9.2.2 Managing Kernel Modules

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

Managing Kernel Modules 0:00-0:46

In this demonstration, we're going to talk about managing kernel modules. Understand that as a Linux administrator, from time to time, you may need to manage the kernel modules, or drivers if you want to think of them that way, that are loaded on your system. You may need to add a module, you might need to remove a module, and so on.

In this demonstration, we're going to review how to use several kernel module management commands to do this. We'll look at 'lsmod', 'rmmod', 'depmod', and 'modprobe'. Before I can do anything, I do have to switch to my root user account because only root is allowed to manage kernel modules.

Let's begin by reviewing how to simply look at a list of all the loaded kernel modules on the system.

'lsmod' Command 0:41-1:35

To do this we type 'lsmod'. When we do, we see a listing of all the kernel modules that are currently loaded on the system. If we scroll up to the top here, we can see what each of these columns in the output mean. The first one is just the name of the module. The second one specifies how much memory that module uses. The last one here is very important. This is the "Used by" column. This specifies the module dependencies. Any module listed over here and used by a particular module specifies that that module is dependent upon this module being loaded first before it can run. For example, down here, the bridge module cannot load until the stp module is loaded first.

'rmmod' Command 1:36-3:17

Notice down here that we have a module for the parallel port in this computer system. Let's suppose that I used to have an old parallel printer connected to my parallel port, and it gave up the ghost--it's died. I'm no longer using it. I've disconnected the printer from the parallel port. I went into the BIOS and disabled the parallel port. Therefore, I no longer need this kernel module loaded. I can remove it. We can do this using the 'rmmod' command.

Before we actually remove this module, there are a couple of important things that you need to be aware of. First of all, you need to be very sure you know what you're doing before you unload a particular kernel module. If you accidentally unload a very critical kernel module, such as the one that supports your hard disk interface, you're going to be in a world of hurt. Also, you need to make sure that the module that you want to remove is not currently in use. If it is in use, then you're going to see a device busy error. This includes any dependent modules.

Going back up here to our earlier example, let's suppose I wanted to unload the stp module. Notice over here that it's actually being used by another module--bridge. If I wanted to unload stp, I would first have to come over here and unload bridge. After I did that, then I could unload the stp module.

In our scenario here, we want to unload parport_pc. I enter 'rmmod parport_pc' in the shell prompt. If I do an 'lsmod' again, we should see that parport_pc has been removed, and it has. It's not there anymore.

'depmod' and 'modprobe' Commands 3:18-5:44

Carrying on with our scenario, let's suppose that I was rummaging around at a garage sale on Saturday and I happened to find a great old parallel port laser printer for five bucks. I couldn't resist it. I bought it. I brought it home, and now I want to connect that behemoth to my Linux system so I can send print jobs to it. In order to do that, I have enabled the parallel port, the BIOS of my system, and now I need to load the kernel module in order to support it.

We could try to use the 'insmod' command to directly load the module, but that's really not the best way to do things, because before you load a module, you really should first generate a module dependency list to make sure any dependencies are accounted for. Really, kernel modules work in much the same way as RPM or Debian software packages. Some kernel modules, as we talked about a minute ago, are dependent upon other modules being loaded first before they can be loaded themselves. The key thing to remember here is that 'insmod' will try to load the kernel module without checking for dependencies. If you're lucky, it might work if there aren't any dependencies, or if the dependent modules just happen to already be loaded, but more than likely you're going to have problems. It's better to generate a dependency list.

Once you do that, then you can use the 'modprobe' command to load the module. This is important because the 'modprobe' command will automatically look for and load any dependent modules first, then it will load the module in question. Let's generate our module dependency list with a 'depmod' command. '-v' option to verbosely list the output from the command. It'll take just a second to run.

It's finished running. If you look at the output up here, it's really not that interesting to you and I, but it is very interesting to the 'modprobe' command. When we try to load a kernel module with the 'modprobe' command, it will look at this information, try to identify any dependencies, and load those dependent modules first before it loads the module that we actually specified. Let's go ahead and use 'modprobe' to load 'parport_pc' again. Now let's run 'lsmod' to make sure that it was loaded. Scroll up to the top. We should see it right there. Now we can send print jobs to that old laser printer that we connected to our parallel port.

Summary 5:45-5:53

That's it for this demonstration. In this demo, we talked about how to manage kernel modules. We talked about how to use the 'lsmod' command, the 'rmmod' command, the 'depmod' command, and the 'modprobe' command.
