 'Inode 309631'
--------------Inode 309631----------------------
Generation Id: 2771183319
uid / gid: 1000 / 1000
mode: rrwxr-xr-x
size: 40
num of links: 1
sectors: 8 (--> 0 indirect blocks).

Inode Times:
Accessed:       1202350961 = Thu Feb  7 03:22:41 2008
File Modified:  1189688692 = Thu Sep 13 15:04:52 2007
Inode Modified: 1189688692 = Thu Sep 13 15:04:52 2007
Deletion time:  0

Direct Blocks: 645627
```

The above is automated and can be done much faster with the command line option `--show-journal-inodes`. This option will find the block that the inode belongs to, then finds all copies of that block in the journal, and subsequently prints only the requested inode from each of these block (each of which contains 32 inodes, as you know), eliminating duplicates:

```
$ ext3grep $IMAGE --show-journal-inodes 309631
Number of groups: 75
Minimum / maximum journal block: 1115 / 35026
Loading journal descriptors... done
Journal transaction 4381435 wraps around, some data blocks might have been lost of this transaction.
Number of descriptors in journal: 30258; min / max sequence numbers: 4379495 / 4382264
Copies of inode 309631 found in the journal:

--------------Inode 309631----------------------
Generation Id: 2771183319
uid / gid: 1000 / 1000
mode: rrwxr-xr-x
size: 0
num of links: 0
sectors: 0 (--> 0 indirect blocks).

Inode Times:
Accessed:       1202350961 = Thu Feb  7 03:22:41 2008
File Modified:  1202351093 = Thu Feb  7 03:24:53 2008
Inode Modified: 1202351093 = Thu Feb  7 03:24:53 2008
Deletion time:  1202351093 = Thu Feb  7 03:24:53 2008

Direct Blocks:

--------------Inode 309631----------------------
Generation Id: 2771183319
uid / gid: 1000 / 1000
mode: rrwxr-xr-x
size: 40
num of links: 1
sectors: 8 (--> 0 indirect blocks).

Inode Times:
Accessed:       1202350961 = Thu Feb  7 03:22:41 2008
File Modified:  1189688692 = Thu Sep 13 15:04:52 2007
Inode Modified: 1189688692 = Thu Sep 13 15:04:52 2007
Deletion time:  0

Direct Blocks: 645627
```

The file is indeed small: only one block. We copy this block with dd as shown before:

```
$ dd if=$IMAGE bs=4096 count=1 skip=645627 of=block.645627
1+0 records in
1+0 records out
4096 bytes (4.1 kB) copied, 0.0166104 seconds, 247 kB/s
```

and then edit the file to delete the trailing zeroes, or copy the first 40 bytes (the given size of the file):

```
$ dd if=block.645627 bs=1 count=40 of=start_azureus
40+0 records in
40+0 records out
40 bytes (40 B) copied, 0.000105397 seconds, 380 kB/s

$ cat start_azureus
cd /usr/src/azureus/azureus
./azureus &
```

Note that it is possible to see all descriptors of a given transaction. The transaction that we used to recover this file was 4382098. The complete transaction can be seen with:

```
$ ext3grep $IMAGE --journal-transaction 4382098
[...]
Prev / Current / Next sequences numbers: 4382097 4382098 4382099
Transaction was NOT COMMITTED!
TAG: 6074=851971 6073=622598 6072=393218 6071=393395 6070=393231 6069=393409 6068=393240 6067=393371 6066=622596
REVOKE: 506451
TAG: 6056=393217 6057=1 6058=393273 6059=393232 6060=403879 6061=393216 6062=491520 6063=506302 6064=0 6065=393219
```

Here you see, for example, the TAG 6072=393218, meaning that block 6072 contains a (old) copy of block 393218. I don't know why it says that the transaction wasn't committed (that seems very unlikely). Probably, the commit block was overwritten and this old journal transaction simply isn't complete anymore.

Recovering files

Of course, it would be annoying to recover larger files, existing of many blocks this way; let alone manually recovering thousands of files! Therefore all of the above can be automated. However, if you recover 50,000 files then there is virtually no way to even check if it worked afterwards: especially when MORE files were recovered than you really wanted; it will be hard to find back all the junk. You really should take care to recover files as accurate as possible.

No such care seems necessary to recover a single file, you can just pass it's path to ext3grep:

```
$ ext3grep $IMAGE --restore-file carlo/bin/start_kvm
[...]
Restoring carlo/bin/start_kvm

$ cat carlo/bin/start_kvm
#! /bin/sh
cd /usr/src/qgt/src
./host-linux 192.168.2.4 &
cd /opt/kvm/winXPpro
sudo modprobe kvm_intel
sudo kvm -m 384 -hda vdisk6GB.img -cdrom /dev/cdrom -localtime -std-vga -net nic,vlan=0,model=rtl8139 -net tap,vlan=0
#-snapshot # -daemonize
killall -9 host-linux
```

Note that this created the directory carlo/bin in the current directory, in order to be able to restore this file. Also note that if carlo/bin/start_kvm already existed in the current directory then it was *not* overwritten!

In order for this to work you will first have to pass stage1 and stage2 of the disk analysis that ext3grep will perform (see below).

It is possible to dump all file names that ext3grep can find, using the command line option --dump-names:

