How not to delete a file:

2023-05-11 — 2026-04-26

So, you have a file and you don't want it anymore.

Everyone knows that moving it to the trash won't delete it, but how about the "permanent" delete button?

> For the record, I did this on Linux 6.19.13 with ext4 formatted disks, but the same holds for most any modern system.

```
# cat > test_file <<EOF
This is some file data.
EOF
# sync
```

Deleting a file only removes the (filesystem) entry telling the computer where it is: it doesn't actually remove the data from the disk.

In real forensics, "deleted" files are recovered by searching the whole disk for anything that looks like data, but this can take a while.

For a demonstration, it's easier to grab the location beforehand:

```
# hdparm --fibmap test_file
test_file:
 filesystem blocksize 4096, begins at LBA 0; assuming 512 byte sectors.
 byte_offset  begin_LBA    end_LBA    sectors
           0  241046952  241046959          8
```

In most cases, a sector is 512 bytes, so it's easy enough to read the data directly:

```
# dd if=/dev/nvme0n1p4 skip=241046952 bs=512 count=9
This is some file data.
8+0 records in
8+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 0.000114052 s, 35.9 MB/s
```

... and now it's time to delete it:

```
# rm test_file
# sync
```

The file is gone from ls, but the data is still there:

```
This is some file data.
8+0 records in
8+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 0.000114052 s, 35.9 MB/s
```

This isn't too unexpected: it's fairly common knowlage that a normal delete isn't quite good enough.

Let's try shred(1): "Overwrite[s] the specified FILE(s) repeatedly, in order to make it harder for even very expensive hardware probing to recover the data".

Ok then:

```
# shred test_file2
# sync
```

Reading the file now returns random data, but...

```
This is some file data.
8+0 records in
8+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 0.000114052 s, 35.9 MB/s
```

Wait what?

The random data from shed got written to a different part of the drive then the original file.

Ok, so the file system clearly can't be trusted: What about using dd to wipe the data directly?

```
# (note, messing this up can ruin your filesystem!)

sudo dd if=/dev/zero of=[DEVICENAME] skip=[START SECTOR] bs=512 count=[SECTOR COU
```

As we've already seen, the filesystem constantly moves files around. Running hdparm and dd will remove the current version of a file, but old copies will still be floating around.

A common "trick" is to create a very big file to fill up all the free disk space.

This should overwrite the file right?

```
dd if=/dev/urandom of=spacefilling_file
sync
rm spacefilling_file
```

Nope:

```
This is some file data.
8+0 records in
8+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 0.000114052 s, 35.9 MB/s
```

While most of the free drive space is filled, a small amount was untouched, which included my file! If the file was very large, only a few fragments would remain, but that's far still from ideal.

Ok, how about nuking the whole drive?

```
sudo dd if=/dev/urandom of=[DEVICENAME]
```

Surely the file's gone right?

Perhaps...

A funny thing about SSDs is that even though flash memory is manufactured in power-of-two sizes, most drive sizes are not: My "1 TB" disc is 953.9 GB instead of 1024 GB.

What are they doing with my 70 GB???

This "spare" space is used to provide redundancy case a block fails, and for Wear leveling: the disc's controller moves frequently accessed blocks around the physical flash memory to even out writes.

This increases the device's lifetime, but does mean that there is some data stored in places that the OS cannot touch.

Even though this space is normally unreachable, hardware level recovery can get to it. If your adversary can desolder a BGA chip, wiping the drive isn't good enough.

There are two ways to ensure data is truly gone:

1. Using the disc's "secure erase" command: This will remove all data from the drive, including spare or failed/flaky blocks (where possible).

2. Never write sensitive data to it in the first place. Set up full-disc encryption when installing the OS, and forget the password: without it, the data is just a few terabytes of random numbers.
