Linux Ext2fs Undeletion mini−HOWTO

Table of Contents

Linux Ext2fs Undeletion mini−HOWTO

Aaron Crane, `aaronc@pobox.com`

v1.3, 2 February 1999

Picture this. You've spent the last three days with no sleep, no food, not even a shower. Your hacking compulsion has at last paid off: you've finished that program that will bring you world−wide fame and recognition. All that you still need to do is tar it up and put it on Metalab. Oh, and delete all those Emacs backup files. So you say `rm * ~`*. And too late, you notice the extra space in that command. You've just deleted your magnum opus! But help is at hand. This document presents a discussion of how to retrieve deleted files from a Second Extended File System. Just maybe, you'll be able to release that program after all...*

1.Introduction

This mini−Howto attempts to provide hints on how to retrieve deleted files from an ext2 file system. It also contains a limited amount of discussion of how to avoid deleting files in the first place.

I intend it to be useful certainly for people who have just had, shall we say, a little accident with `rm`;

however, I also hope that people read it anyway. You never know: one day, some of the information in here could save your bacon.

The text assumes a little background knowledge about UNIX file systems in general; however, I hope that it will be accessible to most Linux users. If you are an outright beginner, I'm afraid that undeleting files under Linux *does* require a certain amount of technical knowledge and persistence, at least for the time being.

You will be unable to recover deleted files from an ext2 file system without at least read access to the raw device on which the file was stored. In general, this means that you must be root, but some distributions (such as Debian GNU/Linux) provide a `disk` group whose members have access to such devices. You also need `debugfs` from the `e2fsprogs` package. This should have been installed by your distribution.

Why have I written this? It stems largely from my own experiences with a particularly foolish and disastrous `rm −r` command as root. I deleted about 97 JPEG files which I needed and could almost certainly not recover from other sources. Using some helpful tips (see section Credits and Bibliography) and a great deal of persistence, I recovered 91 files undamaged. I managed to retrieve at least parts of five of the rest (enough to see what the picture was in each case). Only one was undisplayable, and even for this one, I am fairly sure that no more than 1024 bytes were lost (though unfortunately from the beginning of the file; given that I know nothing about the JFIF file format I had done as much as I could).

I shall discuss further below what sort of recovery rate you can expect for deleted files.

1.1 Revision history

The various publicly−released revisions of this document (and their publication dates) are as follows:

- v1.0 on 18 January 1997
- v1.1 on 23 July 1997 (see section Changes in version 1.1)
- v1.2 on 4 August 1997 (see section Changes in version 1.2)
- v1.3 on 2 February 1999 (see section Changes in version 1.3)

Changes in version 1.1

What changes have been made in this version? First of all, the thinko in the example of file recovery has been fixed. Thankyou to all those who wrote to point out my mistaek; I hope I've learned to be more careful when making up program interaction.

Secondly, the discussion of UNIX file system layout has been rewritten to be, I hope, more understandable. I wasn't entirely happy with it in the first place, and some people's comments indicated that it wasn't clear.

Thirdly, the vast uuencoded gzipped tarball of `fsgrab` in the middle of the file has been removed. The program is now available on my website and on Metalab (and mirrors).

Fourthly, the document has been translated into the Linux Documentation Project SGML Tools content markup language. This markup language can be easily converted to any of a number of other markup languages (including HTML and LaTeX) for convenient display and printing. One benefit of this is that beautiful typography in paper editions is a much more achievable goal; another is that the document has cross−references and hyperlinks when viewed on the Web.

Changes in version 1.2

This revision is very much an incremental change. It's here mainly to include changes suggested by readers, one of which is particularly important.

The first change was suggested by Egil Kvaleberg `egil@kvaleberg.no`, who pointed out the `dump` command in `debugfs`. Thanks again, Egil.

The second change is to mention the use of `chattr` for avoiding deleting important files. Thanks to Herman Suijs `H.P.M.Suijs@kub.nl` for mentioning this one.

The abstract has been revised. URLs have been added for organisations and software. Various other minor changes have been made (including fixing typos and so on).

Changes in version 1.3

Though it is the first release in 17 months, there is very little that is new here. This release merely fixes a few minor errors (typos, dangling URLs, that sort of thing −− especially the non−link to the Open Group), and updates a few parts of the text that have become hopelessly out−of−date, such as the material on kernel versions and on `lde`. Oh, and I've changed `Sunsite' to `Metalab' throughout.

