

When Disaster Strikes: Attack of the rm Command

by Kyle Rankin

(Reprinted From The Linux Journal)

Some commands on the command line are so blunt, so potentially devastating, that every time I use them, I pause for a moment before I press Enter. In my last column, I discussed one of my all-time favorites: dd (which could possibly stand for Destroy Data). Of course, as useful as dd is, I don't use it every single day, so even though I approach the command with reverence, you might argue it doesn't compare to the true master of data destruction: rm. Yes, dd can wipe out your hard drive in a few short keystrokes, but nothing really matches the compact destructive power of `rm -rf /`.

True, most people aren't bitten by that version of the command. Usually, it's its more sinister brother, `rm -rf ./` run from the wrong directory. The scene plays out something like this:

```
rm -rf ./
```

Clicking noises from the hard drive... "Hmm, that's taking longer than I tho...HEY!" Ctrl-C Ctrl-C Ctrl-C. It's too late. By the time you noticed you ran that command in the wrong terminal, half of your home directory is gone. Now when I started out with Linux, I always was told in true UNIX form that when you `rm` a file, it is gone, and there is no way you can get it back. Undelete commands were for DOS users anyway—we Linux users knew better, right? Well, it turns out, we don't. Most Linux users I know have deleted the wrong files at least once in their lives. Now, the best protection against this is a backup (noticing a common thread in this series?), but if you don't have a backup, you aren't completely without hope. Everything you might have been told about the `rm` command isn't entirely true, and by the end of this article, you'll find that Linux does have an undelete of sorts.

Free Space Isn't Free

To understand how to recover a deleted file, it's important to understand what `rm` does. When `rm` deletes a file, it essentially adds those blocks to the available free space on that filesystem. Unless you use a tool like `shred`, the data in those blocks stays intact until another file overwrites them. Blocks aren't reused in any date order, so some freed blocks might stay on the system for days, weeks or even years before they are reallocated to a new file, while others could be reused almost immediately.

Because a Linux system writes files constantly, time is against you when you accidentally delete a file. The first thing you should do if you delete important files is unmount that filesystem. If you can't easily unmount the filesystem, shut down the system. Or, if the files are extra important, you might even pull the plug to ensure no other files are written to disk.

Forensics to the Rescue

It turns out that accident-prone Linux users aren't the only ones who want to recover deleted files. In fact, deleted file recovery is particularly useful for forensics, as attackers might try to delete files to cover their tracks. Forensics tools work with the filesystem on a low level as it is, because they try to gather data traditional tools might miss.

To recover deleted files, you need to install `sleuthkit`. Most distributions these days offer it as a package; otherwise, you can download the source from the project's Web site. It may go without saying, but don't install `sleuthkit` on the filesystem you are recovering! If you need to recover files from the root filesystem, this may mean you have to take the hard drive to a second system or use a rescue disk like Knoppix that includes `sleuthkit`.

Once you have `sleuthkit` installed, you need to get a second disk that is large enough to store any files you want to recover. Unlike some other recovery methods, with `sleuthkit`, you don't have to create a

When Disaster Strikes: Attack of the rm Command

by Kyle Rankin

(Reprinted From The Linux Journal)

complete image of the free space, so you won't need nearly as much storage. You can use the df tool to see how much free space you have:

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       9.4G  7.0G  2.0G  79% /
/dev/sda3       20G  17G  3.6G  83% /home
```

In this case, I have around 2GB of space on my / partition and 3.6GB in /home to which to restore files. For this example, let's assume I have connected the recovery filesystem to this machine, and it has shown up as /dev/sda1. Be sure not to mount this filesystem. Or, if your machine automatically mounted it, be sure to unmount it before you continue, so you won't write to it accidentally. Because /home has more free space, I will recover to it, so I create a directory to store the recovered files and then use the sleuthkit fls (forensic ls) command to create a list of all the deleted files it can find on /dev/sda1:

```
$ mkdir ~/recovery
$ sudo fls -f ext -d -r -p /dev/sdb1 > ~/recovery/deleted_files.txt
```

This command might take some time, depending on how much free space it has to pore through. In the meantime, we can discuss what these different arguments mean. The fls man page goes into more detail, but the -f argument specifies what filesystem fls is scanning (ext is used for ext2 and ext3). If you are unsure what value to use, type fls -f list to see a complete list of filesystems. By default, fls can list all the files on a particular filesystem, but when you specify -d, it lists only deleted ones. The -r option turns on recursion, so it traverses all directories it finds, and the -p option outputs the full path to each file. Without -p, if multiple files have the same name, it might be difficult to tell them apart. Finally, you list the partition you want fls to scan.

Once fls completes, you can open ~/recovery/deleted_files.txt to see a complete list of all the deleted files on the filesystem. It will look something like the following:

```
d/d * 944680:  home/kyle/.mutt
r/r * 943542:  home/kyle/.muttrc
r/r * 910452:  home/kyle/may_lj_article.txt
```

The first field tells you whether the file is a directory (d/d) or a regular file (r/r). Next is an inode number for the file, and then finally, you see the path to the file. Let's say, for this example, I want to recover the /home/kyle/may_lj_article.txt file. I then would use the sleuthkit icat tool to recover it. The icat program is a special version of cat that takes inodes as arguments. In this case, I would specify the inode 910452:

```
$ sudo icat -f ext -r -s /dev/sdb1 910452 >~/recovery/may_lj_article.txt
```

As with fls, this might take some time to complete. You can read about all of its arguments in the icat man page, but here I use -f to specify the filesystem type like with fls. The -r option tells icat to go into a special recovery mode it uses for deleted files. The -s option causes icat to output the full contents of any sparse files it finds. Finally, I specify the partition to recover from and the inode to recover. Once the command completes, I can open ~/recovery/may_lj_article.txt and see whether it was able to restore it.

When Disaster Strikes: Attack of the rm Command

by Kyle Rankin

(Reprinted From The Linux Journal)

This method works fine when you need to recover only a few files, but what if you need to recover hundreds? Well, if you search on-line, you will find a number of different shell scripts people have written to recover all deleted files from fls output automatically. Below is one I originally found at forums.gentoo.org/viewtopic-t-365703.html and then improved a bit:

```
#!/bin/bash

DISK=/dev/sdb1 # disk to scan
RESTOREDIR=/home/kyle/recovery # directory to restore to

mkdir -p "$RESTOREDIR"
cat $1 |
  while read line; do
    filetype=`echo "$line" | awk {'print $1'}`
    filenode=`echo "$line" | awk {'print $3'}`
    filenode=${filenode%:}
    filenode=${filenode% *}
    filename=`echo "$line" | cut -f 2`

    echo "$filename"

    if [ $filetype == "d/d" ]; then
      mkdir -p "$RESTOREDIR/$filename"
    else
      mkdir -p "$RESTOREDIR/`dirname $filename`"
      icat -f ext -r -s "$DISK" "$filenode" \
        > "$RESTOREDIR/$filename"
    fi
  done
```

Save this script under `/usr/local/bin/restore`. To use this script, replace the `DISK` and `RESTOREDIR` variables at the top of the script so they match your environment, give it executable permissions, and then run it with the `fls` output you created before as an argument. All of your recovered files will be wherever you set `RESTOREDIR` nested within their parent directories:

```
$ sudo chmod a+x /usr/local/bin/restore
$ sudo /usr/local/bin/restore ~/recovery/deleted_files.txt
```

Now, don't let this make you too comfortable with `rm`—there's no guarantee a particular file will be complete or even recovered at all. I still say the best policy is to have backups followed by a thoughtful pause before you press Enter on any recursive `rm` command.