BASH SCRIPTING

Bash Scripting

Introduction

This tutorial will give you a solid platform in how to create bash scripts and automate day to day system admin tasks. Certainly, everything cannot be covered in this chapter but you will be equipped with right amount of knowledge to make your own scripts and excel in it if you put in your own efforts. Bash scripting is used by many of the system admins and DevOps geeks to get things done quickly and efficiently. There are so many automation tools in the market like Ansible, Puppet, Chef etc. Which are way more sophisticated but sometimes to get things done quickly in Linux systems we use Bash scripts. Also, scripting will make you understand what automation means and then you can quickly grasp the features that is used in Configuration Management tools like Ansible or puppet.

🧱 SCRIPTBOX PROJECT — FULL STEP-BY-STEP NOTES (Part 1 + Part 2 Combined)

✅ 1. Project Overview

- You are creating a shell scripting practice environment using **Vagrant**.
- There are 3 main CentOS-based VMs: `scriptbox`, `web01`, `web02` and one Ubuntu VM: `web03`.
- You'll write and run scripts **primarily on `scriptbox`** first.
- Later, you'll push the scripts to `web01` and `web02`.

🗂️ 2. Folder Setup and Vagrantfile Preparation

📁 Step-by-step:

Create a directory where your Vagrantfile will live:

bash
CopyEdit

```
mkdir "D:/bash-scripts"
cd "D:/bash-scripts"
```

1.

Copy the Vagrantfile into this folder (from Downloads or other location):

bash
CopyEdit

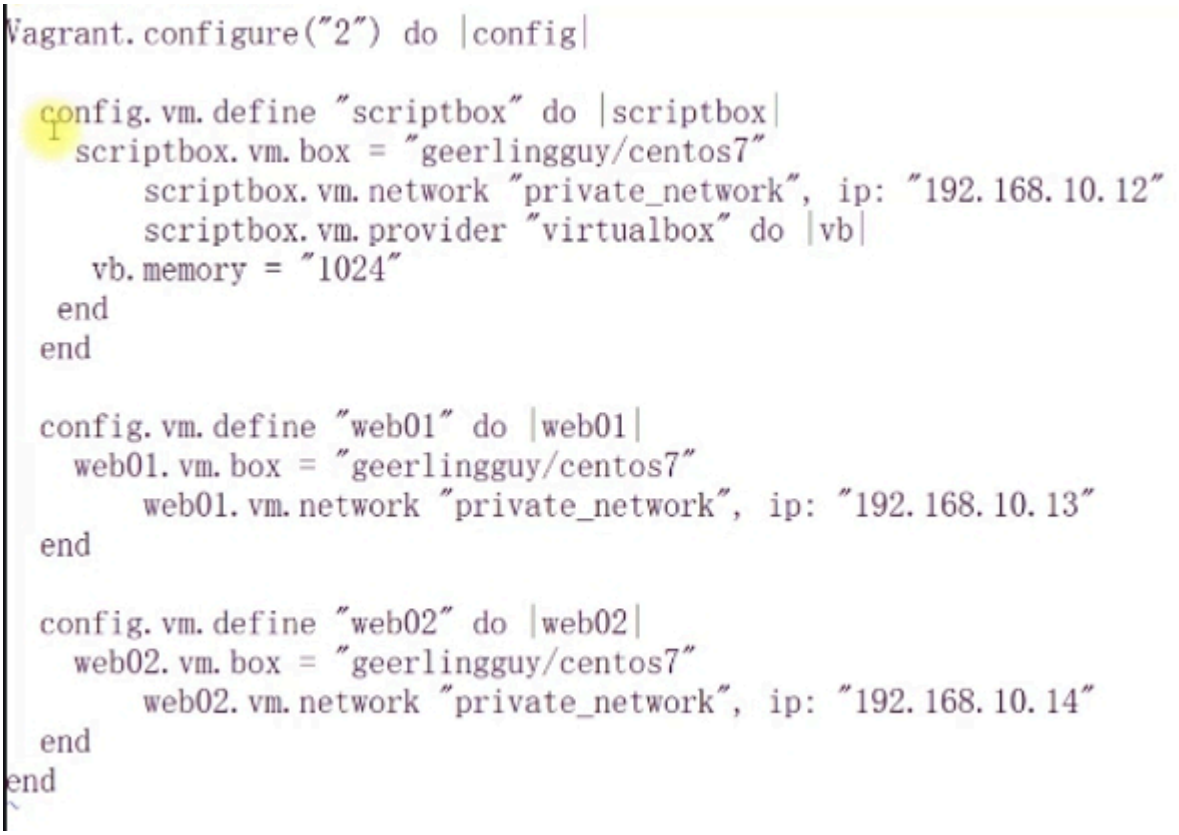
```
cp ~/Downloads/Vagrantfile .
```

2.

📄 3. Full Vagrantfile Used in This Setup

ruby

CopyEdit



```
Vagrant.configure("2") do |config|

  config.vm.define "scriptbox" do |scriptbox|
    scriptbox.vm.box = "geerlingguy/centos7"
        scriptbox.vm.network "private_network", ip: "192.168.10.12"
        scriptbox.vm.provider "virtualbox" do |vb|
     vb.memory = "1024"
   end
  end

  config.vm.define "web01" do |web01|
    web01.vm.box = "geerlingguy/centos7"
        web01.vm.network "private_network", ip: "192.168.10.13"
  end

  config.vm.define "web02" do |web02|
    web02.vm.box = "geerlingguy/centos7"
        web02.vm.network "private_network", ip: "192.168.10.14"
  end
end
```

```
Vagrant.configure("2") do |config|

  config.vm.define "scriptbox" do |scriptbox|
    scriptbox.vm.box = "eurolinux-vagrant/centos-stream-9"
    scriptbox.vm.network "private_network", ip: "192.168.10.12"
    scriptbox.vm.hostname = "scriptbox"
    scriptbox.vm.provider "virtualbox" do |vb|
      vb.memory = "1024"
    end
  end

  config.vm.define "web01" do |web01|
    web01.vm.box = "eurolinux-vagrant/centos-stream-9"
    web01.vm.network "private_network", ip: "192.168.10.13"
    web01.vm.hostname = "web01"
  end

  config.vm.define "web02" do |web02|
    web02.vm.box = "eurolinux-vagrant/centos-stream-9"
    web02.vm.network "private_network", ip: "192.168.10.14"
    web02.vm.hostname = "web02"
```

```
  end

  config.vm.define "web03" do |web03|
    web03.vm.box = "ubuntu/bionic64"
    web03.vm.network "private_network", ip: "192.168.10.15"
    web03.vm.hostname = "web03"
  end
end
```

🚀 4. Booting and Logging into Scriptbox VM

bash
CopyEdit

```
vagrant up scriptbox
vagrant ssh scriptbox
```

> ℹ️ Note: This boots only the `scriptbox` VM. You can bring up all VMs later.

🖥️ 5. Set Hostname Inside the VM

bash
CopyEdit

```
sudo -i
echo "scriptbox" > /etc/hostname
hostname
exit
vagrant ssh scriptbox
```

- After re-login, prompt should reflect hostname as `scriptbox`.

📦 6. Create Scripts Directory

bash
CopyEdit

```
sudo -i
mkdir -p /opt/scripts
cd /opt/scripts
```

✏️ 7. Install vim (if not present)

bash
CopyEdit

```
yum install vim -y
```

🗒️ 8. Writing the First Script

bash
CopyEdit

```
vim firstscript.sh
```

🧠 Script Content:

bash
CopyEdit

```
#!/bin/bash

### This script prints system info ###

# Checking system uptime
echo
echo "######################################"
echo "The uptime of the system is:"
uptime
echo

# Memory Utilization
echo "######################################"
echo "Memory Utilization"
free -m
echo

# Disk Utilization
echo "######################################"
echo "Disk Utilization"
df -h
```

🔎 Key Concepts:

- **Shebang line** `#!/bin/bash`: Tells OS to use bash interpreter.
- **Comments** with `#`: Ignored by shell; used for readability.
- **Commands**: Basic Linux utilities like `uptime`, `free`, `df`.
- **Output formatting** using `echo` for clean display.

🔐 9. Make Script Executable

bash
CopyEdit
```
chmod +x firstscript.sh
```

▶️ 10. Run Script (Both Ways)

➤ Relative path:

bash
CopyEdit
```
./firstscript.sh
```

➤ Absolute path:

bash
CopyEdit
```
/opt/scripts/firstscript.sh
```

📌 11. Important Concepts from Transcript

Concept	Explanation
`Shebang (#!)`	Instructs system which interpreter to use (`bash`, `python`, etc).
`chmod +x`	Adds execute permission so script can run.
`vim`	Used to write scripts inside terminal.
`echo`	Used for printing to output.
`uptime`, `df`, `free`	Commands showing system health info.

Script readability	Comments (#) and structured output make script and output easier to read.
Script is just a text file	With list of shell commands you can re-run anytime.
Best Practice	Always comment your scripts — even if small — so others or you can understand them later.

✅ 12. Script Output Sample (Formatted)

bash
CopyEdit

```
#####################################
The uptime of the system is:
 13:48:27 up 1:23,  2 users,  load average: 0.00, 0.01, 0.05

#####################################
Memory Utilization
              total        used        free      shared  buff/cache
available
Mem:            980         109         691           8         179
722
Swap:           511           0         511

#####################################
Disk Utilization
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1        40G  1.8G   38G   5% /
...
```

🧹 13. Cleanup & Best Practices

- Use `exit` to come out of root.
- `vagrant halt scriptbox` when not working to free resources.
- Always store your scripts in `/opt/scripts` or similar organized directory.

📦 SCRIPTBOX PROJECT — VIDEO 3: Website Setup Script (Standalone Guide)

🎯 Goal of This Script

Instead of manually typing and executing a series of commands to:

- Install Apache
- Download a website template
- Extract and copy it to the `html` directory
- Start Apache

👉 **We automate all of it** via a **Shell script**.

🧱 Step-by-Step Breakdown

✅ 1. Script Name and Location

We create a new script called `websetup.sh`
Location: Inside `/opt/scripts/`

bash
CopyEdit

```
cd /opt/scripts
vim websetup.sh
```

✅ 2. Final Script Content

bash
CopyEdit

```
#!/bin/bash

# ----------------------------------------
# Installing Dependencies
# ----------------------------------------
echo "########################################"
```

```
echo "Installing packages."
echo "#######################################"
sudo yum install wget unzip httpd -y > /dev/null
echo

# ---------------------------------------
# Start & Enable HTTPD Service
# ---------------------------------------
echo "#######################################"
echo "Start & Enable HTTPD Service"
echo "#######################################"
sudo systemctl start httpd
sudo systemctl enable httpd
echo

# ---------------------------------------
# Creating Temp Directory
# ---------------------------------------
echo "#######################################"
echo "Starting Artifact Deployment"
echo "#######################################"
mkdir -p /tmp/webfiles
cd /tmp/webfiles
echo

# ---------------------------------------
# Downloading and Extracting Template
# ---------------------------------------
wget https://www.tooplate.com/zip-templates/2098_health.zip >
/dev/null
unzip 2098_health.zip > /dev/null
sudo cp -r 2098_health/* /var/www/html/
echo

# ---------------------------------------
# Bounce Service
# ---------------------------------------
echo "#######################################"
echo "Restarting HTTPD service"
echo "#######################################"
systemctl restart httpd
```

```
echo

# ---------------------------------------
# Clean Up
# ---------------------------------------
echo "#########################################"
echo "Removing Temporary Files"
echo "#########################################"
rm -rf /tmp/webfiles
echo

# ---------------------------------------
# Status Check
# ---------------------------------------
echo "#########################################"
echo "Checking HTTPD Status and Content Deployed"
echo "#########################################"
sudo systemctl status httpd
ls /var/www/html/
```

🧠 Concepts Covered in the Script

🔧 Concept	🗒️ Explanation
`#!/bin/bash`	Shebang line – tells system to use bash interpreter.
`yum install wget unzip httpd -y`	Installs required packages non-interactively (`-y`).
`> /dev/null`	Hides normal output, but allows errors to appear.
`mkdir -p /tmp/webfiles`	Creates dir only if it doesn't exist — avoids errors.
`wget`	Downloads the zip template from the internet.
`unzip`	Extracts zip file into a folder.
`cp -r`	Recursively copies the website content to `/var/www/html/`.
`sudo`	Required because `/var/www/html` is owned by root.

`systemctl start/enable/restart httpd`	Starts, enables, and restarts the Apache service.
`rm -rf`	Cleans up the temporary download directory.
`systemctl status` + `ls`	Final verification: checks service is running and files are copied.

🔒 3. Make Script Executable

bash
CopyEdit

```
chmod +x /opt/scripts/websetup.sh
```

▶️ 4. Run the Script

bash
CopyEdit

```
/opt/scripts/websetup.sh
```

- You may face "permission denied" if it's not executable.
- Output will be **clean and readable**, with important information shown, unnecessary clutter suppressed.

📌 5. Debugging Notes from Transcript

Issue	Fix
Unzipped folder name mismatch	Make sure you unzip and reference folder `2098_health`, not the zip file directly.
Too much output from install/unzip	Use `> /dev/null` (but **not** **`2>&1`**) to hide only stdout, **retain error messages**.
User experience readability	Use `echo` for section headers. Add line spacing. Keep script user-friendly.

