10.3.4 cron Task Scheduling

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

Cron Task Scheduling 0:00-1:32

In this lesson, we're going to look at how you can use the cron daemon to schedule jobs to run in the future on a schedule that you specify. Now as you know, the at daemon can also do the same thing. You can use it to schedule a job to run at a later time in the future.

However, be aware that the at daemon has one key drawback, and that is the fact that it can schedule a job to run only once in the future. Now, that's not a problem if you only want the job to run once in the future. However, there will be many times when you want a job to run in the future on a regular schedule.

For example, you may want to run your back up every day at a particular time. In this situation, the at daemon really doesn't cut it because you'd have to define a new at job every single day to run your backups, and that's really not all that useful.

Instead, what we need is a tool that can handle a repetitious schedule, and the cron daemon can do just that. Unlike at, cron can run commands on a schedule that you specify over and over and over.

So in this lesson, we're going to talk about how cron works, and then how you can use cron to schedule jobs in the future. So let's begin by talking about how the cron daemon itself works.

The cron daemon runs when the system is booted up, and it runs continuously in the background. It checks special files called crontabs files once every minute to see if there's a job scheduled that it needs to run. And if there is, it runs it; if not, it waits and checks again the next minute.

/etc/crontab 1:33-3:09

Now understand that you can configure cron to run two different types of jobs. You can run system jobs, or you can run user jobs. We're going to look at both of those.

We're going to begin, however, by talking about running system jobs. Now, using cron to run system jobs on a schedule is an extremely useful tool for a Linux administrator. Using cron, you can schedule certain jobs to run on a regular schedule, and that saves you a ton of time and effort.

So to run system jobs, the cron daemon uses this file right here, /etc/crontab. The /etc/crontab file specifies what system jobs the cron daemon should run, and an example of a crontab file is shown right here.

The crontab file contains this one command right here that tells the cron daemon to run scripts that are located in these four directories. The first one in the /etc directory is cron.hourly, which contains cron scripts that should be run every single hour of the day.

There's also a directory in /etc called cron.daily, which contains scripts that should be run once a day. There's also a directory in etc called cron.weekly that contains scripts that cron needs to run once a week.

And then finally, there's a directory called cron.monthly that the cron daemon should run once a month. So any script that's found in any of these four directories is run automatically by cron according to the specified schedule.

---

cron Directories 3:10-3:50

For example, in the /etc/cron.daily directory, there are scripts that perform a wide variety of tasks. For example, we have a script called logrotate. This script contains the commands necessary to rotate your system logs. And because it's in the daily directory, it is run once a day.

This is designed to keep your log files from getting too big, to keep them manageable, and to back them up every so often. So if you have a system task that you need to be run according to one of these four different schedules-- hourly, daily, weekly, or monthly-- all you have to do is create a script file that has the commands that you need within it, and then copy it into the appropriate directory in /etc.

/etc/cron.d 3:51-9:02

What do you do, however, if you have a system job that needs to be run on a schedule, but the schedule required doesn't fit the four different categories represented by those four cron directories? Basically, you need it to run on a custom schedule. What do you do?

Well, in addition to the four directories we just talked about, there is a fifth directory in /etc directory called cron.d. If you have a system job, and you need it to run on a custom schedule, you create what's called a crontab file in this directory.

This crontab file will be read by the cron daemon and the commands contained within it will be run for you according to the schedule that you specify. Now, a sample crontab file is shown here.

The crontab file is just a text file, and it has one cron job per line. Now each line has six different fields, and they're separated by tabs. Here's field 1, field 2, field 3, field 4, field 5, and then the rest is field 6. These different fields configure how the cron job will be scheduled.

For example, let's suppose that I want to back up the /home directory, and I want to do that using the tar command. I want the back up to occur every day of the week except for Sundays, because nobody's at work on Sunday, and I need it to run at 11:05 pm.

Later at night, when everybody should be gone and not working on the system. I want to run the tar command. I want it to backup the /home directory, and I want to save it to a file named home_back.tar, which will be saved on a removable USB hard disk that has been mounted in /media/usb. I can accomplish this using the crontab file shown here.

Let's take a look at what each of the fields in this crontab file do. The first field is the minutes field. This specifies how many minutes past the hour that you want the command to be run. The next field is the hour field. This specifies the hour of the day when you want the command to be run.

So basically, these two fields together specify the time of day when the command should be run. In this case, we're going to run the command at 23:05. Now, as you can see here, cron prefers the 24-hour clock. So if subtract 12 from 23, we get 11pm. And because we put five in the minutes field, this command is going to be run at 11:05.

