8.6.1 File System Maintenance

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

File System Maintenance 0:00-0:49

Just as with any other operating system, you need to monitor and maintain your Linux file system. In this lesson, we're going to talk about how to check for disk space utilization, we're going to talk about checking for open files, we're going to talk about how you can identify processes that have files open, and then we're going to end this lesson by looking at how you check for file system integrity.

Let's begin by discussing how you monitor disk space utilization. Just as with any other operating system, you need to make sure that your Linux file system doesn't run out of space. Because I can tell you from experience that if you do run out of space in your Linux file system, nothing good happens.

There are several utilities you can use in order to keep track of how much disk space your operating system files and your user files are using. Let's take a look at what they are.

Use df 0:50-3:41

Let's first look at the df utility. df is extremely useful. It displays several pieces of information. First of all, it shows you where your hard drive partitions, optical drives, and other storage devices--such as USB drives--are mounted in your Linux file system. And it also shows the total size of the storage device, as well as how much space has been used on that device.

To run df, you just run the df command at the shell prompt, as you see here. However, be aware that by default, df will display space utilization stats in terms of blocks, which is fine if you like doing math and figuring out how many blocks equals how much storage space. I don't like doing that.

In order to make the output easier and faster to read, I always use this option, the -h option, with the df command. -h means human readable. It will cause df to display space utilization information in human-readable format, such as gigabytes, megabytes, terabytes, and so on.

If you want to see what type of file system is used on each device, you can also use the -T option. As you can see here, I have /dev/sbd5, and it uses an ext3 file system. It has a total size of 4.8 GB. 11 MB have been used on the device. 4.6 GB are available still to be used on the device. Only 1% of available space on the device is being utilized, and here's where it's mounted in the file system: /mnt/private.

As you see in this example, if you run df without specifying a particular file or directory or device, it just shows information about all mounted file systems.

However, if you enter df, followed by a particular filename at the prompt--in this case /mnt/shared/myfile.txt--then df will display information about just the file system on which that file resides--in this case, /dev/sdb1. It tells us the size, how much is used, how much is available, and where it's mounted.

Another useful option you can use with df is the -i option, lowercase i. This lists inode usage information instead of disk space usage. And this is important because, remember, that when you create a file system on a device, a fixed number of inodes is assigned.

This can be important if the file system stores a humongous number of very little files. In this situation, it is theoretically possible for that file system to run out of inodes to be assigned to new files, even though there is plenty of disk space still available.

In this situation, you can monitor your inode usage using the -i option with the command. As you can see here on /dev/sdb5, there are a total of 320,000 inodes available, which means I can create up to 320,000 files in this file system. Currently, I'm using only 11 of them, so basically almost all of them are still available.

Use du 3:42-4:21

Another utility that you can use to monitor disk space usage is the du utility. Its function is to provide you with a summary of the disk space used by each file recursively for a specified directory. This utility can be useful, for example, in locating which user has downloaded a whole bunch of 12 GB high-definition video files and is using up all the space on the system.

The syntax is to enter du, followed by the name of the directory you want to look at. And as with df, you can use the -h command to specify that it use human readable space utilization stats--in other words, with kilobytes, megabytes, and gigabytes instead of blocks, as we saw with df.

Use lsof 4:22-6:47

Now let's look at how you check for open files on a Linux file system. This can be really important. I've had many situations where remote users using a different computer system have connected to a Linux system over the network, have opened a file for editing, and then for whatever reason lost their connection to the Linux system.

This happens very commonly with wireless connections when the wireless client is mobile and they move out of range, or interference happens, or something is such that they lose the connection with the Linux box itself. When that happens, the file that they were editing is in an open state, and if they reconnect and try to open that file for editing again, they can't because it's locked.

Well, you can use the command that you see here--lsof--to find that open file, and then you can actually close it. If you enter lsof, which stands for list open files, at the shell prompt without any options, it will list all open files belonging to all active processes on the system.

And you can use the options shown here with lsof; the first one is -s, which displays the size of the file in addition to whether it's open or not. You can also use the -t option with the lsof command, and this can be really useful because it will tell lsof to produce terse output with process identifiers only--in other words, what process currently has this file open.

This is useful because you can then take the output of the lsof command, and then pipe it over to the kill command, which will allow you to locate an open file on the system, find out which process currently has that file open, and then have the kill command kill that process, which will then make the file openable again by another process.

Remember I said earlier that if you enter just lsof at the shell prompt without any other options, it displays all open files belonging to all active processes on the system. If you ever try that, you'll see that it produces a ton of output, more than what you need.

More than likely what you're going to want to do is use this -u option with lsof, because this option tells lsof to display only those open files that are associated with a specific user account. For example, if I had the user who lost the wireless connection to the network and has a particular document open that the user can't get back into because it's locked, then you can use -u to find just the open files that are associated with that user account.

An example of that is shown here. I've entered lsof -u, followed by the name of the user that I want to look for, ksanders. And when I do, all the files that this user account has open are displayed on the screen.

Use fuser 6:48-9:35

In addition to listing open files, there may be times when you also need to identify the processes running on your system that are using a particular file in the file system. Basically, lsof looks at files from the perspective of which file is open on the system.

While this command, fuser, comes from a different perspective; it says what processes on the system have files open. When you run fuser at the command prompt, it will display the process ID numbers of the processes that are using a specific file or file system.