This release is anticipated to be the last one before release 2.0, which will hopefully be a full Howto. I have been working on some substantial changes which will justify an increment of the major version number.

1.2 Canonical locations of this document

The latest public release of this document should always be available in on the Linux Documentation Project site (and mirrors).

The latest release is also kept on my website in several formats:

- SGML source. This is the source as I have written it, using the SGML Tools package.
- HTML. This is HTML, automatically generated from the SGML source.
- Plain text. This is plain text, which is also automatically generated from the SGML source.

2.How not to delete files

It is vital to remember that Linux is unlike MS−DOS when it comes to undeletion. For MS−DOS (and its bastard progeny Windows 95), it is generally fairly straightforward to undelete a file − the `operating system' (I use the term loosely) even comes with a utility which automates much of the process. For Linux, this is not the case.

So. Rule number one (the prime directive, if you will) is:

> **KEEP BACKUPS**

no matter what. Think of all your data. Perhaps, like me, you keep several years' of accumulated email, contacts, programs, papers on your computer. Think of how your life would be turned upside down if you had a catastrophic disk failure, or if −− heaven forbid! −− a malicious cracker wiped your disks. This is not unlikely; I have corresponded with a number of people in just such a situation. I exhort all right−thinking Linux users to go out and buy a useful backup device, work out a decent backup schedule, and to *stick to it*. Myself, I use a spare hard disk on a second machine, and periodically mirror my home directory onto it over the ethernet. For more information on planning a backup schedule, read Frisch (1995) (see section Bibliography and Credits).

In the absence of backups, what then? (Or even in the presence of backups: belt and braces is no bad policy where important data is concerned.)

Try to set the permissions for important files to 440 (or less): denying yourself write access to them means that `rm` requires an explicit confirmation before deleting. (I find, however, that if I'm recursively deleting a directory with `rm −r`, I'll interrupt the program on the first or second confirmation request and reissue the command as `rm −rf`.)

A good trick for selected files is to create a hard link to them in a hidden directory. I heard a story once about a sysadmin who repeatedly deleted `/etc/passwd` by accident (thereby half−destroying the system). One of the fixes for this was to do something like the following (as root):

```
# mkdir /.backup
# ln /etc/passwd /.backup
```

It requires quite some effort to delete the file contents completely: if you say

```
# rm /etc/passwd
```

then

```
# ln /.backup/passwd /etc
```

will retrieve it. Of course, this does not help in the event that you overwrite the file, so keep backups anyway.

On an ext2 file system, it is possible to use ext2 attributes to protect things. These attributes are manipulated with the `chattr` command. There is an `append−only' attribute: a file with this attribute may be appended to, but may not be deleted, and the existing contents of the file may not be overwritten. If a directory has this attribute, any files or directories within it may be modified as normal, but no files may be deleted. The `append−only' attribute is set with

```
$ chattr +a FILE...
```

There is also an `immutable' attribute, which can only be set or cleared by root. A file or directory with this attribute may not be modified, deleted, renamed, or (hard) linked. It may be set as follows:

```
# chattr +i FILE...
```

The ext2fs also provides the `undeletable' attribute (`+u` in `chattr`). The intention is that if a file with that attribute is deleted, instead of actually being reused, it is merely moved to a `safe location' for deletion at a later date. Unfortunately this feature has not yet been implemented in mainstream kernels; and though in the past there has been some interest in implementing it, it is not (to my knowledge) available for any current kernels.

Some people advocate making `rm` a shell alias or function for `rm −i` (which asks for confirmation on *every* file you delete). Indeed, the Red Hat distribution does this by default for all users, including root. Personally, I cannot stand software which won't run unattended, so I don't do that. There is also the problem that sooner or later, you'll be running in single−user mode, or using a different shell, or even a different machine, where your `rm` function doesn't exist. If you expect to be asked for confirmation, it is easy to forget where you are and to specify too many files for deletion. Likewise, the various scripts and programs that replace `rm` are, IMHO, very dangerous.

A slightly better solution is to start using a package which handles `recyclable' deletion by providing a command not named `rm`. For details on these, see Peek, et al (1993) (see section Bibliography and Credits). These however still suffer from the problem that they tend to encourage the user to have a nonchalant attitude to deletion, rather than the cautious approach that is often required on Unix systems.

