

**100 NEW
HACKS**

LINUX SERVER HACKS™

Volume Two

*Tips & Tools for
Connecting, Monitoring,
and Troubleshooting*



O'REILLY®

*William von Hagen
& Brian K. Jones*

HACK
#96

Piece Together Data from the lost+found

`fsck` and similar programs save lost or unlinked files and directories automatically. Here's how to figure out what they are.

The `fsck` utility, created by Ted Kowalski and others at Bell Labs for ancient versions of Unix, removed much of the black magic from checking and correcting the consistency of Unix filesystems. No one wept many tears for the passing of `fsck`'s predecessors, `ichck` and `ncheck`, since `fsck` is far smarter and encapsulates a lot of knowledge about filesystem organization and repair. One of the coolest things that `fsck` brought to Unix filesystems was the notion of the `lost+found` directory at the root of a Unix filesystem. Though actually created by utilities associated with filesystem creation (`newfs`, `mkfs`, `mklost+found`, and so on, depending on the filesystem and version of Unix or Linux that you're using), the `lost+found` directory is there expressly for the use of filesystem repair utilities such as `fsck`, `e2fsck`, `xfs_repair`, and so on.

The idea behind the `lost+found` directory was to preallocate a specific directory with a relatively large number of directory entries, to be used as an electronic catcher's mitt for storing files and directories whose actual locations in the filesystem can't be determined during a filesystem consistency check. When a utility such as `fsck` performs a full filesystem consistency check, its primary goal is to verify the integrity of the filesystem, which means that filesystem metadata such as lists of free and allocated blocks, inodes, or extents (typically stored as bitmaps) are correct, all files and directories in the filesystem are correctly linked into the filesystem, directory and file attributes are correct, and so on. Unfortunately, preserving corrupted data is a secondary concern during filesystem consistency checking and repair. Inconsistent files or directories are usually simply purged during a filesystem consistency check, but the contents of directories that are purged may still themselves be consistent. When this situation occurs during a filesystem consistency check, the contents of such directories are automatically linked to existing (empty) entries in that filesystem's `lost+found` directory. On older Unix systems, the hard links to these "recovered" files and directories were given names corresponding to their inode numbers. On `ext2` or `ext3` Linux filesystems, the hard links to such files and directories are given names beginning with a hash mark (`#`) and followed by the inode number.

When you encounter a severely corrupted filesystem or recover one as part of a repair or recovery [Hack #94], you will almost always find files and directories in that filesystem's `lost+found` directory after `fsck`'ing the filesystem. Here are some tips on how to figure out what they contain, what files and

directories they may have been, and how to put them back into the actual filesystem.



This hack focuses on piecing things together for an *ext2* or *ext3* filesystem, but the procedure for identifying files and directories applies to other filesystems as well. For some ReiserFS-specific tips, see “Repair and Recover ReiserFS Filesystems” [Hack #95].

Exploring the `lost+found`

To explore a filesystem’s *lost+found* directory, you must first mount the filesystem using the standard Linux `mount` command, which you must execute as the root user. Once the filesystem is mounted, `cd` to the *lost+found* directory at the root of that filesystem, which will be located in the directory where you mounted the filesystem. If this directory contains any files or directories, you’re in luck—there’s more data in your filesystem than just the standard files and directories it contains!

The entries in the *lost+found* directory are files and directories whose parent inodes or directories were damaged and discarded during the consistency check. You will have to do a bit of detective work to find out what these are, but two factors work in your favor:

- The names of the files and directories in the *lost+found* directory for an *ext2/ext3* filesystem are based on the numbers of the inodes associated with the lost files or directories.
- The *e2fsck* program simply re-links unconnected files and directories into the *lost+found* directory, which preserves the creation, access, and modification timestamps associated with those files and directories.

The first thing to do when exploring an *ext2* or *ext3* *lost+found* directory is to prepare an area on another disk to which you can temporarily copy files and directories as you attempt to reconstruct their organization. In this hack, I’ll use the example */usr/restore*, but you can use any location. As you proceed with exploration and reconstruction, it is important not to modify the files in the *lost+found* directory in any way other than by copying them elsewhere, or you may lose helpful timestamp information.

