

# GPT fdisk for Linux

Rod Smith

GPT fdisk (aka gdisk) is a text-mode partitioning tool for Linux, FreeBSD, or Mac OS X that works on Globally Unique Identifier (GUID) Partition Table (GPT) disks, rather than on the more common (through 2009) Master Boot Record (MBR) partition tables. If this sounds interesting to you, then read on (or skip straight to the "Obtaining GPT fdisk" link if you don't need the GPT pep talk). If you don't know what a GPT is, be sure to read the first section!

Note: The FreeBSD version of GPT fdisk can't save changes to your partition table if any partition from the disk is mounted. If you want to modify your FreeBSD boot disk, you must do so from an emergency system or from a dual boot to Linux or Mac OS X. This limitation is shared by at least some other FreeBSD partitioning tools, such as gpt and FreeBSD's fdisk. This limitation does not exist in the Linux or Mac OS X versions of gdisk.

## What's a GPT

In order to understand what GPT is, you must first understand the current standard for disk partitioning and its limits. With that knowledge in hand, you can see how GPT can fix MBR's deficiencies.

## MBR, Its Annoyances, and Its Limits

Since the earliest hard disks for x86 computers, these disks have been divided into one or more partitions using a partitioning scheme that has, through the ages, gone by several names, such as MS-DOS disklabels, BIOS partitions, and MBR partitions. By whatever name, the MBR partitioning scheme has several characteristics that have, in one way or another, been limitations or annoyances:

- The MBR system originally provided for only four partitions. As a workaround, one of these original (or primary) partitions can be set aside (and is then known as an extended partition) to hold an arbitrary number of logical partitions. This configuration is awkward and can lead to problems, since some OSes can only boot from primary partitions and since it's not easy to convert a partition from one type to another.
- MBR partitions include one-byte partition type codes to help OSes identify the partitions' purpose. The single byte has proven somewhat limiting, and there are occasional collisions -- a single code with an ambiguous meaning, such as 0x82, which can refer to either Linux swap space or a Solaris disklabel.
- The MBR scheme originally employed cylinder/head/sector (CHS) addresses to identify partitions' locations. This feature can lead to problems because the CHS geometry of disks can change. The CHS scheme also tops out at a maximum disk size of 8 GiB. In part to work around this limitation, MBR has been extended to use 32-bit logical block addressing (LBA), but using both systems itself introduces the opportunity for problems, such as mismatched CHS and LBA definitions for a single partition. In practice, modern OSes seem to mostly ignore the CHS values, and they must ignore the CHS values on partitions that begin or end beyond the 8 GiB mark.
- In conjunction with the near-universal sector size of 512 bytes, the 32-bit LBA pointers used by MBR partitions impose a 2 TiB limit on disk and partition size. This limit, unlike the earlier 8 GiB limit of CHS, is not easily overcome by a simple extension to MBR; there simply isn't space left in the size allocated to MBR data structures. With 2 TiB drives available now and larger drives expected soon, this problem is a very serious one. (Certain types of hardware RAID configurations permit virtual drive sizes to exceed 2 TiB even today.)
- MBR partitions are susceptible to damage. The primary partition table is stored entirely within the first sector of the disk, so if it's destroyed or damaged, it will be hard to recover the disk's partitions.

# GPT fdisk for Linux

Rod Smith

Logical partitions are stored in a linked list data structure that's scattered over the extended partition, so if a single link is broken, access to the remaining logical partitions will be lost.

## Four for the Price of Two?

If you're familiar with MBR data structures, you may think I'm being a little bit pessimistic. This is because MBR records partition locations in terms of the starting sector and the partition's length. Both of these are 32-bit values, so in theory you could use MBR on a 4 TiB disk, so long as all the space after the 2 TiB mark is in a single primary partition, or perhaps in a single extended partition, which could in turn hold many logical partitions. Such a configuration would be somewhat limiting, but it fits within the MBR framework.

To test the practicality of such a configuration, I ran tests using the QEMU machine emulator and its virtual disks, which can appear much larger to a guest OS than to the host OS. I was able to create a 4 TiB disk that was just kilobytes in size on my real disk (although it grew as I manipulated it, of course). I tested with a number of legacy and modern OSes, including Linux, FreeBSD, Windows Me, Windows XP, OS/2, and BeOS.

To make a long story short, the only OSes that seemed capable of handling a partition that spanned the 2 TiB mark were Linux and FreeBSD. Windows Me didn't show a drive icon, and its FDISK reported that it was unable to access the disk. Windows XP saw the disk as being just 8 GiB in size. BeOS reported the disk information correctly in its disk setup tool, but was unable to mount the partition I created in Linux or to create a new filesystem on the partition. OS/2's FDISK reported the already-created partition was smaller than it was; apparently it just ignored everything past the 2 TiB mark. One caveat: It's conceivable that these OSes were interacting with a peculiarity of the QEMU environment to create these problems; however, as Linux and FreeBSD were able to handle the disks, I suspect the real problem was in the OSes themselves.

Because Linux and FreeBSD have relatively mature GPT support, there seems to be limited benefit to carefully crafting a configuration to enable use of MBR on a 2-4 TiB drive -- at least, for the OSes I tested. I was unable to test Windows Vista or Windows 7 in QEMU, and it's conceivable that one or both of them would benefit from such a configuration. Overall, though, I consider 2 TiB to be the practical limit for MBR, since so many OSes can't use MBR's theoretical capacity to manage some configurations of up to twice that size.

## GPT to the Rescue

The heir apparent to MBR is GPT. This new partitioning scheme fixes many of MBR's problems:

- In its most common configuration, GPT supports up to 128 partitions, so there's no need for extended or logical partitions. In case needs change, GPT data structures permit larger partition table sizes, although in practice few tools seem to allow users to adjust this parameter. (GPT fdisk does, though.)
- The GPT partition type code is a 128-bit (16-byte) GUID. In theory, this gives plenty of room for lots of unique partition types. Unfortunately, in practice there are already collisions -- most importantly for Linux users, Linux and Windows use the same type code for their data partitions. To supplement the type code, GPT supports a plain-text name for each partition.
- GPT knows nothing about CHS geometries and it uses 64-bit sector pointers, so 2 TiB drives are puny compared to the total supported disk size of GPT.

# GPT fdisk for Linux

Rod Smith

- GPT adds CRC32 checksums to its data structures and stores those structures twice on the disk - once at the start of the disk and again at the end. These measures help protect the system against accidental damage caused by carelessness or disk errors.

Unfortunately, GPT is not without its problems. These mainly relate to compatibility. Older OSes have no or limited GPT support. For instance, Windows only supports GPT at all on Windows Server 2003, the 64-bit (but not the 32-bit) version of Windows XP, Windows Vista, and later. Through Windows Vista, booting from a GPT disk is impossible unless the system uses the Extensible Firmware Interface (EFI) rather than a legacy BIOS. (Most computers through at least September of 2009 still use a legacy BIOS. All Intel-based Macintoshes use EFI, though, and EFI is starting to appear as a user-configurable option on some mainstream motherboards.) Linux has long supported GPT, but boot support depends on the boot loader. GRUB through version 0.97 doesn't officially support booting from GPT, but patched versions of GRUB 0.97 with GPT support are common. Intel-based Macs use GPT by default.

To protect GPT disks against errant older disk tools, GPT keeps an MBR partition table on the first sector of the disk. This MBR contains a single disk-spanning partition of type 0xEE, which makes older tools think the disk is in use by an unknown OS. Some tools take advantage of this feature to create a hybrid MBR configuration, in which some partitions are accessible via both GPT and MBR definitions. Although this is non-standard, awkward, and delicate, it can help make the transition from MBR to GPT easier by providing a workaround for OSes that don't understand GPT.

In addition to the protective MBR, GPT features two main types of data structure, each of which is stored on the disk twice:

- The GPT headers contain disk meta-data -- information on the locations of the partition tables, the disk GUID, and so on. No actual partitions are defined in the headers, though. Each disk has two headers, a main header and a backup header. If one is damaged, you can use the other to recover the damaged one.
- Just as there are two headers, there are two partition tables. These should be byte-for-byte identical to one another, unlike the headers, which vary because they need to point to each other and to their own partition tables.

Because of the hard 2 TiB limit of MBR partitions, chances are you'll be forced to switch to GPT for at least some disks in the not-too-distant future. MBR is likely to remain useful on smaller devices, such as USB flash drives, for years to come. In 2009, most x86 and x86-64 PCs still use MBR-partitioned disks. Mac hardware is an exception to this rule; Apple has embraced GPT, as well as the EFI, of which the GPT definition is part.

## Why Use GPT fdisk?

If you want or need to use GPT, you have relatively few choices for partitioning software. Under Linux, libparted and the programs that use it (GNU Parted, gparted, and so on) have been your only real choice for a while. As of early 2009, these programs have worked, but have also suffered from several problems. For instance, as of GNU Parted 1.7.1, when creating a partition with a FAT filesystem, Parted marks it with a Microsoft Reserved partition type code, which makes the partition inaccessible to both Windows and Mac OS. As Homer Simpson would say, d'oh!

To work around such problems and to satisfy personal curiosity about GPT, I wrote gdisk. This program's user interface is modeled after that of Linux's fdisk utility for manipulating MBR disks, although gdisk necessarily deviates from fdisk in many respects.

# GPT fdisk for Linux

Rod Smith

Compared to GNU Parted, gdisk has several advantages and disadvantages. Broadly speaking, you should consider using gdisk if:

- You don't mind using relatively new software. gdisk is currently only at version 0.5.0 and has only been tested on a handful of systems. You should be particularly cautious when using the program on a disk with existing partitions that hold valuable data.
- You use Linux, FreeBSD, or Mac OS X. Each OS seems to have its own way of doing certain important low-level disk tasks, such as determining the disk size, so gdisk must be explicitly customized for each OS it supports.
- You want more precise control over your partitioning than Parted provides. For instance, gdisk provides sector-exact control of partition sizes and it enables you to enter any arbitrary GPT partition type.
- You want better control over recovery operations in the event that a corrupt partition table is encountered.
- You want to convert an MBR disk to GPT format. gdisk permits doing this, although doing so will require you to re-install GRUB (if it's a boot disk).
- You want to convert a disk that uses BSD disklabels to GPT format, or you want to convert a BSD disklabel within an MBR or GPT partition, into GPT partitions. This conversion is more trouble-prone than the MBR conversion, but it works in many cases.
- You want to back up and restore your partition table to or from an ordinary disk file.
- You like fdisk's simple text-mode interface.

Chances are you'll be happier with GNU Parted or its GUI consins if:

- You want a stable and mature program. (Note, however, that one of my motivations for writing gdisk was glaring bugs in Parted. I suspect Parted's GPT code is relatively untested compared to its MBR code.)
- You prefer a GUI interface for partitioning. (This is true only of gparted, qtpted, and the like; GNU Parted is text-based.)
- You want to create filesystems at the same time you create the partitions that hold them.
- You want to resize or move filesystems along with their partitions.
- You want to use the software on any OS but Linux, FreeBSD, or Mac OS X. (You can use partitions created by gdisk on other OSes, but gdisk itself runs only on Linux, FreeBSD, and Mac OS X systems.)

