

Tales from the Abyss: UNIX File Recovery

Liam Widdowson and John Ferlito

It is every systems administrator's nightmare -- an important file has been accidentally deleted, falling into the deep abyss where bits and bytes go to die. Typically, this situation presents an inconvenience rather than a tragedy as the files can be recovered from regular backup media.

However, if a systems administrator or developer has significantly modified the file in question since the time of last backup then a large amount of data may be lost. There are also other situations that systems administrators may face, such as incorrectly configured backup scripts, backup hardware failure, or plain old bad luck. Nothing can replace a proper backup strategy, but this article will outline how whole or partial files can be restored directly from the UNIX file system.

The UNIX File System

Files in UNIX file systems are logical containers of data. Each file has an inode (index-node) structure associated with it that contains meta-data such as the physical disk blocks the file is stored on, the file owner, permissions, size, etc. [1]. When files are removed, the inodes are not erased from disk but are marked as free. The actual data contained in the files is still on disk and can potentially be retrieved before being over-written with new data.

A Tale from the Abyss

Some time ago, a graduate in my team was tasked with developing a Perl CGI-based provisioning and support interface to a product we were developing. Earlier on in the project I showed her how she could use UNIX pipes and redirection to debug her Perl script from a shell rather than exclusively through the Web server. Unfortunately, one day she accidentally typed the following:

```
$ ./nph-www.pl > nph-www.pl
```

That command resulted in the Perl code being turned into a zero-byte file. Unfortunately, she had not checked the file into CVS for a week, which meant around 600 lines of code had been lost. When she told me what had occurred, I explained that it wasn't a problem because we could retrieve the file from last night's backup and she would only have to re-create today's changes. I phoned our sysadmin and asked him to restore the file in question. However, I did not get the response I hoped for -- the sysadmin said he had never heard of the development server in question and so it was not being backed up. The graduate sighed and said that it was impossible to restore the deleted file from the UNIX file system, so she would spend the weekend re-coding the script. I told her that it was indeed possible, and that I'd have her file restored in a few hours. Note that this type of bad luck does not only occur to inexperienced administrators. A number of years ago, I accidentally typed `crontab -d` instead of `crontab -e` on a Linux system. How many readers back up `/var/spool/cron/crontabs`? UNIX File Recovery As soon as any files have been accidentally erased, all I/O activities on the relevant partition should cease. The partition should be unmounted, or the system should be placed into single-user mode immediately after the incident. If this is not possible (e.g., the root file system cannot be unmounted), then all logged-in users should cease work, and new logins should be disallowed (e.g., with an `/etc/nologin` file). It is also possible on some UNIX variants to remount the partition read-only. For example:

```
# mount -o ro,remount -n /home
```

The key is to prevent other processes from overwriting the disk blocks or inodes previously used by the erased file. This is most pertinent when the partition is almost full, because it is more likely that the deleted inodes will be reused. Alternatively, operations can be performed directly on the partition while mounted read/write but, as mentioned previously, this reduces the chance of recovery. The following sections outline how files can be restored from UNIX filesystems -- the first section describes how a file with known content can be restored from almost any UNIX variant's file system, while the second

Tales from the Abyss: UNIX File Recovery

Liam Widdowson and John Ferlito

section is specific to restoring files from the Linux ext2 filesystem. Known Content File Recovery Process Once the system is in a safe state, a raw copy of the partition containing the erased file should be made with the dd(1) command. Consider that the /etc/passwd file has been accidentally removed. The following example illustrates the procedure required to create a copy of the root partition and place it in a file on the /export partition:

```
# df -k / /export
Filesystem          kbytes    used   avail capacity  Mounted on
/dev/dsk/c0t3d0s0   123231    18413   92495    17%        /
/dev/dsk/c1t0d0s0   17498618 14327901 3170717   82%       /export
# dd if=/dev/dsk/c0t3d0s0 of=/export/recover.dsk
263088+0 records in
263088+0 records out
# ls -l
-rw-r--r--    1 root      other      134701056 Jul  1 16:54 recover.dsk
```

