

RECOVERING LOST ext2 LINUX FILESYSTEMS

Steve Friedl

WARNING - The procedures described here can be very dangerous if applied to the wrong drive, and it's remarkably easy to do this. Don't do this on a live machine with un-backed-up data unless you are really sure of yourself. Please be careful.

Recently we had the unpleasant experience of having to recover data from a crashed hard drive, and we're documenting our complete success here to hopefully help others in the same boat. It was a very, very long process that was made worse by the irony of the failure's occurrence just as we were bringing the machine down to install a tape drive. Grrrr.

Table of Contents

- [Make a snapshot of the failing drive](#)
- [Trying to locate the partitions](#)
- [Repartitioning](#)
- [Scandrive Reference](#)
- [Suggestions](#)
- [Stuff to do](#)
- [Resources](#)
- [Other Resources](#)
- [Download](#)

We ran some DOS-based low-level diagnostics ([TuffTEST-Pro](#)) that showed only about two dozen blocks were physically bad, so this suggested that we'd be able to recover at least some of the data. But the partition table was hit, so this made it quite a bit more interesting.

These notes were written to reflect work done on Red Hat Linux 6.0 on the Intel platform (the 2.2.5 kernel). This is quite old, but this particular version was required for an embedded development project that this machine was used for. All drives were IDE.

These notes are also not terribly complete. Most of them were written while drives were being copied or scanned or otherwise crunching: once we actually recovered all our data, our interest in writing about this saga dropped rapidly in favor of getting back to actual work. Sorry if it's a bit unpolished.

Make a snapshot of the failing drive

The first step was to make a physical copy of the drive so we'd have something to work with in case the failed drive took a complete dump. We were quite sure this would happen eventually, and we'd like to have as much data onto good media as long as the Gods were smiling. We had a larger drive lying around so installed it into the same system.

IDE drives in Linux are addressed this way:

- **/dev/hda** - primary IDE controller, master device
- **/dev/hdb** - primary IDE controller, slave device
- **/dev/hdc** - secondary IDE controller, master device
- **/dev/hdd** - secondary IDE controller, slave device

In our case, the failed drive was a 20G unit at **/dev/hdb**, and the spare was a much larger 60G unit as **/dev/hdd**. Note that we're using the "direct" device: **/dev/hdb**, which refers to the *entire drive* and

RECOVERING LOST ext2 LINUX FILESYSTEMS

Steve Friedl

ignores the (bogus) partition table. Since we had no valid partition table on the failed drive, there was no such thing as `/dev/hdb1` or `/dev/hdb2` or the like.

Note: SCSI devices are often found as `/dev/sda`, `/dev/sdb`, etc.

Copying a drive is usually done with the `dd` command, but a few non-obvious options are required in order for this:

```
# dd if=/dev/hdb of=/dev/hdd bs=512 conv=noerror,sync
```

Here we have:

- **if=/dev/hdb** - Input File is the failing hard drive
- **of=/dev/hdd** - Output File is the spare hard drive
- **bs=512** - Block Size = 512 bytes
- **conv=noerror,sync** - don't stop on error, pad each input block to **bs**

The final **conv** options are crucial: we can't stop copying when we get an (inevitable) error, and when a bad spot occurs, it must be filled in with zeros instead of just dropping it. Dropping bad blocks during the copy is **guaranteed** to make the filesystem unrecoverable because these dropped blocks will mess up the sector numbers of all the remaining data.

Using a small blocksize makes the copy go much more slowly, but we think that this minimizes the data that gets filled in with zeros on an error: if we used 64k blocksizes, we're afraid of one bad spot zeroing out most of the large block. We really need to research this.

If the bad spots are all known to be early in the drive, it's possible to do a small blocksize copy for just that area and run a second `dd` command with larger blocksizes later, but we weren't in enough of a rush to warrant that much trouble.

But a custom C program holds some promise too. Not only could it manage buffering properly to optimize speed and minimize data loss, but it could also save the list of bad blocks to a file and report on progress (percentage completion would be really helpful). Writing this kind of utility is on our list...