Overall, I believe gdisk will appeal to those who like to use simple tools that provide relatively direct control over the things they manage. If you prefer fdisk to Parted on MBR disks, you'll probably prefer gdisk to Parted on GPT disks. Keep in mind gdisk's relative immaturity, though.

# GPT fdisk for Linux

Rod Smith

## A GPT fdisk Walkthrough

The GPT fdisk package includes a man page that documents the gdisk program in the usual way. If you want to read up on all its options, please refer to that document. This page takes a different approach: It walks you through some common operations, explaining each one.

This walkthrough uses as a starting point the partitioning of a 7.5 GiB USB flash drive so that it contains three partitions: One FAT partition for data exchange among multiple Oses, one ext2fs partition for use by Linux alone, and one UFS partition for use by FreeBSD. This scenario is admittedly somewhat artificial, since most people don't use USB flash drives in this way; but this is similar to what you might do when setting up a hard disk for use by multiple Oses, or perhaps even one OS.

The starting point is a drive with a single FAT partition in an MBR partition table. This starting point illustrates what happens when you start gdisk on a disk with an existing MBR partition table:

```
# gdisk /dev/disk1
GPT fdisk (gdisk) version 0.5.0

Partition table scan:
  MBR: MBR only
  BSD: not present
  APM: not present
  GPT: not present

*****
Found invalid GPT and valid MBR; converting MBR to GPT format.
THIS OPERATON IS POTENTIALLY DESTRUCTIVE! Exit by typing 'q' if
you don't want to convert your MBR partitions to GPT format!
*****

Command (? for help):
```

You launch gdisk in much the same way as fdisk, although gdisk supports very few command-line arguments. (You may pass it the `-l` option to see the partition table and then quit, but that's it.) You must know the name of the device file that's used to access the disk. For Linux, this file is likely to take the form `/dev/sdx` or `/dev/hdx`, where `x` is a letter. In Mac OS X, the device filename takes the form `/dev/disky`, where `y` is a number from 0 up.

When gdisk starts, it performs a scan for four types of existing partition tables and displays the results. MBR is the common Master Boot Record partitioning system; BSD is the Berkeley Standard Distribution (BSD) disklabel system used on some computers that run a BSD (FreeBSD, OpenBSD, etc.); APM is the Apple Partition Map used on 680x0- and PowerPC-based Macintoshes; and GPT is of course the GUID Partition Table. The BSD and APM scans report either present or not present, but GPT can report three states (present, not present, or damaged), and there are four MBR states (MBR only, protective, hybrid, or not present). The normal state for an MBR-only disk is as shown here, and the normal state for a GPT disk is MBR: protective and GPT: present. Hybrid MBRs (described on my Hybrid MBRs page) change the MBR state to hybrid. Other combinations are possible, some of which indicate that a disk has been re-partitioned for a new partitioning system without completely erasing the old partition table.

# GPT fdisk for Linux

Rod Smith

When starting `gdisk` on a disk with existing MBR or BSD disklabel partitions and no GPT, the program displays a message surrounded by asterisks about converting the existing partitions to GPT. This is intended to scare you away if you launch `gdisk` on the wrong disk by accident or if you don't know what you're doing.

The Command (? for help): prompt is taken straight from Linux `fdisk`, except that GPT `fdisk` uses ? as the prompt for help, whereas `fdisk` uses `m`. If you type ?, you'll see a list of available commands:

```
Command (? for help): ?
b      back up GPT data to a file
c      change a partition's name
d      delete a partition
i      show detailed information on a partition
l      list known partition types
n      add a new partition
o      create a new empty GUID partition table (GPT)
p      print the partition table
q      quit without saving changes
r      recovery and transformation options (experts only)
s      sort partitions
t      change a partition's type code
v      verify disk
w      write table to disk and exit
x      extra functionality (experts only)
?      print this menu
```

`fdisk` users will recognize many of these commands, such as `d`, `n`, and `p`. Although some of these don't work in exactly the same way as in `fdisk`, most of them are very similar. For instance, suppose you want to verify that you're working on the disk you think you're working on and then delete the partition:

```
Command (? for help): p
Disk /dev/disk1: 15654912 sectors, 7.5 GiB
Disk identifier (GUID): FC94128F-F9A2-0096-F830-3B1862BF3801
Partition table holds up to 128 entries
First usable sector is 34, last usable sector is 15654878
Total free space is 38 sectors (19.0 KiB)

Number  Start (sector)    End (sector)  Size      Code  Name
   1            63             15654869     7.5 GiB   0700  Linux/Windows data

Command (? for help): d
Using 1

Command (? for help):
```

The disk data shown by `p` is a bit different from `fdisk`'s, but it's similar. One salient point is that, because GPT knows nothing about CHS geometry, `gdisk` measures all partitions in sectors (aka blocks). The original converted MBR partition begins at sector 62 and ends at sector 15,654,878.

The Code column of the output shows the partition type code. Because GPT employs a 16-byte GUID number for storing partition type codes, displaying this raw code would be both space-consuming and difficult for people to parse. Thus, `gdisk` translates the GUID code into a variant of MBR type codes. In

## GPT fdisk for Linux

Rod Smith

particular, the codes used by `gdisk` are the MBR codes multiplied by hexadecimal `0x0100`, and displayed in hexadecimal. The `0x0700` code shown above is therefore equivalent to an MBR code of `0x07`, which is the code for HPFS/NTFS.

"But wait," you say, "I thought the disk had a FAT partition!" Indeed it does. Windows uses a single GUID code for all its data partitions, be they FAT or NTFS. Linux uses the same code for its data partition. Thus, in this case several different MBR codes are all translated into a single GPT GUID code. `gdisk` uses, somewhat arbitrarily, the `0x0700` code (or more precisely, `EBD0A0A2-B9E5-4433-87C0-68B6B72699C7`) for all of these.

The Name column displays a free-form text string that you can modify. Starting with version 0.4.0, `gdisk` places the name associated with the partition type code in this field when it creates or converts partitions, but you can change this default, as described shortly.

With the existence of this single partition as (presumably) sufficient identification that this is the disk you want to modify, the `d` command deletes the partition. Since only one partition exists, `gdisk` doesn't prompt you for a number; it just responds Using 1 and then displays another command prompt.

At this point you presumably want to begin adding your new partitions. As with `fdisk`, you can use `n` to do this task:

```
Command (? for help): n
Partition number (1-128, default 1):
First sector (34-15654878, default = 34) or {+-}size{KMGT}:
Last sector (34-15654878, default = 15654878) or {+-}size{KMGT}: +4G
Current type is 'Unused entry'
Hex code (L to show codes, 0 to enter raw code): L
0700 Linux/Windows data      0c01 Microsoft Reserved    2700 Windows RE
4200 Windows LDM data       4201 Windows LDM metadat   8200 Linux swap
8301 Linux Reserved        8e00 Linux LVM              a500 FreeBSD disklabel
a501 FreeBSD boot          a502 FreeBSD swap          a503 FreeBSD UFS
a504 FreeBSD ZFS           a505 FreeBSD Vinum/RAID    a800 Apple UFS
a901 NetBSD swap           a902 NetBSD FFS            a903 NetBSD LFS
a903 NetBSD RAID           a904 NetBSD concatenated   a905 NetBSD encrypted
ab00 Apple boot            af00 Apple HFS/HFS+        af01 Apple RAID
af02 Apple RAID offline    af03 Apple label           af04 AppleTV recovery
be00 Solaris boot          bf00 Solaris root           bf01 Solaris /usr & Mac
bf02 Solaris swap          bf03 Solaris backup         bf04 Solaris /var
bf05 Solaris /home         bf05 Solaris EFI_ALTSCTR   bf06 Solaris Reserved 1
bf07 Solaris Reserved 2    bf08 Solaris Reserved 3    bf09 Solaris Reserved 4
bf0a Solaris Reserved 5    c001 HP-UX data            c002 HP-UX service
ef00 EFI System            ef01 MBR partition schem    ef02 BIOS boot partition
fd00 Linux RAID
Hex code (L to show codes, 0 to enter raw code): 0700
Changed system type of partition to 'Linux/Windows data'
```

In this example, I pressed the Enter key to accept the default partition number and first sector values. The default starting sector is at the start of the largest contiguous block of free space, which in this case is the beginning of all free space. I then typed `+4G` to create a 4 GiB partition. I typed `L` to see a list of partition type codes for this first partition, just so you could see them. Since the first partition should be a FAT partition, I opted to give it a `0x0700` type code, which `gdisk` translates into the correct GUID.

## GPT fdisk for Linux

Rod Smith

Starting with gdisk 0.4.1, partition locations and sizes can be entered in either absolute or fully relative form. In absolute form, you enter an absolute location or end point in sectors (or KiB, MiB, GiB, or TiB), as in 34 for sector 34 or 4M for the sector that corresponds to the 4MiB position on the disk. Relative values are specified by preceding the number with a plus sign (+) or minus sign (-), in which case the value is taken as relative to the start or end, respectively, of the free disk space area noted in the prompt. You can use this feature to create partitions with gaps between them, or to leave just enough space after a partition to create another one of a given size. For instance, you can specify +128M as the first sector to produce a 128MB gap between the partition you're creating and the previous one; or you can specify -1G as the last sector to fill all but 1GiB of the contiguous free space.

With that task done, it's now time to create two more partitions, but these will have to be smaller:

```
Command (? for help): n
Partition number (2-128, default 2):
First sector (8388642-15654878, default = 8388642) or {+-}size{KMGT}:
Last sector (8388642-15654878, default = 15654878) or {+-}size{KMGT}: +2G
Current type is 'Unused entry'
Hex code (L to show codes, 0 to enter raw code): 0700
Changed system type of partition to 'Linux/Windows data'

Command (? for help): n
Partition number (3-128, default 3):
First sector (12582946-15654878, default = 12582946) or {+-}size{KMGT}:
Last sector (12582946-15654878, default = 15654878) or {+-}size{KMGT}:
Current type is 'Unused entry'
Hex code (L to show codes, 0 to enter raw code): a503
Changed system type of partition to 'FreeBSD UFS'
```

The second partition, which will hold a Linux ext2 filesystem, receives the same 0x0700 type code as the first (FAT) partition. For the final partition, I used the default end partition number rather than enter a value explicitly; and I gave it a 0xa503 type code, which flags it with the GUID for FreeBSD UFS data.