By default, each open file will be followed by a particular letter.

c specifies that the file is in the current directory. e indicates that this is an executable that is currently being run. f, lowercase f, specifies an open file. Capital F is similar, but it specifies that the file is open and being written to. r specifies the route directory, and m specifies a memory-mapped file or a shared library.

When you run the command, some options you can use are shown here. We can use -a to show all files specified. This can be useful because by default, without -a, only the files that are accessed by at least one process will be shown.

We can use the -k option, which is something I actually use fairly frequently in order to kill the process accessing a particular file. You can use the -i option to ask the user for confirmation before killing that process. And you can use the -u option to append the name of the process owner to each process ID number.

Basically, we can see who owns the process that has the file open. Here's an example of using the fuser command. I want to see which user is running the top executable. This is the path to the top executable: /user/bin/top.

When I run the command, I see that the ksanders user is currently using the top executable; it's open. We can see by the e in the output that this is an executable, and here's the process ID number.

At this point, let's shift gears a little bit and talk about fixing file system errors. And just as with any other operating system, Linux partitions with file systems created on them can sometimes encounter problems. And these problems can be caused by a variety of different conditions.

However, my experience has been that the number one source of partition file system corruption is that of a power outage that results in an unclean shutdown of the system. And if your system goes down without properly unmounting a mounted file system, it's very likely that some data corruption will occur. It could be a little bit, and it could also be a lot. I know, it's happened to me.

If something like this happens, then you need to check your file systems for errors, and if necessary make the necessary repairs. And this is done using the file system check utility, or fsck.

Use fsck 9:36-13:01

As its name implies, the fsck utility checks the integrity of the files and directories in your Linux file system. Now be aware that before you can use fsck to check the file system, you first have to umount the file system that you want to check.

In this example, you can see that I want to check the file system on the /dev/sdb1 partition. In order to do that, I first use the umount command to unmount that file system, and then once done, I can then use the fsck command to check the file system on that partition.

I enter file system check, fsck, and then the name of the partition whose file system I want to look at, /dev/sdb1. When I do that, the utility will check the file system and report any errors that it encounters, or the lack thereof. As you can see in this example, the file system is clean. There were no errors encountered, so I can rest easy.

On the other hand, if there were errors that were encountered by fsck, then a code will be displayed that tells you what the error is. For example, an error code of 1 says that file system errors were encountered but they were corrected. 2 tells us that errors were encountered and corrected, but the system is going to have to be rebooted.

4 tells us that, "Hey, I've found errors, but I couldn't fix them so I left them uncorrected." A code of 8 indicates some type of operational error. 16 says that you basically entered the command wrong; you used the wrong syntax.

If you start a check and then manually cancel it, then you'll get an error code 32, which just basically says, "Hey, I was running, but you told me to quit." And then the last one is 128, which indicates that there is a problem somewhere with a shared library.

In this example, I'm running fsck on /dev/sdb1. But notice this text right here. It says that the actual utility that was used to check the file system was the e2fsck utility. And that's because the fsck command is really just a front end to file system-specific repair utilities.

This is really nice because you don't have to know what file system has been created on the device that you're checking. You just run fsck, it figures out what the file system is, and then it runs the appropriate file system-specific repair utility for that file system.

A list of them is shown here. We have a fsck utility for ext3, fsck utility for ext2. We also have the e2fsck utility and the fsck.ext4 utility.

Be aware that the e2fsck utility can actually check ext3, ext2, and ext4 file systems. And, in fact, most of the time my experience has been that this is the one that is actually run for any of these file systems instead of these utilities--but they are there.

If fsck encounters a MS-DOS (FAT) partition, then it will run this file system check utility. If you have a Reiser file system created on a partition, then this is the file system check utility that will actually be used. And be aware that if you're dealing with a Reiser file system, you can also run the fsck.reiserfs file system-specific utility by running this command, the Reiser file system check command.

Either one will accomplish the same thing. After you run file system check and the check is complete, everything looks good, remember to remount the partition, using the mount command. Sometimes new Linux administrators remember to umount the partition, run the file system check, and then they walk away, forgetting to actually remount the file system. Then everybody is mad because they can't get at their files.

Restore a Damaged Superblock 13:02-14:05

Now let's spend a little bit of time looking at that e2fsck utility we just talked about. The reason I want to look at it is because it has one very useful feature that you won't use very often, but when you need it, boy it's a lifesaver. And that is the fact that it can restore a damaged superblock on an ext2, ext3, or ext4 file system.

Remember the superblock is the block at the beginning of the partition that contains information about the overall structure of that entire file system. That's very critical. It's so critical that most Linux file systems will keep a backup copy of the superblock at various locations within the partition. That way, if the original superblock gets corrupted, you can restore it using one of the backup copies.

You can do that with the e2fsck command, you enter e2fsck -f -b, followed by the number of the backup superblock that you want to restore from, and then the name of the device. The -f option specifies that e2fsck should force a check, even if the file system appears to be clean. And then, of course, -b tells it where to go to find the backup copy of the superblock.

Summary 14:06-14:19

That's it for this lesson. In this lesson we reviewed several of the utilities that you can use to monitor, as well as to maintain, your Linux file system. We looked at the du command, the df command, the lsof command, the fuser command, and the file system check (fsck) command.