Trying to locate the partitions

We were ecstatic when we found a link to [gpart](#), a program that scans through a hard drive and attempts to recreate the partition table by looking for clues left behind. We knew that we had an MS-DOS and a Windows partition on the drive, but we didn't care about them at all - all we needed were the Linux filesystems in the extended partitions.

Ultimately we never were able to recover anything with this tool, and we're not sure why. The drive had previously been repartitioned several times, so there were a **lot** of phantom partition blocks left over from prior runs, and the overwhelming amount of data made it hard for us to know what we were looking at, and we think our own inexperience with the ext2 filesystem contributed to this. Looking back we see that the partition we ultimately found **was** pointed out to us by **gpart**.

After looking for several other tools, we decided to do it ourselves. We were pretty sure that the data was still on the drive, and this was as good an opportunity to really dig into filesystem recovery as any. We've done this before with the older (and simpler) System V filesystems, but never with BSD or ext2. Now we know.

RECOVERING LOST ext2 LINUX FILESYSTEMS

Steve Friedl

But one point we'll note: since the geometry of the spare drive and the failed drive are probably not the same, it's important to not rely on anything that refers to a "cylinder boundary". Only if the drives are identical can you take these optimizations.

For very large drives, the simple act of scanning them can take **hours**. So we decided to split up the process into "scanning" and "analysis": one simple program optimized for speed would run through the drive **one time** and create a list of "interesting" sectors, and a later program would take this as input for analysis. When it takes more than two hours to scan a large drive, it's simply too much work to keep fine-tuning this process over and over.

Instead, our "scandrive" program makes one very fast run through the drive and notes everything with a broad definition of "interesting". The idea is that we cast a wide net with a one-time scan, and use later tools to process all the blocks located: this allows us to refine our tools without having to re-scan many gigabytes of data. It also lends itself to using perl for later parts of the process.

Scandrive knows about three kinds of "interesting" items: possible partition tables, possible ext2 superblocks, and possible root directory entries. Each of these is reported with type and sector number, and though we expect to get some mistaken entries in the list, later software can process them in more detail without having to scan the whole drive.

We ran scandrive on the whole drive - it took more than two hours on the 20GB part that was a mirror of the failed drive - and the result was a logfile of several thousand lines. In this process we have found that the partition table entries weren't that helpful, so we've ignored them here.

Winnowing the wheat from the chaff can be a bit trying: not only do we have the routine false positives where random data are seen as something interesting, but any *previous* formatting of the drive can leave remnants behind in the otherwise empty space. This leads to a lot of "clutter" in the output. Much of it could be reduced by a bit of postprocessing, but we have found that just by "eyeballing" the data we can locate the data that forms the most likely suspects.

All our ext2 filesystems use 1k "blocks" (two "sectors"), and the first super block is the **second** "block" on the partition. This means that sectors #0 & 1 are reserved for the OS, and sectors #2 & 3 are the first super block.

In each super block are lots of key parameters, but two of them are of particular interest: the filesystem size in blocks, and the block group number. Each super block inside a filesystem has the same "size" value, measured in (usually) 1 kbyte sizes. Seeing a string of candidate super blocks with the same filesystem size suggests that they all go together.

The second field of interest is the block group number: this is the sequence of super block within the filesystem, and they progress from #0, #1, ... up to the last super block in the filesystem.

If we see a super block #0 with a "sane" size, followed shortly by a root directory, then by a block #1 with the same size, this is a good clue that we have found the start of an ext2 partition. Not a complete clue, but a good one.

We can do a bit of automated processing on the logfile that will help dump some of the entries that are clearly bogus.