You can now examine the partitions you've created:

```
Command (? for help): p
Disk /dev/disk1: 15654912 sectors, 7.5 GiB
Disk identifier (GUID): FC94128F-F9A2-0096-F830-3B1862BF3801
Partition table holds up to 128 entries
First usable sector is 34, last usable sector is 15654878
Total free space is 0 sectors (0 bytes)

Number  Start (sector)    End (sector)  Size      Code  Name
-----  -
1         34                8388641      4.0 GiB   0700  Linux/Windows data
2       8388642           12582945      2.0 GiB   0700  Linux/Windows data
3      12582946           15654878      1.5 GiB   A503  FreeBSD UFS
```

The Name column holds default names, but you can assign your partitions more descriptive names using the c command. In this case, you might want to provide names to help distinguish between the first two partitions, which will hold different filesystems, despite having the same type code:

## GPT fdisk for Linux

Rod Smith

```
Command (? for help): c
Partition number (1-3): 1
Enter name: Shared (FAT) data

Command (? for help): c
Partition number (1-3): 2
Enter name: Linux data

Command (? for help): p
Disk /dev/disk1: 15654912 sectors, 7.5 GiB
Disk identifier (GUID): FC94128F-F9A2-0096-F830-3B1862BF3801
Partition table holds up to 128 entries
First usable sector is 34, last usable sector is 15654878
Total free space is 0 sectors (0 bytes)

Number  Start (sector)    End (sector)  Size      Code  Name
   1            34             8388641     4.0 GiB   0700  Shared (FAT) data
   2         8388642         12582945     2.0 GiB   0700  Linux data
   3        12582946        15654878     1.5 GiB   A503  FreeBSD UFS
```

You can now tell by using the p command what the partition's purpose is. (GNU Parted and many other tools also display partitions' names.)

One caveat about partition names is that gdisk only supports entry and display of ASCII names, although the GPT data structure uses UTF-16 to store the names. What this means is that you won't be able to enter the full range of name data, and if you try to view the partitions on a disk with a non-ASCII name, gdisk is likely to corrupt the name in its display. (gdisk retains the full UTF-16 name internally and should not corrupt it when you save non-name changes, though.)

Before saving your changes, you may want to verify that there are no glaring problems with the GPT data structures. You can do this with the v command:

```
Command (? for help): v
No problems found. 0 free sectors (0 bytes) available in 0
segments, the largest of which is 0 sectors (0 bytes) in size
```

In this case, every available byte on the disk is allocated to partitions and no problems were found. If free space were available for partition creation, you'd see how much was available. If gdisk found problems, such as overlapping partitions or mismatched main and backup partition tables, it would report them here. Of course, gdisk includes safeguards to ensure that you can't create such problems yourself. The v option's sanity checks are intended to help you spot problems that might result from data corruption.

If problems had been detected, you might be able to correct them using various options on the recovery & transformation menu, which is a second menu of options available by typing r:

```
Command (? for help): r

recovery/transformation command (? for help): ?
b      use backup GPT header (rebuilding main)
c      load backup partition table from disk (rebuilding main)
d      use main GPT header (rebuilding backup)
```

# GPT fdisk for Linux

Rod Smith

```
e      load main partition table from disk (rebuilding backup)
f      load MBR and build fresh GPT from it
g      convert GPT into MBR and exit
h      make hybrid MBR
i      show detailed information on a partition
l      load partition data from a backup file
m      return to main menu
o      print protective MBR data
p      print the partition table
q      quit without saving changes
t      transform BSD disklabel partition
v      verify disk
w      write table to disk and exit
x      extra functionality (experts only)
?      print this menu
```

The main data-recovery options are b, c, d, e, f, and l. When using these options, you should keep in mind the distinctions between the various types of GPT data structures described in "What's a GPT?".

Ordinarily, gdisk loads the main GPT header and partition table into memory and uses them almost exclusively, although GPT maintains the secondary header (but not the secondary partition table) independently. (gdisk does check the backup partition table for corruption.) If either header is damaged, the b and d options cause gdisk to rebuild one using the contents of the other. If the main partition table is damaged, you can load the backup in its place with the c option. The e option, which loads the main partition table, is unlikely to be useful because the main partition table is loaded and used to begin with. You might use e if you've tried using the backup but found it's in worse shape than the main table, or if the wrong main table was loaded because of a corrupt main header that you've corrected with b. The f option could be used if the GPT is badly damaged on a disk that held a hybrid MBR, and you want to salvage what you can by using the hybrid MBR's partition definitions. The l option loads a backup of the GPT data structures that you must have previously saved using the b main-menu option.

Transformation options enable you to perform partition table transformations that are not handled automatically when the program starts: convert a GPT to MBR (g), create a hybrid MBR (h), or convert BSD disklabels in a container partition into GPT partitions (t). These options are described in more detail on the Converting to GPT and Hybrid MBR pages of this document.

A third menu, the experts' menu, can be accessed by typing x in either the main menu or the recovery & transformation menu:

```
recovery/transformation command (? for help): x

Expert command (? for help): ?
a      set attributes
c      change partition GUID
g      change disk GUID
i      show detailed information on a partition
m      return to main menu
n      create a new protective MBR
o      print protective MBR data
p      print the partition table
q      quit without saving changes
r      recovery and transformation options (experts only)
```

# GPT fdisk for Linux

Rod Smith

```
s      resize partition table
v      verify disk
w      write table to disk and exit
z      zap (destroy) GPT data structures and exit
?      print this menu
```

You can do some low-level edits, such as changing partition or disk GUIDs (c and g, respectively). The z option immediately destroys the GPT data structures, which you should do if you want to re-use a GPT disk using some other partitioning scheme. If these structures aren't erased, some partitioning tools can become confused by the apparent presence of two partitioning systems.

**Caution:** Version 0.4.2 of gdisk replaced the MBR when using the z option, so if you used fdisk or some other MBR-unaware utility to repartition a GPT disk and then used this option in gdisk to erase the older GPT data structures, you'd lose your new MBR partitions. This is not the case in earlier or later versions of gdisk unless you answer in the affirmative when asked if you want to wipe out the MBR. Nonetheless, if you've already repartitioned using a GPT-unaware tool, you should be cautious; it's possible that the new partitions will occupy some of the space that the older GPT structures once occupied. In this case, GPT fdisk might damage your new partitions.

Generally speaking, the options on both the recovery & transformation menu and the experts' menu shouldn't be used by anything but GPT experts. Non-experts might be forced to use these menus if a disk is damaged, though. Before taking drastic actions, you should use the b main-menu option to create a backup in a file, and store that file on a USB flash drive or some other place that's not on the disk you're modifying. That way, you'll be able to recover your original configuration if you damage your partitions. These precautions won't help you recover from damage done by inappropriate use of the z option on the experts' menu, though.

Once you're done making your changes, be they relatively simple partition creations or recovery from disk damage, you can exit from gdisk. The q command exits without saving your changes, while w writes your changes to disk. Both commands exist on both the main and experts' menus.

```
Expert command (? for help): w

Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING
MBR PARTITIONS!! THIS PROGRAM IS BETA QUALITY AT BEST. IF YOU LOSE ALL YOUR
DATA, YOU HAVE ONLY YOURSELF TO BLAME IF YOU ANSWER 'Y' BELOW!

Do you want to proceed, possibly destroying your data? (Y/N) y
OK; writing new GPT partition table.
Warning: The kernel may continue to use old or deleted partitions.
You should reboot or remove the drive.
The operation has completed successfully.
```

**Note for FreeBSD users:** A limitation that the FreeBSD version of gdisk shares with many (perhaps all) other GPT partitioning tools is that it can't write to a disk if any partitions from that disk are mounted. Thus, if you want to change your boot disk, you'll need to do so from an emergency system. If you know of a way around this limitation, I'd appreciate hearing from you.

You're now back at your shell prompt. Because gdisk doesn't create filesystems, you'll need to do that at your shell prompt. In the case of the 3-partition disk in this example, you can create two filesystems in Linux:

# GPT fdisk for Linux

Rod Smith

```
# mkdosfs /dev/sdc1
# mkfs -t ext2 /dev/sdc2
```

These commands work just as they would on MBR partitions. The example disk's third partition, which is intended for use in FreeBSD, is better formatted in that OS than in Linux.

## Partitioning Advice

In many cases, you can create partitions using GPT in much the same way you'd do with MBR: Place one after the other with no gaps between them and ordering them in whatever way is convenient. In fact, GPT's lack of primary/extended/logical distinctions can make this task easier, by enabling you to place boot partitions or other partitions that must be primary in MBR in any location you want.

Various documents on the Web, however, advise certain procedures when creating GPT partitions. For instance, Apple advises, in its Technical Note TN2166, the following:

- The use of specific partition type GUIDs, which are implemented in gdisk as the various Apple codes.
- Alignment of partitions on 4 KiB boundaries. This advice relates to limitations of the HFS Plus filesystem used by Mac OS X.
- Creation of a 200 MiB EFI System Partition (ESP) as the first partition on disks larger than 2 GiB. This partition is used by EFI in its boot process, so it's beneficial if you intend to boot from an EFI-based system.
- Assignment of 128 MiB of unallocated space after most partitions on disks larger than 1 GiB. (The ESP is an exception to this rule.) The intent is that this space may be used by disk utilities to help them do their jobs.

Windows seems to follow similar rules when creating GPT partitions. Parted can create cylinder-aligned partitions, but doesn't seem to follow Apple's suggestions. Depending on how you want to use the disk, you may want to follow some or all of these rules yourself. In particular, creation of an ESP, which is normally formatted with the FAT filesystem, makes sense on any boot disk. Even if your system uses BIOS, you might find an ESP to be a useful place to put boot loader or other system files. (In this respect, the ESP is similar to a separate Linux /boot partition.)

On BIOS-based systems, a BIOS Boot Partition (GPT fdisk code 0xef02) plays a role that's similar in some ways to the ESP. The BIOS Boot Partition can hold a second-stage boot loader or ancillary boot loader data. This is helpful because some MBR boot loaders hide data in the otherwise unused and unallocated space between the MBR and the start of the first primary partition. This unallocated and unused space may not exist on a GPT disk, so another place is needed to store the data. If you're in doubt, go ahead and create a small BIOS Boot Partition of about 200 KiB. At worst, it'll consume a small amount of disk space.

# GPT fdisk for Linux

Rod Smith

## Converting to GPT

One of the more unusual features of `gdisk` is its ability to read an MBR partition table or BSD disklabel and convert it to GPT format without damaging the contents of the partitions on the disk. This feature exists to enable upgrading to GPT in case the limitations of MBRs or BSD disklabels become too onerous — for instance, if you want to add more OSes to a multi-boot configuration, but the OSes you want to add require too many primary partitions to fit on an MBR disk. (Unfortunately, though, many such OSes can't handle GPT.) I've heard from one user who used this ability to fix an over-2TiB RAID array that had been inappropriately partitioned with MBR. The BSD disklabel support can help simplify and extend a BSD partitioning scheme by removing the need to keep an MBR partition containing a BSD disklabel on the disk of a multi-boot computer.

Note that the details of how to convert a disk, as well as the problems of a conversion, vary with the conversion type. You should read the appropriate section of this page to learn details. If you want to convert an MBR disk with a BSD disklabel partition, you'll have to perform both conversions.

## Converting from MBR to GPT

Conversions from MBR to GPT works because of inefficiencies in the MBR partitioning scheme. On an MBR disk, the bulk of the first cylinder of the disk goes unused -- only the first sector (which holds the MBR itself) is used. Depending on the disk's CHS geometry, this is likely to be sufficient space to store the GPT header and partition table. Likewise, space is likely to go unused at the end of the disk because the cylinder (as seen by the BIOS and whatever tool originally partitioned the disk) will be incomplete, so the last few sectors will go unused. This leaves space for the backup GPT header and partition table.