```
$ ext3grep $IMAGE --dump-names
carlo
carlo/.Trash
carlo/.Xauthority
carlo/.Xauthority-c
carlo/.Xauthority-l
carlo/.Xauthority-n
carlo/.alsaplayer
carlo/.alsaplayer/alsaplayer.m3u
carlo/.alsaplayer/config
[...]
carlo/www/xcw/.svn/tmp/wcprops
carlo/www/xcw/index.html
carlo/www/xmlwrapp-0.5.0.tar.gz
lost+found
lost+found/1st level admin borders (states_provinces)
lost+found/1st level admin names (states_provinces)
lost+found/2002 - cloud cover (0-10%)
[...]
```

The files that will end up in lost+found are files for which no directory could be found (but that still had an inode copy in the journal). Most likely those are files that were deleted long ago and can be disgarded anyway.

Once you are satisfied with the output of --dump-names, you can replace --dump-names with --restore-all, which in effect will cause --restore-file to be called on every file name printed by --dump-names. As mentioned before, it is highly advisable to use a proper --after command line option in order to avoid that ext3grep tries to recover files that are simply too old. Note that at this moment the output of --dump-names is unfiltered, and --restore-file (--restore-all) *only* honors the --after command line option.

For example,

```
$ time ext3grep $IMAGE --restore-all --after=1202351117
Only show/process deleted entries if they are deleted on or after Thu Feb  7 03:25:17 2008.
[...]
Loading md5.ext3grep.stage2... done
Not undeleting "carlo/.Trash" because it was deleted before 1202351117 (32767)
Not undeleting "carlo/.Xauthority" because it was deleted before 1202351117 (32767)
[...]
Cannot find an undeleted inode for file "carlo/.azureus/logs/save/1176594823051_alerts_1.log".
[...]
Restoring carlo/bin/startx
[...]
real    0m3.079s
```

```
sys     0m1.744s
```

where carlo/bin/startx is the only file recovered. It was the *last* file that was deleted, and I set the --after value to one second before it was. Note that it's logical that it was the last file, since I started X by executing this script; hence, it was "in use" until I rebooted.

Considering that it checked over 50,000 files from a in total 10 GB large partition, the 3.1 seconds is extremely fast; this is caused by several factors: 1) The first time ext3grep is run, it does a full analysis of the partition and writes the results to a cache file (in two steps, first stage1 and then stage2). These stages only need to be done once. 2) Since only one file had to be recovered, there wasn't much disk access (besides, I have 4 GB of RAM -- so everything needed was already cached). 3) I have a very fast cpu. It was using 100% cpu during those 3.1 seconds though. Trying to restore many files mainly hangs on disk access, but is relatively still pretty fast (you can just sit and wait for it).

Stage 1

The stage 1 cache file is written to DEVICE.ext3grep.stage1, where DEVICE is replaced with the device name (ie, if $IMAGE is /dev/hda2, then DEVICE is hda2). There is little that can go wrong with stage 1: it just scans the whole disk and finds all blocks that seem to contain a directory.

The format of the stage1 cache file is:

```
$ cat md5.ext3grep.stage1
# Stage 1 data for md5.
# Inodes and directory start blocks that use it for dir entry '.'.
# INODE : BLOCK [BLOCK ...]
2 : 1109 6592 9312
11 : 1110
195457 : 415744
195468 : 2916 4732 17783 403469
195469 : 403470
[...]
929633 : 1885254
929659 : 1885280
# Extended directory blocks.
1178
1179
1182
[...]
1884516
```

In the first part, the first column are inodes, followed by a space, followed by a colon, followed by a space seperated list of block numbers that use that inode for a dir entry with name ".". Obviously, there can only be one directory that uses this inode, so ext3grep has to determine which of those block numbers is the last one that was the real one. The second part lists all block numbers that contain extended directory blocks, that is, directory blocks that are not the first block and do not contain the dir_entry with name ".". It is unknown which directory those belong to without having the original inode. In stage 2 ext3grep will attempt to find out to which directory they belong.

Stage 2

This stage, executed by the function init_directories(), contains most heuristic code. First it determines which blocks are the real directory start blocks, and then assigns each extended directory block to one of those directories (see also TODO, below). As a result it is possible to assign a path name to each (directory) inode, and assign a list of directory blocks to them. Finally, this result is written to a cache file (DEVICE.ext3grep.stage2). In case something goes very wrong here, you might be able to fix it by editing this file (removing incorrect-, or adding correct block numbers), however, do not remove or add comments: ext3grep will get confused if you change the file too much.

Superfluous hardlinks

Because inodes are reused, it happens often that an old directory entry (of a deleted file, or in a deleted directory, or in an old directory block that is not used anymore) refers to an inode that is now used by something else. If that something else is of the same type (both regular files) then there is no way to distinguish it from a hardlink: two files using the same inode. As a result, a recovery results in a lot of WRONG hardlinks.

In order to make it easier to clean these up, ext3grep provides the command line option --show-hardlinks.