- ignore any **ext2** line with a negative size
- ignore any **ext2** line with a negative group number
- ignore any **ext2** line with a size that's greater than drive capacity

RECOVERING LOST ext2 LINUX FILESYSTEMS

Steve Friedl

- ignore any **ext2** line with group#0 and an offset+size that's beyond than drive capacity
- ignore any **ext2** line with a size that's smaller than our defined minimum

We've created a small perl program that follows most of these rules, and it produces an output file that we summarize here. We've highlighted the **#0** lines which draw most of our attention.

```
# Started scan of /dev/hdd at Thu Jan 31 09: 41: 34 2002
```

```
# buffer size = 1024 sectors  
# capacity = 40088160 sectors
```

```
many obviously bogus lines deleted
```

```
...
```

```
ext2      5189252      # size=32098 #0  
root      5189762  
ext2      5205636      # size=32098 #1  
ext2      5222020      # size=32098 #2  
ext2      5238404      # size=32098 #3  
  
ext2      5526617      # size=530113 #0  
root      5527139  
ext2      5543001      # size=530113 #1  
ext2      5559385      # size=530113 #2  
ext2      5575769      # size=530113 #3  
ext2      5592153      # size=530113 #4  
ext2      5608537      # size=530113 #5  
ext2      5624921      # size=530113 #6  
ext2      5641305      # size=530113 #7  
ext2      5657689      # size=530113 #8  
ext2      5674073      # size=530113 #9  
ext2      5690457      # size=530113 #10  
ext2      5706841      # size=530113 #11  
ext2      5723225      # size=530113 #12  
ext2      5739609      # size=530113 #13  
ext2      5755993      # size=530113 #14  
ext2      5772377      # size=530113 #15  
ext2      5788761      # size=530113 #16  
ext2      5805145      # size=530113 #17  
ext2      5821529      # size=530113 #18  
ext2      5837913      # size=530113 #19  
ext2      5854297      # size=530113 #20  
ext2      5870681      # size=530113 #21  
ext2      5887065      # size=530113 #22  
ext2      5903449      # size=530113 #23  
ext2      5919833      # size=530113 #24  
ext2      5936217      # size=530113 #25  
ext2      5952601      # size=530113 #26  
ext2      5968985      # size=530113 #27  
ext2      5985369      # size=530113 #28  
ext2      6001753      # size=530113 #29  
ext2      6018137      # size=530113 #30  
ext2      6034521      # size=530113 #31  
ext2      6050905      # size=530113 #32  
ext2      6067289      # size=530113 #33  
ext2      6083673      # size=530113 #34  
ext2      6100057      # size=530113 #35  
ext2      6116441      # size=530113 #36
```

RECOVERING LOST ext2 LINUX FILESYSTEMS

Steve Friedl

```
ext2 6132825 # si ze=530113 #37
ext2 6149209 # si ze=530113 #38
ext2 6165593 # si ze=530113 #39
ext2 6181977 # si ze=530113 #40
ext2 6198361 # si ze=530113 #41
ext2 6214745 # si ze=530113 #42
ext2 6231129 # si ze=530113 #43
ext2 6247513 # si ze=530113 #44
ext2 6263897 # si ze=530113 #45
ext2 6280281 # si ze=530113 #46
ext2 6296665 # si ze=530113 #47
ext2 6313049 # si ze=530113 #48

ext2 6313802 # si ze=16884283 #0
root 6314450
ext2 6329433 # si ze=530113 #49
ext2 6330186 # si ze=16884283 #1
ext2 6345817 # si ze=530113 #50
ext2 6346570 # si ze=16884283 #2
ext2 6362954 # si ze=16884283 #3
ext2 6379338 # si ze=16884283 #4
ext2 6394969 # si ze=530113 #53
ext2 6395722 # si ze=16884283 #5
ext2 6411353 # si ze=530113 #54
ext2 6412106 # si ze=16884283 #6
ext2 6428490 # si ze=16884283 #7
ext2 6444874 # si ze=16884283 #8
ext2 6461258 # si ze=16884283 #9
ext2 6477642 # si ze=16884283 #10
...
(many lines deleted)
```

The **#0** entries with **root** tags right after are highly likely candidates for partitions. The "size=" parameter is measured in "blocks", and for our purposes we simply assume 1kbyte blocks (though we really ought to compute it from the fields in the super block). Since each super blocks starts at sector #2, we subtract two from each starting offset to get the real partition starting point. From this we can also compute the ending block of the partition, which allows us to find (improper) overlap.

