UTSA

The University of Texas at San Antonio

University Technology Solutions

Express Bash Scripting Tutorial Part 1

Quickly Learn Bash Scripting in Linux

Brent League

Overview

- Getting Started with the vi Text Editor
- Editing and Executing Your Bash Script
- Making Your Script Executable (CHMOD command)
- Documentation: Adding Comments to Scripts
- The Shebang!
- Creating and Using Variables
- Parameters

Getting Started with vi

- vi is a text editor that is included with most Linux distributions
- To create a new file use the `vi` command
- `login4$ vi test.sh`
- Press i to start inserting text
- The “echo” command prints text to the screen:
- Type the following: `echo Birds Up`
- To save and quit, press “ Esc ” key, and enter `:wq!`
- (press the enter key after typing `:wq!`)

Editing and Executing Your Bash Script

- Type the following: `bash test.sh` and press enter.
- You should see the text you typed in your `test.sh` script displayed on the screen
- Now let's edit your script and add some additional text to it. Type the following: `vi test.sh` and press enter. From within the text editor, add additional text, save the file by pressing the "Esc" key, and enter `:wq!` and pressing the enter key
- Type the following: `bash test.sh` and press enter. You should now see the additional text you added

Making Your Script Executable

- Why do we have to type “bash” to execute our script?
- Files by default do not have the “execute” permission
- Use the “chmod” command to make the script executable. Type the following:

```
chmod +x test.sh
```

Making Your Script Executable

- Now type, `test.sh`
- Notice we received an error that the command was not found
- Linux looks for commands in the path, not the current directory. Now type `./test.sh`
- The `./` in front of the file tells the system not to worry about the path, here's the location of the command

Documentation: Adding Comments to Scripts

- Comments are used to document what the different parts of a script does
- They are not displayed to the user and are beneficial to the programmer for documentation as a script grows.
- Open your script in the text editor again, and add the following line at the top of the script:

```
# Displays text to the screen
```

The Shebang!

- Our script is written specifically for the Bash shell but there are other shells out there
- In order to make sure our script is executed in Bash, and we get the results we want, we need to add the following to the top line of the script
- Open your script in the text editor again, and add the following line at the top of the script:

```
#!/usr/bin/env bash
```

Creating and Using Variables

- Let's create a new script called `greetings.sh` and open it in the vi editor
- Enter the following information in your script, starting at the top line, and then saving the file:

```
#!/usr/bin/env bash

FIRST_NAME=Tom
WEATHER="Partly Cloudy"
echo Hi $FIRST_NAME, the forecast is $WEATHER
```

- Use chmod to make the file executable:

```
chmod 755 greetings.sh
```

- Run your script by typing:

```
./greetings.sh
```

Passing Parameters (1)

- Parameters are used for gathering input from users.
- Let's create a new script called `params.sh`, use **chmod** to make it executable, and then open it in the vi editor
- Enter the following information in your script, starting at the **top** line, and then save the file:

```
#!/usr/bin/env bash

echo Hello $1
```

- Now run the params.sh script as follows

```
./params.sh
Hello
```

- Run the `params.sh` script and add a name to the end of it each time you run it. Notice the difference in output?

```
./params.sh Tom
Hello Tom
./params.sh Tammy
Hello Tammy
```

Passing Parameters (2)

- It may be helpful to use variables for holding the parameters passed from the command line. Since the names in the previous example have no meaning, you should assign a variable to the parameter, so the variable can give some meaning to it. Open your `params.sh` script in the vi editor again
- Let's define a variable as such:

```
USER_NAME=$1
```

- Now, let's change our echo command to use our variable:

```
echo Hello $USER_NAME
```

- Run your `params.sh` script again by typing:

```
./greetings.sh Kelly
```

- The results are the same, but now when we review our script it makes more sense to us

Passing Parameters (3)

- Now we are going to practice adding various system commands to our Bash script. Open `params.sh` with vi and add the code highlighted in red

```
#!/usr/bin/env bash
USER_NAME=$1
echo Hello $USER_NAME
echo $(date)
echo $(pwd)

exit 50
```

- Run your script again and add a name to the end of it as shown below:

```
./params.sh Aurin
```

- You should see the following response which includes system time, as well as your current working directory:

```
Hello Aurin
Thu Sep 3 15:32:47 CDT 2020
/home-new/ytf623
```

Passing Parameters (4)

- Scripts that execute without an error should return a “0” to the system. We can check this by typing `echo $?` (you should receive an exit code of “0” indicating there were no errors and the script completed successfully).

- We can change the value that is returned by editing our `params.sh` script. Enter the following information at the end of the script, and then save the script and exit the vi editor:

```
exit 50
```

- If the script executed without error, you should now receive a “50” in response to the `echo $?` command

UTSA

The University of Texas at San Antonio

University Technology Solutions

Express Bash Scripting Tutorial Part 1

Quickly Learn Bash Scripting in Linux

Brent League