The task of converting MBR to GPT therefore becomes one of extracting the MBR data and stuffing the data into the appropriate GPT locations. Partition start and end points are straightforward to manage, with one important caveat: `gdisk` ignores the CHS values and uses the LBA values exclusively. This means that `gdisk` will fail in the conversion of disks that were partitioned with very old software. If the disk is over 8 GiB in size, though, `gdisk` should find the data it needs.

Converting the partition type codes poses another challenge. Sometimes a single MBR type code has two or more possible translations to GPT GUIDs. This is true of `0x82`, for instance, which can mean either a Linux swap partition or a Solaris disklabel. In this case, because `gdisk` is a Linux utility, it translates MBR `0x82` partitions into GPT Linux swap partitions. Such assignments can be changed by the user, of course.

Another conversion challenge relates to BSD disklabels contained within MBR partitions. As described later on this page, BSD disklabels are tricky to convert. Thus, when you perform an initial MBR conversion, your disklabels will be left intact, so that a potentially unwanted or incorrect conversion won't be forced upon you. In the MBR scheme, several disklabel type codes exist, such as `0xA5` for FreeBSD and `0xA9` for NetBSD. Because I know of only one GPT disklabel code (for FreeBSD), GPT `fdisk` converts the type code of all known MBR disklabel partitions to the GPT code for FreeBSD disklabels. FreeBSD, at least, can read such a partition just fine. I'm not sure about other OSes, though. Booting from a GPT-based BSD disklabel partition seems tricky at best; I've not succeeded in getting it working. In theory, you should be able to convert the disklabel partition and then boot directly from the new GPT partitions. I don't know of a GUID code for Solaris disklabels in GPT, and Solaris disklabels are different from BSD disklabels. Thus, I can't recommend converting a disk that contains Solaris disklabels to GPT format unless you know more about this than I do.

Once the conversion is complete, there will be a series of gaps between partitions. Gaps at the start and end of the partition set will be related to the inefficiencies mentioned earlier that permit the

# GPT fdisk for Linux

Rod Smith

conversion to work. Additional gaps before each partition that used to be a logical partition exist because of inefficiencies in the way logical partitions are allocated. These gaps are likely to be quite small (a few kilobytes), so you're unlikely to be able to put useful partitions in those spaces. You could resize your partitions with GNU Parted to remove the gaps, but the risks of such an operation outweigh the very small benefits of recovering a few kilobytes of disk space.

The GPT partition numbers on a converted disk should match the partition numbers under Linux on the original MBR disk. For instance, if a disk had two primary partitions (1 and 3) and two logical partitions (5 and 6), you'll have the same partition numbers on the converted disk. This is perfectly legal for a GPT disk; however, some GPT disk utilities will sort these partition numbers and remove the gaps. You may then need to adjust your `/etc/fstab` and boot loader configurations. If you'd rather not suffer unexpected surprises, you can sort your partitions in `gdisk` with the `s` menu option from the start. You may still need to make `/etc/fstab` and boot loader changes, but at least you'll know to do this from the start and not be taken by surprise by a utility that sorts the partition table but doesn't warn you about the need to make such changes.

As described earlier, MBR's inefficiencies usually provide room for GPT to squeeze in its data structures. This isn't always true, though. Some CHS geometries leave insufficient room at the start of the disk, and it's possible for there to be insufficient room at the end of the disk, depending on the details of the CHS geometry and the total number of sectors on the disk. Sometimes you can use the `s` option on the experts' menu to work around this problem. This option enables you to resize the partition table from 128 entries to some other value. This operation is non-destructive (it doesn't destroy the partition table). Using fewer than 128 entries is highly unusual, though, and it's conceivable such a configuration will cause problems with some environments. In cursory tests with Linux, Windows Vista, and Mac OS X, though, all three OSes seem to handle such disks fine. Another option for end-of-disk problems is to resize the final partition on the disk. Shrinking it by just a few kilobytes should be enough to provide room for the backup GPT header and partition table.

## Converting from BSD Disklabel to GPT

GPT fdisk version 0.4.0 adds BSD disklabel support. You must upgrade to perform BSD disklabel conversions if you're using an older version of the program.

BSD disklabels can be used in either of two ways:

- **Whole-disk disklabels** -- The BSD disklabel can be a hard disk's sole partitioning method. When GPT fdisk detects such a disk, it attempts to automatically convert the BSD disklabel partitions to GPT format, much as it does with an MBR disk
- **Disklabels within a carrier partition** -- A BSD disklabel can be stored within a conventional partition on an MBR or GPT disk. This configuration is popular on x86 and x86-64 systems, particularly when they multi-boot with another OS, since it permits the OSes to share a disk. GPT fdisk does not automatically convert the disklabel partitions in this case; you must do so by using the `'b'` option on the main menu.

In the case of whole-disk conversions, it's likely that the first and/or last partitions will overlap with GPT disk structures. In the case of the first partition, you might be able to reduce the GPT table size to save the partition, but the final partition will probably have to be deleted. (It will be converted, but its presence will block any attempt to save the partition table.) If you can resize the offending partition(s) using GNU Parted or a similar tool, a conversion might work better.

# GPT fdisk for Linux

Rod Smith

Carrier-partition conversions are less likely to run into the problem with overlap of GPT data structures, simply because the BSD disklabel is constrained to the MBR partition size, which in turn is less likely to have size issues. If a carrier conversion completes successfully, the carrier partition is deleted; otherwise it would overlap with its formerly-contained partitions. (Note that a "successful" conversion could be one in which several partitions are dropped in the process!) Thus, you'll have at least one gap in the partition numbering sequence, and the numbers may not match those assigned in Linux or any other OS. You can create a linear partition number sequence starting with 1 by using the `s` (sort) main-menu option.

Like MBR partitions and GPT partitions, BSD disklabel partitions have type codes. Some of these type codes are obsolete or have no clear GPT equivalents and some are ambiguous. For instance, a common type code is 7, for BSD Fast File System (FFS; aka the Unix File System, or UFS). There are, however, several GPT FFS/UFS codes -- for FreeBSD, NetBSD, Apple, and so on. GPT fdisk can't know which to use in the case of a whole-disk BSD disklabel, and even in the case of a carrier-partition disklabel, that conversion occurs after the conversion from MBR to GPT of the carrier partition, so the source OS data has been lost. For the most part, `gdisk` favors FreeBSD in its assignment of type codes; however, a couple do get converted to NetBSD type codes. Type codes with no obvious GPT equivalents are converted to Linux/Windows data partitions. You can of course change the type codes after the conversion.

Unfortunately, the BSD disklabel format is strange, and interpretation varies from one BSD to another. Thus, interpreting BSD disklabels (and therefore converting them to GPT form) is tricky. I've attempted to create a conversion process that works on any BSD disklabel format I know about, but I can't promise that the conversion will succeed. You may find that one or more partitions has disappeared from the conversion or that the converted partitions are unusable because their start and end points are wrong, they overlap with each other, or for other reasons. Therefore, you should be very cautious about your BSD disklabel conversions! Ideally, you should take notes on the disklabel's original contents and compare the converted partitions to your notes to ensure that everything is OK. Be sure to back up your data, and don't convert a booting BSD system unless you're prepared to re-install it.

## Why Not APM?

Another common partition system is the Apple Partition Map (APM). This scheme was used by Apple on all its Macintoshes until Apple switched to Intel CPUs; at that time, Apple also switched to GPT partitioning.

Starting with version 0.4.0, `gdisk` detects the APM signature and displays a warning message if it's found on a disk. GPT fdisk cannot, however, convert APM disks to GPT format. Such a conversion is theoretically possible, albeit with the caveat that the first and/or last partitions might be unconvertable because of overlap with GPT data structures.

In practice, `gdisk` doesn't support APM conversions because these conversions seem least important to me personally. MBR support is useful because of the huge installed base of x86 and x86-64 systems that use them, and BSD disklabel support is useful because some of these MBR-based systems dual-boot a BSD OS. The vast majority of computers that use APM run older versions of Mac OS, which can't handle GPT. APM has been used on removable media, of course, but for them, APM support in an OS seems more sensible than converting a disk from APM to GPT.

## Converting from GPT to MBR

Unfortunately, a complete conversion from GPT to MBR is not possible — at least, not without resizing partitions or hoping to find gaps between partitions, which are far from guaranteed to exist.

# GPT fdisk for Linux

Rod Smith

Any disk with more than four partitions requires use of logical partitions, and as just described, each logical partition requires a small preceding gap to hold its partition descriptor in a linked-list data structure. Because GPT fdisk can't guarantee that such gaps exist, the program can't create logical partitions based on GPT partitions, and converting more than four partitions is not supported.

Primary partitions are another matter, though. The `g` option on the recovery & transformation menu will convert up to four GPT partitions into primary MBR partitions. GPT fdisk then deletes the GPT data structures and terminates. If you have four or fewer partitions and want to convert to MBR, this function can be very useful.

One important caveat of GPT-to-MBR conversions is that the CHS geometry of the converted disk may be very strange. Partitions can begin and end mid-cylinder, particularly if the disk originated as a GPT disk rather than as a conversion from MBR format. Such peculiarities don't seem to cause problems for most modern OSes, but I can't guarantee that older or more obscure OSes will react as well.

A GPT-to-MBR conversion will also, of course, not overcome the 2 TiB limitation of MBR. GPT fdisk will convert any partition that's 2 TiB or smaller in size and that begins below the 2 TiB mark. This means that GPT fdisk can create converted partitions that span the 2 TiB mark. As described in the What's a GPT? section of this document, such partitions work fine in some OSes, but not in others. Thus, you should exercise caution if you create such a partition in a GPT-to-MBR conversion.

## Final Conversion Thoughts

`gdisk` provides no means for backing out of a conversion operation once you've written your changes to disk. (You can of course quit from `gdisk` by typing `q` before you save your changes, leaving the original partition table intact.) The GPT-to-MBR feature may be adequate for some purposes, but as just described, it's got important limitations. You can also create a backup of an MBR or BSD disklabel configuration and restore it if a conversion to GPT format goes badly. Generally speaking, though, you should consider an MBR-to-GPT or disklabel-to-GPT conversion to be irreversible.

If you convert a boot disk from one format to another, chances are good that it will no longer boot. If your OS(es) support booting from the new format, you should be able to correct this problem, as described in the next section.

## Booting from GPT

One of the challenges of GPT is that of booting from it. In early 2009, support for booting from GPT was limited compared to support for booting from MBR. This support varies by OS, as well. Following are my notes and general observations on this issue. Keep in mind, though, that these details are likely to change rapidly.

## General Comments: EFI vs. BIOS

GPT is part of the EFI specification. EFI includes its own boot loader, so of course booting from GPT disks becomes easier if your computer uses EFI. Unfortunately, EFI is still fairly rare on mainstream PCs. Among common computers, Apple Macintoshes are the systems that are most likely to use EFI. A few mainstream motherboard manufacturers are now including EFI support in their products. This support is usually of the United EFI (UEFI) variety. One obscure product, UEFI DUET, enables booting an EFI environment from a floppy disk or USB flash drive on a BIOS-based computer. Unfortunately, I know of no easy-to-use download link for official UEFI DUET binaries, although you can obtain source code from Tianocore.