🌐 6. Access Website in Browser

Get your VM's IP:

bash
CopyEdit
```
ip addr show
```
or

bash
CopyEdit
```
ifconfig
```

1.

Visit:

cpp
CopyEdit
```
http://<your-vm-ip>
```

2.

✅ You should see the **Health Center template site** served by Apache.

✅ 7. Output Example

shell
CopyEdit
```
########################################
Installing packages.
########################################

########################################
Start & Enable HTTPD Service
########################################

########################################
Starting Artifact Deployment
########################################

########################################
Restarting HTTPD service
########################################

########################################
```

```
Removing Temporary Files
########################################

########################################
Checking HTTPD Status and Content Deployed
########################################
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled;
vendor preset: disabled)
   Active: active (running)
...
css/  fonts/  images/  index.html  js/
```

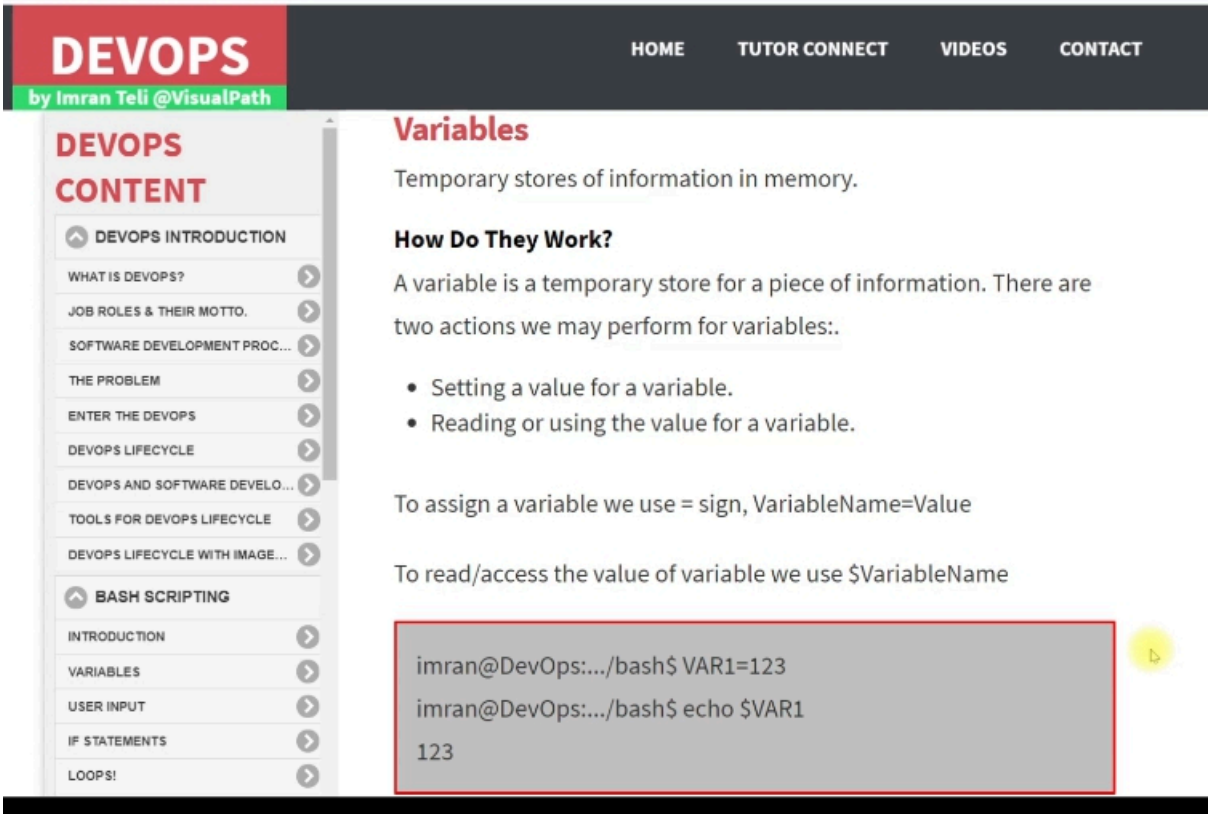
🧹 8. Summary

✅ **What you achieved:**

- Installed Apache
- Deployed a website template from internet
- Verified status
- Cleaned up

🚀 All with **one clean, commented, reusable script**!

📘 DevOps Notes – Web Server Deployment Script Using Variables



🧠 Part 1: Conceptual Foundations

✅ What Are Variables in Bash?

In Bash, a **variable** is a named reference to a value stored temporarily in memory (RAM). These are used to store information that a script can reuse later.

Examples:

bash
Copy code

```
SKILL="DevOps"
echo $SKILL    # Output: DevOps
```

If we echo without `$`, like `echo SKILL`, it prints the literal word `SKILL`.

✅ Why Use Variables in DevOps Scripts?

Variables are used to:

- **Avoid Repetition**: No need to rewrite values like package names or URLs repeatedly.
- **Enable Easy Changes**: Want to switch URLs or service names? Change the variable.
- **Improve Clarity**: Scripts are easier to understand with meaningful variable names like `PACKAGE`, `URL`, `TEMPDIR`, etc.

✅ When to Use a Variable?

Use a variable when:

- A value is used **multiple times** (like `httpd` service).
- A value is **likely to change** (like download URL or artifact name).
- You want **cleaner scripts** that are easier to update and maintain.

🖥️ Part 2: Script Setup Context

We are working inside a **Vagrant-provisioned CentOS VM** named `scriptbox`.

The directory containing all our Bash automation scripts is:

bash
Copy code
`/opt/scripts/`

We previously created:

- `firstscript.sh` — a basic script
- `websetup.sh` — a working script for installing Apache and deploying a template

We now **rename** them and begin using **variables** to improve the script:

bash

Copy code

```
mv firstscript.sh 1_firstscript.sh
mv websetup.sh 2_websetup.sh
cp 2_websetup.sh 3_vars_websetup.sh
```

Now we edit `3_vars_websetup.sh` and refactor it to use variables.

📜 Part 3: Full Script – `3_vars_websetup.sh`

bash
Copy code

```
#!/bin/bash

# Variable Declaration
PACKAGE="httpd wget unzip"
SVC="httpd"
URL='https://www.tooplate.com/zip-templates/2098_health.zip'
ART_NAME='2098_health'
TEMPDIR="/tmp/webfiles"

# Installing Dependencies
echo "########################################"
echo "Installing packages."
echo "########################################"
sudo yum install $PACKAGE -y > /dev/null
echo

# Start & Enable Service
echo "########################################"
echo "Start & Enable HTTPD Service"
echo "########################################"
sudo systemctl start $SVC
sudo systemctl enable $SVC
echo

# Creating Temp Directory and Deploying Artifact
echo "########################################"
echo "Starting Artifact Deployment"
echo "########################################"
mkdir -p $TEMPDIR
```

```
cd $TEMPDIR
wget $URL > /dev/null
unzip $ART_NAME.zip > /dev/null
sudo cp -r $ART_NAME/* /var/www/html/
echo

# Bounce Service
echo "########################################"
echo "Restarting HTTPD service"
echo "########################################"
systemctl restart $SVC
echo

# Clean Up
echo "########################################"
echo "Removing Temporary Files"
echo "########################################"
rm -rf $TEMPDIR
echo

# Final Verification
sudo systemctl status $SVC
ls /var/www/html/
```

🔍 Part 4: Line-by-Line Explanation

🔹 `#!/bin/bash`

- This is the **shebang** line.
- It tells the system to execute the script using `/bin/bash`.

🔹 Variable Declarations

bash
Copy code

```
PACKAGE="httpd wget unzip"
```

- A variable holding **multiple package names** (Apache web server, file retriever, and unzipper).
- Will be passed directly to `yum install`.

bash
Copy code
```
SVC="httpd"`
```

- Service name to be started, enabled, restarted later.

bash
Copy code
```
URL='https://www.tooplate.com/zip-templates/2098_health.zip'
```

- Website template URL to be downloaded.

bash
Copy code
```
ART_NAME='2098_health'
```

- The directory name created after unzipping the template.
- Used later to copy content into the Apache document root.

bash
Copy code
```
TEMPDIR="/tmp/webfiles"`
```

- A temporary location where downloaded files are stored before deployment.

🔹 Installing Packages

bash
Copy code
```
echo "########################################"
```

```
echo "Installing packages."
echo "#######################################"
```

- Section banner for clarity in terminal output.

bash
Copy code
```
sudo yum install $PACKAGE -y > /dev/null
```

- Installs all three packages in one go using `$PACKAGE`.
- `-y` auto-confirms prompts.
- `> /dev/null` silences standard output but **not errors** (very important).
    - Useful to keep script output clean but still show issues.

🔹 Starting and Enabling Apache

bash
Copy code
```
sudo systemctl start $SVC
sudo systemctl enable $SVC
```

- `start`: Immediately runs the Apache (`httpd`) server.
- `enable`: Ensures it **starts on boot**.

🔹 Deploying the Artifact

bash
Copy code
```
mkdir -p $TEMPDIR
cd $TEMPDIR
```

- `mkdir -p`: Creates the temp folder only if it doesn't already exist.

- `cd`: Moves into the temp directory for downloading the zip.

bash
Copy code
```
wget $URL > /dev/null
```

- Downloads the zip file silently.

bash
Copy code
```
unzip $ART_NAME.zip > /dev/null
```

- Extracts the zip file into a folder named `2098_health`.

bash
Copy code
```
sudo cp -r $ART_NAME/* /var/www/html/
```

- Copies all extracted HTML/CSS files into Apache's default root folder.

🔹 Restarting Apache Service

bash
Copy code
```
systemctl restart $SVC
```

- "Bounce" the server. Ensures that Apache restarts with the latest changes.

🔹 Cleanup

bash
Copy code
```
rm -rf $TEMPDIR
```

- Deletes all temp files to free space.
- `-rf` ensures that the command does not prompt or fail due to non-empty folders.

🔹 Final Status and Verification

bash
Copy code

```
sudo systemctl status $SVC
```

- Shows Apache service status. Should be **active (running)**.

bash
Copy code

```
ls /var/www/html/
```

- Verifies that the template files are successfully deployed.

🔁 Optional: Dismantle Everything Before Testing

To ensure a clean test, a script was created beforehand:

bash
Copy code

```
sudo systemctl stop httpd
sudo rm -rf /var/www/html/*
sudo yum remove httpd wget unzip -y
```

Purpose:

- Remove previously installed packages and files.
- Useful to demonstrate that the script works from a clean slate.

✅ Summary of Key Learnings

Concept	Purpose
Variables	Reduce repetition and enable easier updates
yum install	Automates software installation
systemctl	Controls services like Apache (`start`, `enable`, `restart`, `status`)
wget/unzip	Retrieves and unpacks website content
cp	Moves content to the web server's root directory
/dev/null	Suppresses command output but not errors
rm -rf	Safely deletes temp files without confirmation

📦 Output Expectations (After Running)

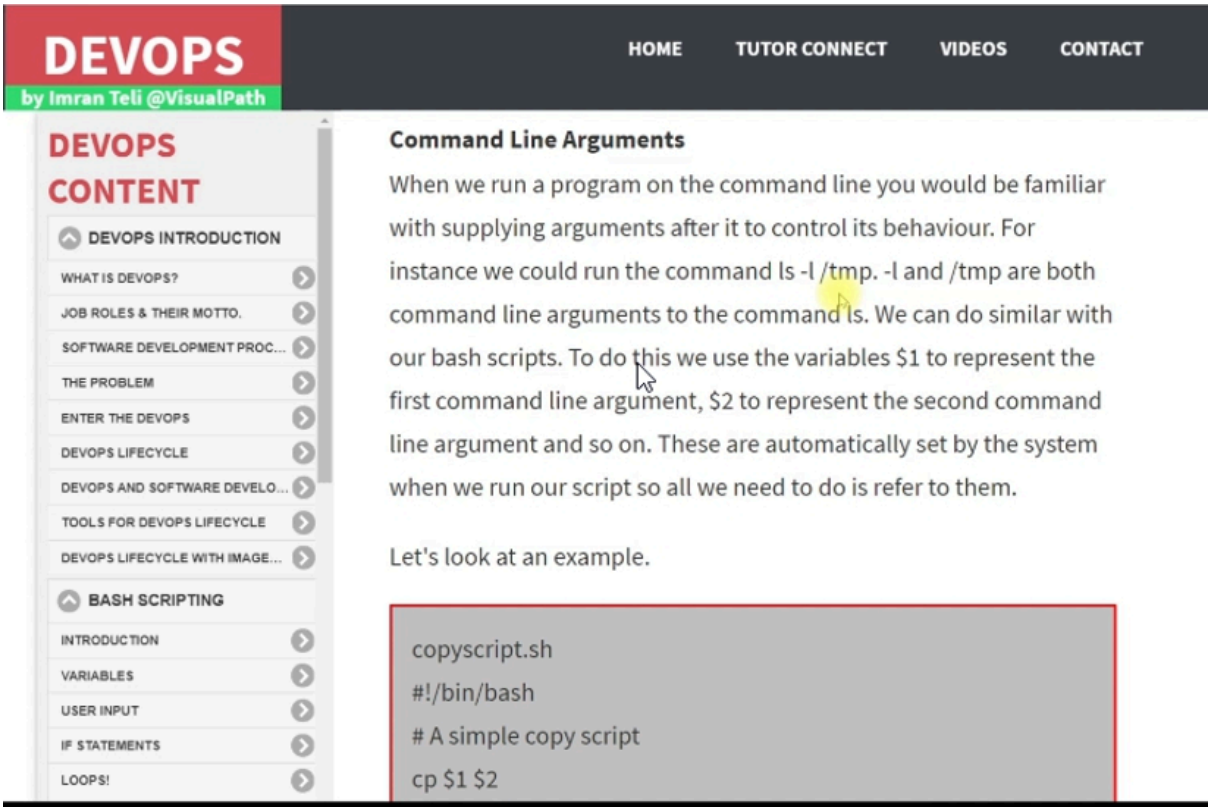
- Apache (`httpd`) is running.
- Static HTML site is deployed at `http://<VM-IP>/`.
- `/var/www/html/` contains all the website files.
- Temporary files are cleaned up.
- The service is enabled to auto-start on reboot.

