Devops

Shell scripting part 2

## ADVANCED SHELL SCRIPTING CONCEPTS

---

### CUSTOM SHELL SCRIPT FOR NODE HEALTH MONITORING

**df Command:** df command is used for printing all info about available disk space on VM.

**Creating Node Health Script:**

We will write a custom shell script to detect node health of our VM and save it in Git repo.

Command: vi nodehealth.sh

**Basic Script:**

#!/bin/bash df -h free -g nproc

Save the file and grant permissions: Command: chmod 777 nodehealth.sh

Execute the script: Command: sh nodehealth.sh

The output will be visible but we know and understand the output because we have written the script. But in order for the script output to be understood by everyone, we can add echo statements in the script to improve readability.

---

### TWO OPTIONS TO IMPROVE SCRIPT READABILITY

**Option 1: Use echo Statements Before Commands**

#!/bin/bash #Author: Sushmita #Date: 05/11/2026

#This script outputs the node health

#Version: v1 ############# echo "print the disk space" df -h echo "print the memory" free -g echo "print the cpu" nproc

In this way save this and now when we execute this script, output will be properly understandable by everyone.

**Problem with echo statements:** Using echo statement all the time is not feasible, especially for large scripts.

**Option 2: Use set -x (Debug Mode)**

set -x is written before the shell script commands. This is used to set the shell script in debug mode.

#!/bin/bash #Author: Sushmita #Date: 05/11/2026

#This script outputs the node health

#Version: v1 ############# set -x df -h free -g nproc

**Output with set -x:**

- df -h Filesystem Size Used Avail Use% Mounted on /dev/root 6.8G 2.4G 4.4G 36% / tmpfs 479M 0 479M 0% /dev/shm tmpfs 192M 904K 191M 1% /run tmpfs 5.0M 0 5.0M 0% /run/lock /dev/xvda16 881M 151M 669M 19% /boot /dev/xvda15 105M 6.2M 99M 6% /boot/efi tmpfs 96M 12K 96M 1% /run/user/1000
- free -g total used free shared buff/cache available Mem: 0 0 0 0 0 0 Swap: 0 0 0
- nproc 1

**Explanation:** The first line will be + the command executed, then the output of that command, and similarly for other commands in the script. This is a better way because if we have a huge script, then always writing echo statements is very tedious.

---

**PROCESS MANAGEMENT COMMANDS**

**ps -ef Command:** ps -ef is used for knowing the processes running.

**Finding a Particular Process:**

Command: ps -ef | grep "amazon"

This will give the processes which are related to amazon which are running.

**Pipe Command:** Here the pipe command sends the output of the first command to the second command.

**Example to Understand Pipe:**

Create a script with: echo 1 echo 11 echo 12 echo 13

When we execute: sh test.sh - Output: 1, 11, 12, 13 sh test.sh | grep 11 - Output: 11

Result:

ubuntu@ip-172-31-22-21:~~$ sh test.sh 1 11 12 13 ubuntu@ip-172-31-22-21:~~$ sh test.sh | grep 11 11

**Important Note:** It is a good practice to write comments in our script so that it could be understood by others as well.

**Understanding Pipe with date Command:**

Command: date | echo "today is" Output: today is

**Why?** Pipe transfers the output or stdout of first command to second command. But in this case, the date command sends output to stdout but echo doesn't read from stdin, so the pipe doesn't send the output of first command to the other command.

**Getting Process ID:**

What if I just want the process ID of amazon processes?

Command: ps -ef | grep "amazon" | awk -F" " '{print $2}'

This prints the column 2 values (process IDs) of processes of amazon.

---

**ERROR HANDLING IN SHELL SCRIPTS**

**Important Commands for Error Handling:**

Whenever we use pipe command in our script, it is necessary to write:

1. set -e - Exits the script when there is an error
2. set -o pipefail - Ensures errors in pipes are caught

**Testing Different Scenarios:**

**Script 1: Without set -e**

#!/bin/bash set -x guguiguiguidsg df -h echo "hello" free -g

**Output:**

- guguiguiguidsg testingpipe.sh: 4: guguiguiguidsg: not found
- df -h Filesystem Size Used Avail Use% Mounted on /dev/root 6.8G 2.4G 4.4G 36% / tmpfs 479M 0 479M 0% /dev/shm tmpfs 192M 900K 191M 1% /run tmpfs 5.0M 0 5.0M 0% /run/lock /dev/xvda16 881M 151M 669M 19% /boot /dev/xvda15 105M 6.2M 99M 6% /boot/efi tmpfs 96M 12K 96M 1% /run/user/1000
- echo hello hello
- free -g total used free shared buff/cache available Mem: 0 0 0 0 0 0 Swap: 0 0 0

**Result:** Script continues execution even after error.

**Script 2: With set -e**

#!/bin/bash set -x set -e guguiguiguids df -h echo "hello" free -g

**Output:**

ubuntu@ip-172-31-22-21:~$ sh testingpipe.sh

- set -e
- guguiguiguids testingpipe.sh: 6: guguiguiguids: not found

**Result:** Script stops execution after error.

**Script 3: With set -e but using pipe**

#!/bin/bash set -x set -e #set -o pipefail guguiguiguids | echo "Hello" df -h echo "hello" free -g