# GPT fdisk for Linux

Rod Smith

The EFI specification includes an EFI System Partition (ESP) as a storage place for EFI drivers. Thus, if you intend to boot a disk using an EFI-based computer, you should create an ESP. Different sources suggest different sizes, but 100-200 MiB seems to be the range.

Many Internet sources, particularly discussion groups, include assertions that it's impossible to boot a GPT disk on a BIOS computer. This is nonsense -- or at least, it's true only of certain OSes. Windows, in particular, is behind the times on this score, as described shortly. I personally have successfully booted both Linux and FreeBSD on GPT-only computers with BIOSes. For the truly adventurous, it's even possible to get Mac OS running on GPT disks on conventional hardware, although this configuration is unsupported by Apple and may even be illegal. I've corresponded with an individual who has used UEFI DUET to boot Windows from GPT-only disks, although I have yet to get the procedure to work for me.

BIOS-based computers, whether they use MBR or GPT, rely on a boot loader in the first sector of the disk to help get the computer booted. In fact, the first 440 bytes of the MBR data structure are devoted to this boot loader. DOS and Windows place a very simplistic boot loader in this space. Other OSes and third-party utilities enable placing more sophisticated boot loaders in the MBR, although these boot loaders usually rely on multiple stages -- the boot loader code loads a secondary boot loader that's located elsewhere, and that boot loader may even load a third stage. In principle, these boot loaders can work just fine when the MBR is in fact a GPT protective MBR. In practice, the boot loader needs to be GPT-aware in order to work. Some boot loaders intended for use with BIOS-based systems and GPT disks require you to have a BIOS Boot Partition (GPT fdisk code 0xEF02) on the disk. This isn't true of all such boot loaders, though (it isn't true of the patched versions of GRUB 0.97 I've used, for instance), but you may want to create such a partition just in case. GRUB2, in particular, places its second-stage boot loader code in the BIOS Boot Partition, without a filesystem. Thus, in the case of GRUB2, the BIOS Boot Partition can be quite small -- perhaps as small as 32KiB.

In my tests, I've found that GNU Parted (version 1.7.1) tends to wipe out the MBR boot loader from GPT disks. Thus, you should be very cautious about using GNU Parted on a GPT boot disk if your computer is BIOS-based. (Parted's behavior in this respect shouldn't affect EFI-based systems.)

## Linux, GRUB, and LILO

Most modern Linux distributions install GRUB as the boot loader. Currently, GRUB 0.97 is the most common choice. Officially, GRUB 0.97 is not GPT-aware and so can't boot anything from a GPT disk. In practice, though, patched versions of GRUB 0.97 are common, and many distributions ship with them. (Ubuntu 8.04 and Fedora 10 definitely have GPT-enabled versions of GRUB 0.97, but openSUSE 11.0 does not.) If your distribution's GRUB lacks GPT support, you can download the System Rescue CD and apply its version of GRUB to your GPT disk:

1. Mount your /boot partition over the SRCD's /boot directory, or copy the contents of your /boot/grub directory over the SRCD's directory.
2. If your drives' device filenames are different under SRCD than under a normal boot of your distribution, edit /boot/grub/devices.map appropriately.
3. Type `grub-install /dev/sda` or `grub-install /dev/hda` to re-install GRUB.
4. Reboot. Assuming GRUB is properly configured, your system should now boot.

Note that if you've converted an MBR disk to GPT format, booting will fail even if you were previously using a GPT-aware version of GRUB. I'm not positive, but I believe this is because GRUB detects the

# GPT fdisk for Linux

Rod Smith

partition table format and sets itself up appropriately depending on what it finds. Re-installing a GPT-aware GRUB, as just described, will correct this problem.

Development on the original GRUB has officially ceased. A new boot loader, known as GRUB 2, is now under development instead. This boot loader includes GPT support; however, it's still officially under development. I've tried it briefly and after some initial user-error troubles, it seemed to work as well as legacy GRUB. Note that the GRUB 2 configuration file format is slightly different. The two versions of GRUB also identify partitions differently; GRUB numbers them starting from 0, whereas GRUB 2 numbers them starting from 1. Confusingly, GRUB 2 continues to number hard disks starting from 0, so that (hd0,1) is the first partition on the first hard disk.

Information on the old Linux Loader (LILO) and GPT is contradictory. Most sources say the two won't get along, but I've read others who opine that the combination does (or at least should) work fine, since LILO just uses sector maps to point to the kernel file. My one attempt at this combination proved inconclusive. LILO was able to load and run the kernel, but the boot then failed with the kernel message mount: could not find filesystem '/dev/root'. This message followed messages that indicated that the computer's LVM configuration was working fine, but somehow handing off to the LVM-based root filesystem was a problem. A GRUB boot of this system worked fine.

Beyond the boot loader, Linux requires GPT support in its kernel to work with GPT disks. This support is common, but there's no guarantee that any given kernel will have it. If you've compiled your own kernel, you can check on this detail by entering the kernel configuration utility (make xconfig or similar) and looking under File Systems -> Partition Types. Be sure that the EFI GUID Partition Support option is checked.

## FreeBSD

FreeBSD supports GPT, and can boot from it; however, this support is very kludgy, as of FreeBSD 7.2. To date, I have been unable to get FreeBSD to boot from a disklabel partition converted from an MBR disk, although the FreeBSD live CD can read that partition just fine. The FreeBSD installer also gets confused by GPT disks; it tends to treat them as MBR disks, which produces a corrupt MBR and a non-working installation.

I have successfully installed FreeBSD on GPT disks twice, but both times involved a conversion process. The simpler procedure, in broad outline, was as follows:

1. I installed FreeBSD as normal on an MBR disk. I ensured that no vital FreeBSD partitions were at the very start or very end of the disk, so as to avoid problems during the conversion stage. I also left a little free space on the disk.
2. I used gdisk to convert the disk to GPT format, including converting the BSD disklabel partitions. I then created a small (30-sector) GPT partition of type "FreeBSD boot" (0xA501 in gdisk). This partition will ultimately hold the GPT-aware FreeBSD boot loader. Note that this partition is not mounted (it's not for the /boot directory; it's more like FreeBSD's version of a BIOS Boot Partition). I created the boot partition immediately before the root partition in one test and immediately after it in another. I don't know if this ordering is critical, though.
3. I booted the FreeBSD live DVD and copied the /boot/pmbr and /boot/gptboot files from the FreeBSD installation (now in GPT form) to the temporary live DVD's filesystem. I then unmounted its filesystem.

# GPT fdisk for Linux

Rod Smith

4. I typed `gpt boot /dev/ad0`. You might need to change the device filename for your system and/or use the `-b` or `-g` options to point to the `pmbr` and `gptboot` files, respectively. This command installs FreeBSD's GPT-aware boot loader. The `pmbr` file is the MBR boot loader, while the `gptboot` file is the GPT-specific second-stage boot loader that ends up in the FreeBSD boot partition. (A key point is that this partition must be large enough to hold the `gptboot` file, but shouldn't be too much larger, since the whole thing is loaded into memory during boot.)
5. I edited `/etc/fstab` so that the system mounted the GPT partitions rather than the BSD disklabel partitions.

The system was then bootable and FreeBSD worked fine from its converted GPT partitions. To create a multi-booting system with FreeBSD and Linux (which I did on just one of my two tests), I had to jump through additional hoops:

1. Using a Linux live DVD, I copied the MBR from the GPT boot disk to a file, as in `dd if=/dev/sda of=sda.mbr bs=512 count=1`. I placed this file in my Linux `/boot` partition.
2. Using the Linux System Rescue CD, I re-installed GRUB on the boot disk. I also created an entry for FreeBSD:

```
title FreeBSD 7.1
root (hd0,2)
chainloader (hd1,5)/sda.mbr
```

In this case, `(hd0,2)` was GRUB's identifier for the FreeBSD root partition and `(hd1,5)` was the identifier for the Linux `/boot` partition where I'd stored the MBR copy.

The effect of this configuration is that, when I selected the FreeBSD entry from the GRUB menu, GRUB launched the copy of the MBR as if it were the boot sector on the disk. This code then took over and launched FreeBSD's native boot loader.

Note that the `gpt` utility is FreeBSD's GPT-handling tool. It includes the ability to create GPT partitions and it has its own MBR-to-GPT facility. This facility can nominally split out a BSD disklabel into GPT partitions; however, when I've tried this it's produced overlapping GPT partitions, so it seems to be buggy (or at least, the version shipped with FreeBSD 7.1 for x86-64 is buggy).

## NetBSD

I have not attempted to boot NetBSD from a pure-GPT disk. The NetBSD 5.01 installer assumes that the system is using MBR. (In fact, I had some install problems for a test installation that may have been related to some remnants of GPT data on the disk.)

I have discovered this page (see also another page on the same project), which describes a new (beta-level, as of July 2009) GPT-aware boot loader for NetBSD. In theory, it should be possible to follow a procedure similar to that described for FreeBSD, but using this new NetBSD boot loader, to get NetBSD to boot from a GPT disk.

## Windows

According to Microsoft's Windows and GPT FAQ, no version of Windows through Vista can boot from a GPT disk unless the computer uses EFI. To boot from a GPT disk, you need a version of Windows for the Itanium CPU or Windows Vista or Windows Server 2008 on EFI-based systems.

# GPT fdisk for Linux

Rod Smith

My own experiments at working around Microsoft's limitations have been unsuccessful, with one partial exception: It's possible to create a hybrid MBR, which is a modified GPT protective MBR that uses up to three of the MBR's primary partitions to point to up to three GPT partitions. (The remaining MBR primary partition contains an EFI GPT partition entry.) You can create a hybrid MBR using the `h` option on GPT fdisk's experts' menu, or you can use the separate `gptsync` utility, which ships with Fedora's `anaconda` package and sometimes in other packages with other distributions.

In my initial experiment, I installed Windows Vista on an MBR disk and converted it to GPT, whereupon Windows refused to boot. Using `gptsync` restored Windows' bootability; however, the configuration was extremely delicate. During my experiments, Windows became unbootable again and I was unable to restore it to bootability. I'm still not sure precisely what happened. In a later experiment, I used GPT fdisk to create a hybrid MBR configuration with Windows 7 RC and Linux. This setup has been more stable, although it required a lot of fiddling to set it up. (I was using this system as a testbed for GPT fdisk's hybrid MBR support.)

Unfortunately, Windows 7 (at least as of the Release Candidate, or RC, version) is no better than its predecessors in terms of GPT support. In my own tests, Windows 7 RC (x86-64 version) refuses to install on a GPT disk, at least on a BIOS-based system. Installing on MBR and converting to GPT renders Windows unbootable, even with the help of a patched GRUB 0.97 or GRUB 2 (Windows loads enough to complain and sulk like a recalcitrant toddler). Extensive Googling about this problem has turned up no solution. My guess (and it's only a guess) is that Windows remains chained, welded, and superglued to the BIOS for the initial parts of its boot process, and this is preventing it from using GPT. This really is stunningly short-sighted of Microsoft. Do they really expect that nobody will need to replace a failed hard disk on a BIOS-based computer during Windows 7's lifetime, and opt to install an over-2TiB drive? With any luck somebody will come out with a workaround at some point. (Hybrid MBRs aren't really a good option, although they can be barely adequate to achieve certain multi-booting goals.) In the meantime, the combination of GPT, a legacy BIOS, and Windows just isn't a good one.