The target partition must have sufficient space to hold the entire slice (in this case ~128 MB). If sufficient space does not exist on the system in question, then alternative measures, such as NFS mounting a remote partition or performing operations directly on the partition device file, can be considered. Once a copy of the file system has been made, the system can be brought back into multi-user mode, and normal system operation can continue while the recovery progresses. If a copy of the partition cannot be made, leave the system in a safe state and simply substitute the device name (e.g., /dev/dsk/c0t3d0s0) for the filename (i.e., /export/recover.dsk) as in examples below. The procedure that follows is more art than science. Consider the example of the erased passwd file. The left-hand side of the following command prints each “line” of the disk with cat(1) (the -n option prints the line number). The output is piped to fgrep(1), which performs a “fast” regular expression search for the “root” entry in the passwd file:

```
# cat -n recover.dsk | fgrep "root:x:0:1"
200600 root:x:0:1:Super-User:/:/sbin/sh
202098 root:x:0:1:Super-User:/:/sbin/sh
332802 Ël root:x:0:1:Super-User:/:/sbin/sh
```

fgrep may not match the supplied pattern. This probably means that the file contents have been overwritten and that recovery is not possible. Otherwise, the pattern can be further generalized in the hope of increasing the possibility of a match. In the above example, there are three places on the disk where versions of /etc/passwd file have been stored. The GNU version of grep(1) provides the -A and -B options that allow a number of lines before/after a particular match to be displayed. This assists in retrieving the entire contents of the erased file. If GNU grep(1) is not installed on your system, the C program (seekcat) in Listing 1 is provided to print out the entire file from a particular byte or line offset. The seekcat program takes two arguments: the first is either a -b or -l flag followed by an integer that specifies a byte or line offset where the listing should commence. The second argument is a -f flag followed by the filename, which in this case is the raw disk image. For example:

```
# fgrep -A 10 "root:x:0:1" recover.dsk > passwd
# cat passwd
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1:/:
bin:x:2:2:/:usr/bin:
...
```

The above functionality can be emulated by providing the starting line number to seekcat. The line number argument is derived from the numbers in the “cat -n” output described earlier in this section.

Tales from the Abyss: UNIX File Recovery

Liam Widdowson and John Ferlito

Essentially, it instructs seekcat to begin printing the file from where a particular match was found. In the following example, ten lines of the raw disk image will be printed from the first match (line 200,600) onwards. If this did not yield suitable output, then the next match (line 202,098) would be specified:

```
# seekcat -l 200600 recover.dsk | head -10 > passwd
# cat passwd
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1:/:
bin:x:2:2:/:usr/bin:
...
```

Unfortunately, parts of a file sometimes may be scattered in a non-contiguous manner over the entire partition. In that case, the above procedure should be repeated with suitable match patterns to obtain the entire contents of the file. In most situations (including the erased Perl script), the entire file tends to be stored in a contiguous fashion that facilitates easy restoration. The most difficult part is determining which version stored on disk is the latest file. Unfortunately, this can only be determined by visual inspection of the content. Fortunately, CVS or RCS headers that include version numbers, dates, and other well-known contents can aid this process. Linux File Recovery Process The following recovery process covers the case where files have been deleted using the rm(1) command on a Linux system. In most cases, this method will result in a perfectly recovered file and does not rely on trial and error. The inherent advantage of this method is that binary files or files with unknown content can be recovered with ease. The tool used during this process is known as the file system debugger, debugfs(8), which is used to examine and change the state of a file system. The examples below are specific to Linux but should in theory be possible on any UNIX-based file system given a tool with the functionality provided by debugfs(8). If such a tool is not available then the known content method of recovery should be used. At this point, it is worth mentioning that debugfs is a powerful but extremely dangerous tool. It provides raw access to the file system so care must be taken throughout its use. debugfs(8) provides a shell-like interface that has three commands that are of interest -- lsdel, cat, and dump. debugfs(8) can be run on the required partition as follows:

```
# debugfs /dev/hda6
debugfs 1.19, 13-Jul-2000 for EXT2 FS 0.5b, 95/08/09
debugfs:
```

At the prompt, lsdel will give a list of all the deleted inodes on the filesystem. This will take some time as all the inodes on the partition need to be scanned:

```
debugfs: lsdel
1844 deleted inodes found.
Inode  Owner  Mode   Size   Blocks   Time deleted
749300  1000 100664 27018  2/ 7 Tue May 9 19:08:17 2000
749301  1000 100444 1671   1/ 1 Tue May 9 19:08:17 2000
.....
944887  1037 100600 597    1/ 1 Sat May 26 18:00:00 2001
717281  1000 100400 1      1/ 1 Sat May 26 18:08:13 2001
 32605  1000 100644 15     1/ 1 Sat May 26 18:09:06 2001
```

Because the list is typically large, it may be useful to redirect the output to a file that may later be examined with an editor or pager as follows:

```
# echo lsdel | debugfs /dev/hda6 > /tmp/lsdel-output
```