Just to be safe, first redirect a long directory listing of the contents of the *lost+found* directory into a file in your restore area, as in the following example:

```
# cd /mnt/baddisk  
# ls -lt > /usr/restore/listing.txt
```

This listing is a precaution against accidental modification of those files. Here's a section of the sample output from the *lost+found* directory from "Recover Lost Partitions" [Hack #93]:

```
# ls -lt
total 2116264
drwx----- 3 root root    16384 2005-06-17 18:14 .
drwxr-xr-x  6 root root     4096 2005-06-17 18:14 ..
-rw-r--r--  1 wvh users 48873341 2005-02-12 08:41 #11993089
-rw-r--r--  1 wvh users 26737789 2005-02-12 08:41 #11993090
-rw-r--r--  1 wvh users 27987253 2005-02-12 08:41 #11993091
-rw-r--r--  1 wvh users 24691821 2005-02-12 08:41 #11993092
-rw-r--r--  1 wvh users 25752913 2005-02-12 08:41 #11993093
-rw-r--r--  1 wvh users 15258373 2005-02-12 08:41 #11993094
-rw-r--r--  1 wvh users 16291065 2005-02-12 08:41 #11993095
-rw-r--r--  1 wvh users 25151049 2005-02-12 08:41 #11993096
-rw-r--r--  1 wvh users 27290257 2005-02-12 08:41 #11993097
-rw-r--r--  1 wvh users    31643 2005-02-12 08:41 #11993098
-rw-r--r--  1 wvh users     2751 2005-02-12 08:41 #11993099
-rw-r--r--  1 wvh users     2670 2005-02-12 08:41 #11993100
-rw-r--r--  1 wvh users 35270097 2005-01-28 05:29 #14811137
-rw-r--r--  1 wvh users 39914258 2005-01-28 05:29 #14811138
-rw-r--r--  1 wvh users 39709879 2005-01-28 05:30 #14811139
-rw-r--r--  1 wvh users 58648049 2005-01-28 05:30 #14811140
-rw-r--r--  1 wvh users 29533858 2005-01-28 05:30 #14811141
-rw-r--r--  1 wvh users 27692066 2005-01-28 05:30 #14811142
-rw-r--r--  1 wvh users 29308352 2005-01-28 05:30 #14811143
-rw-r--r--  1 wvh users     564 2005-01-28 05:30 #14811144
-rw-r--r--  1 wvh users     809 2005-01-28 05:30 #14811145
-rw-r--r--  1 wvh users     156 2005-01-28 05:30 #14811146
drwxr-xr-x  2 lmp users    4096 2005-01-22 21:46 #30507055
drwxr-xr-x  2 lmp users    4096 2005-01-22 21:45 #30507031
-rw-r--r--  1 wvh users 29523256 2005-01-18 05:21 #3063821
[much more output removed]
```

As you can see from this example, the files and directories in my *lost+found* directory are nicely grouped by date and inode number, and many of them were last modified on the same date. This is typical of partitions that are essentially written to once and then used as a source of data. In this case, the partition I lost was a repository for an online music collection for my server's users, consisting of audio files and associated files such as playlists and recording descriptions, so I have a good idea of how the files and directories were originally organized on the disk that went bad. The disk consisted of directories named by artist and date, each of which contained the recordings and associated files for the artist's performance on that date.

Recovering Directories from the *lost+found*

The first thing to do when exploring and recovering the contents of a *lost+found* directory is to copy out any directories that already contain related

sets of files. You can then explore the contents of these directories at your leisure, putting the recovered files back into a live filesystem on your machine.