📘 DevOps Full Notes — Lecture: Command-Line Arguments in Bash

---

🧠 Introduction – What Are Command-Line Arguments?

We've seen many Linux commands that take **arguments**:

- `ls /home` → the path `/home` is an argument.
- `cp file1.txt file2.txt` → both the source and destination are arguments.
- `mv old.txt new.txt` → again, arguments.
- 

➡️ These arguments **change the behavior** of the command.

The question now is:
Can we pass arguments to our own scripts like that?

Yes. And we'll now learn how.

🖥️ Switching to VM: `scriptbox`

From within the `scriptbox` VM terminal:

bash
Copy code
```
clear
```

(Clears previous output.)

📝 Step 1: Create the Argument Script

bash
Copy code
```
vim 4_args.sh
```

Inside this file, type the following:

bash
Copy code
```
#!/bin/bash

echo "Value of 0 is "
echo $0

echo "Value of 1"
echo $1

echo "Value of 2"
echo $2

echo "Value of 3"
echo $3
```

> ❗ `$0`, `$1`, `$2`, etc., are **positional parameters** that hold the script name and arguments respectively.

Save and exit:

- Press `Esc`
- Type `:wq`
- Press `Enter`

🔐 Step 2: Make It Executable

bash
Copy code

```
chmod +x 4_args.sh
```

🧪 Step 3: Basic Variable Behavior Check

Try this:

bash
Copy code

```
x=123
echo $x
```

✅ Output: `123`

Now try accessing an **undefined variable**:

bash
Copy code

```
echo $y
```

❌ Output: nothing (empty string)

➡️ This shows: if you try to access a variable you haven't defined, Bash will **not throw an error** — it will just return an empty string.

▶️ Step 4: Run the Argument Script Without Arguments

bash
Copy code

```
./4_args.sh
```

Output:

csharp
Copy code

```
Value of 0 is
./4_args.sh
Value of 1

Value of 2

Value of 3
```

Explanation:

- `$0` = script name
- `$1`, `$2`, `$3` = empty (no args passed)

▶️ Step 5: Run With Arguments

bash
Copy code

```
./4_args.sh Linux Apache Docker
```

Output:

csharp
Copy code

```
Value of 0 is
./4_args.sh
Value of 1
Linux
Value of 2
Apache
Value of 3
Docker
```

You can pass even more than 3 arguments, up to `$9`.

💡 Positional Parameter Limits

Parameter	Meaning
`$0`	Script name
`$1-$9`	First to ninth argument
`${10}`	Use curly braces for 10+

📦 Step 6: Apply Arguments to Deployment Script

We now want to use arguments in a **real script**:

- Instead of hardcoding the **URL** and **artifact folder name**, we'll pass them as `$1` and `$2`.

📋 Copy existing script

bash
Copy code

```
cp 3_vars_websetup.sh 5_args.sh
```

✍️ Open in editor

bash
Copy code

```
vim 5_args.sh
```

🛠️ Edit the Script to Use Command-Line Arguments

Inside `5_args.sh`, go to the top where variables are defined:

bash
Copy code

```
#URL='https://www.tooplate.com/zip-templates/2098_health.zip'
#ART_NAME='2098_health'
```

Comment out those two lines.

Then scroll to the lines that used `$URL` and `$ART_NAME` and **replace them**:

bash
Copy code
```
wget $1 > /dev/null
unzip $2.zip > /dev/null
sudo cp -r $2/* /var/www/html/
```

Explanation:

- `$1` will be the **zip download URL**
- `$2` will be the **extracted folder name** (used in unzip and copy)

Now the script is generic — you can deploy any website template by passing the URL and folder name.

🛑 Important Warning:

If user doesn't pass arguments, these will be empty. That would run:

bash
Copy code
```
wget
unzip .zip
```

Which will **fail**. So arguments are mandatory.

💣 Dismantle Existing Setup (For Clean Test)

Before running the new script:

bash
Copy code
```
sudo systemctl stop httpd
sudo rm -rf /var/www/html/*
```

```
sudo yum remove httpd wget unzip -y
```

🌐 Get a New Website Template

Go to https://www.tooplate.com

Choose a new template — for example: **Ziggy**

Right-click Download → Inspect (Dev Tools → Network tab)
Click “Download” → get the `.zip` link.

Example:

python
Copy code
```
https://www.tooplate.com/zip-templates/2103_ziggy.zip
```

Folder name (after unzip):

Copy code
```
2103_ziggy
```

▶️ Run the Script With Arguments

bash
Copy code
```
./5_args.sh https://www.tooplate.com/zip-templates/2103_ziggy.zip
2103_ziggy
```

Output: (silent mostly due to `> /dev/null`, but section headers appear)

shell
Copy code
```
##########################################
Installing packages.
##########################################
...
##########################################
Start & Enable HTTPD Service
##########################################
```

```
...
########################################
Starting Artifact Deployment
########################################
...
########################################
Restarting HTTPD service
########################################
...
########################################
Removing Temporary Files
########################################
```

🌐 Access Deployed Website

bash
Copy code

```
ip a
```

Use browser and visit:

cpp
Copy code

```
http://<VM-IP>
```

You'll see:
✅ "Welcome to Ziggy"

🗒️ Full Final Script: `5_args.sh`

bash
Copy code

```
#!/bin/bash

# Variable Declaration
PACKAGE="httpd wget unzip"
SVC="httpd"
# URL and ART_NAME are passed as arguments
TEMPDIR="/tmp/webfiles"
```

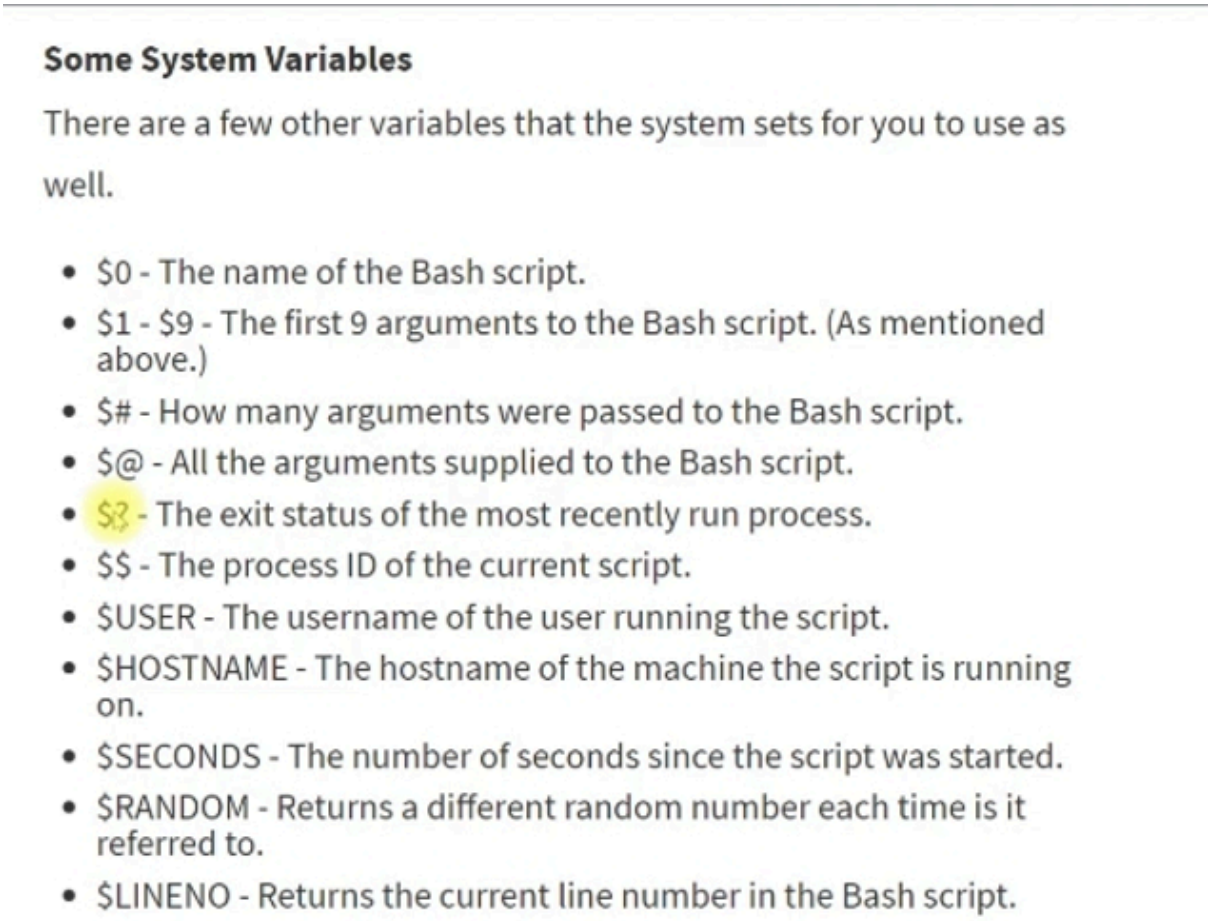
```
# Installing Dependencies
echo "########################################"
echo "Installing packages."
echo "########################################"
sudo yum install $PACKAGE -y > /dev/null
echo

# Start & Enable Service
echo "########################################"
echo "Start & Enable HTTPD Service"
echo "########################################"
sudo systemctl start $SVC
sudo systemctl enable $SVC
echo

# Deploy Artifact
echo "########################################"
echo "Starting Artifact Deployment"
echo "########################################"
mkdir -p $TEMPDIR
cd $TEMPDIR
echo

wget $1 > /dev/null
unzip $2.zip > /dev/null
sudo cp -r $2/* /var/www/html/
echo

# Restart Service
echo "########################################"
echo "Restarting HTTPD service"
echo "########################################"
systemctl restart $SVC
echo

# Clean Up
echo "########################################"
echo "Removing Temporary Files"
echo "########################################"
rm -rf $TEMPDIR
```

```
echo

# Final Check
sudo systemctl status $SVC
ls /var/www/html/
```

🧠 Summary

Concept	Explanation
`$0`	Script name
`$1`, `$2`, ..., `$9`	Arguments passed to the script
Argument Reuse	Makes the script dynamic and reusable
Undefined Variable Access	Returns empty string; doesn't throw error
wget $1	Download site from user-passed URL
unzip $2.zip	Extract based on folder name passed
Clean Testing	Stop service, clean files, reinstall packages if needed

📘 DevOps Notes — Lecture: System Variables and Exit Status (`$?`, `$#`, `$@`, `$USER`, `$HOSTNAME`, `$RANDOM`, etc.)



Some System Variables

There are a few other variables that the system sets for you to use as well.

- $0 - The name of the Bash script.
- $1 - $9 - The first 9 arguments to the Bash script. (As mentioned above.)
- $# - How many arguments were passed to the Bash script.
- $@ - All the arguments supplied to the Bash script.
- $? - The exit status of the most recently run process.
- $$ - The process ID of the current script.
- $USER - The username of the user running the script.
- $HOSTNAME - The hostname of the machine the script is running on.
- $SECONDS - The number of seconds since the script was started.
- $RANDOM - Returns a different random number each time is it referred to.
- $LINENO - Returns the current line number in the Bash script.