The one ray of hope for booting Windows on a pure-GPT configuration (without a hybrid MBR) is to employ UEFI DUET. The basic idea is to have the BIOS boot the DUET software, which implements an EFI environment. This environment can then boot Windows from a GPT disk. This post on the InsanelyMac forum describes a procedure for converting an existing MBR Windows setup to boot from GPT; however, it requires either a BIOS with EFI support or a working UEFI DUET system. It will also reportedly work only with 64-bit versions of Windows Vista or Windows 7, not with Windows XP or any 32-bit version of Windows. To date, I've been unable to get this procedure to work because the UEFI DUET binary I was given hangs after it loads to a boot menu. With any luck somebody will come out with a simple EFI-based boot loader, similar to the ones used to boot Mac OS X on non-Apple hardware, that will enable relatively easy Windows installation to and booting from GPT disks. If you want to investigate the current state of this approach, check the Tianocore Web site, obtain the EFI Dev Kit (EDK), compile it, and deploy DUET on a floppy disk or USB flash drive.

## Mac OS X

As noted earlier, Intel-based Macs employ EFI and GPT by default. Thus, as you might expect, booting Mac OS from a GPT disk is not a problem. Although I've tested USB flash drives with GPT partitions on Mac computers, I haven't done extensive testing of boot disks manipulated by `gdisk` on Macs. Thus, I can't say if there are any caveats for this combination.

## Hybrid MBR Issues

Hybrid MBRs, described in more detail in the Hybrid MBRs section of this document, can create real boot headaches. This is because boot loaders can often become confused by hybrid MBRs. In my

# GPT fdisk for Linux

Rod Smith

experience, GRUB, GRUB2, and Chameleon can all do strange things, typically ignoring either the MBR or the GPT side of a hybrid configuration, even when the boot loader is nominally capable of handling either type. Thus, you may need to experiment to find a configuration that works for you.

## Hybrid MBRs: The Good, the Bad, and the So Ugly You'll Tear Your Eyes Out

GPT is a very flexible partitioning scheme with many advantages over the older MBR system. (For details, see the What's a GPT? section of this document.) GPT does have one glaring problem, though: compatibility. Some OSes, particularly older ones, have limited or no GPT support. When using such an OS, an ugly, flaky, and potentially even dangerous workaround can sometimes be useful: hybrid MBRs. Using a hybrid MBR, you can satisfy your legacy OS's need for up to three partitions defined via an MBR, while keeping additional partitions for more sophisticated OSes in GPT data structures.

### Hybrid MBR 101

As noted on other pages in this document, a conventional GPT disk contains a protective MBR with a single partition, of type 0xEE (EFI GPT), defined. This partition spans the entire size of the disk or 2TiB, whichever is smaller. The intent is to keep GPT-unaware OSes and utilities from trying to modify the disk.

A hybrid MBR is a variant on the normal protective MBR. A hybrid MBR contains a type-0xEE partition, but it also contains up to three additional primary partitions, which point to the same space that's marked out by up to three GPT partitions. For instance, suppose you've got a BIOS-based computer that triple-boots Windows, Linux, and FreeBSD. Linux and FreeBSD are happy on GPT, and so can use GPT partition definitions; but Windows is less capable in this respect. Thus, you'll define your partitions first as GPT partitions (including your Windows partitions), and then you'll modify your protective MBR so that its 0xEE partition is smaller than normal and it contains one to three partition definitions that point to the same disk locations as corresponding GPT partitions. You can then install Windows on these hybridized partitions. Alternatively, you can partition the disk as an MBR disk but leave some space unallocated, install Windows, convert it to GPT with `gdisk`, define additional GPT partitions, and then hybridize the converted disk to regain MBR-based access to the original MBR partitions. This configuration enables Windows (or other GPT-unaware OSes) to boot and use the hybridized partitions, while enabling more GPT-aware OSes to boot and use all of the GPT partitions, including both the hybridized and GPT-only partitions.

One important point is that hybrid MBRs are normally needed only on BIOS-based computers. If your system uses EFI, chances are you'll be installing EFI-aware, and therefore GPT-capable, OSes on it. Recent versions of Windows can boot from GPT disks on EFI-based computers. This wasn't true of some pre-Vista versions of Windows, though, which is why Apple uses hybrid MBRs on its EFI-based Macintoshes when they run Boot Camp to dual-boot with Windows.

### Dangers and Problems of a Hybrid MBR

The preceding description makes hybrid MBRs sound great: You get the best of both MBR and GPT by creating a hybrid MBR. Unfortunately, hybrid MBRs carry with them a lot of baggage. Some of these problems are mere annoyances, but other issues are potential dangers. They include:

- Hybrid MBRs are not part of Intel's EFI GPT standard. Thus, the details of how different OSes and utilities interpret them varies considerably. Some details follow, in the OSes' Reactions to Hybrid MBRs section. Some tools (such as GNU Parted, at least as of version 1.7.1) automatically convert the hybrid MBR to a conventional protective MBR, and it's conceivable that some current or future OS will react badly, too. Apple's Disk Utility has a bug that causes it to fail on disks with hybrid MBRs that contain MBR partition types it doesn't understand. There's no telling how some

## GPT fdisk for Linux

Rod Smith

random disk utility will react to a hybrid MBR; it's conceivable that something intended to rescue your data will end up destroying it. Boot loaders can sometimes interpret hybrid MBRs in unhelpful ways, too, as described in the upcoming section Booting from a Hybrid MBR.

- A limitation of hybrid MBRs is that they don't work well with MBR logical partitions. In theory, you could create a logical partition that in turn points to additional partitions, but coordinating the placement of these partitions between GPT and MBR would be a nightmare, from a programming perspective. I know of no utility that makes the attempt. Thus, you're limited to the three MBR partitions noted earlier. (That said, you can convert an MBR disk to GPT form and then create a hybrid MBR setup that hybridizes one or more former logical partitions; they'll become primary partitions, from an MBR point of view.)
- The protective EFI GPT (type 0xEE) MBR partition must normally begin with the second sector of the disk, in order to protect the primary GPT data structures. If it doesn't, Linux won't find any partitions on the disk, although FreeBSD, Mac OS X, and Windows seem to handle this situation better. This means that if you want to hybridize the first partition on the disk, the EFI GPT partition must be quite small for greatest compatibility. As a result, it's likely that at least some of your disk space will appear to be unallocated to MBR-only utilities, so part of the benefit of the protective EFI GPT partition is lost. An unwary user might try to create MBR partitions that re-use some disk space that's already in use by GPT-only partitions.
- If you have fewer than three partitions you want to hybridize, you may be able to work around the preceding problem by creating an additional protective partition; however, you might not be able to protect all the GPT-only disk space. Furthermore, if you create additional EFI GPT (0xEE) partitions, Mac OS X will treat the disk as an MBR disk rather than a GPT disk, so you may be forced to use some other type code to protect additional areas of the disk. This could be confusing.
- It's possible for the GPT and MBR partition sets to contradict each other. This is most likely to happen when using a GPT-unaware disk partitioning tool or a GPT tool that's unaware of hybrid MBRs. In such a case, you can easily end up with inconsistencies, in which the beginning of an MBR partition points to the middle of a GPT partition and vice-versa. GPT fdisk includes features to minimize this risk. Specifically, when you delete a GPT partition, any corresponding MBR partition is also deleted, and the protective EFI GPT (0xEE) MBR partition is expanded to fill the freed space, if this is possible. GPT fdisk also checks for GPT/MBR inconsistencies when you perform a verification function (v on the menu) and before saving data.
- A hybrid MBR doesn't work around the 2TiB limit for MBR partitions. On an over-2TiB disk, only GPT-aware OSes and utilities will be able to use the whole disk; anything that relies on the MBR side will be limited to 2TiB of disk space.
- The MBR and GPT partition numbers might not match. This could be confusing in some situations, and if your boot loader is inconsistent in its use of the MBR vs. the GPT side of things, it may cause boot problems.
- Although not a problem with hybrid MBRs per se, one issue with GPT fdisk's implementation is that this program doesn't create valid CHS entries for the MBR partitions; it creates only LBA data. Some old OSes and utilities might therefore not work with gdisk-created hybrid MBRs.

Overall, hybrid MBRs should be avoided. I've included support for them in GPT fdisk only because certain popular OSes (read: Microsoft Windows, at least through Windows 7 RC) can't boot from GPT disks on BIOS-based computers. (See the Booting from GPT page for a possible workaround,

# GPT fdisk for Linux

Rod Smith

though.) This fact makes hybrid MBRs a practical necessity in some environments. Apple's Boot Camp, for instance, uses hybrid MBRs to enable Macintoshes to dual-boot both Mac OS X and Windows; and if you want to dual-boot Windows and anything else on a disk of more than 2TiB on a BIOS-based system, you'll need a hybrid MBR (or perhaps some other solution I don't know about). Older non-Windows OSES (BeOS, OS/2, DOS, etc.) are likely to require a hybrid MBR to handle GPT disks, too, although I haven't tested them explicitly.

## Creating a Hybrid MBR

If you've read this far, you're presumably interested in creating a hybrid MBR. In the past, the `gptsync` tool was the only way I knew of to create hybrid MBRs. This program is hard to track down on Linux (most distributions don't include it, although it's part of the `anaconda` package on Fedora) and it has certain limitations. For instance, it blindly converts the first three GPT partitions to MBR form, and it chokes when it sees partition types it doesn't understand. GPT fdisk is more flexible in both respects, but it requires more user interaction. Hybrid MBR support was added to GPT fdisk 0.3.2, so if you're using an older version you should upgrade before trying.

The procedure for creating a hybrid MBR is best illustrated with an example:

```
# gdisk /dev/sdc
GPT fdisk (gdisk) version 0.3.3

Found valid GPT with protective MBR; using GPT.

Command (m for help): x

Expert command (m for help): p
Disk /dev/sdc: 15654912 sectors, 7.5 GiB
Disk identifier (GUID): 846EC387-94E5-09A7-D55A-F1A25F04631F
Partition table holds up to 128 entries
First usable sector is 34, last usable sector is 15654878
Total free space is 0 sectors (0 bytes)

Number  Start (sector)    End (sector)  Size      Code  Name
   1            34                3072033     1.5 GiB   AF00   Mac (HFS)
   2          3072034          7266337     2.0 GiB   0700   Linux (ext2)
   3          7266338          15654878     4.0 GiB   0700   FAT
```

The hybrid MBR function is located on the experts' menu, so after launching `gdisk`, you should type `x` to enter that menu. You'll also need to know partition numbers, so you should probably type `p`, as shown in the example. The example disk (a USB flash drive) has three partitions, containing HFS, ext2, and FAT filesystems.

```
Expert command (m for help): h

WARNING! Hybrid MBRs are flaky and potentially dangerous! If you decide not
to use one, just hit the Enter key at the below prompt and your MBR
partition table will be untouched.
```

The command to create a hybrid MBR is `h`. Typing that command causes a warning to be displayed before `gdisk` prompts for partition information.

