6.1.3 Manage RPM Packages

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

Manage RPM Packages 0:00-0:23

In this demonstration we're going to practice managing RPM packages on a system. We're first going to talk about how to install a new package.

Then we'll talk about how to uninstall an installed package. Then we'll talk about how to test dependencies. We'll then look at verifying installed packages, and we'll end this demonstration by talking about how you can extract individual files from an RPM package.

Install a New Package 0:24-4:23

In order to manage packages on this Linux system, I have to have root level access. I can't do it as my standard rtracy user, so the first thing I will do is switch to root with the 'su -' command.

The first thing we want to do is use the rpm utility to install a new package on the system. I already have some packages downloaded that we can work with.

Let's go ahead and switch to the directory where they reside to make our paths a little simpler with the rpm command. We'll switch to '/home/rtracy/Downloads.' We'll do the 'ls' command. We can see that I have a package here for gcc and a package here for gnuchess.

What I want to do here is install the GCC package on the system. GCC is a C compiler that you can use to compile source code into an executable that can run on the system. It's extremely useful; I use it all the time. Therefore, it's a utility that I want to have installed on most of my Linux systems.

Before I install it, let's first verify that it already isn't installed. We can do that with the rpm command by typing 'rpm -q' and then the name of the package we want to query. In this case, 'gcc'. This will tell us whether or not GCC is already installed. It tells us that the GCC package is not installed.

Let's go ahead and install it. To do this, we use the 'rpm' command again. We have a couple of options we can use to do the installation with. One option is to use the '-i' command. The -i option will tell rpm to go ahead and install the package that we specify on the system.

The -i option actually has a couple of weaknesses, and that is the fact that if we already have GCC installed and we didn't run this command right here to verify that first--but the version of GCC that was installed is an older version of GCC--then rpm -i would just say, "Hey, it's already installed, don't worry about it."

Therefore, what I actually use most of the time when I am installing a new package with rpm is to use the '-U' option. That stands for upgrade.

Upgrade does two different things. First of all, if you specify a package that is not currently installed on the system, such as GCC, then -U will go ahead and just install it for you from scratch.

If GCC were already installed and it was an older version than the version we're trying to install, then -U--which stands for upgrade--would go ahead and upgrade that package.

I think -U is a little bit more flexible. For our purposes today, we know that it's not installed, so let's just go ahead and use the -i option.

I usually use a couple of other options when I'm installing packages, and these two options work regardless of whether you are using -i or -U, by the way. One is to use the '-h' option. -h will cause hash marks to be displayed on the screen as the installation process progresses so you can see where the overall process is, and can kind of estimate how long you will have to wait.

You will also use the '-v' option with -h--for verbose--so you see what is going on during the installation process.

We enter 'rpm -ihv', or -Uhv, if you wanted to go that route. Either one would work in this situation.

Now we have to specify the name of the RPM package that we want to install. In this case it's GCC. I'll type 'gc' and press the 'Tab' key. Tab complete is your best friend when you are working with package names.

Also notice that when we use the -i command, we do have to use the full filename of the package file. Earlier, when we did the rpm -q command, we didn't have to specify the full filename, just the name of the package itself, which is just GCC.

Let's go ahead and press Enter to start the installation process, and the package is being installed. You can see here that the h and v options caused these progress indicators to be displayed, letting me know where the installation process is currently at.

Now that we've installed it with the -i option, let's use the '-q' option again with 'rpm' to verify that GCC truly is installed. We know from the output that it is, because instead of displaying package gcc not installed, this time it tells us the full package name of the GCC package. We know that it's there.

Uninstall a Package 4:24-4:46

You can also use rpm to uninstall an installed package. Let's suppose for whatever reason we decide that, "No, we don't need GCC on the system. Let's go ahead and uninstall it."

To do this, you would type 'rpm -e', or erase, and then just the name of the package that you want to remove. In this case 'gcc'. Note that I don't have to specify the full filename, just the name of the package. Press Enter.

Test Dependencies 4:47-6:47

If we use the '-q' command with 'rpm' for 'gcc', we see that the GCC package has been uninstalled. You can also use the rpm command to test dependencies.

This is a critical thing to understand, because most Linux packages do not include all the software that they need to run. They are dependent upon other packages being first installed. Therefore, before I can install a particular package, I should go in and test and make sure that all the prerequisite software is already there. If not, I can go ahead and install it.

I will point out to you that rpm is not the best utility to use for working through dependencies. It will tell you what the dependent packages are, but then you have to manually go through and install them.

Sometimes the dependency chain gets longer and longer and longer and it takes quite some time. There are better utilities to do this with, which we'll discuss in a different demonstration.

For our purposes here, let's go ahead and use the 'rpm' command to test dependencies to view a list of all the dependent packages that are required for a given package before we can install the package in question.

For our purposes here, we want to check and see what the dependencies are for this package. This is a game. It installs a chess program on the system that we can use to play chess on the computer. We need to see what is required in order to install this package on the system.