3.What recovery rate can I expect?

That depends. Among the problems with recovering files on a high−quality, multi−tasking, multi−user operating system like Linux is that you never know when someone wants to write to the disk. So when the operating system is told to delete a file, it assumes that the blocks used by that file are fair game when it wants to allocate space for a new file. (This is a specific example of a general principle for Unix−like systems: the kernel and the associated tools assume that the users aren't idiots.) In general, the more usage your machine gets, the less likely you are to be able to recover files successfully.

Also, disk fragmentation can affect the ease of recovering files. If the partition containing the deleted files is very fragmented, you are unlikely to be able to read a whole file.

If your machine, like mine, is effectively a single−user workstation, and you weren't doing anything disk−intensive at the fatal moment of deleting those files, I would expect a recovery rate in the same ball−park as detailed above. I retrieved nearly 94% of the files (and these were binary files, please note) undamaged. If you get 80% or better, you can feel pretty pleased with yourself, I should think.

4.So, how do I undelete a file?

The procedure principally involves finding the data on the raw partition device and making it visible again to the operating system. There are basically two ways of doing this: one is to modify the existing file system such that the deleted inodes have their `deleted' flag removed, and hope that the data just magically falls back into place. The other method, which is safer but slower, is to work out where the data lies in the partition and write it out into a new file on another file system.

There are some steps you need to take before beginning to attempt your data recovery; see sections Unmounting the file system, Preparing to change inodes directly and Preparing to write data elsewhere for details. To find out how to actually retrieve your files, see sections Finding the deleted inodes, Obtaining the details of the inodes, Recovering data blocks and Modifying inodes directly.

5.Unmounting the file system

Regardless of which method you choose, the first step is to unmount the file system containing the deleted files. I strongly discourage any urges you may have to mess around on a mounted file system. This step should be performed *as soon as possible* after you realise that the files have been deleted; the sooner you can unmount, the smaller the chance that your data will be overwritten.

The simplest method is as follows: assuming the deleted files were in the `/usr` file system, say:

```
# umount /usr
```

You may, however, want to keep some things in `/usr` available. So remount it read−only:

```
# mount −o ro,remount /usr
```

If the deleted files were on the root partition, you'll need to add a `−n` option to prevent mount from trying to write to `/etc/mtab`:

```
# mount −n −o ro,remount /
```

Regardless of all this, it is possible that there will be another process using that file system (which will cause the unmount to fail with an error such as `Resource busy'). There is a program which will send a signal to any process using a given file or mount point: `fuser`. Try this for the `/usr` partition:

```
# fuser −v −m /usr
```

This lists the processes involved. Assuming none of them are vital, you can say

```
# fuser −k −v −m /usr
```

to send each process a `SIGKILL` (which is guaranteed to kill it), or for example,

```
# fuser −k −TERM −v −m /usr
```

to give each one a `SIGTERM` (which will normally make the process exit cleanly).

6.Preparing to change inodes directly

My advice? Don't do it this way. I really don't think it's wise to play with a file system at a low enough level for this to work. This method also has problems in that you can only reliably recover the first 12 blocks of each file. So if you have any long files to recover, you'll normally have to use the other method anyway. (Although see section Will this get easier in future? for additional information.)

If you feel you must do it this way, my advice is to copy the raw partition data to an image on a different partition, and then mount this using loopback:

```
# cp /dev/hda5 /root/working
# mount −t ext2 −o loop /root/working /mnt
```

(Note that obsolete versions of `mount` may have problems with this. If your `mount` doesn't work, I strongly suggest you get the latest version, or at least version 2.7, as some very old versions have severe security bugs.)

Using loopback means that if and when you completely destroy the file system, all you have to do is copy the raw partition back and start over.

7.Preparing to write data elsewhere

If you chose to go this route, you need to make sure you have a rescue partition somewhere −− a place to write out new copies of the files you recover. Hopefully, your system has several partitions on it: perhaps a root, a `/usr`, and a `/home`. With all these to choose from, you should have no problem: just create a new directory on one of these.