🎯 Objective

Learn about **system variables** (also called environment variables or shell variables) used in Bash scripting:

Variable	Description
`$0`	Name of the script
`$1–$9`	Command-line arguments
`$#`	Number of arguments passed to the script

`$@`	All arguments passed to the script
`$?`	Exit status of the **last executed command**
`$USER`	Current logged-in username
`$HOSTNAME`	Hostname of the system
`$RANDOM`	A random number

🔧 Practical: Test These Variables

Step 1: Create a new script

bash
Copy code

```
vim 6_sysvars.sh
```

Paste this content:

bash
Copy code

```
#!/bin/bash

echo "---- System Variable Demo ----"

echo "Script name (\$0): $0"
echo "Total number of arguments (\$#): $#"
echo "All arguments (\$@): $@"

echo "First argument (\$1): $1"
echo "Second argument (\$2): $2"

echo "Current User (\$USER): $USER"
echo "Hostname (\$HOSTNAME): $HOSTNAME"
echo "Random Number (\$RANDOM): $RANDOM"
echo

echo "---- Testing Exit Status (\$?) ----"
echo "Running a valid command: free -m"
```

```
free -m
echo "Exit status: $?"

echo
echo "Running an invalid command: freeeeeee"
freeeeeee
echo "Exit status: $?"

echo
echo "Running a partially valid command: free -x"
free -x
echo "Exit status: $?"

echo
```

Save & exit:

- Press `Esc`, then type `:wq` and hit `Enter`

Step 2: Make the script executable

bash
Copy code

```
chmod +x 6_sysvars.sh
```

Step 3: Run the script with some arguments

bash
Copy code

```
./6_sysvars.sh linux apache
```

✅ Expected Output:

pgsql
Copy code

```
---- System Variable Demo ----
Script name ($0): ./6_sysvars.sh
Total number of arguments ($#): 2
All arguments ($@): linux apache
```

```
First argument ($1): linux
Second argument ($2): apache

Current User ($USER): vagrant
Hostname ($HOSTNAME): scriptbox
Random Number ($RANDOM): 27054

---- Testing Exit Status ($?) ----
Running a valid command: free -m
# [Output of free -m]
Exit status: 0

Running an invalid command: freeeeeee
bash: freeeeeee: command not found
Exit status: 127

Running a partially valid command: free -x
free: invalid option -- 'x'
Try 'free --help' for more information.
Exit status: 1
```

💡 Explanation of Exit Status

Exit Code	Meaning
`0`	Success
`1`	General error (e.g., wrong option)
`127`	Command not found

This logic is heavily used later for:

- Validating inputs
- Conditional flow control (`if`, `then`)
- Logging failures

🧠 BONUS: Integrate System Variables into `5_args.sh`

Let's now **update** the previous deployment script to use some of these variables.

✅ Full Final Script: `5_args_sysinfo.sh`

bash
Copy code

```
#!/bin/bash

# Variable Declaration
PACKAGE="httpd wget unzip"
SVC="httpd"
TEMPDIR="/tmp/webfiles"

echo "########################################"
echo "Script Metadata"
echo "########################################"
echo "Script: $0"
echo "Total arguments: $#"
echo "Arguments: $@"
echo "Executed by: $USER on $HOSTNAME"
echo "Random ID for session: $RANDOM"
echo

# Check if both arguments are passed
if [ $# -lt 2 ]; then
  echo "❌ Error: Not enough arguments."
  echo "Usage: $0 <URL> <ARTIFACT_NAME>"
  exit 1
fi

# Installing Dependencies
echo "########################################"
echo "Installing packages."
echo "########################################"
sudo yum install $PACKAGE -y > /dev/null
echo "Exit status: $?"

# Start & Enable Service
echo "########################################"
```

```
echo "Start & Enable HTTPD Service"
echo "########################################"
sudo systemctl start $SVC
sudo systemctl enable $SVC
echo "Exit status: $?"

# Creating Temp Directory
echo "########################################"
echo "Starting Artifact Deployment"
echo "########################################"
mkdir -p $TEMPDIR
cd $TEMPDIR

wget $1 > /dev/null
if [ $? -ne 0 ]; then
  echo "❌ Failed to download from $1"
  exit 2
fi

unzip $2.zip > /dev/null
if [ $? -ne 0 ]; then
  echo "❌ Failed to unzip $2.zip"
  exit 3
fi

sudo cp -r $2/* /var/www/html/
echo

# Bounce Service
echo "########################################"
echo "Restarting HTTPD service"
echo "########################################"
systemctl restart $SVC

# Clean Up
echo "########################################"
echo "Removing Temporary Files"
echo "########################################"
rm -rf $TEMPDIR

# Final Check
```

```
echo "########################################"
echo "Deployed site files:"
ls /var/www/html/
echo
sudo systemctl status $SVC
```

💡 New Features Added:

Feature	Purpose
`$USER` / `$HOSTNAME`	Metadata logging
`$RANDOM`	Session ID simulation
`$#` / `$@`	Argument validation
`$?`	Command success/failure
`exit 1/2/3`	Proper exit codes for debugging

✅ How to Run

bash
Copy code

```
bash 5_args_sysinfo.sh
https://www.tooplate.com/zip-templates/2103_ziggy.zip 2103_ziggy
```

🧾 Final Summary Table

Variable	Description	Sample
`$0`	Script name	`./script.sh`
`$#`	No. of arguments	`2`
`$@`	All arguments	`arg1 arg2`
`$1`, `$2`	First and second arguments	URL, folder

`$USER`	Logged-in user	`vagrant`
`$HOSTNAME`	VM hostname	`scriptbox`
`$RANDOM`	Random number	`23728`
`$?`	Exit status of previous command	`0`, `1`, `127`

Command Substitution

Variable defined in the script leave with it and dies after the script dies or completes. If we want to define a variable that is accessible to all the scripts from your current shell we need to export it..

```
1.imran@DevOps:.../testcopy$ file=`ls`
imran@DevOps:.../testcopy$ echo $file
copyscript.sh dir1 dir2
```

```
imran@DevOps:.../testcopy$ files=$(ls)
imran@DevOps:.../testcopy$ echo $files
copyscript.sh dir1 dir2
```

📘 Lecture: Quotes in Bash – Understanding " vs ' in Variable Handling

---

🎯 Objective

Understand how **single quotes (`'`) and double quotes (`"`) behave differently in bash**, especially with **variables** and **special characters** like `$`.

This lesson focuses on:

- Assigning and echoing variables
- Behavior of single vs double quotes
- How to handle literal `$` signs
- How to escape special characters using `\`

🧪 Environment Setup

- **VM Hostname**: `scriptbox`
- **Working Directory**: `/opt/scripts`
- **Logged-in User**: `root`

Terminal prompt confirms this:

csharp
CopyEdit
`[root@scriptbox scripts]#`

🧱 Section 1: Assigning and Accessing Variables

🔹 Set a variable with double quotes:

bash

CopyEdit
`SKILL="DevOps"`

Double quotes are used to assign the string `DevOps` to the variable `SKILL`.

🔸 Print the variable:

bash
CopyEdit
`echo $SKILL`

✅ Output:

nginx
CopyEdit
`DevOps`

Explanation:

- `$SKILL` retrieves the value of the variable `SKILL`
- Since double quotes allow variable expansion, the value is displayed correctly

---

🔹 Now assign using single quotes:

bash
CopyEdit
`SKILL='DevOps'`

Again, we assign the same string. Even though we use single quotes, the value is still correctly assigned.

🔸 Print the variable again:

bash
CopyEdit
`echo $SKILL`

✅ Output:

nginx
CopyEdit

```
DevOps
```

Explanation:

- Single or double quotes can be used for plain string assignments
- Since the variable is used **outside quotes**, bash still expands it

---

🧪 Section 2: Printing Variables Inside Quotes

✅ Print inside double quotes:

bash
CopyEdit

```
echo "I have got $SKILL skill."
```

✅ Output:

css
CopyEdit

```
I have got DevOps skill.
```

Explanation:

- Inside **double quotes**, bash expands variables like `$SKILL`
- The full sentence is interpolated properly

❌ Print inside single quotes:

bash
CopyEdit

```
echo 'I have got $SKILL skill.'
```

❌ Output:

nginx
CopyEdit

```
I have got $SKILL skill.
```

Explanation:

- Inside **single quotes**, **variable names are not expanded**
- Bash treats `$` and everything after as **plain text**
- This is the key difference

📌 Summary Table: Variable Behavior

Quote Type	Variable Expanded?	Use Case Example
Double Quotes	✅ Yes	`"I have $VAR"` → expands `VAR`
Single Quotes	❌ No	`'$VAR'` → literal string: `$VAR`

🧪 Section 3: Printing Literal $ with Variable in Same Line

Let's try printing:

nginx
CopyEdit
```
Due to $VIRUS virus company have lost $9 million.
```

But we want:

- `$VIRUS` to be **expanded**
- `$9` to be treated **literally**, as **"$9"**

🔹 Step 1: Set the variable

bash
CopyEdit
```
VIRUS="covid19"
```

❌ Step 2: Print with double quotes — wrong output:

bash
CopyEdit

```
echo "Due to $VIRUS virus company have lost $9 million."
```

❌ Output:

css
CopyEdit

```
Due to covid19 virus company have lost  million.
```

Explanation:

- `$9` is interpreted as **ninth positional argument** (which doesn't exist)
- Result: `$9` is replaced with **empty string**
- So `9 million` disappears

❌ Step 3: Print with single quotes — wrong output:

bash
CopyEdit

```
echo 'Due to $VIRUS virus company have lost $9 million.'
```

❌ Output:

nginx
CopyEdit

```
Due to $VIRUS virus company have lost $9 million.
```

Explanation:

- Everything is printed **literally**
- `$VIRUS` is **not expanded**

- This defeats the purpose

✅ Step 4: Escape the `$` character

bash
CopyEdit
```
echo "Due to $VIRUS virus company have lost \$9 million."
```

✅ Output:

nginx
CopyEdit
```
Due to covid19 virus company have lost $9 million.
```

Explanation:

- We use `\$` to **escape** the dollar sign
- `\$` tells bash **"treat the `$` as a literal character"**
- `$VIRUS` still expands, because it's **not escaped**

🧠 Concept: Escaping Special Characters

Character	Description	Escape Method
`$`	Variable prefix	`\$`
`"`	Double quote inside string	`\"`
`'`	Single quote inside `'...'`	Use `'\''` or switch to `"`
`\`	Escape character itself	`\\`

📜 Final Examples – Complete Commands to Try Yourself

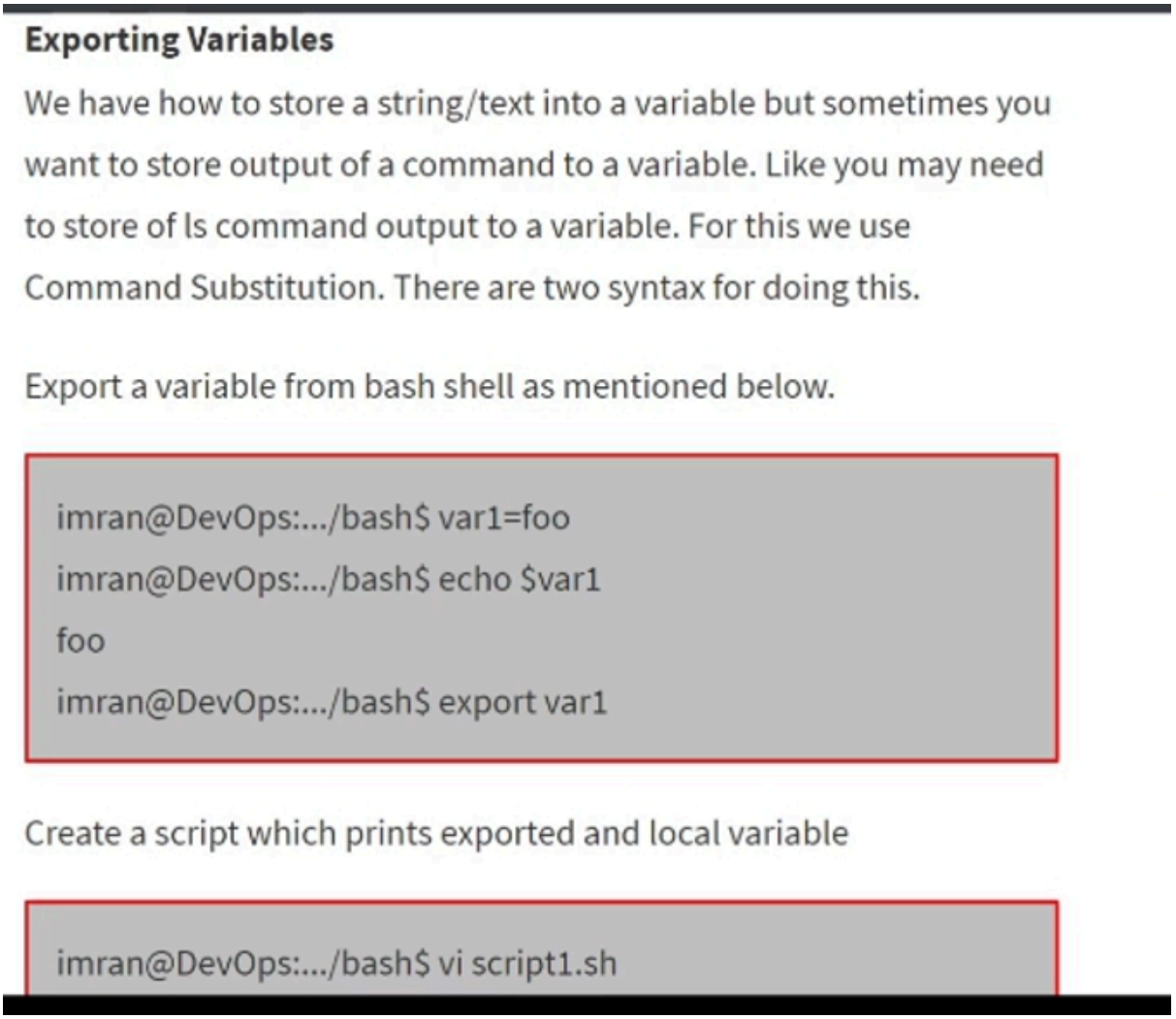
bash
CopyEdit

```
# Assign variables
SKILL="DevOps"
VIRUS="covid19"

# Demonstrating variable expansion
echo "I have got $SKILL skill."              # ✅ expands
echo 'I have got $SKILL skill.'              # ❌ no expansion

# Special character handling
echo "Due to $VIRUS virus company have lost $9 million."   # ❌
wrong, $9 is positional
echo 'Due to $VIRUS virus company have lost $9 million.'   # ❌
wrong, literal output
echo "Due to $VIRUS virus company have lost \$9 million."  # ✅
correct
```

🧠 Key Takeaways

- **Double quotes (`"`)**: expand variables and special characters
- **Single quotes (`'`)**: treat everything literally, no expansion
- **Escape special characters** using `\` when inside double quotes
- **`$9`, `$1`, etc.** refer to **command-line arguments**, so use `\$` when you mean literal dollar values

Exporting Variables and Making Them Permanent



Exporting Variables

We have how to store a string/text into a variable but sometimes you want to store output of a command to a variable. Like you may need to store of ls command output to a variable. For this we use Command Substitution. There are two syntax for doing this.