As you can see from the previous code listing, my *lost+found* directory contains two directories, *#30507055* and *#30507031*. Listing both of these shows the following:

```
# ls -l \#30507055 \#30507031

#30507031:
total 0

#30507055:
total 222380
-rw-r--r--  1 lmp users      915 2005-01-22 21:45 monroe1967-05-15d2.ffp.txt
-rw-r--r--  1 lmp users 11694266 2005-01-22 21:45 monroe1967-05-15d2t01.flac
-rw-r--r--  1 lmp users 14046056 2005-01-22 21:45 monroe1967-05-15d2t02.flac
-rw-r--r--  1 lmp users 21405678 2005-01-22 21:45 monroe1967-05-15d2t03.flac
-rw-r--r--  1 lmp users 10724376 2005-01-22 21:45 monroe1967-05-15d2t04.flac
-rw-r--r--  1 lmp users 19590818 2005-01-22 21:45 monroe1967-05-15d2t05.flac
-rw-r--r--  1 lmp users 13981201 2005-01-22 21:45 monroe1967-05-15d2t06.flac
-rw-r--r--  1 lmp users 13576225 2005-01-22 21:45 monroe1967-05-15d2t07.flac
-rw-r--r--  1 lmp users 12057959 2005-01-22 21:45 monroe1967-05-15d2t08.flac
-rw-r--r--  1 lmp users 15432553 2005-01-22 21:45 monroe1967-05-15d2t09.flac
-rw-r--r--  1 lmp users 19475592 2005-01-22 21:46 monroe1967-05-15d2t10.flac
-rw-r--r--  1 lmp users 13427860 2005-01-22 21:46 monroe1967-05-15d2t11.flac
-rw-r--r--  1 lmp users 16973390 2005-01-22 21:46 monroe1967-05-15d2t12.flac
-rw-r--r--  1 lmp users 12077969 2005-01-22 21:46 monroe1967-05-15d2t13.flac
-rw-r--r--  1 lmp users 26182260 2005-01-22 21:46 monroe1967-05-15d2t14.flac
-rw-r--r--  1 lmp users  6718719 2005-01-22 21:46 monroe1967-05-15d2t15.flac
-rw-r--r--  1 lmp users      405 2005-01-22 21:46 playlist.m3u
```

The directory *#30507031* is empty and can safely be ignored, but the directory *#30507055* seems to contain an intact collection of related files. Based on the filenames, I know that this is a live performance by the bluegrass artist Bill Monroe from May 15, 1967, and that it was created by the user *lmp*. (By the way, you will rarely be this lucky!) To preserve this directory, I'll recursively copy it to my restore area, giving it an appropriate name:

```
# cp -rp \#30507055 /usr/restore/monroe1967-05-15
```

Note the use of the `cp` command's `-p` option, to preserve user and group ownership and timestamps.

If I can't easily identify the contents of a directory in the *lost+found*, I generally copy it to my restore area, giving it a name based on the directory's timestamp. The inode number in the old filesystem is meaningless after a copy, but a visual clue for knowing when the directory was last updated may be useful when trying to figure out what it contains, especially if a project or system user or group owns the directory.

Recovering Recognizable Groups of Files

When recovering files that are essentially preorganized by their creation dates, I usually create recovery directories in my restore area based on the timestamps and use this as a preliminary organizer when copying the files there. The previous code listing shows two groups of files, one created on Feb 12, 2005 (2005-02-12) and another created on January 28, 2005 (2005-01-28). I would thus create two corresponding directories and use wildcards to copy the associated files into those directories, as in the following example:

```
# mkdir /usr/restore/2005-02-12 /usr/restore/2005-01-28
# cp -p \#[11993]??? /usr/restore/2005-02-12
# cp -p \#[148111]? /usr/restore/2005-01-28
```

Next, let's try to figure out what each of these directories actually contains. Change directory to one of the restore directories and examine its contents using the file command:

```
# cd /usr/restore/2005-02-12
# file *
#11993089: data
#11993090: data
#11993091: data
#11993092: data
#11993093: data
#11993094: data
#11993095: data
#11993096: data
#11993097: data
#11993098: JPEG image data, JFIF standard 1.01
#11993099: ASCII English text, with CRLF line terminators
#11993100: ASCII text, with CRLF line terminators
#11993101: ASCII English text
```