If you have only a root partition, and store everything on that, things are slightly more awkward. Perhaps you have an MS−DOS or Windows partition you could use? Or you have the ramdisk driver in your kernel, maybe as a module? To use the ramdisk (assuming a kernel more recent than 1.3.48), say the following:

```
# dd if=/dev/zero of=/dev/ram0 bs=1k count=2048
# mke2fs −v −m 0 /dev/ram0 2048
# mount −t ext2 /dev/ram0 /mnt
```

This creates a 2MB ramdisk volume, and mounts it on `/mnt`.

A short word of warning: if you use `kerneld` (or its replacement `kmod` in 2.2.x and later 2.1.x kernels) to automatically load and unload kernel modules, then don't unmount the ramdisk until you've copied any files from it onto non−volatile storage. Once you unmount it, `kerneld` assumes it can unload the module (after the usual waiting period), and once this happens, the memory gets re−used by other parts of the kernel, losing all the painstaking hours you just spent recovering your data.

If you have a Zip, Jaz, or LS−120 drive, or something similar, it would probably be a good choice for a rescue partition location. Otherwise, you'll just have to stick with floppies.

The other thing you're likely to need is a program which can read the necessary data from the middle of the partition device. At a pinch, `dd` will do the job, but to read from, say, 600 MB into an 800 MB partition, `dd` insists on reading but ignoring the first 600 MB. This takes a not inconsiderable amount of time, even on fast disks. My way round this was to write a program which will seek to the middle of the partition. It's called `fsgrab`; you can find the source package on my website or on Metalab (and mirrors). If you want to use this method, the rest of this mini−Howto assumes that you have `fsgrab`.

If none of the files you are trying to recover were more than 12 blocks long (where a block is usually one kilobyte), then you won't need `fsgrab`.

If you need to use `fsgrab` but don't want to download and build it, it is fairly straightforward to translate an `fsgrab` command−line to one for `dd`. If we have

```
fsgrab −c count −s skipdevice
```

then the corresponding (but typically much slower) `dd` command is

```
dd bs=1k if=device count=count skip=skip
```

I must warn you that, although `fsgrab` functioned perfectly for me, I can take no responsibility for how it performs. It was really a very quick and dirty kludge just to get things to work. For more details on the lack of warranty, see the `No Warranty' section in the `COPYING` file included with it (the GNU General Public Licence).

8.Finding the deleted inodes

The next step is to ask the file system which inodes have recently been freed. This is a task you can accomplish with `debugfs`. Start `debugfs` with the name of the device on which the file system is stored:

```
# debugfs /dev/hda5
```

If you want to modify the inodes directly, add a `−w` option to enable writing to the file system:

```
# debugfs −w /dev/hda5
```

The `debugfs` command to find the deleted inodes is `lsdel`. So, type the command at the prompt:

```
debugfs:  lsdel
```

After much wailing and grinding of disk mechanisms, a long list is piped into your favourite pager (the value of `$PAGER`). Now you'll want to save a copy of this somewhere else. If you have `less`, you can type `-o` followed by the name of an output file. Otherwise, you'll have to arrange to send the output elsewhere. Try this:

```
debugfs:  quit
# echo lsdel | debugfs /dev/hda5 > lsdel.out
```

Now, based only on the deletion time, the size, the type, and the numerical permissions and owner, you must work out which of these deleted inodes are the ones you want. With luck, you'll be able to spot them because they're the big bunch you deleted about five minutes ago. Otherwise, trawl through that list carefully.

I suggest that if possible, you print out the list of the inodes you want to recover. It will make life a lot easier.

9.Obtaining the details of the inodes

`debugfs` has a `stat` command which prints details about an inode. Issue the command for each inode in your recovery list. For example, if you're interested in inode number 148003, try this:

```
debugfs:  stat <148003>
Inode: 148003   Type: regular    Mode:  0644   Flags: 0x0   Version: 1
User:   503   Group:   100   Size: 6065
File ACL: 0    Directory ACL: 0
Links: 0   Blockcount: 12
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x31a9a574 -- Mon May 27 13:52:04 1996
atime: 0x31a21dd1 -- Tue May 21 20:47:29 1996
mtime: 0x313bf4d7 -- Tue Mar  5 08:01:27 1996
dtime: 0x31a9a574 -- Mon May 27 13:52:04 1996
BLOCKS:
594810 594811 594814 594815 594816 594817
TOTAL: 6
```

If you have a lot of files to recover, you'll want to automate this. Assuming that your `lsdel` list of inodes to recover in is in `lsdel.out`, try this:

```
# cut -c1-6 lsdel.out | grep "[0-9]" | tr -d " " > inodes
```

This new file `inodes` contains just the numbers of the inodes to recover, one per line. We save it because it will very likely come in handy later on. Then you just say:

```
# sed 's/^.*$/stat <\0>/' inodes | debugfs /dev/hda5 > stats
```

and `stats` contains the output of all the `stat` commands.

