Linux on your laptop: A closer look at EFI boot options

A review of what EFI boot (UEFI firmware) is, how it works, how it interacts with Linux installations, and a few tips and personal experiences on how to configure and maintain it. By J.A. Watson for Jamie's Mostly Linux Stuff

What I am going to look at is how the boot sequence differs from the previous standard, and how it should be configured and managed for a Linux installation. I am likely to use the terms "EFI" and "UEFI" interchangeably in this post; that is certainly not correct in general, but for purposes of what I am writing they both mean more or less the same thing.

Before EFI, the standard boot process for virtually all PC systems was called "MBR", for Master Boot Record; today you are likely to hear it referred to as "Legacy Boot". This process depended on using the first physical block on a disk to hold some information needed to boot the computer (thus the name Master Boot Record); specifically, it held the disk address at which the actual bootloader could be found, and the partition table that defined the layout of the disk. Using this information, the PC firmware could find and execute the bootloader, which would then bring up the computer and run the operating system.

This system had a number of rather obvious weaknesses and shortcomings. One of the biggest was that you could only have one bootable object on each physical disk drive (at least as far as the firmware boot was concerned). Another was that if that first sector on the disk became corrupted somehow, you were in deep trouble.

Over time, as part of the Extensible Firmware Interface, a new approach to boot configuration was developed. Rather than storing critical boot configuration information in a single "magic" location, EFI uses a dedicated "EFI boot partition" on the desk. This is a completely normal, standard disk partition, the same as which may be used to hold the operating system or system recovery data.

The only requirement is that it be FAT formatted, and it should have the *boot* and *esp*partition flags set (esp stands for EFI System Partition). The specific data and programs necessary for booting is then kept in directories on this partition, typically in directories named to indicate what they are for. So if you have a Windows system, you would typically find directories called 'Boot' and 'Microsoft', and perhaps one named for the manufacturer of the hardware, such as HP. If you have a Linux system, you would find directories called opensuse, debian, ubuntu, or any number of others depending on what particular Linux distribution you are using.
It should be obvious from the description so far that it is perfectly possible with the EFI boot configuration to have multiple boot objects on a single disk drive.

Before going any further, I should make it clear that if you install Linux as the only operating system on a PC, it is not necessary to know all of this configuration information in detail. The installer should take care of setting all of this up, including creating the EFI boot partition (or using an existing EFI boot partition), and further configuring the system boot list so that whatever system you install becomes the default boot target.

If you were to take a brand new computer with UEFI firmware, and load it from scratch with any of the current major Linux distributions, it would all be set up, configured, and working just as it is when you purchase a new computer preloaded with Windows (or when you load a computer from scratch with Windows). It is only when you want to have more than one bootable operating system – especially when you want to have both Linux and Windows on the same computer – that things may become more complicated. The problems that arise with such "multiboot" systems are generally related to getting the boot priority list defined correctly.

When you buy a new computer with Windows, this list typically includes the Windows bootloader on the primary disk, and then perhaps some other peripheral devices such as USB, network interfaces and such. When you install Linux alongside Windows on such a computer, the installer will add the necessary information to the EFI boot partition, but if the boot priority list is not changed, then when the system is rebooted after installation it will simply boot Windows again, and you are likely to think that the installation didn't work.

There are several ways to modify this boot priority list, but exactly which ones are available and whether or how they work depends on the firmware of the system you are using, and this is where things can get really messy. There are just about as many different UEFI firmware implementations as there are PC manufacturers, and the manufacturers have shown a great deal of creativity in the details of this firmware.

First, in the simplest case, there is a software utility included with Linux called *efibootmgr*that can be used to modify, add or delete the boot priority list. If this utility works properly, and the changes it makes are permanent on the system, then you would have no other problems to deal with, and after installing it would boot Linux and you would be happy. Unfortunately, while this is sometimes the case it is frequently not. The most common reason for this is that changes made by software utilities are not actually permanently stored by the system BIOS, so when the computer is rebooted the boot priority list is restored to whatever it was before, which generally means that Windows gets booted again.

The other common way of modifying the boot priority list is via the computer BIOS configuration program. The details of how to do this are different for every manufacturer, but the general procedure is approximately the same. First you have to press the BIOS configuration key (usually F2, but not always, unfortunately) during system power-on (POST). Then choose the Boot item from the BIOS configuration menu, which should get you to a list of boot targets presented in priority order. Then you need to modify that list; sometimes this can be done directly in that screen, via the usual F5/F6 up/down key process, and sometimes you need to proceed one level deeper to be able to do that. I wish I could give more specific and detailed information about this, but it really is different on every system (sometimes even on different systems produced by the same manufacturer), so you just need to proceed carefully and figure out the steps as you go.