Export a variable from bash shell as mentioned below.

```
imran@DevOps:.../bash$ var1=foo
imran@DevOps:.../bash$ echo $var1
foo
imran@DevOps:.../bash$ export var1
```

Create a script which prints exported and local variable

```
imran@DevOps:.../bash$ vi script1.sh
```

🎯 Objective

By the end of this section, you'll understand:

Concept	Covered ?
Temporary vs Persistent variables	✅
Use of `export`	✅
Variable scope: current shell vs child shell	✅

How to pass variables to scripts	✅
Making variables persistent across reboots	✅
Modifying `.bashrc`, `.bash_profile`, `/etc/profile`	✅
Precedence between system-wide and user variables	✅

🗂️ Main Code Files Used

✅ `/opt/scripts/testvars.sh`

bash
CopyEdit

```
#!/bin/bash

echo "The $SEASON season is more than expected, this time."
```

✅ `/root/.bashrc` (Root User's Bash Configuration)

(at the end of the file add this line)

bash
CopyEdit

```
export SEASON="Monsoon"
```

✅ `/etc/profile` (System-Wide Global Shell Config)

(at the very end of the file add)

bash
CopyEdit

```
export SEASON="Winter"
```

🧪 Step-by-Step Terminal Session

Follow exactly what was done in the terminal, word for word.

🔹 Step 1: Define and Access a Variable in Current Shell

bash
CopyEdit

```
[root@scriptbox scripts]# SEASON="Monsoon"
[root@scriptbox scripts]# echo SEASON
SEASON
[root@scriptbox scripts]# echo $SEASON
Monsoon
```

> 🔍 `$SEASON` expands the value, `SEASON` alone does not.

🔹 Step 2: Log Out and Log Back In

bash
CopyEdit

```
[root@scriptbox scripts]# exit
logout
[vagrant@scriptbox ~]$ sudo -i
[root@scriptbox ~]# echo $SEASON
```

> ❌ No output — the variable is **lost** when shell session ended. That's because it was **not exported or persisted**.

🔹 Step 3: Create and Run the Test Script

bash
CopyEdit

```
[root@scriptbox scripts]# vim testvars.sh
```

Paste:

bash
CopyEdit

```
#!/bin/bash
echo "The $SEASON season is more than expected, this time."
```

Make it executable:

bash
CopyEdit
```
[root@scriptbox scripts]# chmod +x testvars.sh
```

Try running it **before exporting the variable**:

bash
CopyEdit
```
[root@scriptbox scripts]# ./testvars.sh
The  season is more than expected, this time.
```

> ❌ `$SEASON` is empty in child shell.

🔹 Step 4: Export the Variable to Pass to Script

bash
CopyEdit
```
[root@scriptbox scripts]# export SEASON
[root@scriptbox scripts]# ./testvars.sh
The Monsoon season is more than expected, this time.
```

✅ Now it works. `export` sends variable into **child shell environments** like your script.

🔹 Step 5: Log Out and Confirm Variable Disappears Again

bash
CopyEdit
```
[root@scriptbox scripts]# exit
logout
[vagrant@scriptbox ~]$ sudo -i
[root@scriptbox ~]# echo $SEASON
```

> ❌ Again, it's gone — because we haven't persisted it.

📌 Part 2: Making Variables Permanent

✅ Option A: Per-User (Root) – Use `.bashrc`

bash
CopyEdit

```
[root@scriptbox ~]# vim ~/.bashrc
```

➡️ Add this line to the end:

bash
CopyEdit

```
export SEASON="Monsoon"
```

Save (`:wq`) and exit.

Now log out and log back in:

bash
CopyEdit

```
[root@scriptbox ~]# exit
logout
[vagrant@scriptbox ~]$ sudo -i
[root@scriptbox ~]# echo $SEASON
Monsoon
```

> ✅ Root user's `~/.bashrc` worked. Variable is now available at login for root.

✅ Option B: Global – Use `/etc/profile`

bash
CopyEdit

```
[root@scriptbox ~]# vim /etc/profile
```

➡️ Add this line at the bottom:

bash
CopyEdit

```
export SEASON="Winter"
```

Save (`:wq`) and exit.

Now **log out from everyone**, even vagrant user:

bash
CopyEdit

```
[root@scriptbox ~]# exit
logout
[vagrant@scriptbox ~]$ exit
logout
Connection to 127.0.0.1 closed.
```

Reconnect to box:

bash
CopyEdit

```
$ vagrant ssh scriptbox
[vagrant@scriptbox ~]$ echo $SEASON
Winter
```

✅ Global setting worked — vagrant inherited variable from `/etc/profile`.

🔄 What About Precedence?

Log in as root again:

bash
CopyEdit

```
[vagrant@scriptbox ~]$ sudo -i
[root@scriptbox ~]# echo $SEASON
Monsoon
```

🧠 **Explanation:**

- `/etc/profile` set it as `Winter` (for everyone)
- But root's `.bashrc` overwrote it to `Monsoon`

✅ **`.bashrc` > `/etc/profile` for root user**

🔚 Summary of File Precedence

Location	Affects	Overridden by
`/etc/profile`	All users	User `.bashrc`/`.bash_profile`
`~/.bash_profile`	Current user only	n/a
`~/.bashrc`	Current user only	n/a

✅ Best Practices

- Use `export` if your variable needs to be accessed by child scripts.
- Modify `.bashrc` or `.bash_profile` for **user-specific permanent values**.
- Modify `/etc/profile` for **system-wide values** (e.g., needed for all users).
- Always logout and login again (or `source ~/.bashrc`) to apply.

Lecture: Taking User Input in Bash Scripts

📌 Objective

Understand how to:

- Make Bash scripts interactive.
- Take input from the user at runtime.
- Securely collect sensitive input like passwords.
- Recognize why interactive scripts are **not preferred in DevOps** automation.

🧠 Why Learn This?

Interactivity is useful for basic scripting and quick user-driven automation. But in DevOps pipelines, **scripts are often run without human intervention**, so knowing both *how* and *when* to use interactive input is key.

🧪 Basic Syntax: `read` Command

bash
CopyEdit
```
read VARIABLE
```

- Pauses the script and waits for user input.
- Whatever user types gets stored in `VARIABLE`.

Example:

bash
CopyEdit
```
read NAME
```

User types “Imran” → Now `NAME="Imran"`

🧑‍💻 Script Demonstration

bash
CopyEdit

```
#!/bin/bash

echo "Enter your skills:"
read SKILL

echo "Your $SKILL skill is in high Demand in the IT Industry."

read -p 'Username: ' USR
read -sp 'Password: ' pass

echo
echo "Login Successful: Welcome USER $USR,"
```

🔍 Explanation Line-by-Line

1. `read SKILL`

- Waits for the user to input a skill (e.g., “CloudComputing”).
- The value is stored in the `SKILL` variable.

2. `read -p 'Username: ' USR`

- `-p`: Displays a prompt and reads the input on the **same line**.
- Stores input into `USR`.

3. `read -sp 'Password: ' pass`

- `-s`: “Silent mode” — does not echo input to the terminal (for passwords or secrets).
- Stores the input into `pass`.

🧪 Live Run Example Output

yaml
CopyEdit

```
[root@scriptbox scripts]# ./7_userInput.sh
Enter your skills:
CloudComputing
Your CloudComputing skill is in high Demand in the IT Industry.
Username: Imran
Password:
Login Successful: Welcome USER Imran,
```

Note: Password is not shown while typing.

💡 Other Read Options

Option	Description
`-p`	Prompt user before reading input
`-s`	Suppress typed characters (secrets)
`-t`	Timeout if user does not input
`-n`	Accept only *n* characters
`-a`	Store input into an array

🚫 Why NOT Recommended in DevOps

Interactive scripts are:

- ❌ **Non-automatable** – CI/CD tools (like Jenkins, Ansible) can't supply input in real-time.
- ❌ **Error-prone** – Relying on humans leads to mistakes or delays.
- ❌ **Not idempotent** – Scripts behave differently based on interaction.

✅ Alternatives for Automation

Pass values as **script arguments**:

bash
CopyEdit
```
./deploy.sh admin mypass123
```

-
- Use **environment variables** or `.env` files.
- Pull secrets from **vaults or secret managers**.

📁 Script Location

bash
CopyEdit
```
cd /opt/scripts/
ls
# => 7_userInput.sh
```

To make it executable:

bash
CopyEdit
```
chmod +x 7_userInput.sh
```

To run:

bash
CopyEdit
```
./7_userInput.sh
```

🧠 Summary

Concept	Notes
`read`	Accepts user input and stores into a variable
`-p`	Prompts user for input on the same line
`-s`	Hides typed input (ideal for passwords)
Exported Variables	Not needed for `read`, but can help with reuse
DevOps Usage	Avoid interactivity; use automation-friendly methods instead

---

🔚 Conclusion

- Interactivity helps when writing personal or learning scripts.
- Use `read`, `-p`, `-s` for quick CLI tools.
- Avoid them for production-grade DevOps tools — always design with automation in mind.

✅ Bash Scripting: Decision Making with `if`, `else`

🎯 Objective

Learn how to make Bash scripts **interactive and intelligent** by enabling them to make **decisions**. Until now, scripts were a **linear list of commands**. With conditional logic, scripts can:

- Check **user input** or **system values**
- Execute commands **only if** a certain condition is met
- Take **different actions** based on values

This is a **core concept** in automation and DevOps scripting.

📘 Theory: Why "Decision Making"?

Without decision making, a script behaves like this:

bash
CopyEdit

```
echo "Starting"
echo "Installing Apache"
echo "Done"
```

It doesn't **check conditions**, so:

- If Apache is already installed → still runs again
- If an error occurs → script doesn't know what to do

With decision making:

You can say:

- *If Apache is not installed, install it*
- *If memory is less than 100MB, alert the user*
- *If the number is greater than 100, do something special*

🧠 Understanding the `if` Statement

🔹 Basic Syntax

bash
CopyEdit

```
if [ CONDITION ]
then
   # commands if condition is true
fi
```

⚠️ Syntax Rules

- Use **spaces inside brackets**:
  - ✅ `[ $NUM -gt 100 ]`
  - ❌ `[$NUM -gt 100]`
- `fi` ends the `if` block.
- You must **evaluate a test expression**.
- Use `read` to get input from the user.

🔢 Bash Comparison Operators

Operator	Description
`-eq`	Equal to
`-ne`	Not equal to

`-gt`	Greater than
`-lt`	Less than
`-ge`	Greater or equal to
`-le`	Less or equal to

These are **only for integers**.

🧪 Full Working Example 1 — Simple `if`

📄 File: `8ifl.sh`

bash
CopyEdit

```
#!/bin/bash

read -p "Enter a number: " NUM
echo

if [ $NUM -gt 100 ]
then
   echo "We have entered in IF block."
   sleep 3
   echo "Your Number is greater than 100"
   echo
   date
fi

echo "Script execution completed successfully."
```

📝 Line-by-Line Explanation

- `#!/bin/bash`: Shebang – tells the OS to run script with Bash shell
- `read -p "Enter a number: " NUM`: Prompt user for a number and store it in `NUM`

- `if [ $NUM -gt 100 ]`: Test if user input is greater than 100
- `then`: Start block that runs if condition is true
- `sleep 3`: Wait 3 seconds (for dramatic effect!)
- `echo "..."`: Output message
- `date`: Print current date/time
- `fi`: End of `if` block
- `echo "Script execution..."`: Always runs, regardless of input

✅ Make Script Executable

bash
CopyEdit

```
chmod +x 8ifl.sh
```

▶️ Run Script

bash
CopyEdit

```
./8ifl.sh
```

Example Output:

Input: 120

csharp
CopyEdit

```
We have entered in IF block.
Your Number is greater than 100

Sat Sep  4 14:37:21 UTC 2021
Script execution completed successfully.
```

Input: 50

nginx
CopyEdit

```
Script execution completed successfully.
```

❗ **Why did the IF block not run?** → Because `50 -gt 100` is FALSE.

🔁 Full Working Example 2 — `if` with `else`

📄 File: `9_if1.sh`

bash
CopyEdit

```
#!/bin/bash

read -p "Enter a number: " NUM
echo

if [ $NUM -gt 100 ]
then
   echo "We have entered in IF block."
   sleep 3
   echo "Your Number is greater than 100"
   echo
   date
else
   echo "You have entered a number less than or equal to 100."
   echo "Try again with a bigger number!"
fi

echo "Script execution completed successfully."
```

📝 New Things Added

- `else`: Runs **only if** `if` condition is false.
- `fi`: Still used to end the whole block.
- Useful to handle **both** possibilities (true/false).

