

# When Disaster Strikes: Restoring a Master Boot Record

By Kyle Rankin

(Reprinted from the Linux Journal)

I have to admit, I've learned more about how Linux works by breaking it and fixing it, than I have by any other method. There really is nothing quite like the prospect of losing valuable data, or the idea that your only computer won't boot, to motivate you to learn more about your system. In this month's installment of "When Disaster Strikes", I discuss a surprisingly small part of your computer that plays a surprisingly large role in booting and using it—the Master Boot Record, or MBR for short. I cover some of my favorite ways to destroy an MBR and a few ways to restore it once you have.

Before you can fully understand how to restore the MBR, you should have a good idea of what it actually is. The MBR comprises the first 512 bytes of a hard drive. Now that's *bytes*, not megabytes or even kilobytes. In our terabyte age, it's hard to appreciate how very small that is, but to give you an idea, at this point in the column, I've already written about three MBRs worth of text.

This 512-byte space then is split up into two smaller sections. The first 446 bytes of the MBR contain the boot code—code like the first stage of GRUB that allows you to load an operating system. The final 66 bytes contain a 64-byte partition table and a 2-byte signature at the very end. That partition table is full of information about the primary and extended partitions on a disk, such as at which cylinder they start, at which cylinder they end, what type of partition they are and other useful data you typically don't think much about after a disk is set up—at least, until it's gone.

## A Routine Lecture on Backups

This is the part of the column where I repeat some of the best disaster recovery advice I know—make backups. In this case, we are talking about MBR disasters, so here are a few ways to back up your MBR. After all, it's only 512 bytes; there's no reason why you can't afford to back it up. Heck, it's small enough to tattoo on your arm, except I guarantee once you do you'll end up migrating to a new system or changing the partition layout.

The best tool to back up the MBR is coincidentally the best tool at destroying it (more on that later), `dd`. In fact, `dd` is one of those ancient, powerful and blunt UNIX tools that blindly does whatever you tell it to, and it's adept at destroying all sorts of valuable data (more precisely, it's adept at following your explicit orders to destroy your valuable data). The following command backs up the MBR on the `/dev/sda` disk to a file named `mbr_backup`:

```
$ sudo dd if=/dev/sda of=mbr_backup bs=512 count=1
```

Basically, this tells `dd` to read from `/dev/sda` 512 bytes at a time and output the result into `mbr_backup`, but to do only one 512-byte read. Now you can copy `mbr_backup` to another system or print it out and do the tattoo thing I mentioned before. Later on, if you were to wipe out your MBR, you could restore it (likely from some sort of rescue disk) with a slight twist on the above command. Simply swap the input and output sources:

```
$ sudo dd if=mbr_backup of=/dev/sda bs=512 count=1
```

# When Disaster Strikes: Restoring a Master Boot Record

By Kyle Rankin

(Reprinted from the Linux Journal)

## More than One Way to Skin an MBR

There are a number of elaborate ways you can destroy some or all of your MBR. Please be careful with this first command. It actually deletes your MBR at the very least, and with a typo, it potentially could delete the entire disk, so step lightly. Let's start with the most blunt, dd:

```
$ sudo dd if=/dev/zero of=/dev/sda bs=512 count=1
```

This command basically blanks out your MBR by overwriting it with zeros. Now, unless you are masochistic, or you are like me and used this in a demonstration of MBR recovery tools, you probably wouldn't ever run this command. Most people end up destroying part of their MBR in one of two ways: mistakes with bootloaders and mistakes with fdisk or other partitioning tools.

Mistakes with partitioning tools probably are the most common way people break their MBRs, or more specifically, their partition tables. It could be that you ran fdisk on sda when you meant to run it on sdb. It could be that you just made a mistake when resizing a partition, and after a reboot, it wouldn't mount. The important thing to keep in mind is that when you use partitioning tools, they typically update only the partition table on the drive. Even if you resize a drive, unless you tell a partitioning tool to reformat the drive with a fresh filesystem, the actual data on the drive doesn't change. All that has changed are those 64 bytes at the beginning of the drive that say where the partitions begin and end. So, if you make a partitioning mistake, your data is fine. You just have to reconstruct that partition table.

It would figure that the first time I really destroyed my MBR, it was through the second, less-common way—mistakes with bootloaders. In my case, it was a number of years ago, and I was struggling to get an early version of GRUB installed on a disk. After the standard command-line commands didn't work, I had the bright idea that maybe I could use the GRUB boot floppy image. After all, it was 512 bytes and so was my MBR, right? Well, it sort of worked. GRUB did appear; however, what I didn't realize was that in addition to writing GRUB over the first 446 bytes of my MBR, I also wrote over the last 66 bytes, my partition table. So although GRUB worked, it didn't see any partitions on the drive.