Looking at the text files in any directory usually provides some information about the contents of that directory. Let's use the head command to examine the first 10 lines of each of the text files:

```
$ head *99 *100 *101
==> #11993099 <==
EAC extraction logfile from 8. February 2005, 23:22 for CD
Cheap Trick 1981-01-22d1t / Unknown Title

Used drive   : HP      DVD Writer 300n  Adapter: 1  ID: 1
Read mode    : Burst
Read offset correction : 0
Overread into Lead-In and Lead-Out : No

Used output format : Internal WAV Routines
                    44.100 Hz; 16 Bit; Stereo
==> #11993100 <==
EAC extraction logfile from 8. February 2005, 23:49 for CD
Cheap Trick 1981-01-22d2t / Unknown Title
```

Piece Together Data from the lost+found

```
Used drive   : HP          DVD Writer 300n  Adapter: 1  ID: 1
Read mode    : Burst
Read offset correction : 0
Overread into Lead-In and Lead-Out : No
```

```
Used output format : Internal WAV Routines
                    44.100 Hz; 16 Bit; Stereo
```

```
==> #11993101 <==
1981-01-22d1t01 Stop This Game.shn
1981-01-22d1t02 Go For The Throat (Use Your Own Imagination).shn
1981-01-22d1t03 Hello There.shn
1981-01-22d1t04 I Want You To Want Me.shn
1981-01-22d1t05 I Love You Honey But I Hate Your Friends.shn
1981-01-22d1t06 Clock Strikes Ten.shn
1981-01-22d1t07 Can't Stop It But I'm Gonna Try.shn
1981-01-22d1t08 Baby Loves To Rock And Roll.shn
1981-01-22d1t09 Gonna Raise Hell.shn
1981-01-22d2t01 Heaven Tonight.shn
```

This tells me that the first two files contain logfiles produced when ripping audio from the CDs that originally contained these live recordings, while the last (*#11993101*) contains a playlist for the files in the original directory. Let's see if looking at more of one of the logfiles can tell us more about the files in this directory:

```
$ head -20 *99
```

```
EAC extraction logfile from 8. February 2005, 23:22 for CD
Cheap Trick 1981-01-22d1t / Unknown Title
```

```
Used drive   : HP          DVD Writer 300n  Adapter: 1  ID: 1
Read mode    : Burst
Read offset correction : 0
Overread into Lead-In and Lead-Out : No
```

```
Used output format : Internal WAV Routines
                    44.100 Hz; 16 Bit; Stereo
```

```
Other options      :
  Fill up missing offset samples with silence : Yes
  Delete leading and trailing silent blocks : No
  Installed external ASPI interface
```

```
Track 1
```

```
  Filename G:\Cheap Trick\Cheap Trick 1981-01-22 Dallas, Tx(Reunion Arena)\
    1981-01-22d1t01 Stop This Game.wav
```

Hooray! This appears to be a live concert by the band Cheap Trick from January 22, 1981, recorded in Dallas. Let's verify that one of the files reported as data actually contains consistent data that is in the lossless Shorten (SHN)

format, as listed in the playlist file. We can do this using the `shninfo` command, which is part of the Linux Shorten command suite:

```
# shninfo *11993089
-----
---
file name:                #11993089
handled by:               shn format module
length:                   8:19.10
WAVE format:              0x0001 (Microsoft PCM)
channels:                  2
bits/sample:              16
samples/sec:              44100
average bytes/sec:        176400
rate (calculated):        176400
block align:              4
header size:              44 bytes
data size:                88047120 bytes
chunk size:               88047156 bytes
total size (chunk size + 8): 88047164 bytes
actual file size:         48873341 (compressed)
compression ratio:        0.5551
CD-quality properties:
  CD quality:              yes
  cut on sector boundary:  yes
  long enough to be burned: yes
WAVE properties:
  non-canonical header:    no
  extra RIFF chunks:       no
Possible problems:
  inconsistent header:     no
  file probably truncated: n/a
  junk appended to file:   n/a
Extra shn-specific info:
  seekable:                no
```

