2.5.2 Manage Environment Variables

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

Managing Environment Variables 0:00-0:10

In this demonstration, we're going to spend some time working with environment variables. There are many different ways to view the values of these variables.

View Environment Variables Using env 0:11-1:01

For example, if you want to view the values of all your environment variables at once, you can use the env command. Simply type 'env' at the shell prompt, hit Enter, and a list of all your environment variables are listed along with their current values. If you look at the output of the env command, you'll see that a specific syntax is used. First of all, we have the name of the environment variable, followed by an equals sign, followed by the value that's been assigned. In this example, we have an environment variable named HOME, and it's used to define the home directory for the current user.

You can see that my HOME environment variable is currently set to /home/rtracy. These environment variables are unique to each user. If I were to log in as a different user, say, lmorgan, then the value of the home environment variable would be /home/lmorgan instead of rtracy.

---

echo 1:02-2:14

In addition to the env command, you can also view the value of a single environment variable using the echo command. Let's clear the screen here. Then we enter 'echo' followed by the name of the environment variable that we want to view. Before we do that, you have to understand that the echo command, by default, simply writes text to the screen.

For example, if I say 'echo home', the 'echo' command simply writes "home" on the screen because that's what you told it to do. If you want, instead, for the echo command to write the value for the home environment variable, you have to tell it that. To do this, we type 'echo', and then we put a dollar sign, '$', followed by the name of the environment variable. This dollar sign, right here, is extremely important. That dollar sign tells echo that what comes after is not literal text that needs to be written to the screen, but is, instead, the name of an environment variable, and that echo should go and grab whatever value is currently in that environment variable and display it on the screen instead.

Notice, here, that when I do 'echo $ home', that the value of the home environment variable is displayed.

PATH Environment Variable 2:15-3:37

Another commonly used environment variable is the PATH environment variable. We could type 'echo $PATH', and when we do, we see a list of all the different directories that are contained within the PATH environment variable. These are the directories that the Linux shell is going to look in if you enter a command at the command prompt. For example, I could type the pwd command and it runs. It tells me what my working directory currently is. It's the /home/rtracy directory. I'm in my home directory.

If I do an ls command, there is no pwd executable in my home directory. How did the shell know where pwd exists? Well, it has to exist in one of these directories in the PATH environment variable. In fact, if we type the which command, we can see that the pwd command resides in /usr/bin. If we look up here, we see that that directory is included in the PATH environment variable. Each directory in the PATH environment variable is separated by a colon, so /user/bin is included in the PATH environment variable, so the shell knows where to go to find the pwd command. As I said at the beginning of this demonstration, the values of these various environment variables are automatically populated as soon as you start up a shell session.

Assign a New Value to an Environment Variable 3:38-8:33

But you're not stuck with those default values; you can change them if you want to. Let's use the echo command to look at the value of another environment variable. In this case, let's look at the PS1 environment variable. PS1 defines the format of your shell prompt, right here, what information is included. Press Enter so you can see that there are some literal characters within the PS1 environment variable. For example, we have this bracket right here, opening bracket, and then this closing bracket, right here. That's what's displayed in the prompt. This bracket and this bracket are these two literal characters.

Then we have some other values defined within those brackets. Some of these values are literals. For example, this @ sign, right here, is a literal, and so it's displayed literally in the prompt, just like the brackets were. But these other values are what we call escaped values, which are used to display other types of information. For example, \u tells the shell prompt to display the name of the current user. u equals user. Pretty straightforward, right? Then we have an @ sign, which is a literal. It doesn't have a backslash on it to escape it. So it's displayed in the prompt as is, as an @ sign. Then the next part of the prompt is \h, which displays the hostname of the system, in this case, FS1.

Then there's a space, which is a literal character. So, we have a space, and then we have \W, which displays the current working directory, which, in this case is my home directory, so we see a tilde right there. We're not stuck with this. We can change it if we want. All we have to do is change the value of the environment variable. To do this, we simply type the name of the variable, 'PS1', and an equals sign, just like we saw in the output of the env command, and then the new value that we want to assign to the environment variable. Let's suppose that, in this case, we want to keep the brackets. We'll put the beginning and ending brackets, and we'll do a dollar sign at the end as well. We also want the username to continue to be displayed in the prompt.

