Linux – Day 08: Shell Scripting Simplified
"Automate the routine, simplify the complex."

Today, let's break down a simple yet powerful **custom shell script** that every DevOps or Cloud Engineer should know how to build. Shell scripts can automate anything from monitoring system resources to deploying infrastructure.

Let's Write a Shell Script

File Name: test.sh
You can use vi, vim, or nano to create the file:

vi test.sh (I am using vi editor)

```
vi test.sh
```

Sample Script Content

```
#!/bin/bash
##########################
# Author : Venkat Uppugunduru
# Date   : 14-05-2025
# Purpose: For Demo Purpose
##########################

set -e           # Exit on any error
set -o pipefail  # Catch pipeline errors
set -x           # Debug mode

df -h            # Check disk space
nproc            # Show number of processors
ps -ef           # List all processes
free -g          # Show memory in GB
```

chmod +x test.sh # Make the script executable

./test.sh # Execute it

Note : we can user chmod 777 test.sh

```
chmod +x test.sh    # Make the script executable
./test.sh           # Execute it
```

Line-by-Line Breakdown

Line 1 – Shebang

#!/bin/bash

Known as **"Shebang" or "Hashbang"**, it tells the OS which interpreter to use for the script. You can also use:

- #!/bin/sh
- #!/bin/dash

Check your default shell using:

echo $SHELL

Change it with: chsh -s /bin/dash

Line 2–4 – Script Metadata

Always include author, date, and purpose. It's good practice for readability and collaboration.

Line 5 – set -e

Exits the script if **any command fails**. This prevents partial execution which can cause major issues in automation.

Example use case:
If you're creating a user and a file afterward — the file should only be created *if the user creation succeeds*. set -e ensures that.

Line 6 – set -o pipefail

By default, errors in piped commands might go unnoticed. This ensures the script exits if **any** part of a pipeline fails.

Line 7 – set -x

Enables **debug mode**, which prints each command before executing it — very useful for troubleshooting and learning.

Why Shell Scripting Matters?

In a world of automation, knowing how to write and manage shell scripts can:

- Automate health checks
- Deploy applications
- Monitor systems

- Save hours of repetitive work!

Up Next: We'll build more advanced scripts to monitor live system health, automate user creation, and more!