Akshat Patel Follow

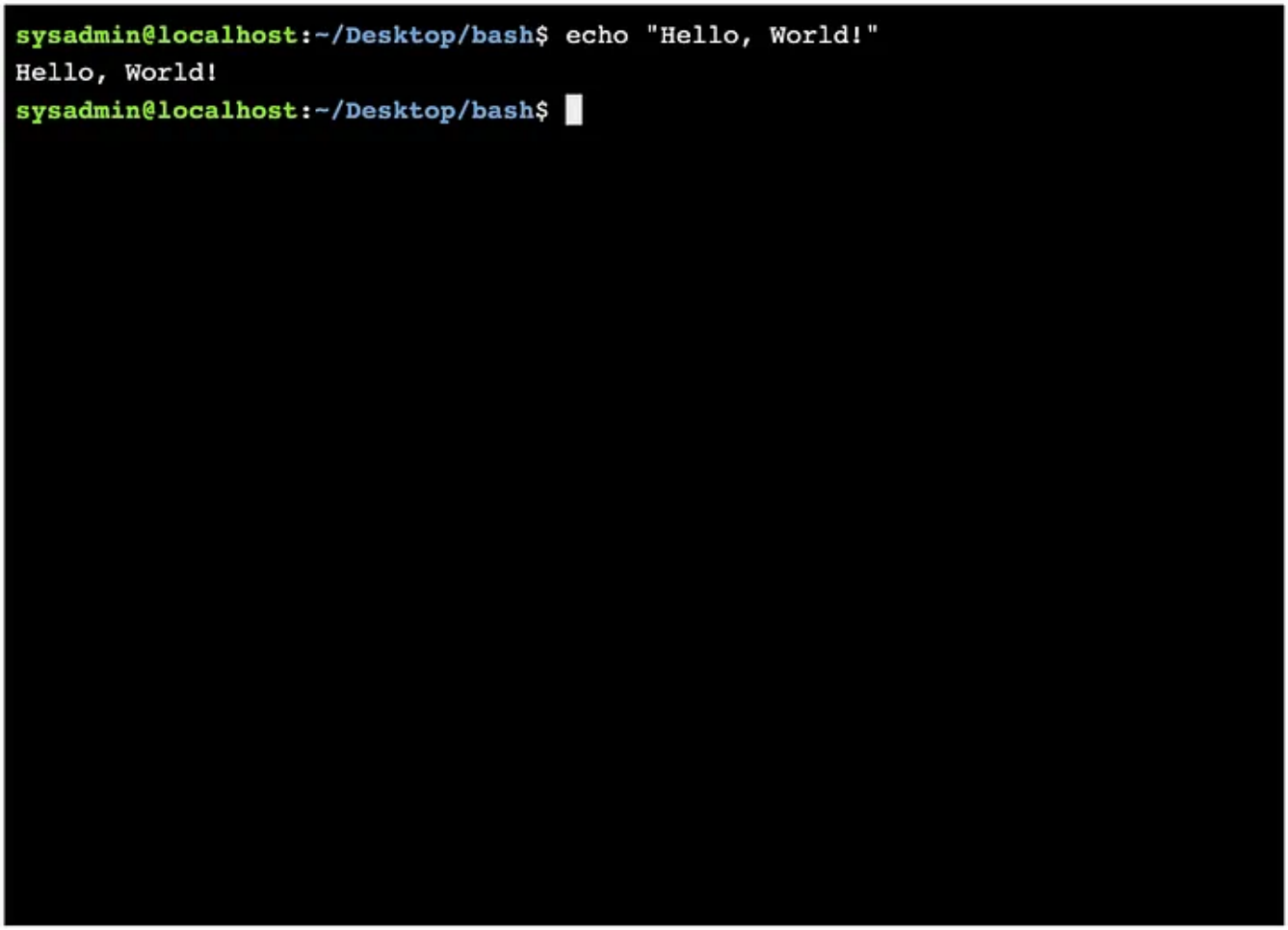
Feb 1 · 5 min read · Listen

Save

Advanced Linux Shell Scripting

A shell script is a collection of executable commands saved in a text file. Each command is executed when the file is run. Shell scripts have access to all shell commands, including logic. A script can thus check for the presence of a file or look for specific output and modify its behavior accordingly. Scripts can be written to automate repetitive parts of your work, freeing up your time and ensuring consistency each time you use the script. For example, if you run the same five commands every day, you can automate them with a shell script, reducing your work to just one command.

Example of Shell Scripting :



In the preceding example, the script is first run as an argument to the shell. The script is then executed directly from the shell. Because the current directory is rarely found in the binary search path $PATH, the name is prefixed with./ to indicate that it should be run from the current directory.