## Guessing Games Fix a Partition Table

I had at least used Linux long enough that after I made my mistake, I realized my actual data was still there and that there *must* be some way to restore the partition table. This was when I first came across the wonderful tool called gpart.

gpart is short for Guess Partition, and that is exactly what it does. When you run the gpart command, it scans through a disk looking for signs of partitions. If it finds what appears to be the beginning of a Windows FAT32 partition, for instance, it jots it down and continues until eventually it sees what appears to be the end. Once the tool has scanned the entire drive, it outputs its results to the screen for you to check and edit. It also optionally can write this reconstructed partition table back to the disk.

gpart has been around for quite some time and is packaged by all of the major distributions, so you should be able to install it with your standard package manager. Don't confuse it with gparted, which is a graphical partitioning tool. Of course, if your main system is the one with the problem, you need to

# When Disaster Strikes: Restoring a Master Boot Record

By Kyle Rankin

(Reprinted from the Linux Journal)

find a rescue disk that has it. Knoppix and a number of other rescue-focused disks all include gpart out of the box.

To use gpart, run it with root privileges and give it the disk device to scan as an argument. Here's gpart's output from a scan of my laptop's drive:

```
greenfly@minimus:~$ sudo gpart /dev/sda
```

```
Begin scan...
```

```
Possible partition(Linux ext2), size(9773mb), offset(0mb)
```

```
Possible partition(Linux swap), size(980mb), offset(9773mb)
```

```
Possible partition(SGI XFS filesystem), size(20463mb), offset(10754mb)
```

```
End scan.
```

```
Checking partitions...
```

```
Partition(Linux ext2 filesystem): primary
```

```
Partition(Linux swap or Solaris/x86): primary
```

```
Partition(Linux ext2 filesystem): primary
```

```
Ok.
```

```
Guessed primary partition table:
```

```
Primary partition(1)
```

```
type: 131(0x83)(Linux ext2 filesystem)
```

```
size: 9773mb #s(20016920) s(63-20016982)
```

```
chs: (0/1/1)-(1023/254/63)d (0/1/1)-(1245/254/56)r
```

```
Primary partition(2)
```

```
type: 130(0x82)(Linux swap or Solaris/x86)
```

```
size: 980mb #s(2008120) s(20016990-22025109)
```

```
chs: (1023/254/63)-(1023/254/63)d (1246/0/1)-(1370/254/58)r
```

```
Primary partition(3)
```

```
type: 131(0x83)(Linux ext2 filesystem)
```

```
size: 20463mb #s(41909120) s(22025115-63934234)
```

```
chs: (1023/254/63)-(1023/254/63)d (1371/0/1)-(3979/184/8)r
```

```
Primary partition(4)
```

```
type: 000(0x00)(unused)
```

```
size: 0mb #s(0) s(0-0)
```

```
chs: (0/0/0)-(0/0/0)d (0/0/0)-(0/0/0)r
```

To hammer home the point about how easy it is to back up the MBR, now I have an extra backup of my laptop partition table—in this magazine.

As you can see, it correctly identified the two primary partitions (/ and /home) and the swap partition on my laptop and noted that the fourth primary partition was unused. Now, after reviewing this, if I decided that I wanted gpart to write its data to the drive, I would run:

```
$ sudo gpart -W /dev/sda /dev/sda
```

# When Disaster Strikes: Restoring a Master Boot Record

By Kyle Rankin

(Reprinted from the Linux Journal)

That isn't a typo; the `-W` argument tells `gpart` to which disk to write the partition table, but you still need to tell it which drive to scan. `gpart` potentially could scan one drive and write the partition table to another. Once you specify the `-W` option, `gpart` gives you some warnings to accept, but it also prompts you to edit the results from within `gpart` itself. Personally, I've always found it a bit more difficult to do it that way than it needs to be, so I skip the editor, have it write to the disk, and then use a tool like `fdisk` or `cfdisk` to examine the drive afterward and make tweaks if necessary.

## **gpart Limitations**

`gpart` is a great tool and has saved me a number of times, but it does have some limitations. For one, although `gpart` works very well with primary partitions, it is much more difficult for it to locate extended partitions, depending on which tool actually created them. Second, take `gpart` results with a grain of salt. It does its best to reconstruct drives, but you always should give its results a sanity check. For instance, I've seen where it has identified the end of a partition one or two megabytes short from the actual end. Typically, when we partition drives, we put one partition immediately after another, so these sorts of errors are pretty easy to find.

## **Reload the Boot Code**

Now, if you have destroyed only the partition table, you hopefully should be restored at this point. If you managed to destroy the boot code as well, you need to restore it too. These days, most Linux distributions use GRUB, so with your restored partition table, if you are currently booted into the affected system, run:

```
$ sudo grub-install --recheck /dev/sda
```

Replace `/dev/sda` with the path to your primary boot device. If you use an Ubuntu system, you optionally could use the `update-grub` tool instead. If you are currently booted in to a rescue disk, you first need to mount your root partition at, say, `/mnt/sda1`, and then use `chroot` to run `grub-install` within it:

```
$ sudo mkdir /mnt/sda1
$ sudo mount /dev/sda1 /mnt/sda1
$ sudo chroot /mnt/sda1 /usr/sbin/grub-install
```

```
--recheck /dev/sda
```

If the `chrooted` `grub-install` doesn't work, you typically can use your rescue disk's `grub-install` with the `-root-directory` option:

```
$ sudo /usr/sbin/grub-install --recheck
```

```
--root-directory /mnt/sda1 /dev/sda
```

Well hopefully, if you didn't have a profound respect for those 512 bytes at the beginning of your hard drive, you do now. The MBR is like many things in life that you don't miss until they are gone, but at least in this case, when it's gone, you might be able to bring it back.