UNIX & LINUX

MBR converted to GPT based system, how to boot Linux and Windows

Asked 2 years, 3 months ago Modified 2 years, 3 months ago Viewed 1k times

2

How to boot GPT based system to Linux and Windows? This is not a question of starting from a fresh GPT based system, but starting from a MBR converted to GPT based system.

My Asus laptop initial setup,

- I disabled the Secure Boot Control, and
- I enabled `[Launch CSM]` (Compatibility Support Module)
- I partitioned my HD using MBR
- All my systems on my Asus laptop were boot from such BIOS/MBR/CSM mode, including Win8 and all my Linux

However, I found that my USB is booted only in EFI style, and Windows 10 is refusing to be installed to my BIOS/MBR/CSM mode system when my USB is booted in EFI style.

So I converted my MBR disk to GPT, and of course, as Krunal warned, such practice ruined my system boot, and I need make everything bootable again.

Alright, so now is my question.

- In BIOS/MBR/CSM mode, I have an active MBR partition, all my systems were boot from it (using `extlinux`), including Win8 and all my Linux.
- In GPT mode, however, this is where the problem begins for me.
  - To mark a GPT Partition bootable under Linux, I saw I need to set the "boot flag" in `GParted`, which I did (marked my previous active MBR partition as type code of `EF00` which stands for "EFI System"), and `GParted` consequently set `ESP` (EFI System Partition) flag as well.

- However, according to GUID Partition Table (GPT) specific instructions from wiki.archlinux, I need a type code `ef02` flag `bios_grub` in order to boot. But I don't have any spare room for such partition.

So I basically don't know which way to go, and don't want to further mess up with my already-messed-up and unbootable system.

My current partition schema (I didn't think it is relevant but since somebody asked for it):

```
Disk /dev/sda: 698.65 GiB, 750156374016 bytes, 1465149168 sectors
Disk model: HGST HTS541075A9
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disklabel type: gpt
Disk identifier: AA9AB709-8A5D-468D-990E-155BA6A2FBB3

Device          Start        End   Sectors   Size Type
/dev/sda1        2048  129026047 129024000  61.5G Microsoft basic data
/dev/sda2  129026048  169986047  40960000  19.5G EFI System
/dev/sda3  169988096  186372095  16384000   7.8G Linux filesystem
/dev/sda4  186374144  200710143  14336000   6.9G Linux filesystem
/dev/sda5  200712192  215046143  14333952   6.9G Linux filesystem
/dev/sda6  215048192  231432191  16384000   7.8G Linux filesystem
/dev/sda7  231434240  247818239  16384000   7.8G Linux filesystem
/dev/sda8  247820288  264204287  16384000   7.8G Linux filesystem
/dev/sda9  264206336  276494335  12288000   5.9G Linux filesystem
/dev/sda10 276496384  288784383  12288000   5.9G Linux filesystem
/dev/sda11 288786432  329746431  40960000  19.5G Linux filesystem
/dev/sda12 329748480  452628479 122880000  58.6G Microsoft basic data
/dev/sda13 452630528  493590527  40960000  19.5G Linux swap
/dev/sda14 493592576  903192575 409600000 195.3G Linux filesystem
/dev/sda15 903194624 1465147391 561952768   268G Linux filesystem
```

So clear instructions on how to boot Linux (and Windows) from such environment is really appreciated. Thanks.

UPDATE/Conclusion:

`/dev/sda2` was my active MBR partition before, and as explained above, I changed its type from `Linux filesystem` to `EFI System`. But it is in `ext4` format so it cannot be used as an EFI System.

So, all I need to fix my above problems are:

- revert `sda2` 's type back to `Linux filesystem`,
- split out a FAT32 partition as the ESP partition from my `sda13` 19.5G Linux swap,
- change the firmware from CSM to native EFI mode
- then follow advice below from @telcoM

boot dual-boot gpt mbr asus

Share Follow edited Sep 5, 2021 at 20:52 asked Sep 4, 2021 at 14:04

xpt
1,376 3 16 36

Please would you add the current partition schema to your question? `sudo fdisk -l` or `cat /proc/partitions` – Lutz Willek Sep 4, 2021 at 14:08

Hmm... It's MBR converted to GPT, and I saw a bunch of GPT partition (15 of them). So @LutzWillek, what would you like to know, and what specific output of `sudo fdisk -l` or `cat /proc/partitions` that you're interested? The machine is currently unbootable and running from a live system. I need to manually re-type them here. That's why I want to know why it matters and what exactly you'd be looking at. thx – xpt Sep 4, 2021 at 14:17

I can show you what my MBR partitions looked like before @LutzWillek, if you think it'll help. – xpt Sep 4, 2021 at 14:18

1 @JohanMyréen Independent *for Linux* perhaps, but unfortunately Windows ties those choices together: you must boot Windows on GPT system disk using UEFI style, and on MBR system disk using legacy BIOS style. As this will be a dual-boot system, the use of a bios_grub partition (and thus a legacy BIOS version of GRUB) is in my opinion a poor recommendation in this case. – telcoM Sep 4, 2021 at 19:58