I have seen a few rare cases of systems where neither of these methods works, or at least they don't seem to be permanent, and the system keeps reverting to booting Windows. Again, there are two ways to proceed in this case. The first is by simply pressing the "boot selection" key during POST (power-on). Exactly which key this is varies, I have seen it be F12, F9, Esc, and probably one or two others. Whichever key it turns out to be, when you hit it during POST you should get a list of bootable objects defined in the EFI boot priority list, so assuming your Linux installation worked you should see it listed there. I have known of people who were satisfied with this solution, and would just use the computer this way and have to press boot select each time they wanted to boot Linux.

The alternative is to actually modify the files in the EFI boot partition, so that the (unchangeable) Windows boot procedure would actually boot Linux. This involves overwriting the Windows file bootmgfw.efi with the Linux file grubx64.efi. I have done this, especially in the early days of EFI boot, and it works, but I strongly advise you to be extremely careful if you try it, and make sure that you keep a copy of the original bootmgfw.efi file. Finally, just as a final (depressing) warning, I have also seen systems where this seemed to work, at least for a while, but then at some unpredictable point the boot process seemed to notice that something had changed and it restored bootmgfw.efi to its original state – thus losing the Linux boot configuration again. So, that's the basics of EFI boot, and how it can be configured. But there are some important variations possible, and some caveats to be aware of.

At the beginning of the description, I said that when you install Linux it will add the necessary boot information to an existing EFI boot partition alongside any existing configuration of Windows or other Linux distributions. In fact you may have more than one EFI boot partition on a disk, so you might choose to create a new (additional) EFI boot partition, and use that for your Linux installation. Some Linux distributions (notably Fedora and its derivatives) do this by default, while others give you the option of doing it, and will simply add to the first existing EFI boot partition if you don't do so. A few distributions (notably Ubuntu and its derivatives) don't give you a choice, they just install to the first EFI boot partition.

As I mentioned at the beginning, most Linux distributions use a unique name, usually derived from the name of the distribution, for their EFI boot directory. Unfortunately not all of them do; in particular, Linux Mint and a few other Ubuntu derivates still use the name *ubuntu* for the boot directory. This is not a problem, unless you try to install two distributions which both use the same name for that directory; then the second installation will overwrite the first, and you will end up only being able to boot one of them. In this case you are basically forced to use a separate EFI boot partition for at least one of them, which is what I did with my new HP laptop, which kicked off all the latest excitement that led to this article.

The last thing to mention here is that the Linux utility for creating the EFI boot configuration has been improved quite a bit over the past few years. It no longer requires a long and complex sequence of command line options to specify the configuration, it is able to create a functional minimum configuration entirely on its own. So if something happens and your boot data gets corrupted or destroyed, you can recreate it, in the correct location, by simply running "grub-install /dev/xxx" (or grub2-install, depending on the distribution), where the xxx is replaced by your primary disk name, generally sda.

This covers the general overview of EFI booting, and how to configure (or coax) it into working for Linux/Windows dual boot (or multi-boot). Please note that one thing I did not talk about here are "boot fix" utilities, which claim to create or repair such a configuration for you. There are a few simple reasons for this; first, I don't like them, because I don't like things in general that supposedly offer a "magic fix" to a difficult problem; second, because as can be seen from the post, the details of EFI boot configuration are still changing, and a "magic utility" which works today may well not work tomorrow, or which works for one distribution or for dual boot, might not work for another distribution or for multi-boot (more than two boot objects). I think it is much more advisable to actually take the time to understand what you are doing if you are this deep into system administration, so that you know what to change, how to change it, and how to fix it if something goes wrong. Call me old-fashioned.

Oh, one last thing. There is also a utility package available called rEFInd that will help you set up and manage EFI boot configuration. I have used it, and it works (or did when I last used it several years ago), so if you are struggling and can't get anything to work, that might be worth a try. I don't consider it to be a "magic utility", because you can see everything that it does, and the accompanying documentation explains it all. Basically it tries to install itself as the first boot object, and then make a list of all the other possible boot targets on your computer and present them to you for selection. If it works, that's fine – and it is actually a good way to learn how all of this works – but I am not convinced that it is really necessary anymore. Most systems today come with EFI firmware that is sufficiently configurable that you can set up a properly working system without too much trouble, in which case rEFInd is generally overkill.