Ext2fs Undeletion of Directory Structures mini−HOWTO

Tomas Ericsson

tomase@matematik.su.se

Revision History

Revision v0.1.1	14 November 2000	Revised by: T.E.
Initial release.		

This document is supposed to be as an complementary to the *Ext2fs−Undeletion mini−HOWTO* written by Aaron Crane. I really recommend you to carefully study that one before reading this.

Here I will describe a straight forward way of recovering whole directory structures, instead of file by file, that have been removed by a misplaced *rm −rf*.

Table of Contents

1. Introduction

1.1. Disclaimer

The author is not responsible for damages due actions taken based on this document. You will do everything at your own risk.

1.2. License

1.3. Feedback

Any kind of feedback is welcome. Corrections of errors in this text are of great interest. If someone find things described here useful I really would appreciate to hear that from you.

1.4. New versions of this document

The latest version of this document will be found at http://www.matematik.su.se/~tomase/ext2fs−undeletion/

1.5. Acknowledgements

Thanks to the following persons for correctures etc (in alphabetic order).

- Gabriel Kihlman
- Richard Nyberg
- Mats Oldin
- Tobias Westerblom

1.6. Background

This text has been written while solving my own undeletion problems that I had some time ago. I was about to move some directories from one disk to another. The problem was that the extra disk went corrupted directly after the move for some reason.

So I wanted to get the moved directories back from my original disk. There were totally about 40000 files to recover and I did not feel very much like to search and rename each one of them by hand.

I wanted to get back the whole structure of directories. The same situation would have appeared if I had made an *rm −rf* to those directories.

2. Preconditions

It is really important that you unmount the affected partition immediately before doing anything else with it. If you have copied around some files in this partition after the accident, then the chance for this method to work has lowered a lot.

Also you must have a quite new kernel, because the 2.0.x and below zeroes indirect blocks, which will not make this process to work for files with more than 12 blocks of data.

I will describe one method of recovery and I will not leave out much for error handling. If you suspect that some step described below went wrong, I do not recommend you to go any further.

3. Preparation

Unmount the partition where you got your deleted files at. I will denote that partition with /dev/hdx1.

```
        # umount /dev/hdx1
```

Check the size in blocks of /dev/hdx1.

```
        # fdisk −l /dev/hdx
```

Now you need to have an extra partition of the same size as /dev/hdx1, to be able to act safely. Suppose that you have an empty harddrive located at /dev/hdy.

```
        # fdisk /dev/hdy
```

Create a partition with the same size as /dev/hdx1. Here *size* is the size in blocks (each block is 1024 kB) of /dev/hdx1 taken from above.

> **Note:** I am using *fdisk* version 2.10f and if you have another version the *fdisk* interaction below may not be the same.

```
        fdisk: n      <− Add a new partition.
        fdisk: p      <− Primary partition.
        fdisk:        <− Just press return to chose the default first cylinder.
        fdisk: +sizeK <− Make a partition of the same size as /dev/hdx1.
        fdisk: w      <− Write table to disk and exit.
```

Now copy the content of the original partition to that extra one.

```
        # dd if=/dev/hdx1 of=/dev/hdy1 bs=1k
```

This process may take quite a long time dependent of how big the partition is. You can get the job done faster by increasing the blocksize, *bs*, but then you need to get the division of the partition by *bs* to have a remainder of zero.

From now on we will only work with that copy of the source partition to be able to step back if something goes wrong.

4. Finding inodes for deleted directories

We will try to find out the inode numbers of the deleted directories.

```
        # debugfs /dev/hdy1
```

Walk to that place in the structure where the directories were located before the deletion. You can use *ls* and *cd* inside *debugfs*.

```
        debugfs: ls -l
```

Example of output from the above command.

```
        179289  20600      0      0       0 17-Feb-100 18:26 file-1
        918209  40700    500    500    4096 16-Jan-100 15:18 file-2
        160321  41777      0      0    4096  3-Jun-100 06:13 file-3
        177275  60660      0      6       0  5-May-98  22:32 file-4
        229380 100600    500    500   89891 19-Dec-99  15:40 file-5
        213379 120777      0      0      17 16-Jan-100 14:24 file-6
```

Description of the fields.