10.Recovering data blocks

This part is either very easy or distinctly less so, depending on whether the file you are trying to recover is more than 12 blocks long.

10.1 Short files

If the file was no more than 12 blocks long, then the block numbers of all its data are stored in the inode: you can read them directly out of the `stat` output for the inode. Moreover, `debugfs` has a command which performs this task automatically. To take the example we had before, repeated here:

```
debugfs:  stat <148003>
Inode: 148003   Type: regular    Mode:  0644   Flags: 0x0   Version: 1
User:   503   Group:   100   Size: 6065
File ACL: 0    Directory ACL: 0
Links: 0   Blockcount: 12
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x31a9a574 -- Mon May 27 13:52:04 1996
atime: 0x31a21dd1 -- Tue May 21 20:47:29 1996
mtime: 0x313bf4d7 -- Tue Mar  5 08:01:27 1996
dtime: 0x31a9a574 -- Mon May 27 13:52:04 1996
BLOCKS:
594810 594811 594814 594815 594816 594817
TOTAL: 6
```

This file has six blocks. Since this is less than the limit of 12, we get `debugfs` to write the file into a new location, such as `/mnt/recovered.000`:

```
debugfs:  dump <148003> /mnt/recovered.000
```

Of course, this can also be done with `fsgrab`; I'll present it here as an example of using it:

```
# fsgrab -c 2 -s 594810 /dev/hda5 > /mnt/recovered.000
# fsgrab -c 4 -s 594814 /dev/hda5 >> /mnt/recovered.000
```

With either `debugfs` or `fsgrab`, there will be some garbage at the end of `/mnt/recovered.000`, but that's fairly unimportant. If you want to get rid of it, the simplest method is to take the `Size` field from the inode, and plug it into the `bs` option in a `dd` command line:

```
# dd count=1 if=/mnt/recovered.000 of=/mnt/resized.000 bs=6065
```

Of course, it is possible that one or more of the blocks that made up your file has been overwritten. If so, then you're out of luck: that block is gone forever. (But just imagine if you'd unmounted sooner!)

10.2 Longer files

The problems appear when the file has more than 12 data blocks. It pays here to know a little of how UNIX file systems are structured. The file's data is stored in units called `blocks'. These blocks may be numbered sequentially. A file also has an `inode', which is the place where information such as owner, permissions, and type are kept. Like blocks, inodes are numbered sequentially, although they have a different sequence. A directory entry consists of the name of the file and an inode number.

But with this state of affairs, it is still impossible for the kernel to find the data corresponding to a directory entry. So the inode also stores the location of the file's data blocks, as follows:

- The block numbers of the first 12 data blocks are stored directly in the inode; these are sometimes referred to as the *direct block*s.
- The inode contains the block number of an *indirect block*. An indirect block contains the block numbers of 256 additional data blocks.
- The inode contains the block number of a *doubly indirect block*. A doubly indirect block contains the block numbers of 256 additional indirect blocks.
- The inode contains the block number of a *triply indirect block*. A triply indirect block contains the block numbers of 256 additional doubly indirect blocks.

Read that again: I know it's complex, but it's also important.

Now, the kernel implementation for all versions up to and including 2.0.36 unfortunately zeroes all indirect blocks (and doubly indirect blocks, and so on) when deleting a file. So if your file was longer than 12 blocks, you have no guarantee of being able to find even the numbers of all the blocks you need, let alone their contents.

The only method I have been able to find thus far is to assume that the file was not fragmented: if it was, then you're in trouble. Assuming that the file was not fragmented, there are several layouts of data blocks, according to how many data blocks the file used:

0 to 12

The block numbers are stored in the inode, as described above.

13 to 268

After the direct blocks, count one for the indirect block, and then there are 256 data blocks.