1 The Windows active flag or in Linux seen as boot flag is used by Windows to know which partition has Windows boot files. Grub does not use boot flag, but a few UEFI/BIOS require one even if only Linux system. Some Linux BIOS boot loaders like Lilo also need boot flag. The boot flag with an ESP is really to assign a very long GUID which identifies that FAT32 partition as the one with UEFI boot files. All systems put boot files in separate folders in the ESP. Somewhat like having multiple drives with different boot loaders. en.wikipedia.org/wiki/GUID_Partition_Table – oldfred Sep 4, 2021 at 20:11

1 Answer

Sorted by: Highest score (default)

3

As far as I know, Windows does not handle such a converted system disk as a special case once the conversion is done, so it should be treated exactly the same as a disk in a "fresh" GPT-based system.

In particular, Windows imposes the limitation that GPT-partitioned system disks must always boot Windows in UEFI native style: that is, using BIOS-style boot process on GPT-partitioned disks is not allowed.

First, a primer on differences between MBR and GPT partitioning:

- In GPT, there is no division to primary/extended/logical partitions like MBR partitioning has. All GPT partitions are just partitions.
- Although there is a "legacy BIOS bootable" partition attribute bit in GPT partition table, *it is not used at all when booting in UEFI style*.
- MBR-partitioned disk normally has a gap of unused disk blocks between the block #0 (the actual Master Boot Record) and the beginning of the first partition. On modern systems, the first partition is usually aligned exactly 1 MiB from the beginning of the disk, so if the common 512-byte blocks are used, the first partition will begin at block #2048. The gap between MBR and the first partition is used by bootloaders like GRUB. On GPT-

partitioned disks, this area is occupied by the actual GPT partition table and cannot be used.

- On MBR-partitioned disks, partition type is identified by a single byte. On GPT-partitioned disks, the type of each partition is identified by an UUID.
- MBR-partitioned disks have a 32-bit *disk signature*; GPT-partitioned disks have a 128-bit UUID for the same purpose. Each GPT partition also has an unique UUID in the partition table: it can be used to uniquely identify a partition even if the filesystem used in it is unknown. Linux displays this as a `PARTUUID` ; for MBR-partitioned disks, a combination of the MBR disk signature and partition number is used in lieu of a real partition UUID.
- The MBR partition table exists in block #0; if extended partitions are used, the beginning of each logical partition has an add-on partition table. The GPT partition table starts in block #1 and occupies multiple blocks; there is also a *backup GPT partition table* at the very end of the disk. This often causes a surprise if you are used to wiping the partitioning off a disk by just zeroing a number of blocks at the beginning of a disk only.

Since partition type UUIDs are inconvenient for humans to use, different partitioning programs have used various methods to shorten them. `gdisk` will use four-digit type codes; Gparted represents the different partition type UUIDs by various flags (which is, in my opinion, an unfortunate choice).

A native UEFI-style boot process is also very different from classic BIOS-style boot process:

- A BIOS-style boot process begins by (usually) assigning the BIOS device ID 0x80 (=first BIOS hard disk) to the device that is currently selected by the BIOS settings as the boot drive. When booting in UEFI style, the firmware settings ("BIOS settings" in an UEFI system) define a *boot path*: it can take many forms, but the most common one for installed operating systems will specify a *partition UUID* and a *boot file pathname*.
- When booting BIOS-style, the firmware checks for a 2-byte boot signature at the end of block #0 of the selected boot disk, and then just executes the about 440 bytes of machine code that fits in the MBR block in addition of the actual partition table. When booting UEFI-style, the firmware has a built-in capability to understand some types of *filesystems*: the UEFI specification says a compliant UEFI firmware **must** understand FAT32, but it may understand other filesystem types too. An UEFI "bootable disk" must contain a partition with a special type UUID: this is called the EFI System Partition, or ESP for short. The firmware will look for an ESP partition whose unique UUID matches the one specified by the boot path, and then attempts to load the specified boot file from that partition.
- When booting UEFI-style from a removable media, or from a disk that has not previously been configured to the firmware settings, the firmware looks for an ESP partition that contains a filesystem the firmware can read, and a file with a particular pathname. For 64-bit x86 hardware, this *UEFI fallback/removable media boot path* will be `\EFI\boot\bootx64.efi` when expressed in Windows-style, or `<ESP mount point>/EFI/boot/bootx64.efi` in Linux-style.

The ESP partition has a standard structure: each installed OS must set up a sub-directory `\EFI\<vendor or distribution name>\` and only place their bootloader files within it. The `\EFI\boot\` sub-directory is reserved for the fallback/removable-media bootloaders, which follow the *Highlander* rule: *there can be only one* (for each system architecture).