#	Line from scandrive	Partition start	Size (sectors)	Last sector
#1	ext2 5189252 # si ze=32098 #0	5189250	64196 (31.3MB)	5253445
#2	ext2 5526617 # si ze=530113 #0	5526615	6586843 (517.7M)	12113457
#3	ext2 6313802 # si ze=16884283 #0	6313800	33768566 (16.4G)	40082364

We see that partition #2 overlaps with partition #3 - this couldn't be valid - and since we remember having a very large root partition, we suspect that #2 is a phantom leftover from previous partitioning

RECOVERING LOST ext2 LINUX FILESYSTEMS

Steve Friedl

and that #3 is the real one. We are also pretty sure that #1 is the `/boot` partition, which is usually small and always contained within the first 4G of the drive.

Repertitioning

Once we have a handle on what partitions are what, we need to put the partition table back onto the drive. We're assuming here that the current partition table is bogus and must be recreated from scratch. There are several tools that can be used for partitioning under Linux, but we only use **sdisk**. This tool allows much more control over *exactly* where the partition boundaries lie, and we have used this tool for other projects well enough to be comfortable with it.

Always start by taking a snapshot of the current partitioning of the drive just for good measure: better to have it and not need it than to need it and not have it:

```
# sfdisk -d /dev/hdd > /tmp/partition.save
```

The `-d` parameter reads the table and dumps it to the standard output in a format that can be fed back to **sdisk** to put the table back.

Each partition has three parts: a starting sector, a size, and a type. It's important that they don't overlap (fortunately, **sdisk** will do a lot of checking for you). Using our constructed table from above, we are able to create the **sdisk** input.

HOWEVER - The partition table at the start of the drive only holds four entries, and if more than four partitions are required, they are done with "extended partitions". This substantially complicates the calculations due to the linked list of extra partition tables, and though we understand this process pretty well, it's much easier to just deal with up to four partitions at a time.

Remember that **all** **sdisk** parameters are measured in sectors, and if a super block is at sector n , then the partition starts at sector $n-2$. We also must provide a "type", which gives a hint as to the contents of the partition. **83** means "Linux native", and **0** means "unused". **sdisk --list-types** shows the whole list of types known to this program.

The constructed **sdisk** input file now looks like:

```
# partition table of /dev/hdd
unit: sectors

/dev/hdd1 : start= 5189250, size= 64196, id=83
/dev/hdd2 : start= 6313800, size= 33768566, id=83
/dev/hdd3 : start= 0, size= 0, id= 0
/dev/hdd4 : start= 0, size= 0, id= 0
```

The partition file created, we must now apply it to the drive. Fortunately, we're able to "test" the configuration before we actually "make it so".

```
# sfdisk -n /dev/hdd < /tmp/new.partition
```

The `-n` parameter says to go through the motions but to not actually write to the disk, and it will allow you check your work before writing to the drive. **sdisk** usually complains about things not being aligned on a cylinder boundary, but these can be ignored. Modern drives no longer really have fixed

RECOVERING LOST ext2 LINUX FILESYSTEMS

Steve Friedl

geometries (outer tracks have higher densities than inner tracks), so "cylinder/head/sector" is more an addressing scheme than a real physical geometry.

But partition overlap and those that extend beyond the size of the drive are to be taken seriously. Fix them in the partition file and try again. Once done, we can actually apply the changes. This is done with the **--force** command-line parameter, which insists that **sfdisk** do its work even if it doesn't care for what it sees. This **will** write to the drive, so making the mistake of mentioning the wrong drive can be very painful.

```
# sfdisk --force /dev/hdd < /tmp/new.partition
```

This produces a fair amount of information, causes the kernel to reread the drive's new partition table, and when finished the **/dev/hddX** devices are available to the user.