269 to 65804

As before, there are 12 direct blocks, a (useless) indirect block, and 256 blocks. These are followed by one (useless) doubly indirect block, and 256 repetitions of one (useless) indirect block and 256 data blocks.

65805 or more

The layout of the first 65804 blocks is as above. Then follow one (useless) triply indirect block and 256 repetitions of a `doubly indirect sequence'. Each doubly indirect sequence consists of a (useless) doubly indirect block, followed by 256 repetitions of one (useless) indirect block and 256 data blocks.

Of course, even if these assumed data block numbers are correct, there is no guarantee that the data in them is intact. In addition, the longer the file was, the less chance there is that it was written to the file system without appreciable fragmentation (except in special circumstances).

You should note that I assume throughout that your blocksize is 1024 bytes, as this is the standard value. If your blocks are bigger, some of the numbers above will change. Specifically: since each block number is 4 bytes long, blocksize/4 is the number of block numbers that can be stored in each indirect block. So every time the number 256 appears in the discussion above, replace it with blocksize/4. The `number of blocks required' boundaries will also have to be changed.

Let's look at an example of recovering a longer file.

```
debugfs:  stat <1387>
Inode: 148004   Type: regular    Mode:  0644   Flags: 0x0   Version: 1
User:   503   Group:   100   Size: 1851347
File ACL: 0    Directory ACL: 0
Links: 0   Blockcount: 3616
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x31a9a574 -- Mon May 27 13:52:04 1996
atime: 0x31a21dd1 -- Tue May 21 20:47:29 1996
mtime: 0x313bf4d7 -- Tue Mar  5 08:01:27 1996
dtime: 0x31a9a574 -- Mon May 27 13:52:04 1996
BLOCKS:
8314 8315 8316 8317 8318 8319 8320 8321 8322 8323 8324 8325 8326 8583
TOTAL: 14
```

There seems to be a reasonable chance that this file is not fragmented: certainly, the first 12 blocks listed in the inode (which are all data blocks) are contiguous. So, we can start by retrieving those blocks:

```
# fsgrab −c 12 −s 8314 /dev/hda5 > /mnt/recovered.001
```

Now, the next block listed in the inode, 8326, is an indirect block, which we can ignore. But we trust that it will be followed by 256 data blocks (numbers 8327 through 8582).

```
# fsgrab −c 256 −s 8327 /dev/hda5 >> /mnt/recovered.001
```

The final block listed in the inode is 8583. Note that we're still looking good in terms of the file being contiguous: the last data block we wrote out was number 8582, which is 8327 + 255. This block 8583 is a doubly indirect block, which we can ignore. It is followed by up to 256 repetitions of an indirect block (which is ignored) followed by 256 data blocks. So doing the arithmetic quickly, we issue the following commands. Notice that we skip the doubly indirect block 8583, and the indirect block 8584 immediately (we hope) following it, and start at block 8585 for data.

```
# fsgrab −c 256 −s 8585 /dev/hda5 >> /mnt/recovered.001
# fsgrab −c 256 −s 8842 /dev/hda5 >> /mnt/recovered.001
# fsgrab −c 256 −s 9099 /dev/hda5 >> /mnt/recovered.001
# fsgrab −c 256 −s 9356 /dev/hda5 >> /mnt/recovered.001
# fsgrab −c 256 −s 9613 /dev/hda5 >> /mnt/recovered.001
# fsgrab −c 256 −s 9870 /dev/hda5 >> /mnt/recovered.001
```

Adding up, we see that so far we've written 12 + (7 * 256) blocks, which is 1804. The `stat' results for the inode gave us a `blockcount' of 3616; unfortunately these blocks are 512 bytes long (as a hangover from UNIX), so we really want 3616/2 = 1808 blocks of 1024 bytes. That means we need only four more blocks. The last data block written was number 10125. As we've been doing so far, we skip an indirect block (number 10126); we can then write those last four blocks.

```
# fsgrab −c 4 −s 10127 /dev/hda5 >> /mnt/recovered.001
```

Now, with some luck the entire file has been recovered successfully.

11.Modifying inodes directly

This method is, on the surface, much easier. However, as mentioned above, it cannot yet cope with files longer than 12 blocks.