🧪 Run Example:

Input: 60

css
CopyEdit
```
You have entered a number less than or equal to 100.
Try again with a bigger number!
Script execution completed successfully.
```

Input: 150

pgsql
CopyEdit
```
We have entered in IF block.
Your Number is greater than 100

<timestamp>
Script execution completed successfully.
```

🔐 Best Practices

Always **quote variables** if they might contain spaces:

bash
CopyEdit
```
if [ "$NUM" -gt 100 ]
```

-
- Use `read -p` to prompt inline.
- Use `sleep` to simulate processing delays.
- Add input validation to avoid script crashing on invalid input.

⚡ BONUS: Input Validation (for numeric input)

bash
CopyEdit
```
read -p "Enter a number: " NUM
```

```
if ! [[ "$NUM" =~ ^[0-9]+$ ]]
then
  echo "❌ Invalid input! Please enter a number."
  exit 1
fi
```

🗒️ Summary

Concept	Example
`if` condition	`if [ $X -gt 10 ]`
`else` fallback	`else ... fi`
`fi` closing block	Always required
`read` user input	`read -p "Enter:" VAR`
`sleep` demo delay	`sleep 3`
Integer operators	`-eq`, `-gt`, `-lt`, `-ne`, etc.
Input validation (opt)	`[[ "$NUM" =~ ^[0-9]+$ ]]`

🚀 Next Steps

After `if-else`, we can go further with:

- `elif` (else-if chains)
- Nested `if`
- Case statements (good for multi-option input)

Let me know if you'd like those covered too!

🧠 What This Lecture Teaches

We learn how to make scripts **intelligent** by enabling **conditional execution** of code using:

- `if`
- `else`
- `elif` (else if)

This helps scripts take different actions based on different inputs or system states.

🛠️ Problem We're Solving

We want to:

- Detect how many active **network interfaces** (excluding the loopback) exist.
- Based on the count:
    - Print if there's 1 active interface
    - Print if there are multiple active interfaces
    - Print if none are active

🧪 Commands Used and What They Do

bash
CopyEdit
`ip addr show`

Shows all network interfaces and their properties.

bash
CopyEdit
`grep -v LOOPBACK`

Filters out any line that contains the word "LOOPBACK" (used to exclude the loopback interface).

bash
CopyEdit
```
grep -ic mtu
```

Counts (`-c`) how many lines (case-insensitively `-i`) contain the string `mtu` (common in each interface's info).

So:

bash
CopyEdit
```
ip addr show | grep -v LOOPBACK | grep -ic mtu
```

This pipeline gives us the **count of active, non-loopback network interfaces**.

📦 Final Bash Script (`10_ifelif.sh`)

bash
CopyEdit
```
#!/bin/bash

# Get the number of active network interfaces excluding loopback
value=$(ip addr show | grep -v LOOPBACK | grep -ic mtu)

if [ $value -eq 1 ]
then
  echo "1 Active Network Interface found."

elif [ $value -gt 1 ]
then
  echo "Found Multiple active Interface."

else
  echo "No Active interface found."
fi
```

🧵 Script Logic Flow

- `value=$(...)`: runs the pipeline and stores the count in the variable `value`.
- `if [ $value -eq 1 ]`: Checks if there's **1** active interface.
- `elif [ $value -gt 1 ]`: Checks if it's **greater than 1**.
- `else`: If neither, assumes **no active interface**.
- `fi`: Closes the `if` block.

🔐 Notes on Syntax

- Always have **spaces** around brackets: `[ $value -eq 1 ]`
- `fi` is the closing keyword for `if`
- `elif` is used **between** `if` and `else` for **additional checks**

🧪 Sample Output (From Your Image)

bash
CopyEdit

```
Found Multiple active Interface.
```

This indicates the condition `elif [ $value -gt 1 ]` was **true**, so multiple interfaces were detected.

📝 Summary

Concept	Purpose
`if`	Executes block if condition is true
`elif`	Optional extra check if initial `if` is false

`else`	Executes if none of the above are true
`fi`	Ends the `if-elif-else` block
`$(command)`	Command substitution — stores output in var
`grep -v`	Inverts match (i.e., exclude lines)
`grep -ic`	Count matching lines, ignoring case

💻 HTTPD Process Monitoring Script — Full Beginner-Friendly Notes

🎯 Goal:

- Write a Bash script to monitor the `httpd` (Apache web server) process.
- If the process is NOT running → Start the process automatically.
- If it fails to start → Notify the user.
- Schedule this script to run automatically every minute using **cron job**, so your system stays self-healing.

📁 File Structure:

Your script directory (inside your VM) should be:

bash
CopyEdit
`/opt/scripts/`

We will place our monitoring script in this folder.

🛠️ Step 1: Write the Monitoring Script (exit code method)

📄 File Name: `/opt/scripts/11_monit.sh`

🔥 Full Script with Comments:

bash
CopyEdit

```
#!/bin/bash

# Print Header
echo "#####################################################"
```

```
date  # Prints current date and time for tracking

# Check if the PID file exists for httpd process
ls /var/run/httpd/httpd.pid &> /dev/null

# $? holds the exit code of the last command
if [ $? -eq 0 ]
then
   echo "Httpd process is running."
else
   echo "Httpd process is NOT Running."
   echo "Starting the process"

   systemctl start httpd  # Attempt to start the process

   # Check if starting process succeeded
   if [ $? -eq 0 ]
   then
      echo "Process started successfully."
   else
      echo "Process Starting Failed, contact the admin."
   fi
fi

echo "#######################################################"
echo
```

🧩 Detailed Step-by-Step Explanation:

✅ `#!/bin/bash` → Shebang. Tells the system to use Bash interpreter to run this script.

✅ `echo "#######"` → Prints separator lines for readability.

✅ `date` → Prints current system date/time, so logs show when the check ran.

✅ `ls /var/run/httpd/httpd.pid &> /dev/null`

- Checks if the file `/var/run/httpd/httpd.pid` exists.
- This file contains the Process ID of `httpd`. If it exists → Process is running.

- `&> /dev/null` → Redirects both standard output and errors to `/dev/null`, hiding them. We only care about the exit code.

✅ `$?` → Holds exit code of the last command:

- `0` → Success (PID file exists → Process running).
- Non-zero → Failure (PID file missing → Process not running).

✅ If process not running:

- Attempt to start with `systemctl start httpd`.
- Again check `$?` to see if start was successful.

✅ Print messages for clarity.

🛠️ Step 2: Alternate Script using `-f` Operator

Instead of relying on `$?`, we can directly use:

bash
CopyEdit
```
if [ -f /var/run/httpd/httpd.pid ]
```

📄 File: `/opt/scripts/12_monit.sh`

bash
CopyEdit
```
#!/bin/bash

echo "#######################################################"
date

# Directly check if file exists
if [ -f /var/run/httpd/httpd.pid ]
then
```

```
   echo "Httpd process is running."
else
   echo "Httpd process is NOT Running."
   echo "Starting the process"

   systemctl start httpd

   if [ $? -eq 0 ]
   then
      echo "Process started successfully."
   else
      echo "Process Starting Failed, contact the admin."
   fi
fi

echo "######################################################"
echo
```

✅ `-f` checks if a file exists and is a regular file.
✅ Logic is same, just another method.

🕑 Step 3: Scheduling with Cron Job

We want this script to run **every minute**, automatically.

Open Crontab:

bash
CopyEdit
```
crontab -e
```

Add this line at the bottom:

bash
CopyEdit
```
* * * * * /opt/scripts/11_monit.sh &>> /var/log/monit_httpd.log
```

🧩 Crontab Timing Breakdown:

text
CopyEdit

```
* * * * * COMMAND
| | | | |
| | | | +---- Day of week (0-6, Sunday=0)
| | | +------ Month (1-12)
| | +-------- Day of month (1-31)
| +---------- Hour (0-23)
+------------ Minute (0-59)
```

Our Example:

bash
CopyEdit

```
* * * * * /opt/scripts/11_monit.sh &>> /var/log/monit_httpd.log
```

✅ Runs every minute, every hour, every day, every month, every day-of-week.
✅ `&>> /var/log/monit_httpd.log` → Appends both standard output and errors to the log file.

📂 Step 4: Verify the Setup

✅ Stop `httpd` manually:

bash
CopyEdit

```
systemctl stop httpd
```

✅ Wait a minute…
✅ Check Logs:

bash
CopyEdit

```
cat /var/log/monit_httpd.log
```

You should see messages like:

text
CopyEdit

```
#####################################################
```

```
Sat Jun 22 11:45:01 IST 2025
Httpd process is NOT Running.
Starting the process
Process started successfully.
##################################################
```

✅ It keeps running every minute to ensure `httpd` stays up.

💡 Extra: Cron Job Example at Specific Time

Run script at **8:30 PM, Monday to Friday only**:

bash
CopyEdit

```
30 20 * * 1-5 /opt/scripts/11_monit.sh &>> /var/log/monit_httpd.log
```

Breakdown:

text
CopyEdit

```
30 20 * * 1-5 COMMAND
|  |  | |  +--- Day of week: 1-5 (Mon to Fri)
|  |  | +------ Month: Every month
|  |  +-------- Day of month: Every day
|  +----------- Hour: 20 (8 PM)
+-------------- Minute: 30
```

🎉 Summary:

✅ Two monitoring scripts shown: one with `$?`, another with `-f`.
✅ Cron job explained with timing breakdown.
✅ All steps beginner-friendly with clear logic.
✅ Logs stored for troubleshooting.

🔁 Bash Loops (For Beginners - Full Explanation)

🎯 Goal of Loops in Bash

Loops help you run the same set of commands multiple times without manually typing them again and again. Ideal when:

- Repeating tasks (e.g., adding users, processing files, running commands on servers)
- Automating sequences of repetitive actions

Two popular loops in Bash:
✅ `for` loop → Runs for a definite, known number of times
✅ `while` loop → Runs until a condition is met (infinite or unknown length)

In this lesson, we focus on **for loops**.

🛠️1️⃣Basic For Loop Syntax

bash
CopyEdit

```
for VARIABLE in ITEM1 ITEM2 ITEM3 ...
do
   # Commands to execute
done
```

Logic:

- Takes each ITEM one by one from the list
- Stores ITEM value in `VARIABLE`
- Runs commands inside `do ... done` block for each ITEM

🔥 Example 1: Simple Programming Language Loop

📄 File: `13_for.sh`

bash
CopyEdit

```
#!/bin/bash

for VAR1 in java .net python ruby php
do
  echo "Looping....."
  sleep 1  # Slows down the loop, for visibility
  echo "#############################################"
  echo "Value of VAR1 is $VAR1."
  echo "#############################################"
  date  # Prints current date & time
  echo
done
```

🧩 Step-by-Step Explanation

✅ `#!/bin/bash` → Shebang line
✅ `for VAR1 in java .net python ruby php` → Loop iterates over 5 items
✅ `do` → Start of commands block
✅ Inside block:

- Print "Looping..."
- Sleep for 1 second
- Print current item value
- Print date
  ✅ `done` → End of loop

Loop flow:

text
CopyEdit

```
1. VAR1 = java → Run commands
2. VAR1 = .net → Run commands
3. VAR1 = python → Run commands
4. VAR1 = ruby → Run commands
5. VAR1 = php → Run commands
```

📊 Sample Output:

bash
CopyEdit

```
Looping.....
################################################
Value of VAR1 is java.
################################################
Mon Jun 24 15:00:01 IST 2025

Looping.....
################################################
Value of VAR1 is .net.
################################################
Mon Jun 24 15:00:02 IST 2025
...
```

🛠️2️⃣ Real-World Example: Adding Multiple Users

📄 File: `14_for.sh`

bash
CopyEdit

```
#!/bin/bash

MYUSERS="alpha beta gamma"  # List of users

for usr in $MYUSERS
do
```

```
    echo "Adding user $usr."
    useradd $usr    # Add the user
    id $usr         # Show user details (UID, GID)
    echo "#####################################"
done
```

🧩 Explanation

✅ `MYUSERS="alpha beta gamma"` → 3 usernames to process
✅ `for usr in $MYUSERS` → Loop over each username
✅ Commands:

- `echo` prints progress
- `useradd $usr` adds the user
- `id $usr` shows user's UID, GID, groups
- Separator for readability

Total Loop Runs:

text
CopyEdit

```
1. usr = alpha → Add alpha
2. usr = beta → Add beta
3. usr = gamma → Add gamma
```

📊 Output Example:

bash
CopyEdit

```
Adding user alpha.
uid=1001(alpha) gid=1001(alpha) groups=1001(alpha)
#####################################

Adding user beta.
uid=1002(beta) gid=1002(beta) groups=1002(beta)
```

```
####################################

Adding user gamma.
uid=1003(gamma) gid=1003(gamma) groups=1003(gamma)
####################################
```

⚡ Common Notes & Tips

✅ You must run the script with root privileges for `useradd` to work
✅ Always use `echo` separators for better readability
✅ Loop variable can be any name (commonly `i`, `usr`, etc.)
✅ You can use `{1..10}` syntax for numeric loops (more advanced)

🔧 Making Script Executable

bash
CopyEdit

```
chmod +x 14_for.sh
./14_for.sh
```

🚀 Final Takeaways

Concept	Purpose
`for` loop	Run commands multiple times over a list
`$VARIABLE`	Stores current item in the loop
`useradd`	Adds system users
`id`	Shows user details
`sleep 1`	Adds delay for clear output

🔄 Bash `while` Loop — Beginner's Full Explanation

🎯 What is a While Loop?