# GPT fdisk for Linux

Rod Smith

```
Type from one to three GPT partition numbers, separated by spaces, to be
added to the hybrid MBR, in sequence: 1 3
Place EFI GPT (0xEE) partition first in MBR (good for GRUB)? (Y/N): y
```

You specify the partitions you want to hybridize by listing their numbers, separated by spaces. In this example, I've hybridized partitions 1 and 3 (the HFS and FAT partitions). They'll become MBR partitions 1 and 2 or 2 and 3, depending on the answer to the next question.

GPT fdisk can create the EFI GPT (0xEE) placeholder partition either before or after the hybridized partitions in the MBR table. (Note that this has nothing to do with the disk sectors this partition protects.) Each placement has its advantages. Putting the 0xEE partition first in the table works best with GRUB and GRUB2, which treat the disk as an MBR disk if the first slot in the MBR is not a 0xEE partition. This first-position placement of 0xEE, however, can render Windows unable to read subsequent partitions if the disk is a removable disk, such as a USB flash drive. Overall, if the disk is a hard disk, I recommend putting the 0xEE partition first; if it's a removable disk, putting it later in the table may work better.

```
Creating entry for partition #1
Enter an MBR hex code (suggested AF): af
Set the bootable flag? (Y/N): y

Creating entry for partition #3
Enter an MBR hex code (suggested 07): 0c
Set the bootable flag? (Y/N): n

Unused partition space(s) found. Use one to protect more partitions? (Y/N): y
Enter an MBR hex code (EE is EFI GPT, but may confuse MacOS): 0a
```

The program now prompts you for some information about each partition. In each case, gdisk asks you for an MBR hex code. Although the program could convert automatically, the ambiguity surrounding Linux/Windows data partitions (GPT fdisk type code 0700) makes explicit prompting at this point more sensible than a blind conversion. In this case, since the second hybridized partition (#3 in GPT) holds a FAT filesystem, I chose to use a type code of 0x0C. Note that you must enter a two-character hexadecimal value, without a leading "0x" indicator. There's no default value, so pressing the Enter key alone will give the partition a type of 0x00, which is an error. (This is a known bug in version 0.3.3; it will be fixed in a future release.) You're also asked if you want to set the bootable flag for each partition. Ordinarily, you'd set this flag on Windows boot partitions, but not on data partitions. (At least one boot loader behaves strangely if you set this flag, though.) For illustrative purposes, I've set this flag on one partition.

Because I only hybridized two partitions, there was one unused space in the partition table, meaning that GPT fdisk could add one more MBR partition to protect free space from accidental modification. The program will fill the largest empty (to MBR) space on the disk with the partition it creates. You're asked if you want to do this, and in this example I responded in the affirmative. The program then asks for the MBR hex code for the created partition. Although 0xEE is a logical choice, using it will cause Mac OS X to interpret the disk as an MBR disk, thus removing the benefits of GPT for that OS. In this example, I used 0x0A, which is the code for an OS/2 Boot Manager partition. My reason for using this code is simply that few disk utilities will attempt to do anything with such a partition. Many other codes will work as well, but all non-0xEE codes will be misleading or ambiguous. Note that Apple's Disk Utility has a bug that renders it unable to manipulate disks with unknown MBR type

## GPT fdisk for Linux

Rod Smith

codes, so you should be careful with what you select, or simply don't create this extra partition. In my tests, the 0x0A value worked, but 0x77 did not.

One drawback to using a non-0xEE extra protective partition is that gdisk's own data integrity checks may flag this partition as a problem. In particular, if the partition spans multiple GPT partitions or covers an area that doesn't correspond to any GPT partition, gdisk will complain when you save your partitions or perform a disk verification (via the v menu option). If your spare protective partition is of type 0xEE, though, gdisk won't complain. Overall, if you're using Mac OS X, it may be best to forego this partition. If you use Linux, FreeBSD, or Windows, but not Mac OS X, I know of no reason not to create an extra 0xEE partition.

Note that even with the extra protective partition, significant parts of a disk could go unprotected. In the case of this example, it's just a few sectors at the end of the disk; however, if you hybridize two non-contiguous partitions, the last of which is not at the end of the disk, either the partitions at the end of the disk or the space between the partitions will be unallocated in an MBR sense. If you hybridize three partitions, it's likely that even more space will be unallocated in the MBR scheme. On a sub-2TiB disk, you can maximize the protected space by placing the partitions to be hybridized at the end of the disk. That way, the 0xEE partition will span from sector 2 all the way up to the first hybridized partition.

Before writing your changes, you should check the status of the hybrid MBR:

```
Expert command (m for help): o
MBR disk identifier: 0x00000000
MBR partitions:
Number  Boot   Start (sector)  Length (sectors)  Type
  1                1                33                0xEE
  2      *      34              3072000            0xAF
  3                7266338          8388541            0x0C
  4                3072034          4194304            0x0A

Disk size is 15654912 sectors (7.5 GiB)
```

Typing o on the experts' menu displays the MBR data, and you can see the hybrid partition definitions.

**Caution:** The o command, when typed at the main menu, creates a fresh GPT, including a new protective MBR.

If you compare the start and length values to the start and end values for the GPT partitions, you should see that they match for the hybridized partitions. (You'll need to convert length to end value, of course: End value is start value plus length minus 1.) The EFI GPT (0xEE) partition covers the GPT data structures, and in some cases it can cover more. If you don't hybridize the partition that comes first in disk order, the EFI GPT partition will cover its space, for instance. In this example, the extra 0x0A partition covers the area of the second GPT partition, since that was the single largest area that was unallocated in the MBR scheme. In other cases, you might have an MBR partition that spans multiple GPT partitions or that just covers a bit of unused space at the end of the disk (protecting the secondary GPT data structures and perhaps one or more partitions at the end of the disk). If you hybridize only one GPT partition and choose to create the extra protective partition, GPT fdisk will be able to protect all the available disk space in the combination of the regular 0xEE partition, the extra protective partition you create, and the hybridized partition itself.

# GPT fdisk for Linux

Rod Smith

If necessary, you can modify the hybrid MBR with Linux's fdisk or other GPT-unaware partitioning tools. GPT-aware programs, however, are likely to modify the GPT side alone, or perhaps change the hybrid MBR back into a standard protective MBR, as GNU Parted does. Because GPT doesn't use CHS addressing, you're likely to find that your partitions don't begin or end on cylinder boundaries, as viewed by fdisk or other MBR tools. This isn't a problem with modern OSes, but if you need to boot a truly ancient OS, such as DOS, you may not be able to use even the hybrid MBR partitions. GPT fdisk makes no effort to create valid CHS partition data in its hybrid MBR, so any OS or utility that relies on this data won't work properly with the hybrid MBR.

Note that the MBR entries may not be in order. In the preceding example, the entry for the 0x0A partition comes at the end of the table, although its disk area is between the two real partitions. If you choose to place the 0xEE partition's entry after the main partitions' entries, the order can become even more confused. For the most part, these deviations won't cause problems, although some utilities have specific quirks, as already noted.

## OSes' Reactions to Hybrid MBRs

In testing hybrid MBRs, I've found that different OSes react to them in different ways. Further complicating matters, the boot loaders you use to boot your OSes have their own quirks, as detailed shortly, in Booting from a Hybrid MBR. Unfortunately, these differences can make use of a hybrid MBR a complex and frustrating task. OS-by-OS peculiarities include:

- **Linux** -- Linux seems to require that at least one 0xEE (EFI GPT) partition cover at least part of the GPT data structures. Linux isn't fussy about which partition (1-4) this is, though. In practice, this means that if you hybridize the earliest partition on a disk, you'll end up with a very short EFI GPT partition, as in the preceding example. Note that gptsync also starts its 0xEE partition at the beginning of the disk, so it creates a Linux-compatible hybrid MBR. If you adjust the hybrid MBR with fdisk so that the 0xEE partition covers a later area of the disk, Linux becomes unable to find any partitions on the disk.
- **FreeBSD** -- My testing of hybrid MBRs with FreeBSD has been fairly minimal; however, I do know that FreeBSD handles hybrid MBRs created by both gptsync and gdisk. Unlike most OSes, FreeBSD provides access to partitions on a hybrid MBR disk using both the MBR and the GPT systems. On a test USB flash drive, GPT-style partitions were named /dev/da0px, while MBR-style partitions were named /dev/da0sy, where x and y are partition numbers. In basic tests, FreeBSD 7.1-RELEASE has shown no problems with disks with multiple 0xEE MBR partitions (unlike Mac OS X) or with a single 0xEE partition that does not begin at the very start of the disk (unlike Linux). Thus, overall it appears that FreeBSD may have the best hybrid MBR support -- although I'm concerned that the multiple device files pointing to single partitions could cause confusion or even data corruption if somebody attempted to access them simultaneously.
- **Mac OS X** -- Mac OS X can handle just about anything in the way of hybrid MBRs, including some configurations (which GPT fdisk doesn't create) that give Linux and/or Windows fits. There's one important exception, though: More than one partition of type 0xEE causes Mac OS X to treat the disk as an MBR disk rather than a GPT disk. This is the reason that GPT fdisk prompts you for the type of any additional protective partitions, as shown earlier: You may use 0xEE if you don't intend to use the disk with Mac OS X, or another code if the disk might be accessed by Mac OS X. Although it's not critical to basic Mac OS X access to your disks, Disk Utility's problem with unusual MBR partition type codes should be kept in mind when creating hybrid MBRs. Fortunately, you can easily change a type code if it becomes necessary.

# GPT fdisk for Linux

Rod Smith

- **Windows** -- When confronted with a hybrid MBR, Windows tries to treat the disk as an MBR disk, ignoring the GPT partitions completely. At least, that's what I've found so far. This limitation means that in a dual-boot environment, you must hybridize all of your Windows partitions, and none of those partitions should span the 2TiB mark on the disk, if the disk is larger than that. (You can use GPT with a conventional non-hybridized protective MBR on a second physical disk, at least with recent versions of Windows.) Windows treats removable disks as "superfloppies," meaning they should either not be partitioned or they should have a single partition. Thus, if you hybridize a USB flash drive, Zip disk, or similar medium, you should be sure to put the EFI GPT (0xEE) partition after the hybridized partitions. Furthermore, you should put whatever partition Windows must access first in the partition table, by specifying its number first in the list when GPT fdisk asks for partition numbers to hybridize. Another important Windows issue is setup order: If you install Windows, convert to GPT, and then create a hybrid MBR, it's conceivable that Windows will refuse to boot. (This happened to me in my tests.) The Windows install disc may be able to restore your system to bootability (it couldn't for me), but you may then need to restore any non-Microsoft boot loader you're using. It's probably safest to install Windows to an already-hybridized disc -- although again, you'll then need to re-install your multi-OS boot loader. Note that I've done my testing on Windows 2000 and Windows 7 RC on x86 and x86-64 platforms. I don't know how Windows Me or earlier, or as-yet-to-be-released versions of Windows, will handle things. I also don't know if these rules might be different on EFI-based systems rather than BIOS-based systems, or on non-x86/x86-64 platforms.