We do 'rpm', and then we use the '-i' option as if we're going to install it, but instead of installing it, we use the '--test' option to say, "No, actually I don't really want you to install the package. I simply want you to test whether or not it can be installed."

Then rpm will go through and see what will be required in order for the rpm package specified to be installed, and lets you know what needs to happen.

We enter rpm -i --test and then the name of the package file.

Notice we had to use the full filename of the package. Press Enter, and notice here that I would not be able to install this package with rpm -i because there is a failed dependency. Before this package can be installed, I've got to install the Ruby package on the system.

Verify Installed Packages 6:47-8:14

You can also use the rpm utility to verify an installed package, basically making sure that no dependencies have gotten broken--that all the files necessary for the application to run are there.

This is done by using the 'rpm' command and the -V option. You can verify just one particular package or you can actually use the -Va option in order to verify all of the packages on your system.

This is actually something that I do on occasion, because after a system has been in use for a while, there is always a risk that somebody has deleted a file that they shouldn't have, broke a dependency somewhere that they should not have, and it's good to just go through and look at all the installed packages on the system to see if anything is amiss. You can do that by running rpm -Va.

Hit Enter. This command is going to take a little while to run because there are a lot of packages installed on the system, and the rpm command is going to have to go and look at all of them. While this is running, let's go ahead and open up a new terminal window, and let's

switch gears a little bit while the rpm command is running.

There are times when you may need to grab just one single file out of an rpm package. Maybe you have a file that got corrupted, and rather than uninstalling and reinstalling the whole application, you just need to grab the original version of that file and copy it over.

You can't actually get into an rpm package directly. Instead, what you have to do is convert the package to an archive file, and then use the appropriate utility to extract the necessary files out of the archive file.

Extract Files 8:15-12:13

Basically, the archive file that we're going to create is like a winzip file that doesn't have any compression. We have lots of different files all archived together into one single file.

One way you can do this is to use the 'rpm2cpio' command. This will take an rpm file and convert it to a cpio archive that you can manipulate with the cpio command.

Before we do this we do need to switch to our root user account. Let's switch back to '/home/rtracy/Downloads' where we were working before. For the purposes of our demo today, let's go ahead and grab this chess game package file and convert it to a cpio archive file, which we can then extract files from.

To do this I enter 'rpm2cpio', followed by the name of the package file that I want to convert to a cpio archive. We'll just use the chess package. We'll use tab complete so we don't have to type the full name out.

At this point, I'm going to do something different because by default, this utility will write its output to the screen, which is practically useless for what we want to do. Instead what we're going to do is redirect the output of the command to a file in the file system. Let's just name it 'gnuchess.epio'.

We use that extension to indicate that this is now a cpio archive. It won't hurt the original file. It will leave it in place and it will just copy it over into this archive file that we specify here.

Hit Enter, wait just a minute while it runs. Use the 'ls' command, and now we see that we have an archive file. Now we can use the cpio utility to extract individual files out of that archive.

Okay, it looks like the rpm command is now finished in our original window. We can look at the output of the rpm command over here. Essentially what happens is the rpm command, when you use the -V option with a, it will go through and look at every single package installed on the system and look for anything that isn't right.

When it's done, you need to come over here and look at the output and then use the codes that are displayed to determine what is wrong.

For example, with the /etc/plymouth/plymouthd.conf file, the S right here indicates that there is a problem with the size of the file. It doesn't tell us what is exactly wrong with it, but somehow there is something wrong with the size of the file.

Also the 5 right here indicates there is actually a problem with the md5 checksum of that file. This would be consistent with the size being off.

That means some change has been introduced into this file that has caused its filesize to either increase or decrease which of course then causes the checksum of the file to be mismatched with the original version of the file.

I also notice here that there is a T. This T indicates that there is a problem with the modification time of this file. It's possible that this file has been corrupted in some way. It could be minor and insignificant and not affect the system, or it could be really serious and could cause major problems with the system.

Notice that this file right here, var/run/wpa_supplicant is just downright missing. It should be there. There are executable on the system that are expecting it to be there, but it's not.

Down here for this file, usr/lib/os-release, we have an L. The L indicates that there is a problem with this file's symbolic link. Again it doesn't tell us what the problem is, it just tells us that there is a problem somewhere.

I might point out too--I forgot to mention--that if you see a C right here, it tells us that those files are configuration files. Notice that this file and this file are not configuration files, while these other files are.

If you see an M in the output of the command, it tells us that there is a problem with the file's mode. Again, it doesn't tell us what the problem is, just that there is something wrong with the permissions that have been assigned to this file.

Down here a G tells us that there is something wrong with the file's group. Perhaps the group that is specified as the owning group of this file doesn't exist anymore.

Summary 12:14-12:29

That's it for this demonstration. In this demo we talked about how to manage your rpm packages on a Linux system. We first talked about how to install a new package. Then we talked about how to remove an install package. We talked about testing dependencies. We talked about extracting files from an rpm package, and then we ended this demonstration by verifying all of the installed packages on the system.