Basics of shell scripting in centos

Topics : Centos Server
Written on March 05, 2024

Shell scripting allows you to automate tasks and execute commands in a sequence, making it a powerful tool for system administration and automation on CentOS. Here are the basics of shell scripting:

1. Choose a Shell:

- CentOS uses the Bash shell (`/bin/bash`) by default, which is the most common shell on Unix-like systems.
- Other shells like Zsh, Ksh, and Dash are also available but less commonly used.

2. Create a Script File:

- Create a new file with a `.sh` extension (e.g., `script.sh`) to write your shell script.
- Use a text editor like `nano`, `vim`, or `gedit` to create and edit the script file.

3. Set the Shebang Line:

- Start the script with a shebang line (`#!`) followed by the path to the shell interpreter.
- For Bash scripts, use `#!/bin/bash`.

4. Write Shell Commands:

- Write shell commands in the script file to perform specific tasks or execute commands.
- Commands can include system commands, utilities, redirections, pipes, variables, loops, conditionals, functions, and more.

5. Execute the Script:

- Make the script executable using the `chmod` command:

```
chmod +x script.sh
```

- Run the script by typing its filename preceded by `./`:

```
./script.sh
```

6. Variables:

- Use variables to store data or values that can be reused throughout the script.
- Declare variables without spaces around the equals sign:

```
variable_name=value
```

- Access variables using the $ prefix:

```
echo $variable_name
```

7. Control Structures:

- Use control structures like loops and conditionals to control the flow of execution in the script.
- Example of a `for` loop:

```
for i in {1..5} do echo "Number: $i" done
```

8. Functions:

- Define functions to encapsulate code blocks that perform specific tasks.
- Example of a function:

```
my_function() { echo "Hello, world!" } my_function
```

9. Input and Output:

- Read input from users using the `read` command:

```
echo "Enter your name:" read name echo "Hello, $name!"
```

- Redirect output to files or pipes using redirection operators (>, >>, <, |, etc.).

10. Comments: - Use comments to add explanatory notes or annotations to your script for better readability. - Comments start with the # symbol and extend to the end of the line.

11. Error Handling: - Implement error handling mechanisms to handle errors gracefully and provide informative error messages to users. - Use `exit` to terminate the script with a specific exit code in case of errors.

12. Debugging: - Use debugging techniques like `set -x` to enable debugging mode and trace the execution of commands in the script. - Print debugging messages using the `echo` command to troubleshoot issues.