```
$ ext3grep $IMAGE --show-hardlinks
[...]
Inode 309562:
  carlo/bin/pc++ (309540)
  carlo/bin/pcc (309540)
  carlo/bin/pcc.unlock (309540)
Inode 702474:
  carlo/projects/libcwd/libcwd/.svn/entries (700387)
  carlo/projects/libcwd/libcwd/testsuite/tst_flush.o (700609)
[...]
```

Here, the hardlinks for inode 309562 are correct. The hardlink for inode 702474 is wrong, and one of the files should be deleted. After you manually determined which file is wrong and deleted it; it will not show up again when you rerun this command: Only those hardlinks are reported that still exist in the output directory: you can only use --show-hardlinks *after* running --restore-all, or it will not result in any output since no output file exists.

TODO

The program has been written *while* I was learning how ext3 works. It's earliest functionality is therefore not depending on things that I wrote later. An advantage is that those functionalities are faster and will still work if the later code is broken; they are also more down-to-earth, so you can use them to check what is really going on without depending on the more complex (and heuristic) code that was added later. However, there are also disadvantages: The filtering code that I wrote for --ls is not being used by the later written code that handles --dump-names and

code written later. It is not easy to change that because that code uses the results of stage 2. I think that a better algorithm to find which blocks are the correct ones for the last copy of a directory would be the same as how I finally recovered files: by finding the last non-deleted inode of that directory in the journal. This is not how it currently works though.

Command line options

All command line options are listed by providing `--help` on the command line:

```
$ ext3grep $IMAGE --help
Usage: /home/carlo/c++/ext3/main/ext3grep [options] [--] device-file
Options:
  --version, -[vV]       Print version and exit successfully.
  --help,                Print this help and exit successfully.
  --superblock           Print contents of superblock in addition to the rest.
                         If no action is specified then this option is implied.
  --print                Print content of block or inode, if any.
  --ls                   Print directories with only one line per entry.
                         This option is often needed to turn on filtering.
  --accept filen         Accept 'filen' as a legal filename.
                         Can be used multiple times.
  --journal              Show content of journal.
  --show-path-inodes     Show the inode of each directory component in paths.
Filters:
  --group grp            Only process group 'grp'.
  --directory            Only process directory inodes.
  --after dtime          Only entries deleted on or after 'dtime'.
  --before dtime         Only entries deleted before 'dtime'.
  --deleted              Only show/process deleted entries.
  --allocated            Only show/process allocated inodes/blocks.
  --unallocated          Only show/process unallocated inodes/blocks.
  --reallocated          Do not suppress entries with reallocated inodes.
                         Inodes are considered 'reallocated' if the entry
                         is deleted but the inode is allocated, but also when
                         the file type in the dir entry and the inode are
                         different.
  --zeroed-inodes        Do not suppress entries with zeroed inodes. Linked
                         entries are always shown, regardless of this option.
  --depth depth          Process directories recursively up till a depth
                         of 'depth'.
Actions:
  --inode-to-block ino   Print the block that contains inode 'ino'.
  --inode ino            Show info on inode 'ino'.
                         If --ls is used and the inode is a directory, then
                         the filters apply to the entries of the directory.
                         If you do not use --ls then --print is implied.
  --block blk            Show info on block 'blk'.
                         If --ls is used and the block is the first block
                         of a directory, then the filters apply to entries
                         of the directory.
                         If you do not use --ls then --print is implied.
  --histogram=[atime|ctime|mtime|dtime|group]
                         Generate a histogram based on the given specs.
                         Using atime, ctime or mtime will change the
                         meaning of --after and --before to those times.
  --journal-block jblk   Show info on journal block 'jblk'.
  --journal-transaction seq
                         Show info on transaction with sequence number 'seq'.
  --dump-names           Write the path of files to stdout.
                         This implies --ls but suppresses it's output.
  --search-start str     Find blocks that start with the fixed string 'str'.
  --search str           Find blocks that contain the fixed string 'str'.
  --search-inode blk     Find inodes that refer to block 'blk'.
  --search-zeroed-inodes Return allocated inode table entries that are zeroed.
  --inode-dirblock-table dir
                         Print a table for directory path 'dir' of directory
                         block numbers found and the inodes used for each file.
  --show-journal-inodes ino
                         Show copies of inode 'ino' still in the journal.
  --restore-file 'path'  Will restore file 'path'. 'path' is relative to root
                         of the partition and does not start with a '/' (it
                         must be one of the paths returned by --dump-names).
                         The restored directory, file or symbolic link is
                         created in the current directory as ./'path'.
  --restore-all          As --restore-file but attempts to restore everything.
                         The use of --after is highly recommended because the
                         attempt to restore very old files will only result in
                         them being hard linked to a more recently deleted file
                         and as such polute the output.
  --show-hardlinks       Show all inodes that are shared by two or more files.
```

New functionality was more or less added top down, so this also gives a historic overview of how the program was written.

Download

There is no download yet. Please email me at «carlo ât alinoe.com» and I'll send you the source code.