The next field specifies the day of the month that you want the command to be run. Notice here that I used an asterisk character. That means every day of the month. If you had a command that you wanted to run only on a particular day of the month, you could put the number of that day of the month here.

For example, you may have a command that needs to be run on the 15th of every month and only on the 15th of every month. Well, you'd put that number 15 in this field.

The next field is the month field. This specifies the month of the year when the command should be run. In this case, again I've used an asterisk, so that means I want the command to be run every single month of the year.

If that weren't the case, I would put the number of the month in this field instead. And then finally, we have the day of the week that we want the command to be run on. Now as far as cron is concerned, Sunday is day zero and Saturday is day six.

So you could specify one particular day of the week when you want the command to be run. For example, if I put five in this field, the command would only be run on Fridays. Or I can put a range, as I've done here, one to six.

This indicates that the command should be run on Monday, Tuesday, Wednesday, Thursday, Friday, and Saturday--but not on Sunday. And then the sixth field over here is the actual command that we want the cron daemon to run.

Now notice that we had to use the full path over here to the tar command, and that's different than what you're used to when running a command at the shell prompt. Just be aware that the cron daemon does not use the path environment variable to find the command to run, so you have to put the full path.

And if you don't know what the full path is to the command that you want to run, use the which command at the shell prompt; type which followed by the name of the command, and it'll tell you where that command is located.

So, using this crontab file, my /home directory will be backed up to a file named /media/usb/home_back.tar at 11:05 of every day of the month on Monday through Saturday.

Now, in addition to creating system cron jobs, individual users on your Linux system can also create their own cron jobs, running commands on the schedule that they specify. And this is done by using a separate crontab file that's associated with just their user account.

Now, unlike the system crontab file, which is saved in the /etc directory, user crontab files are created and stored in a different directory. They're stored in /var/spool/cron/tabs. So if an individual user on your system has created their own crontab file, it'll be saved in the tabs directory under his or her username.

User cron Jobs 9:03-10:29

Now before proceeding, I should point out that not all Linux system administrators actually want to allow their users to create their own cron jobs. Some will have no problem with it whatsoever. Others will say, "No, we don't allow that."

So, if allowing users to create their own crontab files and run their own programs on a particular schedule makes you a bit nervous, you can lock the system down to prevent them from doing this. The cron daemon will use the two files that you see here, cron.allow and cron.deny--both in the /etc directory--to determine who can and who cannot create crontab schedules.

Now be aware that by default, this file does not exist. The only file that exists when the system is initially set up is the cron.deny file. If you use just cron.deny, then everybody on the system, all users on the system, will be allowed to create cron jobs except for anyone whose account is listed in the cron.deny file.

Now that's one way to approach cron security. Another approach is to go ahead and create a cron.allow file in the /etc directory. If you do this, everything changes.

Basically, if cron.allow exists, then cron.deny is moot. It's not even used. If cron.allow exists, then no users on the system are allowed to create crontab files except for those listed in this file.

User crontab Files 10:30-12:32

So with this in mind, let's take a look at how users can create their own crontab files. In order to do this, they log in as themselves-- in this example I'm logged in as the rtracy user-- and then they type the crontab -e command.

When you run crontab -e, what it actually does is load the vi editor and it creates a new blank crontab file and loads it in the text editor. Then, within this file, the user adds lines for each cron job they want to run, and it uses exactly the same syntax as we saw before for system cron jobs.

This is the minutes field. This is the hour field. This is the day of the month field. Month, and day of the week, and then the command that you actually want to run.

So in this example, the tar command is going to be used to back up my user's home directory--that's the tilda right here--to a file called myhome.tar. And it will be run at what time? Well, 17 is in 24-hour time. So if we subtract 12 from it, we see that it's 5 pm.

So at 5:10 pm of every day of the month, of every month, of every day of the week, the tar command is going to be run. Notice once again these specified the full path to the tar command here.

Now, once you're done creating your crontab file, you exit out of the vi editor just as you would if you're editing any other text file, and you save it.

And when you do, a new crontab file for your user account is created in /var/spool/cron/tabs, and the cron service will be notified that a new crontab file has been created, so it actually reloads itself so that the new configuration that you specified in the crontab file can be applied.

Now there are a couple of other options you can use with the crontab command. We already looked at -e. You can use -l just to view your user's crontab file; it'll just print it out on the screen. And if for some reason you decide, "I don't want a crontab file anymore for my user account," use the -r option with crontab, which will remove your user's crontab file.

Summary 12:33-12:47

That's it for this lesson. In this lesson, we reviewed how you can use the cron daemon on a Linux system to schedule tasks to run automatically on a schedule that you specify. We talked about how cron works. We talked about how to create system cron jobs. Then we ended this lesson by talking about how you can create user specific cron jobs.