1. Inode number.
2. First two (or one) numbers represents the kind of inode we got:

   2 = Character device

   4 = Directory

   6 = Block device

   10 = Regular file

   12 = Symbolic link

   Last four numbers are the usual Unix rights.

3. Owner in number representation.
4. Group in number representation.
5. Size in bytes.
6. Date (Here we can see the Y2K bug =)).
7. Time.
8. Filename.

Now dump the mother directory to disk. Here *inode* is the corresponding inode number (do not forget the '<' and '>').

```
        debugfs: dump <inode> debugfs-dump
```

Get out of *debugfs*.

```
        debugfs: quit
```

5. Analyse of directory dump

View the dumped inode in a readable format.

```
        # xxd debugfs−dump | less
```

Every entry consist of five fields. For the first two fields the bytes representing the field comes in backward order. That means the first byte is the least significant.

Description of the fields.

1. Four bytes − Inode number.
2. Two bytes − Directory entry length.
3. One byte − Filename length (1−255).
4. One byte − File type (0−7).

   0 = Unknown

   1 = Regular file

   2 = Directory

   3 = Character device

   4 = Block device

   5 = FIFO

   6 = SOCK

   7 = Symbolic link

5. Filename (1−255 characters).

If an entry in the directory is to be deleted, then the second field at the entry before will be increased by the value of the deleted entrys second field.

If a filename is renamed to a shorter one, then the third field will be decreased.

The first entry you will see is the directory itself, represented by a dot.

Suppose that we have the following directory entry.

```
         c1 02 0e 00 40 00 05 01 'u' 't' 'i' 'l' 's'
```

Then the inode would be e02c1 in hexadecimal representation or 918209 in decimal. The next entry would be located after 64 bytes (40 in hex). We see that the filename consist of 5 bytes ("utils") and the file type (01) is a regular file.

Now recalculate the directories inode numbers in decimal representation.

If you do not like to calculate this by hand I have made a small program in C that will do this for you. The program takes as input a directory dump (created by *debugfs* as described in Section 4). The output (at stdout) consist of each entrys inode number and filename.

Before you run the program you need to load the dump into a hexeditor and change the *directory entry length* field at the entry before the one you want to get back. But it is simple. If we name the field before to *x* and the field at the entry you want to get back to *y*. Then change *x* to *x − y*.

The program called *e2dirana* (ext2fs directory analyse) can be found at http://www.matematik.su.se/~tomase/ext2fs−undeletion/

6. Locating deleted inodes

Get a list of all deleted inodes.

```
        # echo lsdel | debugfs /dev/hdy1 > lsdel.out
```

One problem here is that *debugfs* will not give the inode numbers to files that are zero in size (you probably have some in your /etc directory for instance). I will tell how to solve this problem in Section 9 and Section 11.

Load "lsdel.out" into a texteditor. The list of inodes should be sorted in time. Try to remember exactly what time you did that *rm −rf*. Probably it was the the last items you deleted and then they will be located at the bottom of the list, because it is sorted in time. Delete all lines that has nothing of interest. Save as "lsdel.out−selected".

Now we will cut away everything except the inodes.

```
        # cut −b 1−8 lsdel.out-selected | tr −d " " > inodes
```

To be sure, check that the deleted directories found above have their inodes in the list.

```
        # grep ^inode$ inodes
```

Where *inode* is the corresponding inode number.

7. Activating inodes

Now it is time for changing some flags of the deleted inodes.

Copy the following 6 lines into a file named "make−debugfs−input".

```
        #!/bin/sh
        awk '{ print "mi <" $1 ">\n"\
                     "\n\n\n\n\n\n\n"\
                     "0\n"\
                     "1\n"\
                     "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n" }'
```

This will simulate the user input when editing the inodes manually. We will set *deletion time* to 0 and *link count* to 1.

> **Note:** I am using *debugfs* version 1.18 and if you have another version you should check if you need to adjust the number of returns above.

Now change the inodes.

```
        # ./make−debugfs−input < inodes | debugfs −w /dev/hdy1 | tail −c 40
```

If all went well it should end with "Triple Indirect Block [0] debugfs:".