Tales from the Abyss: UNIX File Recovery

Liam Widdowson and John Ferlito

From the information given in the lsdell listing, it should be possible to narrow down which deleted inodes are relevant. The most useful fields in the output are -- Owner, Size, Mode, and Time deleted. If no further disk operations occurred after deleting the file(s) then the required inodes should be those at the end of the list. The owner field is the uid of the owner of the file, which can be found in the third field of /etc/passwd. In this example, the file was owned by the user johnf, uid 1000, and contained the string "important_data". From the listing above, inode 32605 is the last entry and thus a good candidate for initial investigation. The contents of the inode can be listed using the following command:

```
debugfs: cat <32605>
important_data
```

The deleted file has been found. The dump command can now be used to write the file out to disk:

```
debugfs: dump -p <32605> /tmp/recovered_file
```

The -p ensures that the same owner, group, and permissions are retained for the file. This approach is sufficient if a single file has been deleted but can be tedious if a large number of files must be restored. Fortunately, Tom Pycke has written a utility called recover [2] that automates file recovery.

The installation of recover is fairly straightforward:

```
# tar xzf recover-1.3.tar.gz
# cd recover-1.3
# make
# make install
```

Recover is installed into the /usr hierarchy by default. Please read the README file for instructions on installing into a different directory hierarchy. Recover asks a number of simple questions such as:

Who is the owner of the files?
When were the files deleted?
What is the approximate size of the files?

Using this information, it executes debugfs to recover the inodes that match the given criteria and places them in a user-specified directory. Unfortunately, it is not possible to recover filenames in addition to content and results in a directory full of files named dumpinode-number. If a directory such as /etc was actually removed, this could amount to hundreds of files. At this stage, simple UNIX utilities can be used to sort through the recovered files and the two most useful tools are strings(1) and file(1). strings(1) displays sequences of ASCII characters for a given input file and is useful for extracting text from an otherwise undecipherable binary file. file(1) is able to determine the type of a file (i.e., whether it is a JPEG image or a postscript file). It does this by performing a set of magic number tests to uniquely determine the file type. For example, the output of file(1) in a directory filled with recovered files is as follows:

```
# file *
dump34578:directory
dump98568:PGP armored text signed message
dump896545:gzip compressed data, deflated, last modified: Sun Jan 28 03:31:21 2001,
os: Unix
dump78890: ASCII text dump67245: 'diff' output text
dump76345:JPEG file dump9723: MPEG 1.0 layer 3 audio stream data,128 kBit/s
dump8976: ASCII C program text
```

Tales from the Abyss: UNIX File Recovery

Liam Widdowson and John Ferlito

```
dump57654: Bourne shell script text executable
dump3463: troff or preprocessor input text
dump56789: ELF 32-bit LSB executable, Intel 80386, version 1, dynamically linked
(uses shared libs), stripped
dump9876: ELF 32-bit LSB shared object, Intel 80386, version 1, stripped
```

At this point a simple shell script to sort files of different types by adding file extensions would be useful. For example:

```
# for i in `file * | grep "ASCII C program text" | awk -F: '{print $1}'`;
do mv $i $i.c; done
```

Once sorted, it is time to begin the process of trying to establish the identity of each file. For text, C, image, and audio files this is a fairly straightforward process as each file can be visually inspected with the appropriate tools and the original file name can be guessed. However, binary files such as executables, libraries, and database files are more difficult to inspect. In this situation, the strings(1) utility can be used to print out any ASCII text strings in a binary file. For example:

```
# strings dump45678.bin
/lib/ld-linux.so.2
__gmon_start__
libstdc++-libc6.2-2.so.3
_DYNAMIC
__rtti_user
....
/build/build/groff-1.16/src/include/stringclass.h
groff.cc
GROFF_COMMAND_PREFIX
troff
....
```

From the above output, a guess can be made that the file is the groff(1) executable. Executing the file with a --help argument confirms this. Libraries are a little more difficult because they cannot be executed, however, the objdump(1) command can provide suitable assistance. For example:

```
# objdump -p dump34756.lib | grep SONAME
SONAME      libmenu.so.4
```

The aforementioned method is only useful if the file has been deleted through use of a utility such as rm(1) or the unlink(2) function. If the file has been overwritten, then the inodes that originally contained the data have probably been overwritten. It is still possible that some fragments of the data are held elsewhere on disk. For example, many editors make temporary copies of files that later get deleted, or perhaps not all the inodes in the file have been overwritten. If the inodes are lost, then the known content recovery method already described will be more applicable.

Conclusion

No file recovery method can take the place of thorough, regular, and reliable system backups. The processes described above are designed to provide last-ditch assistance to systems administrators perched on the edge of an abyss.

References

1. Vahalia, U. 1996, UNIX Internals -- The New Frontiers. Prentice-Hall, Upper Saddle River, NJ
2. Pycke, T., 2001, Recover. <http://recover.sourceforge.net/linux/recover/>