That's very useful, so we'll put a '\u', and then we'll put '@', the literal, and then another '\h' to display the host. Therefore, the username@hostname will still be displayed in the prompt. Then let's add a space, which is a literal again. This time, let's suppose we want to add other information. Say, maybe, the time. There's actually a lot of different things we could put inside the prompt. If we put, for example, '\d', we could add the date to the prompt. We could use '\t' to add the time to the prompt in 24-hour format, or you could also put '\@' to display the time in 12-hour format.

You might, at first, be confused, saying, "Well, wait a minute. You have @ here, and you're putting @ here. How's that going to be different?" The difference is the \ character. That makes this @ mean something totally different than just the literal character @. By doing this, we're going to put the time, and then let's go ahead and continue putting the working directory in the prompts. We do '\w' again. At this point, if I press 'Enter', the value of the PS1 environment variable will be changed. Before we do that, we need to place all of these in quotations.

The quotes will not be added to the PS1 environment variable. The quotes simply tell the shell what to put inside of the environment variable. In this case, it will put everything between the quotes. If I press Enter, we see that the shell prompt has now changed according to what we put in PS1. We still have our opening and closing brackets; we have our username, '\u'; the @; the hostname, '\h'; a space; and then the current time, '\@'; a space; and then '\w' for the current directory, the working directory. It's important to note at this point that the value that we just assigned here will not be persistent between shell sessions. If I open up a new shell prompt, PS1 will go back to its default value.

That's because we haven't exported the value of this variable yet. Let's go ahead and do that now. Let's do 'export PS1', press Enter, and now the value of PS1 will be persistent between shell sessions, and my prompt will always look the same. But if I were to reboot the system, I would lose the changes that I just made. It would go back to the default value of PS1 when the system came up again. In order to fix this, I would need to add these two commands right here--this one and this one--to my shell configuration file. Which file that is depends upon which distribution you're using. Most likely, it will be the .bash_profile file within your home directory or the .bashrc file within your home directory.

By doing that, every time the system boots, PS1 will be assigned the value that we specified right here.

New Value to the End of an Environment Variable 8:34-11:42

Before we end this demonstration, we need to talk about how to modify the value of an environment variable without altering the values that are already there. One scenario where this is commonly needed is one where we need to add a directory to the end of our PATH environment variable. Notice, here, when we assign this value to PS1, it completely overwrote the original value. In fact, we can do an 'echo $PS1' here, and you can see that the new value that we specified overwrote the old value that was in there. That's the way environment variables work by default.

Let's suppose that I'm an application developer, and I want to create a folder in my home directory where I can put applications that I'm currently working on, where I can execute them where they won't hurt anything else. I'm going to make a new directory called temp_apps. That's where I'm going to drop my applications that I want to test. The problem here is that this directory, '/home/rtracy/temp_apps', is not in my PATH environment variable. Therefore, to run the applications I'm testing, I would have to provide the path for the application. That can get really old, especially if the path name is really long.

One option for eliminating that is to actually add this path to my existing PATH environment variable. However, if I were to simply say 'PATH=/home/rtracy/temp_apps', if I were to press Enter at this point (and I'm not going to because it would require me to do a little bit of work to fix it), it would wipe out the current value of the PATH environment variable and replace it with this. As soon as I did that, any applications that I had within that directory would run just fine, but nothing else would. None of my commands would work without providing the full path to those applications. So I don't want to do that. Instead, what I want to do is retain the current value of the PATH environment variable and then tack on the new directory to the end.

To do this, I enter 'PATH=', and then I specify '$PATH'. That says take whatever value is currently in the PATH environment variable and assign it to the new value of the PATH environment variable. Basically, we're retaining all of these paths that are already in there. Then I can put a colon because, remember, the paths up here, in the PATH environment variable, are separated by a full colon. I put a full colon down

here, and then I enter the additional path that I want to add: '/home/rtracy/temp_apps'. Press Enter. If I do 'echo $PATH', notice that my original list of paths has been retained, but a new path has been added on to the end so I can drop my apps in there and run them directly.

Once again, in order for this to be persistent across shell sessions, I would need to do an export PATH. If I wanted to make that persistent across reboots, I'd have to go in and edit the appropriate bash shell configuration file.

Summary 11:43-12:03

That's it for this demonstration. In this demo, we talked about managing environment variables. We first talked about what environment variables are. We talked about how to view environment variables with the 'env' command. We talked about how to use echo to view the value of a single environment variable. We then talked about how to assign a new value to an environment variable. And then we talked about how to add a new value to the end of an existing environment variable.