Another success! Unfortunately, there's no way to verify which of the recovered files is which of the files listed in the playlist, but let's see if we got all of the Shorten files that were in the original directory. We can do this in a number of ways, but the easiest is to count the number of lines in the playlist file and compare this number against the number of files in the recovered directory:

```
$ wc -l *101
18 #11993101
$
$ ls -l | wc -l
14
```

Unfortunately, this shows that the playlist file contains 18 entries, while there are only 14 files in the recovered directory, 3 of which are text files and 1 of which is a JPEG file. This means that we only recovered 10 of the files containing music in the original directory: the others apparently were

located on disk blocks that had gone bad on the original disk or were otherwise inconsistent. Oh well—10 is definitely better than 0!

To complete the recovery process for this directory, I would rename the directory with something more meaningful than its creation date (perhaps *cheaptrick1981-22-01_dallas*) and then play the Shorten files one by one, renaming them once I recognized them.

Examining Individual Files

The end of the listing of our *lost+found* directory at the beginning of this hack showed one file, `#3063821`, that was not accompanied by files with similar inode numbers or timestamps. This means either that the file is the only one that could be recovered from a damaged directory, or that the file was located at the top level of the recovered filesystem but could not be re-linked into the filesystem correctly.

Examining individual files in a *lost+found* directory is similar to examining a group of files. First, use the `file` command to try to figure out the type of data contained in the file, as in the following example:

```
# file \#3063821
#3063821: FLAC audio bitstream data, 16 bit, stereo, 44.1 kHz, 11665332
samples
```

Depending on the type of data contained in the file, you can use utilities associated with that file type to attempt to get more information about its contents. For text files, you can simply use utilities such as *cat* or *more*. For binary files in a nonspecific format, you can either make an educated guess based on the type of files that you know were stored on the filesystem, or you can use generic utilities such as *strings* to search for text strings in the binary file that may give you a clue to its identity. In this case, the file is a lossless FLAC audio file, so we can use the `metaflac` command's `-list` and `-block-number` options to examine the comments in the FLAC header that are stored in block number 2, and see if we can get any useful information:

```
# metaflac --list \#3063821 -block-number=2
METADATA block #2
  type: 4 (VORBIS_COMMENT)
  is last: false
  length: 254
  vendor string: reference libFLAC 1.1.0 20030126
  comments: 8
  comment[0]: REPLAYGAIN_TRACK_PEAK=0.64492798
  comment[1]: REPLAYGAIN_TRACK_GAIN=-5.84 dB
  comment[2]: REPLAYGAIN_ALBUM_PEAK=0.98718262
  comment[3]: REPLAYGAIN_ALBUM_GAIN=-4.77 dB
  comment[4]: ALBUM=Old Waldorf SF
  comment[5]: ARTIST=Pere Ubu
```

```
comment[6]: DATE=79  
comment[7]: GENRE=Avantgarde
```

I am indeed lucky! The creator of this file was thoughtful enough to include comments, which identify this file as a recording by Pere Ubu, created in 1979 at the Old Waldorf in San Francisco. Unfortunately, the title isn't listed, but I can now play the file using *flac123* in the hopes of identifying it so that I can copy it to the */usr/restore* area with a meaningful filename.

Summary

The examples provided in this hack show a variety ways of examining and reorganizing files that were saved by the *e2fsck* program in a filesystem's *lost+found* directory. I was quite lucky in these examples (modulo the fact that I had filesystem consistency problems in the first place), since the disk that had problems contained a large number of sets of files that were for the most part organized in a specific way. However, you can use these same techniques to examine the contents of any *lost+found* directory—and even if you've lost many files and directories, remember that recovering anything is always much better than losing everything.

See Also

- “Recover Lost Partitions” [Hack #93]
- “Recover Data from Crashed Disks” [Hack #94]
- “Recover Deleted Files” [Hack #97]