8.1.1 MBR Disk Partitions

Click one of the buttons to take you to that part of the video.

MBR Disk Partitions 0:00-0:37

In this lesson, we're going to discuss managing master boot record disk partitions on a Linux systems hard drive. The MBR partition formats have been around for quite a while, since the early 1980s, and it's still used a lot on Linux systems. Therefore, you need to be familiar with managing these types of partitions.

Be aware that there are many newer disk partitioning schemes available that are much more robust, and much more flexible, than the old MBR partitioning scheme is. But for the time being, there's a lot of systems that still use MBR, so you need to be familiar with it.

---

fdisk Utility 0:38-3:35

You manage MBR partitions on a Linux system using the fdisk utility. There are other utilities you could use, but fdisk is the one that is common across all Linux distributions.

You can use fdisk to view disk partitions, you can use it to create new partitions, or you can use it to delete partitions from the shell prompt. Do be aware that in order to run fdisk, you do have to be logged in to the system as the root user, because it does require root level access to the system.

The syntax for using with fdisk is shown here. First we run 'fdisk', and then we specify the name of the hard disk device that we want to manage partitions on. Notice in this diagram that we have three hard disks within this Linux system.

You need to be familiar with the device-naming syntax in order to use fdisk correctly. The first hard disk in the system is sda, the second hard disk in the system is sdb, and the third hard disk in the system is sdc.

Notice that all of these hard disks are addressed using a file in the /dev directory. If I wanted to manage the partitions on this hard disk right here--the second hard disk in the system--then I would use this command here: 'fdisk /dev/sdb'.

You might be wondering how the system decides which hard disk is sda, which one's sdb, and which one's sdc. It's all based on which connector the particular hard disk is connected to on your hard disk controller.

For example, on a system that uses SATA hard disk drives, then the connector that is connected to the lowest numbered SATA connector, for example SATA0, is going to be sda. The hard disk connected to the next higher numbered SATA connector, such as SATA1, is going to be sdb. And the hard disk connected to the next highest numbered SATA connector, such as SATA2, is going to be sdc, and so on.

At this point, you need to enter commands to tell fdisk what exactly it is you want to do with the partitions on the hard disk.

As you can see here, once fdisk is running, you're provided with a command prompt. At this command prompt, you enter the various fdisk commands that manipulate the partitions on the disk.

Before you go any further, it's really a good idea at this point to enter the m command as shown here, because when you do, it will display a list of all the different commands you can use with fdisk. For example, you can use n to create a new partition. You can use p just to view partitions. You can use t to change a partition type.

Notice right here, you also have the option with d to delete an existing partition. You should be very careful before you use this action, by the way, because if you delete a partition with fdisk, then you're going to lose all the data that was on that partition.

Once those changes are committed to disk, there's no going back. The changes aren't reversible, so before you delete a partition, make sure that it doesn't have any data on it that you want to save. If it does, back it up first or you're going to be really sad.

'p' --View Existing Partitions 3:36-6:55

Before you do anything with fdisk, it's really a good idea just to see what's already on the disk to see if there are already any existing partitions. The way you do that is by entering p at the command prompt, for print. It will just print out a list of all the different partitions currently on the drive.

This will help you determine, first of all, if there's sufficient space on the drive to even create a new partition; and if so, then what numbers needs to be assigned to that new partition when you created it on the disk.

As you can see here in the output of this p command, there are no partitions. We can go ahead and just add new partitions to this disk without worrying about managing any existing partitions.

With that in mind, let's talk about creating a new partition. To do that, you'll use the n command at the fdisk prompt. When you do this, you'll have to specify whether or not you want to create a primary partition or whether you want to create an extended partition.

Here's a key thing that you have to remember when you're dealing with MBR partitions, and that is the fact that any hard disk in your system that uses MBR partitions can have only four partitions maximum defined in this partition table, and that can be a limitation.

If you intend to just create four partitions or less, maybe two, then it's not a big deal. You can just use primary partitions. However, if you need more than four partitions on the disk, then we've got a problem.

By the way, on a Linux system, it's very common to need more than four partitions on the hard disk drive. In this situation, what you do is include at least one extended partition in your four partitions.

Extended partitions are great. Using an extended partition, you can create many logical partitions within that extended partition. Notice right here that it says an extended partition is really just a container for logical partitions.

Let's take a look at an example. Let's suppose that we need six partitions on the hard drive in this Linux system. We already know that we could have a maximum of only four, so we already know that one of those four partitions has to be an extended partition.

To do this, what we could do is create three regular partitions right here, right here, right here--these are just standard primary partitions. And then we could take the rest of the disk space and define this as an extended partition instead of a regular partition. This is cool because what we then do is create lots of logical partitions inside of this extended partition.

We've got three primaries defined. We need three more created within this extended partition. We could define a logical partition here, and a logical partition here, and a logical partition here. We have one, two, three logical partitions within the one extended partition.

In effect, what we have then is six partitions on the disk when we're technically allowed to have only four. Please be aware that you can have only one extended partition per disk.

The general rule of thumb is that you create your primary partitions first--however many it is you want to create, one, two, or three--and then you create your extended partition using the remaining space on the drive. Then, define all of the logical partitions that you want to use within that one extended partition.