8. Adding directory entries

Start up *debugfs* in read−write mode.

```
        # debugfs −w /dev/hdy1
```

Now you should add the deleted directories to the directory were they were located before deletion.

```
        debugfs: link <inode> directoryname
```

Where *inode* is the inode number and *directoryname* is the directoryname.

After you have added the links you will notice that the directories have popped up in the current directory. You can now list their contents (from *debugfs*).

But the size shown for each directory is zero and that need to be fixed, otherwise they will look empty with *ls* from the shell.

Get out of *debugfs*.

```
        debugfs: quit
```

9. Recalculation

Now it is time to call *e2fsck* to recalculate the sizes and checksums.

> **Note:** I am using *e2fsck* version 1.18 and if you have another version you should check if some parameters or interaction have changed.

If you know that you do *not* have any files with the size of zero that you want to recover you can do the following and skip the rest of this section (Of course you can leave out the *y* parameter, but then you need to answer all questions by hand and it may take a lot of time.).

```
        # e2fsck −f −y /dev/hdy1 > e2fsck.out 2>38;1
```

If you instead want the zero files back you have to answer *no* to all questions about clearing entries and *yes* to the other ones.

Copy the following 7 lines into a file named "e2fsck−wrapper".

```
        #!/usr/bin/expect −f
        set timeout −1
        spawn /sbin/e2fsck −f $argv
        expect {
            "Clear<y>? " { send "n" ; exp_continue }
            "<y>? "      { send "y" ; exp_continue }
        }
```

Run the script.

```
        # ./e2fsck−wrapper /dev/hdy1 > e2fsck.out 2>38;1
```

Examine "e2fsck.out" to see what *e2fsck* thought about your partition.

10. If /lost+found not empty

Some of the directory and files may not pop−up at their right places. Instead they will be located in /lost+found with names after their inode numbers.

In this case the pointer at the ".." directory entry probably have been increased and will point to one of the later entrys in the directory (of some reason I do not know, maybe it is a fs bug).

Examine *pass 3* (where directory connectivity is checked) of "e2fsck.out". There you will find which directories that are affected. Dump those to disk (as described in Section 4).

Run *e2dirana* both with and without the *p* parameter (it changes the pointer at the ".." directory entry). Here *dump* is the dumped directory.

```
        # e2dirana dump > dump1
        # e2dirana −p dump > dump2
```

Compare the outputs.

```
        # diff dump1 dump2
```

If they are not the same there are missing files in that directory. Then move those files from /lost+found to their right place. Here *dest* is a symbolic link to the destination directory. Put the output in a script and run it if you agree.

```
        # diff dump1 dump2 |\
          tail −n $[`diff dump1 dump2 | wc −l`−1] | cut −b 3− |\
          sed −e 's/^\([^ ]*\) \(.*\)$/mv lost+found\/#\1 dest\/"\2"/' |\
          sed −e 's/!/"\\\!"/g'
```

Repeat all this until /lost+found is empty.

11. Final touch

If in Section 9 you choosed to get files with the size of zero back you now have a problem, because those files has a non−zero deletion time and their link count is zero. That means that every time *e2fsck* is running it will prompt to remove (clear) those files.

The easy way to solve this is to copy the whole directory structure to another place (it can be the same partition) and then delete the old directory structure. Otherwise you will have to locate those inodes and change them with *debugfs*.

Now if everything have went well, things should have been restored to its original state before deletion. At least it did for me and in those tests I have made while writing this. Be sure that you have brought up to the preconditions described in Section 2.

12. References

Linux Ext2fs Undeletion mini−HOWTO, v1.3

- Aaron Crane

Design and Implementation of the Second Extended Filesystem, http://e2fsprogs.sourceforge.net/ext2intro.html

- Rémy Card, Laboratoire MASI−−Institut Blaise Pascal
- Theodore Ts'o, Massachussets Institute of Technology
- Stephen Tweedie, University of Edinburgh

Kernel Source for Linux 2.2.16

- linux/include/linux/ext2_fs.h
- linux/fs/ext2/namei.c