I don't know how other OSes, such as BeOS, OS/2, DOS, Solaris, or NetBSD, react to hybridized MBRs. If you have problems with such an OS and the hybrid MBRs created by GPT fdisk, you might want to give gptsync a try. It does things a bit differently, so it's conceivable that one tool might work better in some environments and the other will work better in others. There may also be problems because of the non-existence of CHS-based partition entry data in the hybrid MBR created by GPT fdisk. I would expect this problem to affect only very old OSes, though, since any disk over 8GiB requires LBA data to be fully accessible.

## Booting from a Hybrid MBR

Hybrid MBRs are often used on boot disks with multiple OSes, and so the issue of bootability is an important one. Unfortunately, this is an area that's even more poorly documented than the murky area of booting GPT disks generally. I've done some experimenting, but I don't claim to have all the answers. Some information I consider reliable includes the following:

- **GRUB and GRUB2** -- These workhorse boot loaders from the Linux world work well with both MBR and GPT disks (although you need a patched GRUB 0.97 or GRUB2 for GPT support). They appear to be very fussy about hybrid MBRs, though. In particular, if the EFI GPT (0xEE) partition resides anywhere but in the first position of the MBR partition table, these boot loaders treat the disk as if it were MBR-only. The likely result is that some or all of your OSes won't boot. For this reason, if you expect to use GRUB or GRUB2 with a hybrid GPT, I suggest you place the EFI GPT partition first in the partition table. One caveat: My tests with these boot loaders were on a system with a BIOS Boot Partition defined. It's conceivable they'd work differently without that partition in place. You should also be aware that, when you place the EFI GPT partition first in the MBR table, you should use the GPT partition numbers for your hybrid partitions, even if the OSes they contain are GPT-unaware. These identifiers are for the use of GRUB/GRUB2 only. Once the OS boots, it will read the MBR and figure out its own way of referring to partitions.
- **Chameleon** -- This is a popular boot loader among those who run Mac OS X on standard PC hardware. It tends to boot straight through to Windows, however, if you mark a Windows partition as bootable in the MBR. Fortunately, Windows (at least, Windows 7 RC) seems to be able to boot

# GPT fdisk for Linux

Rod Smith

without this flag set, so the solution is simple: Don't set the bootable flag in the MBR, or remove it if it's already set.

- **Windows MBR & BCD** -- Windows' default method of booting is as old as the MBR hills: It involves an MBR boot loader that in turn loads the boot sector of the partition that's marked as active in the MBR. This boot sector then loads additional boot files, which are known today as the Boot Configuration Data (BCD). This configuration works just fine on a hybrid MBR, provided that you only want to boot one partition and it's marked as active. Some third-party utilities, such as EasyBCD, extend this system by enabling BCD to boot other partitions. I don't know if any such tools enable booting GPT-based OSes, though.

Unfortunately, the non-standard nature of hybrid MBRs means that you'll be venturing into uncharted territory when you use this type of configuration with most tools. Generally speaking, Mac OS X users seem to be the most experienced with such setups, thanks to Apple's Boot Camp and the OSx86 (OS X on stock PC) communities.

Most OSes try to install their own boot loaders to the MBR when you install them, at least on BIOS-based computers. Thus, when you install multiple OSes, you often end up overwriting the earlier OSes' boot loaders. You can generally use GRUB or some other flexible boot loader in the MBR to redirect the boot process to any OS; however, sometimes this doesn't work. One trick that can be helpful is to back up the MBR, or at least its boot code (the first 440 bytes) after every OS installation. In Linux, you can use `dd` to do the job:

```
# dd if=/dev/sda of=/boot/sda-mbr-backup-1.img bs=512 count=1
```

With a backup of the MBR's boot loader from after each OS installation in hand, you can try redirecting the boot process through such a backup file, if you can't find a more direct way to boot the OS. In GRUB, the configuration would look like this:

```
title OpenSolaris 2008.11 captured MBR
    root (hd0,1)
    chainloader (hd1,4)/hda-solaris-mbr.img
```

This example is taken from an actual OpenSolaris installation on one of my computers, although the installation in question doesn't use a hybrid MBR. The details will vary with boot loaders other than GRUB, of course.

## Hybrid MBR Strategies

As I've said before, hybrid MBRs are clunky, ugly, and dangerous. If at all possible, I recommend you avoid them. If possible, use two disks rather than a hybrid MBR. Place your GPT-unaware OSes on the MBR disk and use GPT on the other disk. Recent versions of Windows (Vista and later, and some earlier versions, depending on platform) can read GPT disks just fine once booted; they just can't boot from GPT. This fact gives you an "out" if you need to use large disks or employ GPT for another OS's benefit, since you can boot Windows from an MBR disk and store data on a separate GPT disk. Some RAID controllers enable you to configure a RAID array as two or more virtual disks, which you can then partition using MBR and GPT.

Sometimes, though, you might really have no alternative but to use a hybrid MBR. This might be necessary on a laptop computer on which you want to install lots of OSes, for instance. In such cases, I recommend you treat your GPT configuration as the standard one. If possible, create the initial partitions using GPT fdisk or some other GPT partitioning tool; converting from MBR to GPT and then

# GPT fdisk for Linux

Rod Smith

hybridizing presents more opportunities for problems to crop up. (On the other hand, creating MBR partitions can more easily create them on cylinder boundaries, which some OSes may appreciate.) If you run into problems or need to repartition, restore a standard GPT protective MBR (using the `n` option on GPT fdisk's experts' menu), do whatever you need to do to your partitions, and then create a fresh hybrid MBR.

In my experience, Windows doesn't react well to changes in its partitions. (This has been true for many years; it's not a GPT-specific issue.) Thus, if possible you should install Windows after you create hybrid MBR partitions for it. Plan to re-install your MBR-based boot loader after installing Windows.

On a sub-2TiB disk, place the partitions you must access via MBR at the end of the disk, to enable the EFI GPT (0xEE) partition to protect the non-MBR partitions. If you can keep your hybridized partitions contiguous, and numbered at just 1 or 2, you'll also be able to create a second protective partition to keep the secondary GPT data safe from meddling by GPT-unaware utilities.

On an over-2TiB disk, having just 1 or 2 hybridized partitions becomes even more important, since you'll then be able to create a second protective partition after those partitions. Alternatively, you could place your to-be-hybridized partitions so that they're contiguous and terminate as close as possible to, but before, the 2TiB mark. This layout will protect the preceding GPT partitions (via the 0xEE MBR partition), and the subsequent GPT partitions will be protected by virtue of falling after the MBR's 2TiB limit.

Discussion forums are filled with posts about hybrid MBRs, mostly from people using Boot Camp with Mac OS X. If you run into problems, try posting to such a forum. Although I may be able to help, I hope to minimize my own use of hybrid MBRs, and my impression is that hybrid MBR problems can be quite quirky and configuration-dependent, so your best bet in getting help is to post your problems in a well-read forum.

## GPT fdisk for Linux

GPT fdisk is available via its Sourceforge page. This is the preferred method of downloading the program. If that fails, though, you can try the following direct link to obtain the source code:

[gdisk-0.5.0.tgz](#)

The Sourceforge page includes binary RPMs and Debian packages for x86, x86-64, and PowerPC, as well as a binary package for Mac OS X.

GPT fdisk is currently beta-quality software! Although I know of no data-corruption bugs in the current version, I can't guarantee that the program won't damage your system. I welcome bug reports and reports of successful use of the program, particularly if you're using it on disks (or hardware RAID arrays) over 2 TiB in size.

## Revisions

The following summarizes changes in GPT fdisk's public releases:

- **0.5.0** -- The bump to version 0.5.0 mostly reflects a new menu structure, which includes the main menu and the experts' menu from previous versions, along with a recovery & transformation menu. Other new features include a new GPT-to-MBR conversion, the ability to reload an MBR as a way of creating a new GPT, and generation of valid CHS values when creating hybrid MBRs or in the

# GPT fdisk for Linux

Rod Smith

new GPT-to-MBR conversion. This version also incorporates a few bug fixes, including a fix for one that could cause data loss when using the "zap" function in version 0.4.2.

- **0.4.2** -- The only significant new feature in this version is the ability to work on disk image files as well as actual disks. This feature is intended to enable editing of disk images used in emulators, although you'll only be able to work on "raw" disk images, not the more advanced formats used by QEMU, VMWare, etc. This version also includes a number of minor bug fixes and feature improvements, such as additional prompts or warnings in certain potentially destructive commands.
- **0.4.1** -- Two improvements appear in this version: First, the method of specifying partition start and end points has been expanded to enable both absolute and relative sector specifications. See the Walkthrough page for details. Second, hybrid MBR synchronization has been improved. If you delete a GPT partition for which a corresponding MBR partition exists, the corresponding partition is deleted and, if possible, the protective 0xEE (EFI GPT) partition is expanded to fill the newly freed disk space. MBR/GPT correspondence checks have also been added to the verify (v on the main menu) command and before writing disk changes.
- **0.4.0** -- This version marks some major extensions to GPT fdisk: It now runs correctly on big-endian systems (an ancient PowerPC running Linux is my test platform) and it compiles and runs correctly on FreeBSD. I've also added support for converting BSD disklabels to GPT format -- but BSD disklabels are tricky, so use that feature cautiously. (See the updated Converting to GPT page for details.) New partitions are now given a default name based on the partition type. This version also includes several smaller bug fixes and some code reorganization.
- **0.3.4** -- Fixed a few minor bugs, including one that enabled accidental entry of MBR partition type 0x00 during hybrid MBR creation and another that prevented acceptance of default end sector number under certain conditions when creating new GPT partitions. Added Debian binary packages to distribution set.
- **0.3.3** -- This version gives the user the ability to specify whether to create the EFI GPT (0xEE) partition before or after other partitions in a hybrid MBR configuration.
- **0.3.2** -- This version adds a couple of new features, namely the ability to create a hybrid MBR and the ability to clear the GPT data structures from a disk (in case you want to re-use it as an MBR or some other type of disk). See the Hybrid MBRs section of this document for details on that feature.
- **0.3.1** -- This version fixes a few bugs, including one that could cause a segfault under specific circumstances. It also adds Mac OS X support, thanks to two anonymous contributors.
- **0.3.0** -- This version added the ability to enter "raw" GUIDs as a single string rather than as five separate numbers. It also fixed a few very minor bugs. I bumped up the second version number from 2 to 3 and called this version the first beta release because I'd received several reports of successful uses of earlier versions of the program and no bug reports, aside from one compilation issue. I also added precompiled RPMs for 32-bit (Fedora 11) and 64-bit (OpenSUSE 11) distributions.
- **0.2.2** -- Fixed a compilation problem with GCC 4.4.0.
- **0.2.1** -- This was the first version released on Sourceforge and on my own Web page. It fixed a number of minor bugs, added a few partition type codes, and cleaned up the code a bit.

# GPT fdisk for Linux

Rod Smith

- **0.2.0** -- This was the first version to be shown to anybody aside from myself.

Additional details can be found in the CHANGELOG file in the main tarball (or other package files).