- Runs repeatedly **as long as the condition is TRUE**
- Checks the condition **before every iteration**
- Commonly used for:
  ✅ Monitoring
  ✅ Waiting for tasks to finish
  ✅ Infinite loops for services
  ✅ Repeated calculations

🔧 Basic Syntax

bash
CopyEdit

```
while [ CONDITION ]
do
   # Commands to run repeatedly
done
```

- `[ CONDITION ]` → Evaluates TRUE or FALSE
- If TRUE → Runs commands inside the block
- Keeps looping until condition becomes FALSE

🛠️1️⃣ Practical Example: Count from 0 to 4

📄 Script: `15_while.sh`

bash
CopyEdit

```
#!/bin/bash

counter=0  # Initialize counter to 0

while [ $counter -lt 5 ]  # Condition: counter < 5
do
  echo "Looping...."
  echo "Value of counter is $counter."

  counter=$(( $counter + 1 ))  # Increment counter by 1

  sleep 1  # Pause for 1 second for clarity
done

echo "Out of the loop"
```

🧩 Detailed Step-by-Step Logic

✅ `counter=0` → Start at 0
✅ `while [ $counter -lt 5 ]` → Loop runs if counter < 5
✅ Inside the loop:

- Print messages
- Increment counter with `counter=$(( $counter + 1 ))`
- Sleep 1 second to slow down output
  ✅ When counter = 5, condition becomes FALSE, exits loop
  ✅ Prints `"Out of the loop"`

📊 Sample Output

bash
CopyEdit

```
Looping....
Value of counter is 0.
Looping....
Value of counter is 1.
Looping....
Value of counter is 2.
Looping....
Value of counter is 3.
Looping....
Value of counter is 4.
Out of the loop
```

⚠️ Common Mistake: Forgetting to Change the Condition

If you forget to increment the counter:

bash
CopyEdit

```
counter=0
while [ $counter -lt 5 ]
do
  echo "Value of counter is $counter."
done
```

This causes an **infinite loop** because `$counter` stays 0, the condition is always TRUE.

Use `Ctrl + C` to manually stop the loop in terminal.

🛠️2️⃣ Example: Infinite Loop with `true`

📄 Script: `16_while.sh`

bash
CopyEdit

```
#!/bin/bash
```

```
counter=2

while true  # Infinite loop, true is always TRUE
do
  echo "Looping...."
  echo "Value of counter is $counter."

  counter=$(( $counter * 2 ))  # Multiply by 2 each time

  sleep 1
done

echo "Out of the loop"  # This never runs
```

🧩 How it Works

✅ `counter=2`
✅ `while true` → Runs forever, no condition to break the loop
✅ Multiplies counter by 2 repeatedly
✅ You must use `Ctrl + C` to stop the loop manually

📊 Sample Output

bash
CopyEdit

```
Looping....
Value of counter is 2.
Looping....
Value of counter is 4.
Looping....
Value of counter is 8.
Looping....
Value of counter is 16.
Looping....
Value of counter is 32.
...
```

If you remove `sleep 1`, loop runs super fast, consuming CPU resources.

⚡ Best Practices for `while` Loops

Practice	Why Important
Update your condition	Avoid infinite loops accidentally
Use `sleep` for visibility	Prevent fast, uncontrollable output
Use `Ctrl + C` to exit infinite loops	Manual termination for testing
Consider infinite loops for monitoring	Like scripts that check service health

---

🔧 Common Conditional Operators in While Loops

Operator	Meaning
`-lt`	Less than
`-le`	Less than or equal to
`-gt`	Greater than
`-ge`	Greater than or equal to
`-eq`	Equal to
`-ne`	Not equal to
`-f file`	True if file exists
`true`	Boolean true (always runs)

🔥 Quick Comparison: For vs While

`for` loop	`while` loop
Runs over a known sequence	Runs based on a condition
Good for fixed tasks	Good for unknown duration tasks
Example: For each file, do this	Example: Keep running until success

---

🧩 Summary

✅ Use `while` loop to run commands repeatedly based on a condition
✅ Be careful of infinite loops
✅ Use `sleep` for controlled output
✅ Can create monitoring or background tasks

🔥 Full Notes: Remote Command Execution with Bash, SSH & SCP (Multi-OS Setup) 🔥

✅ Goal of this Lecture

- Remotely execute scripts/commands from **scriptbox** to:
    - `web01` (CentOS)
    - `web02` (CentOS)
    - `web03` (Ubuntu)
- Use **`devops`** user for execution (not `vagrant`)
- Deploy a website template automatically
- Support **multi-OS logic** in the script

🏗️ Step 1: Vagrant Multi-VM Setup

Vagrantfile Example (Final Clean Version):

ruby
CopyEdit

```
Vagrant.configure("2") do |config|

  config.vm.define "scriptbox" do |scriptbox|
    scriptbox.vm.box = "geerlingguy/centos7"
    scriptbox.vm.network "private_network", ip: "192.168.10.12"
    scriptbox.vm.provider "virtualbox" do |vb|
      vb.memory = "1024"
    end
  end

  config.vm.define "web01" do |web01|
    web01.vm.box = "geerlingguy/centos7"
    web01.vm.network "private_network", ip: "192.168.10.13"
  end
```

```
  config.vm.define "web02" do |web02|
    web02.vm.box = "geerlingguy/centos7"
    web02.vm.network "private_network", ip: "192.168.10.14"
  end

  config.vm.define "web03" do |web03|
    web03.vm.box = "ubuntu/bionic64"
    web03.vm.network "private_network", ip: "192.168.10.15"
  end

end
```

IP Summary:

VM Name	IP	OS
scriptbox	192.168.10.12	CentOS 7
web01	192.168.10.13	CentOS 7
web02	192.168.10.14	CentOS 7
web03	192.168.10.15	Ubuntu 18.04

🛠️ Step 2: Update /etc/hosts on scriptbox

Inside scriptbox VM, edit `/etc/hosts`:

bash
CopyEdit
```
sudo vi /etc/hosts
```

Add:

CopyEdit
```
192.168.10.13   web01
192.168.10.14   web02
192.168.10.15   web03
```

Test:

bash
CopyEdit
```
ping web01
ping web02
ping web03
```

✅ Should resolve names to correct IPs.

👤 Step 3: Prepare Remote Machines

Add user `devops` on all VMs:

On web01 & web02 (CentOS):

bash
CopyEdit
```
sudo useradd devops
sudo passwd devops
```

Grant sudo without password:

bash
CopyEdit
```
sudo visudo
```

Add:

bash
CopyEdit
```
devops ALL=(ALL) NOPASSWD: ALL
```

Ubuntu Extra Step (web03):

1. Can't SSH with password by default.

2. Enable password login:

bash
CopyEdit
```
sudo vi /etc/ssh/sshd_config
```

Find:

nginx
CopyEdit
```
PasswordAuthentication no
```

Change to:

nginx
CopyEdit
```
PasswordAuthentication yes
```

Restart SSH:

bash
CopyEdit
```
sudo systemctl restart ssh
```

3. Add user devops:

bash
CopyEdit
```
sudo adduser devops
sudo visudo
```

Add same sudoers line:

bash
CopyEdit
```
devops ALL=(ALL) NOPASSWD: ALL
```

🚀 Step 4: Multi-OS Website Setup Script

Script: `multios_websetup.sh`

bash
CopyEdit

```
#!/bin/bash

URL='https://www.tooplate.com/zip-templates/2098_health.zip'
ART_NAME='2098_health'
TEMPDIR="/tmp/webfiles"

# Check if CentOS (yum exists)
yum --help &> /dev/null

if [ $? -eq 0 ]; then
    PACKAGE="httpd wget unzip"
    SVC="httpd"

    echo "Running Setup on CentOS"
    sudo yum install $PACKAGE -y > /dev/null

else
    PACKAGE="apache2 wget unzip"
    SVC="apache2"

    echo "Running Setup on Ubuntu"
    sudo apt update
    sudo apt install $PACKAGE -y > /dev/null
fi

# Start & Enable Service
echo "########################################"
echo "Starting Web Service: $SVC"
echo "########################################"
sudo systemctl start $SVC
sudo systemctl enable $SVC

# Deploy Artifact
mkdir -p $TEMPDIR
cd $TEMPDIR
wget $URL > /dev/null
unzip $ART_NAME.zip > /dev/null
sudo cp -r $ART_NAME/* /var/www/html/
```

```
# Restart Web Service
sudo systemctl restart $SVC

# Cleanup
rm -rf $TEMPDIR

# Verification
sudo systemctl status $SVC
ls /var/www/html/
```

Make it executable:

bash
CopyEdit
```
chmod +x multios_websetup.sh
```

🌐 Step 5: Remote Execution Script

File `remhosts`

nginx
CopyEdit
```
web01
web02
web03
```

Script: `remote_exec.sh`

bash
CopyEdit
```
#!/bin/bash

USR='devops'

for host in `cat remhosts`
do
   echo
   echo "############################################################"
   echo "Connecting to $host"
```

```
    echo "Pushing Script to $host"
    scp multios_websetup.sh $USR@$host:/tmp/

    echo "Executing Script on $host"
    ssh $USR@$host sudo /tmp/multios_websetup.sh

    echo "Cleaning up"
    ssh $USR@$host sudo rm -rf /tmp/multios_websetup.sh

    echo "#############################################################"
    echo
done
```

Make executable:

bash
CopyEdit
`chmod +x remote_exec.sh`

Run the Automation:

bash
CopyEdit
`./remote_exec.sh`

🎯 Concept Summary

✅ `scp` → Securely copy files to remote hosts
✅ `ssh user@host cmd` → Run command remotely without login shell
✅ `sudo` with `NOPASSWD` avoids password prompts during automation
✅ Script auto-detects OS:

- CentOS → Installs `httpd`
- Ubuntu → Installs `apache2`
  ✅ Website template deployed to `/var/www/html/`
  ✅ Full automation for multi-server, multi-OS environments

⚠️ Common Errors

Issue	Fix
SSH permission denied	Enable `PasswordAuthentication yes`
`sudo` password prompt	Configure NOPASSWD in sudoers file
Script fails on one OS type	Validate `yum` or `apt` logic
IP/hostname mismatch	Correct `/etc/hosts` on scriptbox

📌 Optional Improvements

- Use SSH keys for passwordless auth
- Add error handling to scripts
- Use `rsync` instead of `scp` for better performance
- Use arrays for hosts instead of `cat remhosts`

🛠️ Remote Execution & SSH Key-Based Login — Full Lecture Notes

Goal:

- Execute commands remotely from `scriptbox` to `web01`, `web02`, and `web03`
- Use both password-based and key-based authentication
- Automate secure, passwordless SSH with `ssh-keygen` and `ssh-copy-id`

🖥️ Machines Setup (Vagrantfile Excerpt):

ruby
CopyEdit

```
config.vm.define "scriptbox" do |scriptbox|
  scriptbox.vm.box = "geerlingguy/centos7"
  scriptbox.vm.network "private_network", ip: "192.168.10.12"
  scriptbox.vm.provider "virtualbox" do |vb|
    vb.memory = "1024"
  end
end

config.vm.define "web01" do |web01|
  web01.vm.box = "geerlingguy/centos7"
  web01.vm.network "private_network", ip: "192.168.10.13"
end

config.vm.define "web02" do |web02|
  web02.vm.box = "geerlingguy/centos7"
  web02.vm.network "private_network", ip: "192.168.10.14"
end

config.vm.define "web03" do |web03|
  web03.vm.box = "ubuntu/bionic64"
  web03.vm.network "private_network", ip: "192.168.10.15"
end
```

🖥️ Hostname Setup (Run on Each Machine):

bash
CopyEdit

```
sudo vi /etc/hostname
# Set hostname as web01, web02, web03 respectively
```

Check IP Addresses:

bash
CopyEdit

```
ip a
```

Update `/etc/hosts` in `scriptbox` for name resolution:

bash
CopyEdit

```
sudo vi /etc/hosts

# Add:
192.168.10.13 web01
192.168.10.14 web02
192.168.10.15 web03
```

Test:

bash
CopyEdit

```
ping web01
ping web02
ping web03
```

🛡️ Add DevOps User on All Machines:

bash
CopyEdit

```
sudo adduser devops
sudo passwd devops
```

Add `devops` to sudoers (use `visudo`):

bash
CopyEdit

```
export EDITOR=vim
sudo visudo

# Add at the end:
devops ALL=(ALL) NOPASSWD: ALL
```

⚠️ On Ubuntu (`web03`), enable password login:

bash
CopyEdit

```
sudo vi /etc/ssh/sshd_config

# Change:
PasswordAuthentication yes

sudo systemctl restart ssh
```

🛠️ SSH Remote Execution Basics:

Password-Based Example:

bash
CopyEdit

```
ssh devops@web01 uptime
```

It asks:

bash
CopyEdit

```
The authenticity of host 'web01' can't be established...
Are you sure you want to continue connecting (yes/no)? yes
devops@web01's password: vagrant
```

Then outputs uptime and exits.