Our first step is to try to inspect each partition to see if it might be any good: this is what the **fsck** is for:

```
# fsck /dev/hdd1
# fsck /dev/hdd2
```

In the first go-round, we don't normally allow writes to the drive (we say "no" to all questions) while we get a feel for what kind of shape the filesystem is in. If the damage looks minimal, we may actually run it again and allow full repair, but serious damage is often a sign that the partitioning is not right.

To actually get to the data, we mount the filesystem readonly onto a common **/mnt/hdd** mount point. In our case we had only two partitions, so we could be reasonably sure which was which:

```
# mkdir /mnt/hdd
# mount -r /dev/hdd2 /mnt/hdd          <-- root partition
# mount -r /dev/hdd1 /mnt/hdd/boot    <-- boot partition
```

but in more advanced schemes, you might have to mount the partitions one at a time in order to look around. Once mounted, we suggest taking all the data off, either by backing up to tape or by copying the data to a "good" drive (not the spare used for recovery).

We do not trust the partitioning enough to actually go with this live. Instead, we copy the data elsewhere, repartition the spare drive via the "normal" routes, reformat the filesystems, and copy the data back.

Scandrive Reference

Our little scandrive program is meant to be more or less self-contained. It works exclusively on a readonly basis, so it won't ever hurt anything, but we think it's probably best to run it on a drive that's idle. In most circumstances, it has to be run as root.

Building the program is very easy:

```
# cc -o scandrive.cpp -o scandrive
```

RECOVERING LOST ext2 LINUX FILESYSTEMS

Steve Friedl

and it takes these command-line parameters:

-v *DEV*

(REQUIRED) *DEV* is the device to scan. Most of this time this is the full drive (*/dev/hdX*), but it can be an individual partition if this is appropriate for the current situation.

-B *size*

(OPTIONAL) *size* is the number of 512-byte sectors to use for I/O buffering. The default is probably fine.

-C *capacity*

(OPTIONAL) *capacity* is the size of the drive in 512-byte sectors. This is used to limit the range of the scan - we'll stop when we hit the end - as well as to report percent-done status. When scanning a really large drive it's nice to know how far along in the process it is. If omitted, the whole drive is scanned.

-L *FILE*

(OPTIONAL) Append summarized results into logfile *FILE*. This file has a format intended to be easy to parse and process, and it is a bit more terse than the standard output. The intent is that the standard output need not be captured if this option is used.

The output file created with the **-L** parameter is in straight ASCII, and it always *appends* to the file if found: scanning is too slow of an operation to risk blowing an important previous file away.

Blank lines and those starting with a **#** should be treated as comments and ignored, and the rest are lines with a "type" and the starting sector number, and possibly a comment after. A sample output file:

```
# Started scan of /dev/hda at Thu Jan 31 15: 50: 51 2002
# buffer size = 256 sectors
pt      0
ext2    65      # size=24066 #0
root    575
pt      4033
pt      5869
ext2    16449   # size=24066 #1
pt      48195
ext2    48260   # size=1496045 #0
root    52362
pt      69234
pt      69283
pt      72122
pt      72171
pt      75010
```

The types known are:

pt

This entry might be a Partition Table, but the bar is very low for this test: *any* sector with 55AA as the last two bytes will be reported this

RECOVERING LOST ext2 LINUX FILESYSTEMS

Steve Friedl

	way.
ext2	This sector might be an ext2 filesystem super block as recognized by the magic number being in the right place. Each filesystem can have more than one super block (they are spread out for redundancy), so we also report the size of the partition in "blocks" and the number of the super block group (#0, #1, ...).
root	This sector might contain an ext2 root directory as recognized by the "." and ".." directories at inode number 2. This should be found shortly after the first super block inside a partition, and it may help isolate where the real filesystem starts.