For each inode you want to recover, you must set the usage count to one, and set the deletion time to zero. This is done with the `mi` (modify inode) command in `debugfs`. Some sample output, modifying inode 148003 from above:

```
debugfs:  mi <148003>
                          Mode    [0100644]
                       User ID    [503]
                      Group ID    [100]
                          Size    [6065]
                 Creation time    [833201524]
             Modification time    [832708049]
                   Access time    [826012887]
                 Deletion time    [833201524] 0
                    Link count    [0] 1
                   Block count    [12]
                    File flags    [0x0]
                     Reserved1    [0]
                      File acl    [0]
                 Directory acl    [0]
              Fragment address    [0]
               Fragment number    [0]
                 Fragment size    [0]
               Direct Block #0    [594810]
               Direct Block #1    [594811]
               Direct Block #2    [594814]
               Direct Block #3    [594815]
               Direct Block #4    [594816]
               Direct Block #5    [594817]
               Direct Block #6    [0]
               Direct Block #7    [0]
               Direct Block #8    [0]
               Direct Block #9    [0]
              Direct Block #10    [0]
              Direct Block #11    [0]
                Indirect Block    [0]
         Double Indirect Block    [0]
         Triple Indirect Block    [0]
```

That is, I set the deletion time to 0 and the link count to 1 and just pressed return for each of the other fields. Granted, this is a little unwieldy if you have a lot of files to recover, but I think you can cope. If you'd wanted chrome, you'd have used a graphical `operating system' with a pretty `Recycle Bin'.

By the way: the `mi` output refers to a `Creation time' field in the inode. This is a lie! (Or misleading, anyway.) The fact of the matter is that you cannot tell on a UNIX file system when a file was created. The `st_ctime` member of a `struct stat` refers to the `inode change time', that is, the last time when any inode details were changed. Here endeth today's lesson.

Note that more recent versions of `debugfs` than the one I'm using probably do not include some of the fields in the listing above (specifically, `Reserved1` and (some of?) the fragment fields).

Once you've modified the inodes, you can quit `debugfs` and say:

```
# e2fsck −f /dev/hda5
```

The idea is that each of the deleted files has been literally undeleted, but none of them appear in any directory entries. The `e2fsck` program can detect this, and will add a directory entry for each file in the `/lost+found` directory of the file system. (So if the partition is normally mounted on `/usr`, the files will now appear in `/usr/lost+found` when you next mount it.) All that still remains to be done is to work out the name of each file from its contents, and return it to its correct place in the file system tree.

When you run `e2fsck`, you will get some informative output, and some questions about what damage to repair. Answer `yes' to everything that refers to `summary information' or to the inodes you've changed. Anything else I leave up to you, although it's usually a good idea to say `yes' to all the questions. When `e2fsck` finishes, you can remount the file system.

Actually, there's an alternative to having `e2fsck` leave the files in `/lost+found`: you can use `debugfs` to create a link in the file system to the inode. Use the `link` command in `debugfs` after you've modified the inode:

```
debugfs:  link <148003> foo.txt
```

This creates a file called `foo.txt` in what `debugfs` thinks is the current directory; `foo.txt` will be your file. You'll still need to run `e2fsck` to fix the summary information and block counts and so on.

12.Will this get easier in future?

Yes. In fact, I believe it already has. Although as of this writing, current stable kernels (in the 2.0.x series) zero indirect blocks, this does not apply to development kernels in the 2.1.x series, nor to the stable 2.2.x series. As I write this on 2 February 1999, kernel 2.2.1 was released a few days ago; Linux vendors are likely to start producing distributions containing and supporting 2.2.x kernels a month or two from now.

Once the indirect−zeroing limitation has been overcome in the production kernels, a lot of my objections to the technique of modifying inodes by hand will disappear. At the same time, it will also become possible to use the `dump` command in `debugfs` on long files, and to conveniently use other undeletion tools.

13.Are there any tools to automate this process?

As it happens, there are. Unfortunately, I believe that they currently suffer from the same problem as the manual inode modification technique: indirect blocks are unrecoverable. However, given the likelihood that this will shortly no longer be a problem, it's well worth looking these programs out now.

I have written a tool called `e2recover`, which is essentially a Perl wrapper around `fsgrab`. It makes a reasonable amount of effort to deal with zeroed indirect blocks, and seems to work fairly well as long as there was no fragmentation. It also correctly sets the permissions (and when possible the ownership) of recovered files, and even makes sure that recovered files have the correct length.