'n' --Create New Partitions 6:56-10:47

To create a primary partition, you enter a p after entering n. If you want to create an extended partition, you would enter e instead of p. In this example, we used p to create a primary partition, and then you have to specify what number you want to assign to that partition.

By default, what it's going to do is take a look at the existing partitions on the disk, and then it will suggest using a number one higher than the number assigned to the last partition. In the example that you see here, there are no partitions on the disk; therefore, the default number that's going to be assigned is 1.

You could use a different partition number if you wanted to, but I rarely do. I usually just use the default, because it works out beautifully. fdisk knows what the next partition number probably should be.

In this case, I just hit Enter to use the default value of 1, and then the next thing we have to do is specify how big we want the partition to be. This is done by specifying the beginning and ending sectors that we want to use for the partition on the disk.

First, we have to specify the first sector that we want to begin with on the partition. By default, what fdisk is going to do is specify the first available sector where a partition could begin. You don't have to use it if you don't want to, but to use space effectively, it's a good idea to do it.

In this situation, there are no partitions defined, so it's going to suggest you use the first available sector, which is 2048. That will place our beginning sector right down around here. Then you have to specify what the last sector will be.

Notice that there are several different options that you can use for specifying the last sector. You could specify the last sector you want to use. If you're good at math, you can calculate that out--that's fine.

Or you can specify a total number of sectors that you want to use, and let fdisk determine what the actual ending sector number should be--what I usually do to specify a size. How big do I want the partition to be? You can specify it in kilobytes, megabytes, gigabytes, terabytes, and so on.

Notice over here that I said I just want this partition to be 10 GB in size, so fdisk then calculates where the ending sector should be, right there, and set it up for me. When I am done, I've got a new 10 GB partition on the hard disk drive.

After you create a new partition, it's really a good idea to run the p command again so you can see all of the partitions on the disk. Basically, make sure that A: the partition got created, and B: it got created with the parameters that you specified.

As you can see here, we created the first partition on the sdb on our disk drive. Here's the starting sector. Here's the ending sector. Here's the total number of sectors, size, and the partition type--it's a standard Linux partition.

Notice over here that once we create a partition, that we use a different device file to refer to that partition. Instead of referring to it as /dev/sdb, which points to the hard disk itself, we point to the first partition on that hard drive, which has its own device file /dev, which is /dev/sdb1. This indicates the first partition on the second hard drive in the system.

It's important to know at this point that the partition actually hasn't been written to disk yet. All the changes that we're making to the disk partitioning are only saved in memory. It hasn't actually been committed to the disk yet. This is really nice, because it allows you to play with the partitions before you actually make the changes.

If you go through a whole partition scheme and realize, "Boy, I don't really like this. I want to start over." All you have to do is enter q at the fdisk command prompt, and none of the changes that you specified will be written to disk. Everything will be back the way it was before you started using fdisk.

When you're running fdisk, I always tell students, "q is your best friend." Before you commit the partition to disk, you may need to change the partition type.

't' --Change Partition Types 10:41-11:45

When we created that partition just a second ago, it was created as a standard Linux partition, and that's the default behavior for fdisk. Usually, this is probably what you want to do, because you want to store data on that partition. But what would you do if you wanted to create not a data partition, but instead you wanted to create a swap partition that will be used for virtual memory?

Well, to do this, you have to use a different type of partition. This is done by entering t at the command prompt and then entering the ID number of the partition type that you want to use. If you don't know what the ID number of the partition type that you want to use is, you can enter the l command, and it will bring up a list of all the possible partition types.

For example, if we wanted to create a swap partition instead of a data partition, we would change its type to 82 right here. Notice that a standard Linux partition is type 83, that's the type of partition you store data in. Partition type 82 is used to create a swap partition.

'w' -- Commit Changes to Disk 11:46-13:10

At this point, we have defined our new partitioning scheme, and we're ready to commit those partitions to disk. To do this, you use the w command. Once you enter w, things are a little more serious because it will make the changes to the partitioning.

If you decided to delete a partition, that partition gets deleted. If you decided to create a new partition, it gets created. So be very careful before you use the w command. Remember you can always enter q to back out without applying the changes if you're not happy.

After you've written your partitions to disk, be aware that you may actually need to reboot the system before the Linux kernel will recognize the partition changes you've made with fdisk. That is actually a pain, because a lot of times you don't want to take the system down just to allow the kernel to see new partitions.

That's particularly true on a server system, where there may be active connections going on with network clients. You don't want to take that server out of commission, even if it's only for a few minutes to reboot. Therefore, what you can do is run the partprobe command.

The partprobe command will force the kernel to see the new partition table without rebooting the system. The syntax is shown here: you type partprobe -s. The -s option causes partprobe to show a summary of what it finds. As you can see here, it found the new partition that was just created on /dev/sdb.

Summary 13:11-13:27

That's it for this lesson. In this lesson, we discussed managing MBR partitions. We talked about how to use the fdisk utility to view existing partitions, create new partitions, change partition types, and delete partitions. We ended this lesson by talking about how we can use the partprobe command to update the kernel with the updated partition table.