Suggestions

1. Make an offline copy of your partition tables and filesystem layouts to give you some vital help should things go south: this would have saved me probably two weeks of time. Save the output of these commands into an offline file or onto hardcopy:

```
2. # date
3. # uname -a
4. # sfdisk -d /dev/hda
5. # cat /etc/fstab
```

2. Create your partitions on cylinder boundaries. This is not easy to do with the standard tools, but with a bit of effort you should be able to make every single partition lie correctly on a cylinder boundary. We always use the **sfdisk** tool for this, and we've gotten even complex partitioning schemes working right. During system installation this might not be so easy.
3. Label your filesystems. The **mke2fs** command permits a **-L label** option that puts an up to 16 byte string in the super block, and this can help identify a filesystem that's being recovered later. After the fact the Linux **e2label** command can be used, but we're pretty sure that the filesystem has to be unmounted first.

Stuff to do

- Create a high-speed drive copy program. This should use optimal buffering, and perhaps even asynchronous I/O to do simultaneous reads and writes when split across IDE controllers. It might even be possible to talk to the IDE driver to have it drive the copy operation itself without needing the host CPU.
- Write tools to do better post-scandrive analysis of the output and make even better guesses. We made some attempts at this, but we had problems with drives larger than 4G because the **lseek()** operation can't go beyond a 32-bit offset. The **llseek()** function is supposed to allow 64-bit offsets, but we kept running into problems with it.

RECOVERING LOST ext2 LINUX FILESYSTEMS

Steve Friedl

- Each ext2 super block contains not only a group number, but the size of each group is included as well. We should be able to use this to calculate the starting offset of each partition plus how many overall groups, and this would allow us to exclude a *lot* of bogus super blocks.
- Be a little smarter about analyzing "partition tables", as these can also be Windows/NT "master boot blocks". This probably should be done in the post-processing phase.

Resources

In the source of this research we ran across quite a few resources of varying helpfulness to our situation. In no particular order:

- TuffTEST-Pro, MS-DOS level diagnostics that seem to do a very thorough job in testing all aspects of PC hardware, including the hard drive. We were able to confirm that only certain blocks were actually bad, and this gave us hope that we'd be able to recover our data.
- Understanding the Linux Kernel, O'Reilly. Chapter 17 is about the ext2 filesystem, and it was really helpful in understanding the filesystem layout in order to search for the start of ours.
- gpart, a Linux utility that attempts to guess a drive's partition table by scanning. This didn't really help us much, and we're not sure why. It think this tool only looks for the partition-table signatures on the drive (with the 55AA at the end), and it seems that in our case all of the extended partitions were damaged. This tool has a lot of promise for others.
- Analyzing an ext2 filesystem - this very helpful paper picks apart the ext2 filesystem format and was instrumental in our figuring out what to look for in scanning. Thank you, David Morgan.
- IBM's Drive diagnostics. The drive that failed happened to be an IBM drive, and the "Drive Fitness Test" (which runs on a bootable PC floppy) can analyze the drive and even reformat it. It was able to completely repair (by erasing) our failed drive. A superficial test shows that it works OK with non-IBM ATA drives, so this is encouraging for the general case.
- Ranish Partitioning Utilities includes software and a great Partitioning Primer. These tools probably aren't directly helpful while *recovering* a failed drive, but they would be great for *partitioning* a new one.

Other Resources

The information in this Tech Tip is quite dated, and we've simply not done any updates since this problem was in front of us. We have since learned of other resources that may aid in filesystem recovery. No endorsement: this is just a listing of what we've heard from others.

- Stellar Information Systems — commercial disk data recovery software; reportedly effective.
- dcfldd — an enhanced version of **dd** which provides (among other things) a progress display while it copies, otherwise taking mostly the same parameters. It's helpful to watch and get a handle on how long it will take to do a big copy.

We'll list others as we learn of them.

Download

The C++ language source for **scandrive** can be found here on our web site:

- scandrive.cpp

If using the GNU tools, compile it with **g++**, not **gcc** (so as to get the C++ linkages).

RECOVERING LOST ext2 LINUX FILESYSTEMS

Steve Friedl

This tool is in the public domain - have fun.