I originally wrote `e2recover` for the forthcoming major update to this Howto; unfortunately this means that much of the useful documentation for `e2recover` is scheduled for inclusion in that update. Be that as it may, it should be useful now; it can be downloaded from my web site, and soon from Metalab.

Scott D. Heavner is the author of `lde`, the Linux Disk Editor. It can be used as both a binary disk editor, and as an equivalent to `debugfs` for ext2 and minix file systems, and even for xia file systems (though xia support is no longer available in 2.1.x and 2.2.x kernels). It has some features for assisting undeletion, both by walking the block list for a file, and by grepping through disk contents. It also has some fairly useful documentation on basic file system concepts, as well as a document on how to use it for undeletion. Version 2.4 of `lde` is available on Metalab and mirrors, or on the author's web site.

Another possibility is offered by the GNU Midnight Commander, `mc`. This is a full−screen file management tool, based AFAIK on a certain MS−DOS program commonly known as `NC'. `mc` supports the mouse on the Linux console and in an xterm, and provides virtual file systems which allow tricks like `cd`−ing to a tarfile. Among its virtual file systems is one for ext2 undeletion. It all sounds very handy, although I must admit I don't use the program myself −− I prefer good old−fashioned shell commands.

To use the undeletion feature, you have to configure the program with the `--with-ext2undel` option; you'll also need the development libraries and include files that come with the `e2fsprogs` package. The version provided in Debian GNU/Linux is built in this way; the same may apply to packages for other Linux distributions. Once the program is built, you can tell it to `cd undel:/dev/hda5`, and get a `directory listing' of deleted files. Like many current undeletion tools, it handles zeroed indirect blocks poorly −− it typically just recovers the first 12k of long files.

The current version may be downloaded from the Midnight Commander ftp site.

14.Colophon

I intend to produce regular updates to this document as long as I have both enough time to do it, and something interesting to say. This means that I am eager to hear comments from readers. Could my writing be clearer? Can you think of something that would make matters easier? Is there some new tool that does it all automatically? Whatever. If you have something to say about this document or about the `fsgrab` or `e2recover` tools, drop me a line on `aaronc@pobox.com`.

15.Credits and Bibliography

> `If I have seen farther than others, it is because I was standing on the shoulders of giants.'
> (Isaac Newton)

This mini−Howto was originally derived from a posting in the `comp.os.linux.misc` newsgroup by Robin Glover `swrglovr@met.rdg.ac.uk`. I would like to thank Robin for graciously allowing me to rework his ideas into this mini−Howto.

I would also like to take this opportunity to thank once again all the people who've written to me about the Howto. Receiving grateful comments makes the effort worth while.

Some bibliographic references:

- **Frisch**, Æleen (1995), *Essential System Administration*, second edition, O'Reilly and Associates, Inc., ISBN: 1−56592−127−5.
- **Garfinkel**, Simson, Daniel **Weise** and Steven **Strassmann** (1994), *The Unix−Haters Handbook*, IDG Books, ISBN: 1−56884−203−1. Much of this book is merely the adolescent whinings of people who think that *their* operating system was so much better than Unix, and much of the rest simply doesn't apply if you have a well−written user−space such as GNU. But there is some wheat among the chaff; for example, the discussion of how easy it is to delete files under Unix is well worth reading.
- **Glover**, Robin (31 Jan 1996), *HOW−TO : undelete linux files (ext2fs/debugfs)*, comp.os.linux.misc Usenet posting.
- **Peek**, Jerry, Tim **O'Reilly**, Mike **Loukides** et al (1993), *UNIX Power Tools*, O'Reilly and Associates, Inc./Random House, Inc., ISBN: 0−679−79073−X. Second edition, 1998.

16.Legalities

All trademarks are the property of their respective owners. Specifically:

- *MS−DOS* and *Windows* are trademarks of Microsoft.
- *UNIX* is a trademark of the Open Group.
- *Linux* is a trademark of Linus Torvalds in the USA and some other countries.

The author requests but does not require that parties intending to sell copies of this document, whether on computer−readable or human−readable media, inform either him or the Linux HOWTO Coordinator of their intentions.

The Linux HOWTO Coordinator is currently Tim Bynum `linux-howto@metalab.unc.edu`.
