Devops

Shell scripting part 1

WHAT IS AUTOMATION?

Automation is a process where we try to reduce manual activities.

Example: If we want to create 100 files using touch command, we can do it manually. But if we need to create 1000 files, then it becomes tedious to do manually. So we will use automation.

Shell Scripting in Linux: Shell scripting in Linux is a process of automating your day-to-day activities on Linux compute. Irrespective of your AWS hosted Linux machine or your laptop Linux machine, as far as you are using the same shell in shell scripting, we can define which shell to execute the script. So it can be executed anywhere.

HOW TO WRITE SHELL SCRIPT

To write a shell script, the basic thing that is required is a file. Inside our file we write a script.

Step 1: Create a Shell Script File

Command: touch first-shell-script.sh

Use ls to check all the files and folders present in current directory. Use ls -ltr for detailed info of each file or folder.

man Command: man command with other command gives info about each command. Example: If we do man touch, it tells the info about this command.

Why use touch command instead of vi? We can basically create a file using vi command, then why use touch command? Touch command is used in automation. We cannot use vi for automation purpose.

WRITING THE SHELL SCRIPT

Step 2: Open and Edit the File

Command: vi first-shell-script.sh

Shebang Line: Start writing the script with #!/ - This is called as shebang. This is the first line everybody writes in a shell script file.

First line options:

- #!/bin/sh
- #!/bin/dash
- #!/bin/ksh
- #!/bin/bash

These are different executables of our shell script and this depends upon the type of our Linux machine. There is ksh, sh, bash or dash which are executables that I want to execute my shell script. Each of them has a little difference in syntax. Bash is widely used.

Difference between sh and bash: Previously, there was an option called linking so /sh would redirect to /bash. But these days or in a year, some of the OS are using /dash as a default. So now it is important to write /bash only if you want bash scripting and not /sh. Always the linking is not linked to /bash.

Write Your First Script:

#!/bin/bash echo "Sushmita Hubli"

Then save this file (Esc + :wq!)

CHECKING FILE CONTENT

If I want to check the content of this file, every time I cannot open this file and check the content using vi command. What if I want to copy the content of this file name and look at the content of the file? I will use cat command.

Command: cat first-shell-script.sh

This displays the content of the file without opening it.

EXECUTING THE SHELL SCRIPT

Method 1: Command: sh first-shell-script.sh

Method 2: Command: ./first-shell-script.sh

When we execute this command using ./, we will get permission denied.

Why Permission Denied? Because in Linux, although we have created a file, Linux terminal wants to understand who can execute this file. The main purpose of Linux is

security. For that reason, whenever we create a file, we need to grant permissions to a file.

GRANTING PERMISSIONS USING chmod

chmod Command: chmod stands for:

- ch = change
- mod = mode

3 Things Required as Arguments:

1. What are permissions for root user
2. What are the permissions for group
3. What are the permissions for all users

In Linux, we categorize permissions using numbers.

Default Permission for Everyone: Command: chmod 777 first-shell-script.sh

After doing this, if I write sh first-shell-script.sh, my file will be executed.

Understanding Permission Numbers:

7 is a magic number:

- 7 for myself
- 7 for my group
- 7 for everyone

Linux Formula 421:

- 4 = read
- 2 = write
- 1 = execute

How to Calculate:

- 7 = 4+2+1 = read + write + execute
- 6 = 4+2 = read + write
- 5 = 4+1 = read + execute
- 4 = read only

- 2 = write only
- 1 = execute only

Examples:

chmod 444 xyz This means: Give read permission for everyone (me, group, and all people), where xyz is the name of the file.

chmod 444 first-shell-script.sh After this, we will not have permission to edit this file (only read permission).

USEFUL COMMANDS

history Command: If I want to see all the commands that I have executed so far, I will use history command to check everything.

pwd Command: Used to know the present working directory.

mkdir Command: mkdir myfirstfolder - Used to create a folder with name "myfirstfolder"

cd Command: cd myfirstfolder - To go to that folder cd .. - Go back to previous directory

---

PRACTICAL EXAMPLE: CREATING A SHELL SCRIPT

Step 1: Create the Script File

Command: touch sample-shell-script.sh

Step 2: Write the Script

Command: vi sample-shell-script.sh

Write the following:

#!/bin/bash #This is a comment #create a folder mkdir Sushmita

we will create a file

cd Sushmita touch firstfile secondfile

Save the file (Esc + :wq!)

Step 3: Grant Permissions

Command: chmod 777 sample-shell-script.sh

Step 4: Execute the Script

Command: sh sample-shell-script.sh

Step 5: Verify

Command: ls You can see a folder named "Sushmita"

Navigate to the folder: Command: cd Sushmita

Check files inside: Command: ls You can see firstfile and secondfile being created inside.

This is the way we execute a simple script.

ROLE OF SHELL SCRIPTING IN DEVOPS

What is the role of shell scripting in organization? In DevOps?

As a DevOps engineer:

- We have infra maintenance tasks
- Maintain code using Git
- We also do configuration management using shell scripting

All of this is done with the help of shell scripting.

Real-World Example:

Person: John Role: DevOps Engineer at Amazon

Scenario:

- John notices that Amazon has 10,000 VMs in his team
- He has to manage these VMs and monitor the node health of these VMs
- Developers are facing issues with VMs like:
    - Memory problems
    - CPU issues
    - Linux becoming slow
    - Processes running on VMs are slow

Without Automation: Every time John has to go to each of these VMs and use Linux commands to fix the issues.

With Automation: Instead of this, he will:

1. Write a script and save it to Git repo

2. Whenever some DevOps engineer says there is an issue with 9000th VM
3. He will execute shell script on his local machine
4. This will log into the 9000th VM
5. Looks at all the parameters
6. Returns info about the VM
7. He will understand what exactly is the issue in that VM

Advanced Automation: If he receives an issue like "memory is full" on a frequent basis, he decides to execute the script automatically:

1. The script will login to each of the 10,000 machines
2. Look at the status of each machine
3. Send a mail to John saying the issues with the VMs

This is one of the use cases for the DevOps engineer.

MONITORING COMMANDS

free Command: Used to monitor the memory of my device Shows memory usage statistics

top Command: Tells the processes running on machine and how much memory they are using Shows real-time system statistics

nproc Command: To check the CPU related info Shows number of CPU cores

Ctrl+C: Used to stop the execution of current script or command

---

SUMMARY

Key Learning Points:

1. Automation reduces manual repetitive tasks
2. Shell scripting is used for automating Linux activities
3. Shebang (#!/bin/bash) is the first line in shell scripts
4. chmod 777 gives full permissions (read, write, execute) to everyone
5. Permission formula: 4 (read) + 2 (write) + 1 (execute) = 7

6. Shell scripting is crucial for DevOps tasks like infrastructure maintenance and configuration management
7. Scripts can be used to automate monitoring and maintenance of thousands of VMs