By setting the GParted "boot flag" on a non-ESP partition, you effectively changed the type UUID of that partition to ESP type UUID. That was a mistake: now the disk has two partitions with type ESP. You should change the type of the partition you changed back to what it originally was. In GParted, that would mean removing the "boot" and "esp" flags; in `gdisk`, it would probably mean setting the type code to `8300` ("Linux filesystem") or perhaps `8304` ("Linux x86-64 root").

Since you also have Windows on the same disk, trying to use a BIOS-Boot partition (`gdisk` type code `ef02`) is not recommended: that would usually force you to go to firmware settings and enable/disable CSM each time you wanted to switch between operating systems. Instead, you would want to use the live Linux boot media to mount your on-disk installation to e.g. `/mnt`, and then chroot to it to replace the current BIOS-style bootloader (usually GRUB with the `i386-pc` architecture type) with a native UEFI one (e.g. GRUB with `x86_64-efi` architecture type). Basically (all the following commands as `root`):

```
mount <your root filesystem device> /mnt
mount -o rbind /dev /mnt/dev
mount -t proc none /mnt/proc
mount -t sysfs none /mnt/sys
chroot /mnt /bin/bash
```

Now your session will be using the environment of your installed Linux OS, and you should be able to use package manager and any other tools pretty much as usual (caveat: if you have parts of the standard system like `/var` as separate partitions, mount them now too!)

The first step should be adding a mount point for the ESP and mounting it. First run `lsblk -o +UUID` to find the UUID of your ESP partition; since its filesystem type is most likely FAT32, it should be of the form `xxxx-yyyy`. Replace `<ESP UUID>` in the following commands with the actual UUID:

```
mount UUID=<ESP UUID> /boot/efi
echo "UUID=<ESP UUID> /boot/efi vfat umask=0077,shortname=winnt,flush 0 2"
>>/etc/fstab
```

The next step is switching the bootloader type.

Unfortunately you didn't mention which Linux distribution you're using. If it's Debian, or Ubuntu, or some distribution derived from those, it would be a matter of using the standard package management tools to remove the `grub-pc` and `grub-pc-bin` packages and install `grub-efi-amd64` and `grub-efi-amd64-bin` in their stead, then running `grub-install /dev/sda` (or whichever disk contains your ESP partition), and finally running `update-grub` to rebuild the GRUB configuration.

At this point, you can exit the chroot, undo the mounts and see if your system can boot now.

```
(if you had to mount any extra partitions, unmount them now)
exit
umount /mnt/dev
umount /mnt/proc
umount /mnt/sys
umount /mnt
reboot
```

You might also want to install the `efibootmgr` utility, as it allows you to view, backup and modify the firmware boot settings while Linux is running. (Windows can do the same with its `bcdedit` command, but in my opinion that command is much more awkward to use than `efibootmgr` .)

Share Follow

answered Sep 4, 2021 at 19:55
telcoM
91k 3 122 241

Thanks a lot for your detailed answer telcoM! I updated my OP while you were writing it, here are further details -- /dev/sda2 was my active MBR partition before, and It is in ext4 format and I used extlinux to boot from it. I changed its type from Linux filesystem to EFI System. And yes, I use Debian / Ubuntu. So, I need to revert its type back to `Linux filesystem` , split out a FAT32 partion as the ESP partition, then follow your above advices, right? Have I missed anything? thx – xpt Sep 4, 2021 at 20:15

Yes. Although 100 MB might be enough for the ESP partition, you might want to allocate 260 MB to be reasonably future-proof in case you ever clone your system to a disk with a 4096-byte block size (the FAT32 filesystem type has a minimum number of blocks requirement that works out to about 254 MB when using 4k blocks). Once you've done that, you should be able to fix Windows boot by running the boot repair function of a Windows installation media. When it detects GPT partitioning and an existing ESP, it should just install a UEFI Windows bootloader onto it. Then the procedure in my answer. – telcoM Sep 4, 2021 at 20:36

Thanks a lot for all your help, Mr telcoM, including the supplement ESP partition size and fixing Windows boot. I've now get everything on the Linux side working, just fixing Windows boot via Windows installation media failed. But that'd be a different quest. Thx again! – xpt Sep 5, 2021 at 20:46

1 Just a word of warning: given the chance, Windows 10 will usually "self-heal" its firmware boot entry if you manage to get Windows booting even once. In the process, if there is no existing Windows boot entry in the firmware (i.e. in the `efibootmgr` list), it will usually usurp `Boot0000` for itself, regardless of whether or not it is already in use. Also, if the `os-prober` package is installed, then `update-grub` should auto-detect the Windows UEFI boot manager (`\EFI\Microsoft\boot\bootmgfw.efi`) and add it to the GRUB menu. – telcoM Sep 5, 2021 at 21:37

1 On the other hand, if there *is* an existing Windows boot entry, I have never had it delete non-Windows ones even once on any of my dual-boot systems, not even across upgrades. (I've heard of certain vendors' UEFI firmware doing so, however.) – u1686_grawity Sep 6, 2021 at 4:54