The Permission denied error indicates that the script has not been marked as executable. After a quick chmod, the script is operational. The command chmod is used to change the permissions of a file, which will be covered in greater detail in a later chapter.

There are numerous shells, each with its own language syntax. As a result, more complex scripts will indicate a specific shell by including the absolute path to the interpreter as the first line, prefixed by #!, as shown:

```
#!/bin/sh
echo "Hello, World!"
```

```
or

#!/bin/bash
echo "Hello, World!"
```

The two characters #! are traditionally known as the hash and the bang, respectively, which results in the shortened form “shebang” when used at the start of a script.

The shebang (or crunchbang) is also used in traditional shell scripts and other text-based languages such as Perl, Ruby, and Python. As long as the script is run directly, any text file marked as executable will be run under the interpreter specified in the first line. If the script is run as an argument to an interpreter, such as sh script or bash script, the given shell is used regardless of what is in the shebang line.

Editing of Shell scripting

There are numerous text editors available on UNIX. The advantages of one over the other are frequently debated. The LPI Essentials syllabus specifically mentions two: The GNU nano editor is a simple text editor that works well with small text files. The Visual Editor, vi, or its newer version, VI improved (vim), is a powerful editor with a steep learning curve. We’ll concentrate on nanotechnology.

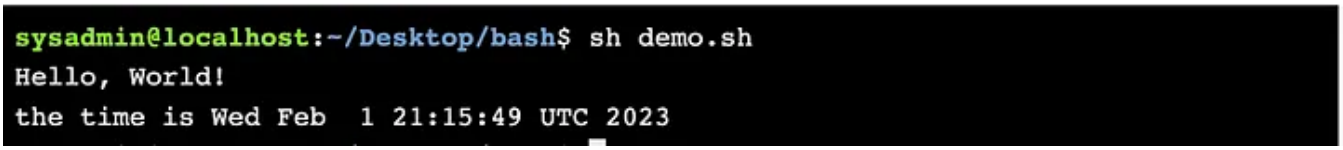
When you run nano test.sh, you’ll see something like this:

```
neno demo.sh
```

```
#!/bin/sh

echo "Hello, World!"
echo -n "the time is "
date
```

The nano editor has a few features to get you started. Simply type with your keyboard, moving around with the arrow keys and deleting text with the delete/backspace button. Along the bottom of the screen are some commands that are context-sensitive and change depending on what you're doing. You can also use the mouse to move the cursor and highlight text if you're directly on the Linux machine rather than connecting via a network. Start typing a simple shell script inside nano to familiarise yourself with the editor.



Scripting Basics

Earlier in this chapter, we gave you your first taste of scripting by introducing a simple script that ran a single command. The script began with the shebang (or hashbang) line, which instructed Linux to use /bin/bash (Bash) to execute the script.

Aside from running commands, there are three topics you must learn:

Variables are variables in the script that hold temporary information. Conditionals allow you to do different things based on the tests you write Loops allow you to do the same thing over and over again.

Variables are a key part of any programming language. A very simple use of variables is shown here:

```
#!/bin/bash

ANIMAL="penguin"
echo "My favorite animal is a $ANIMAL"
```

A directive to assign some text to a variable follows the shebang line. ANIMAL is the variable name, and the equals sign assigns the string penguin. Consider a variable to be a container in which you can store items. Following the execution of this line, the word penguin appears in the ANIMAL box.

There should be no spaces between the variable's name, the equals sign, and the item to be assigned to the variable. If there is a space there, an unusual error such as "command not found" will occur. Although capitalizing the variable name is not required, it is a useful convention for distinguishing variables from commands to be executed.

The script then sends a string to the console. The variable's name is preceded by a dollar sign in the string. When the interpreter sees the dollar sign, it recognizes that it will be substituting the variable's contents, a process known as interpolation. The script's output is then A penguin is my favorite animal.

So remember: to assign to a variable, simply use the variable's name. To access the variable's contents, prefix it with a dollar sign. In this example, we see a variable receiving the contents of another variable!

```
#!/bin/bash

ANIMAL=penguin
```

```
SOMETHING=$ANIMAL
echo "My favorite animal is a $SOMETHING"
```

ANIMAL includes the string penguin (as there are no spaces; in this example, the

common in larger scripts because it allows you to construct a larger command and execute it!

Another method of assigning to a variable is to use the output of another command as the variable's contents by enclosing the command in backticks:

```
#!/bin/bash
CURRENT_DIRECTORY=`pwd`
echo "You are in $CURRENT_DIRECTORY"
```

This pattern is frequently used in text processing. You could take text from one variable or an input file and run it through another command, such as sed or awk, to extract specific parts and save the result in a variable. The sed command is used to edit STDIN streams, while the awk command is used for scripting.

Shell Script Variables Linux Shell Bash Script


Download on the
App Store


GET IT ON
Google Play