**Output:**

ubuntu@ip-172-31-22-21:~$ sh testingpipe.sh

- set -e
- echo Hello
- guguiguiguids testingpipe.sh: 7: guguiguiguids: not found Hello
- df -h Filesystem Size Used Avail Use% Mounted on /dev/root 6.8G 2.4G 4.4G 36% / tmpfs 479M 0 479M 0% /dev/shm tmpfs 192M 900K 191M 1% /run tmpfs 5.0M 0 5.0M 0% /run/lock /dev/xvda16 881M 151M 669M 19% /boot /dev/xvda15 105M 6.2M 99M 6% /boot/efi tmpfs 96M 12K 96M 1% /run/user/1000
- echo hello hello
- free -g total used free shared buff/cache available Mem: 0 0 0 0 0 0 Swap: 0 0 0

**Result:** set -e stops the script execution if there is any error, but if we have anything which is not a command with | echo "hello", hello will be printed and other commands will be executed. But we want to stop execution if something in the script is written which is not a command.

**Script 4: With set -e and set -o pipefail**

#!/bin/bash set -x set -e set -o pipefail guguiguiguids | echo "Hello" df -h echo "hello" free -g

**Output:**

ubuntu@ip-172-31-22-21:~$ sh testingpipe.sh

- set -e

- set -o pipefail testingpipe.sh: 6: set: Illegal option -o pipeline

**Note:** Correct syntax is "pipefail" not "pipeline"

**Best Practice - Combine All Three:**

set -exo pipefail

This can be written as one line at the beginning of the script.

---

**LOG FILE ANALYSIS - MAJOR DEVOPS USE CASE**

**Scenario:** Let's say there are 100 apps that are running. If one of the apps fails, then look into the log file. Every time you run into an error, there is one success mantra: Go into your log file and find error in the log file.

This is the general practice that every DevOps engineer in every company does. Whenever an app fails, they simply go to the log file and try to find the error.

**Problem:** Log files are very very huge.

**Finding Errors in Log Files:**

Command: cat logfile | grep "error"

**The Twist:** There are lots of logs, so what people do is they upload the logs to some storage platforms. Let's say you upload it to Google Storage or S3. So this is not present on your VM. So how will you retrieve this info?

---

**curl AND wget COMMANDS**

**curl Command:** This can be done with the help of curl command.

Command: curl "url of the place where your logs are stored"

curl command can be used to get request to different pages like google.com, etc.

**wget Command:** There is another command called wget.

Command: wget url

**Difference between curl and wget:**

curl: Gets the output directly in the command itself wget: Downloads the file and stores it locally

**Problem with wget:** Using wget we have to execute 2 commands (download, then read), but using curl, we can get the output of log file in command itself.

---

**find COMMAND**

**Purpose:** We have find command with the help of which we can find the location of a file.

**Examples:**

Command: find / Result: Finds everything in the system

Command: sudo find / -name pam.d Result: Finds the file/folder named "pam.d"

**Why use sudo?**

Usually we don't use root user in our organization because:

- Root is a powerful user
- We can delete anything
- Deleted items cannot be retrieved back unless we have snapshots

**Switching to Root User:**

Command: sudo su - Result: Takes you to the root user

**Switching to Another User:**

Command: sudo su - priya Result: Goes to priya user

**Why sudo with find?** When we use find command to find a file or folder, we get "permission denied" in lots of places, so we use:

Command: sudo find / -name pam.d

Result: Now we will get the location of this pam.d file in the system.

---

**LOOPS IN SHELL SCRIPTING**

**1. if-else Loop:**

a=4 b=10 if [ $a -gt $b ] then echo "a is greater than b" else echo "b is greater than a" fi

**Output when we run this script:** b is greater than a

**2. for Loop:**

Script: for i in {1..100}; do echo $i done

This will print numbers from 1 to 100.

---

**trap COMMAND**

**What is trap command?** trap command is used for trapping signals.

**Background:** When we want to kill a process:

Command: kill -9 processid

When we execute this command, there is a signal passed behind the scene to Linux machine. There are lots of signals which are used in Linux. When we use kill command, SIGKILL signal is used behind the scenes.

**Purpose of trap Command:** trap command is used to trap a signal.

**Example:** If you have trap Ctrl+C, then whenever Ctrl+C is pressed, the signal is trapped and this does not stop the execution of the script.

**Use Case:** This is useful when you want to:

- Perform cleanup operations before script exits
- Prevent accidental termination of critical scripts
- Execute specific commands when certain signals are received
- Ensure graceful shutdown of processes

**Basic Syntax:**

trap 'commands' SIGNAL

Example: trap 'echo "Ctrl+C pressed but script continues"' SIGINT

This traps the SIGINT signal (Ctrl+C) and executes the echo command instead of stopping the script.

---

**SUMMARY**

Key Learning Points:

1. set -x enables debug mode in scripts
2. Pipe (|) sends output of first command to second command
3. set -e exits script on error
4. set -o pipefail catches errors in pipes
5. Best practice: set -exo pipefail at start of script
6. curl is better than wget for reading remote log files

7. find command locates files in system (use sudo to avoid permission errors)
8. trap command prevents signal interruption of scripts
9. Always add comments in scripts for better understanding
10. Log file analysis is a major DevOps responsibility