8.4.2 File System Creation

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

File System Creation 0:00-0:47

In this lesson, we're going to spend a few minutes talking about creating Linux file systems. Let's suppose you got a Linux system running, and you use it every day to do a lot of different things and you're running out of disk space.

In order to get extra storage space, you go out to your local big box store, you find a hard disk drive. You install that hard disk drive on the computer, and then you load either the fdisk or maybe the gdisk utility and you create a new Linux partition on it.

You've got all that extra storage space that you can save data on, right? Wrong. Actually, you don't. Why? Even though you partition the hard disk drive, you have not yet created a file system on that partition. After you've created a partition on Linux, you can't use that space to store data until you format it with a file system.

Role/Function of a File System 0:48-2:20

Basically, creating a file system is analogous to formatting a partition on a Windows system. You make a file system on a partition on a hard disk in the Linux system using the mkfs commands, and you might guess, mkfs stands for make file system.

You can use the mkfs utility to make an ext2 file system, ext3, ext4, and so on.

You can even use it to create a FAT32 or even an NTFS file system on a partition. These file system types would allow that partition to be used on both Linux and in Windows. Be aware that the mkfs utility is really just a front end for the real commands that actually make each specific type of file system, and these are shown here. You'll actually find all of these commands on your Linux system.

For example, to create an ext4 file system, mkfs actually runs the mkfs.ext4 command, same for ext3, ext2, btr file system, FAT file system, and NTFS file system, and so on. If you wanted to, you could skip running mkfs and actually run the particular real command if you wanted to, and these are located in the /sbin directory, by the way.

However, most Linux administrators just use the mkfs front end to run the appropriate command. If you do decide to use one of these commands instead of mkfs, you'll need to go and look at the man page for that particular mkfs command to see what syntax it uses.

---

mkfs Command 2:21-7:14

To create a new file system on a partition in a Linux system, you first run the mkfs command and then you use the -t option to specify which type of file system you want to create. In this example, we're going to create an ext4 file system, which would then run the mkfs.ext4 command that we just talked about, and then you have to specify which partition you want to create that file system on.

In this case, we're going to create on /dev/sdb5.

When you run the mkfs command, there are several other options you can use in addition to -t to customize the way that file system is going to be created. For example, you can use the -b option to specify the size of the data blocks on the system.

If you use the -b option, you need to specify the block size that you want to use. There are three values that are allowed, 1024, 2048, or 4096, and you'll have to decide for yourself which one is best for the particular implementation you are installing.

The general rule of thumb is if your Linux system will store very large files, then you can use a very large block size. If on the other hand, your Linux system will be storing a lot of little, teeny, tiny files that change frequently, then a large block size will not use disk space efficiently, and you should use a smaller one such as 1024.

You can also use the -N option to specify how many inodes are going to be created within the file system. Remember an inode stores basic information about a file in the Linux file system. It specifies the size of the file, the device where that file exists, who owns that file, and all the permissions that have been assigned to the file.

Here is the key thing that you need to remember. Each file has its own inode with the exception of hard link files, which share inodes.

With every other Linux file, each file has its own inode; therefore, specifying a hard number of inodes effectively limits the total number of files that will be allowed to be created on that file system, because when that limit is reached, you'll not be allowed to create any more. You can also use the -i option to specify the size of the inode.

Typically, I don't do this. I just use the defaults, which I believe by default will size the inode to be the same as the block size of the file system, and then you can use the -j option to create a journal on the file system. We don't use this option very much anymore.

For a while, we used it a lot back in the days when we were migrating from ext2 to ext3 because you could use -j essentially to convert an ext2 file system to an ext3 file system by simply adding a journal to the ext2 file system. Because there aren't very many ext2 file systems around anymore, you probably won't use that option very often.

If you don't include any options when you're in the mkfs command, then default values will be used for you, and to be honest, that's usually what I do. On very rare occasions, I might override the defaults, but for the most part, using the default values determined by mkfs usually works pretty good.

