14.2.1 Bash Shell Environments and Shell Variables

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

Bash Shell Environments and Shell Variables 0:00-0:40

One of the important tools available to a Linux administrator is shell scripting. A well-crafted script can save you time and reduce errors when you're configuring services or performing other complex administrative tasks. You'll find that the most helpful scripts are ones that automate the repetitive tasks you find yourself doing every day, like adding new users.

To write helpful shell scripts, you need a good understanding of the environment that the script runs in. In this lesson, we're going to talk about some of the building blocks of shell scripting, shell environments, environment variables, and shell variables as they relate to the bash shell. These concepts also apply to other shells, as they all function similarly.

Shell Environment 0:41-1:24

Let's start by describing a shell environment. Linux is a multi-processing operating system, and each shell environment is spawned as a process. As a process is created, environment information is gathered from a variety of files and settings on the system, and it's made available to the shell process. This shell environment is implemented as key-value pairs or environment variables. Each environment variable has a name that's assigned a value or multiple values. By convention, the name of an environment variable is written in uppercase. If you create custom variables using lowercase names, you are less likely to accidently modify an environment variable.

It's important to understand how environment variables are added to a shell process, but how this works depends on the shell environment type.

Shell Environment Types 1:25-2:17

If you've interacted with a Linux console or a terminal window, you're familiar with an interactive shell environment. An interactive shell reads from standard-input (the keyboard) and writes to standard-output (the display screen). There are two kinds of interactive shells. If you're required to provide a user ID and password, the environment is classified as a login shell. If not, it's a non-login shell. If you logged in using the Linux console or through a remote SSH session, it's a login shell. An example of a non-login shell is when you open a terminal window in the Gnome desktop environment.

There's one more shell environment type, the non-interactive shell. One way this shell environment is used is when a cron job starts. And here's the big secret: it's also used when a shell script is run. Linux creates a new process, initiates a shell environment, and runs the script in that environment.

Bash Shell Startup Files 2:18-2:58

You're probably aware of the shell startup files that are run when a bash shell is initiated. There are bash startup files that are run for all users, like /etc/profile and /etc/bashrc, and hidden bash startup files that are specific to each user, like .bash_profile and .bashrc, which are found in the user's home directory. Here's another big secret: these bash files only run for interactive shells. This means that bash scripts running in a non-interactive shell environment can't rely on any configurations that are implemented in the startup scripts. This brings up an important question: how do you add the configurations the bash script needs?

Environment Variables 2:59-3:39

One important way to do this is with environment variables. Each Linux process has a parent process. You can see the parent process ID, or PPID, for your process using the ps -f command. Here's the third big secret: when you run a script and a child shell is created, it inherits environment variables from its parent process.

There's a couple of caveats to this. Any environment variable added to the parent shell after the child shell is created won't be available to the child shell. Also, the child shell can manipulate these environment variables without affecting the parent's environment variables. Essentially, a copy of the parent shell's environment variables is given to the child shell.

Create Environment Variables 3:40-4:28

Environment variables are mostly used by the shell. For example, the PATH environment variable is a list of directories used when the shell searches for executable files, and HOSTNAME is the name the shell uses when referring to the host. The printenv command displays a list of environment variables.

To create an environment variable to be used by a script, on one line, you enter the variable's name followed by the equal sign and then the value you'd like to assign to the name. This creates a shell variable. To make it an inheritable environment variable, on the next line, you use the export command followed by the variable name. Alternately, you can create and export an environment variable at the same time using the declare -x command.

This environment variable can be referenced in a script since it's inherited by any child shells.

Shell Variables 4:29-6:24

Many people confuse shell variables with environment variables, since their names are in uppercase and they seem to be available in every shell session. The HISTFILE shell variable is a good example of a shell variable. It points to the file where past shell commands used in an interactive shell are stored. Another example is PS1, which contains the characters used to define the appearance of the shell prompt used in an interactive shell.

Shell variables aren't inherited like environment variables. There are a few that are explicitly created by the bash shell when it's invoked. But most are created by shell startup scripts.

One way to see the difference between environment variables and shell variables is to run two commands, ‘printenv' and ‘set', and compare the results. The printenv command lists only environment variables, while the set command lists all variables, including environment variables. Any variable listed in the set command output that isn't in the print-env output is, strictly, a shell variable.

Another way to see the difference is to add the printenv command to a script. If you run printenv interactively, its results are identical to a script run without it. However, when you do the same thing using the set command, the two outputs will be dramatically different. This is due to the shell variables that are added by the shell startup files that are only run in interactive shell environments. You might think that when the set command is run in a script, its output should be the same as the printenv output. But there are several shell variables that are added when the shell initializes, independent of whether the shell is interactive or non-interactive.

The important thing in all of this is that environment variables--those that are listed by the printenv command--are inherited from parent shells by child shells. On the other hand, shell variables are created during shell initialization or by shell startup scripts.

Create Shell Variables 6:25-7:05

When we discussed creating environment variables that could be used by a script, we listed the steps that you use to create a shell variable. You enter the variable's name followed by the equal sign and then the value you'd like to assign to the name.

Be aware that that a shell variable isn't inherited by a child shell. It's only available in the current shell. You can use the same steps to create a variable in a script. However, we normally classify variables created in a script as user variables.

To convert a shell variable to an environment variable, on the next line, you use the export command. If you need to convert an environment variable back to a shell variable, you use the export -n command.

Summary 7:06-7:30

Now let's review what we've talked about. In this lesson, we described the shell environment. We introduced two shell environment types; interactive shells, including login and non-login shells; and non-interactive shells, which are the type that a shell script runs in. Then we described environment variables that are inherited by child shells and shell variables that are added during shell initialization and by shell startup script. Finally, we discussed how to create environment variables and shell variables in a script.