This is because an optimal block size and an optional number of inodes will be automatically calculated for you based upon the size of the partition. In the example you see right here, I'm using the mkfs command to create an ext4 file system on the /dev/sdb5 partition. When mkfs ran, it looked at the size of the partition, which is actually really tiny, then it determined these optimal parameters for that partition.

Be aware that these are not the default values used by mkfs in all cases. Remember, it automatically determines what the best sizes should be for the particular partition and file system in question. In this case, it decided that the best block size was four kilobytes 4096.

It also decided that this many blocks and this many inodes should be created in the file system. In fact, if you multiply the total number of blocks by the block size, you can determine the size of the overall file system, the amount of storage space in the file system, which in this case, I think works out to five gigabytes or so, four or five gigabytes.

Also, notice over here that it specifies that superblock backups will be stored in these blocks in the file system.

The superblock is the block at the very beginning of the partition, and that is called block 0. This is a very important block, because it contains information about the overall structure of the entire file system on that partition.

In this file system, what mkfs is going to do is place redundant copies of this block in these locations. That way if something happens to this block for some reason, we have backups available.

---

mkreiserfs Command 7:15-9:09

In addition to the mkfs command, there's also another utility that you can use to create a file system on a partition on a hard drive in the Linux system, and it's not used very often anymore, and that is the mkreiserfs command. This utility creates a Reiser file system on a partition.

At one time about a decade ago, Reiser was the up and coming file system that everybody was using, but then due to some legal problems that the guy who wrote the Reiser file system got into, everybody pretty much abandoned it and starting using ext3 and 4 instead.

Be aware that before you can use the mkreiserfs command to create a Reiser file system on a partition, you do need to have the reiserfs-utils package installed on your system. For example, if I wanted to create a Reiser file system on a partition within a Fedora system, I would use the yum command to install the reiserfs-utils package.

Once done, you can then create a Reiser file system on a partition in your Linux system. To do this, you run the mkreiserfs command and then specify the device file for the partition you want to create the file system on. In this case, /dev/sdb1, and by the way, if you don't have this package installed, you can still create a Reiser file system with the mkfs command.

You just specify reiserfs with the -t option with mkfs, and you can still accomplish the same thing. You can see here that as the file system is created, the default values for mkreiserfs are used.

You can customize these parameters if you want to like we did with mkfs. You can go ahead and look at the mkreiserfs utilities man page to see what these parameters are. Usually, my experience has been, however, that these default parameters work really well because just like mkfs, mkreiserfs will determine what the best parameters probably should be based upon the size of the partition and the file system that you're creating.

Use mkfs to Create Non-Linux File Systems 9:10-10:21

In addition to creating ext2, 3, and 4, as well as Reiser partitions, you can also use the mkfs command to create XFS file systems. The XFS file system was originally created by Silicon Graphics, and it's a very fast, very flexible file system, and it's been ported over to run on Linux.

Be aware that before you can create an XFS file system, XFS support has to be added to the Linux kernel. This is done by default during the installation of some distributions like Fedora, but it's not included with many other distributions. Two key ones I can think of right off the top of my head are openSUSE and Ubuntu. They do not support XFS by default.

In addition to creating Linux file systems with the mkfs command, you can also create non-Linux file systems with the mkfs command. For example, you could create a FAT partition or even an NTFS partition.

To create a FAT partition, which would be readable between Linux and a Windows system, you use the -t vfat option with the mkfs command. Likewise, you can even create an NTFS file system on a partition; you use -t ntfs.

Summary 10:22-10:42

That's it for this lesson. In this lesson, we discussed how to create file systems on Linux partitions. First, we discussed the role and function of a file system. Then, we looked at using the mkfs command to make ext2, ext3, and ext4 file systems. We also looked at the mkreiserfs command to make Reiser file systems, and then we ended this lesson by discussing how you can use mkfs to create non-Linux file systems, such as FAT or NTFS.