🔑 SSH Key-Based Authentication (Safer Method):

Generate Key on `scriptbox`:

bash
CopyEdit

```
ssh-keygen
# Hit enter 3 times to accept defaults
```

Output:

swift
CopyEdit

```
Your identification has been saved in /root/.ssh/id_rsa
Your public key has been saved in /root/.ssh/id_rsa.pub
```

Key Analogy:

- `id_rsa.pub` → Public "Lock" to apply on servers
- `id_rsa` → Private "Key" kept safe on scriptbox

Push Public Key to Servers:

bash
CopyEdit

```
ssh-copy-id devops@web01
ssh-copy-id devops@web02
ssh-copy-id devops@web03
```

Enter password when prompted.

🧪 Test Passwordless Login:

bash

CopyEdit

```
ssh devops@web01 uptime
```

No password asked! It uses:

bash
CopyEdit

```
ssh -i ~/.ssh/id_rsa devops@web01 uptime
```

🔍 Verify Key Files:

bash
CopyEdit

```
ls ~/.ssh

cat ~/.ssh/id_rsa.pub  # Public key (short)
cat ~/.ssh/id_rsa      # Private key (long, keep secure)
```

If `.ssh/id_rsa` contains:

vbnet
CopyEdit

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAwsl4w9...
...
-----END RSA PRIVATE KEY-----
```

And `.ssh/id_rsa.pub` looks like:

CopyEdit

```
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAwsl4w...
```

The public key is applied to `~devops/.ssh/authorized_keys` on servers.

🛠️ Remote Automation Script (`push_script.sh`):

bash
CopyEdit

```
#!/bin/bash

USR='devops'

for host in `cat remhosts`
do
   echo
   echo "############################################################"
   echo "Connecting to $host"
   echo "Pushing Script to $host"
   scp multios_websetup.sh $USR@$host:/tmp/
   echo "Executing Script on $host"
   ssh $USR@$host sudo /tmp/multios_websetup.sh
   ssh $USR@$host sudo rm -rf /tmp/multios_websetup.sh
   echo "############################################################"
   echo
done
```

`remhosts` file:

nginx
CopyEdit

```
web01
web02
web03
```

🌐 What Happens:

- Sends `multios_websetup.sh` to `/tmp/` of each server
- Executes setup remotely with `sudo`

- Cleans up the script
- No password required due to SSH key setup

📦 `multios_websetup.sh` Supports Multi-OS:

bash
CopyEdit

```
#!/bin/bash

URL='https://www.tooplate.com/zip-templates/2098_health.zip'
ART_NAME='2098_health'
TEMPDIR="/tmp/webfiles"

yum --help &> /dev/null

if [ $? -eq 0 ]; then
    PACKAGE="httpd wget unzip"
    SVC="httpd"
    sudo yum install $PACKAGE -y
else
    PACKAGE="apache2 wget unzip"
    SVC="apache2"
    sudo apt update
    sudo apt install $PACKAGE -y
fi

sudo systemctl start $SVC
sudo systemctl enable $SVC

mkdir -p $TEMPDIR
cd $TEMPDIR
wget $URL
unzip ${ART_NAME}.zip
sudo cp -r $ART_NAME/* /var/www/html/

sudo systemctl restart $SVC
```

```
sudo systemctl status $SVC
ls /var/www/html/
```

🎯 Final Checks:

- Websites running on `http://192.168.10.13`, `.14`, `.15`
- Apache (httpd) on CentOS, apache2 on Ubuntu
- No manual password entry for SSH due to key-pair setup

⚠️ Notes:

- Keep private key secure (`id_rsa`)
- Public key can be shared to servers
- Remove password login on production for higher security
- Always test key login before disabling passwords

🛠️ Grand Finale Bash Scripting — Remote Multi-OS Web Setup (Full Notes, No Compromise)

This Grand Finale combines everything we've learned so far — variables, conditions, remote execution, and multi-OS scripting. It's inspired by real-world automation tools like Ansible. Here's the entire breakdown:

🎯 Goal

✅ Automate Web Server setup (HTTPD/Apache) on multiple remote servers
✅ Handle both CentOS and Ubuntu systems using the same script
✅ Use SSH key-based login for passwordless remote execution
✅ Deploy website template automatically

1️⃣ SSH Key-Based Login Setup

🗝️ Generate SSH Key Pair

bash
CopyEdit
`ssh-keygen`

Outputs:

- Private Key saved to `/root/.ssh/id_rsa`
- Public Key saved to `/root/.ssh/id_rsa.pub`

🔑 Analogy:

- Public Key = Lock
- Private Key = Key to open the lock
- Public key goes on remote servers (web01, web02, web03)

- Private key stays on scriptbox

🛠️ Copy Public Key to Remote Servers

bash
CopyEdit

```
ssh-copy-id devops@web01
ssh-copy-id devops@web02
ssh-copy-id devops@web03
```

Now:

- SSH login happens via keys
- No password prompt during SSH

Test:

bash
CopyEdit

```
ssh devops@web01 uptime
```

2️⃣ Prepare Remote Hosts List

bash
CopyEdit

```
mkdir /opt/scripts/remote_websetup
cd /opt/scripts/remote_websetup
vim remhosts
```

Example `remhosts` file:

nginx
CopyEdit

```
web01
web02
web03
```

3️⃣ Test Remote Loop Execution

Print Hostnames

bash
CopyEdit

```
for host in `cat remhosts`; do echo $host; done
```

Run Remote Commands

bash
CopyEdit

```
for host in `cat remhosts`; do ssh devops@$host hostname; done
```

Or check uptime:

bash
CopyEdit

```
for host in `cat remhosts`; do ssh devops@$host uptime; done
```

4️⃣ Multi-OS Setup Script (CentOS + Ubuntu)

Start by copying existing script:

bash
CopyEdit

```
cp /opt/scripts/3_vars_websetup.sh
/opt/scripts/remote_websetup/multios.sh
```

Edit `multios.sh` for multi-OS:

bash
CopyEdit

```
#!/bin/bash

# Website Artifact Info
URL="https://www.tooplate.com/zip-templates/2098_health.zip"
ART_NAME="2098_health"
TEMPDIR="/tmp/webfiles"

# Detect OS Type using yum
```

```
yum --help &> /dev/null

if [ $? -eq 0 ]
then
    echo "   Running Setup on CentOS"

    PACKAGE="httpd wget unzip"
    SVC="httpd"

    echo "########################################"
    echo "Installing Packages"
    echo "########################################"
    sudo yum install $PACKAGE -y > /dev/null
    echo

    echo "########################################"
    echo "Starting & Enabling $SVC Service"
    echo "########################################"
    sudo systemctl start $SVC
    sudo systemctl enable $SVC
    echo

    echo "########################################"
    echo "Starting Artifact Deployment"
    echo "########################################"
    mkdir -p $TEMPDIR
    cd $TEMPDIR
    wget $URL > /dev/null
    unzip $ART_NAME.zip > /dev/null
    sudo cp -r $ART_NAME/* /var/www/html/
    echo

    echo "########################################"
    echo "Restarting $SVC"
    echo "########################################"
    sudo systemctl restart $SVC
    echo

    echo "########################################"
    echo "Removing Temporary Files"
    echo "########################################"
```

```
    rm -rf $TEMPDIR
    echo

    sudo systemctl status $SVC
    ls /var/www/html/

else
    echo "   Running Setup on Ubuntu"

    PACKAGE="apache2 wget unzip"
    SVC="apache2"

    echo "########################################"
    echo "Installing Packages"
    echo "########################################"
    sudo apt update
    sudo apt install $PACKAGE -y > /dev/null
    echo

    echo "########################################"
    echo "Starting & Enabling $SVC Service"
    echo "########################################"
    sudo systemctl start $SVC
    sudo systemctl enable $SVC
    echo

    echo "########################################"
    echo "Starting Artifact Deployment"
    echo "########################################"
    mkdir -p $TEMPDIR
    cd $TEMPDIR
    wget $URL > /dev/null
    unzip $ART_NAME.zip > /dev/null
    sudo cp -r $ART_NAME/* /var/www/html/
    echo

    echo "########################################"
    echo "Restarting $SVC"
    echo "########################################"
    sudo systemctl restart $SVC
    echo
```

```
    echo "########################################"
    echo "Removing Temporary Files"
    echo "########################################"
    rm -rf $TEMPDIR
    echo

    sudo systemctl status $SVC
    ls /var/www/html/
fi
```

5️⃣ Test Locally First

bash
CopyEdit

```
bash multios.sh
```

Outputs:

- Detects OS (CentOS or Ubuntu)
- Installs correct packages
- Deploys Website
- Restarts Service
- Cleans Temp Files

6️⃣ Remote Execution on All Hosts

bash
CopyEdit

```
for host in `cat remhosts`; do scp multios.sh devops@$host:/tmp/; done
for host in `cat remhosts`; do ssh devops@$host "bash /tmp/multios.sh"; done
```

7️⃣ OS Detection Logic Explanation

We check:

bash
CopyEdit
`yum --help &> /dev/null`

If `yum` exists → CentOS
Else → Ubuntu

We use exit status `$?`:

- `0` → Command Success
- Non-zero → Command Failed

8️⃣ Why `/dev/null`?

We don't want command output cluttering the screen. We're only interested in success/failure using `$?`.

9️⃣ Common Issues

✔️ Yum not found on Ubuntu — handled via conditional logic
✔️ SSH key-based login must be set up properly
✔️ Hostnames/IPs must resolve correctly in `/etc/hosts`

🏁 End Result

✅ Automated, OS-aware Web Setup across multiple servers
✅ Single, reusable, portable script
✅ Foundation for real-world tools like Ansible

🔟 Extras: Commands Recap

- Stop HTTPD

bash
CopyEdit
```
systemctl stop httpd
```

- Remove User

bash
CopyEdit
```
userdel username
```

- Check if process running (old method)

bash
CopyEdit
```
ls /var/run/httpd/httpd.pid &> /dev/null
if [ $? -eq 0 ]; then echo "Running"; else echo "Not Running"; fi
```

🚀 GRAND FINALE: Full Remote Web Deployment Automation with Bash

This builds on everything from the previous sessions — remote execution, OS detection, automation, variables, and loops. You're basically simulating what tools like Ansible do under the hood.

🎯 Goal Summary

✔️ Push your web setup script to multiple remote machines (CentOS + Ubuntu mix)
✔️ Execute the script remotely to install HTTPD/Apache, deploy website
✔️ Clean up temp files after execution
✔️ Achieve this using plain Bash, SSH, and SCP

1️⃣ SCP (Secure Copy) Basics

SCP works over SSH to copy files securely between systems. You need working SSH (preferably key-based login).

📂 Push File to Remote `/tmp` Directory

bash
CopyEdit

```
scp testfile.txt devops@web01:/tmp/
```

✅ Works — `/tmp` is writable by normal users

❌ Push to `/root` Fails for Non-Root

bash
CopyEdit

```
scp testfile.txt devops@web01:/root/
```

Output:

bash
CopyEdit

```
scp: /root/testfile.txt: Permission denied
```

Reason: `/root` is owned by root, normal `devops` user has no permission

---

✅ Bypass with SSH Key as Root

bash
CopyEdit

```
scp -i ~/.ssh/id_rsa testfile.txt devops@web01:/root/
```

Still fails unless user has root rights or you switch to actual `root` user on target

---

✅ Safe Path for Normal Users

bash
CopyEdit

```
scp -i ~/.ssh/id_rsa testfile.txt devops@web01:/home/devops/
```

Success — Files land in `/home/devops/`

2️⃣ Write Full Automation Script

Script Name:

bash
CopyEdit

```
webdeploy.sh
```

Full Script with All Details:

bash
CopyEdit

```
#!/bin/bash

USR="devops"

for host in `cat remhosts`
```

```
do
    echo
    echo "################################################"
    echo "Connecting to $host"
    echo "Pushing Script to $host"
    scp multios_websetup.sh $USR@$host:/tmp/

    echo "Executing Script on $host"
    ssh $USR@$host sudo /tmp/multios_websetup.sh

    echo "Cleaning Up Script on $host"
    ssh $USR@$host sudo rm -rf /tmp/multios_websetup.sh
    echo "################################################"
    echo
done
```

Notes:
✔️ `$USR` variable holds username (devops)
✔️ Push script to `/tmp/` for safety — writable by normal users
✔️ Use `sudo` to run script since setup needs root permissions
✔️ After setup, delete script from `/tmp/` to avoid clutter

3️⃣ Execute Automation

Make your script executable:

bash
CopyEdit
```
chmod +x webdeploy.sh
```

Run the automation:

bash
CopyEdit
```
./webdeploy.sh
```

4️⃣ Behind the Scenes Breakdown

✅ `scp` sends your `multios_websetup.sh` to each remote host `/tmp/`
✅ `ssh` runs the script remotely using `sudo`
✅ Script detects OS (`yum --help`)
✅ Installs correct packages:

- CentOS → `httpd`
- Ubuntu → `apache2`
  ✅ Downloads website zip, extracts, copies to `/var/www/html`
  ✅ Restarts service (`systemctl restart`)
  ✅ Final cleanup removes temp files

5️⃣ Accessing the Deployed Website

Browser examples (based on IP):

cpp
CopyEdit

```
http://192.168.10.13  --> web01
http://192.168.10.14  --> web02
http://192.168.10.15  --> web03
```

Check `/etc/hosts` on your system if hostnames don't resolve:

bash
CopyEdit

```
192.168.10.13 web01
192.168.10.14 web02
192.168.10.15 web03
```

6️⃣ Key Learnings & Real-World Connection

✔️ Remote automation with loops
✔️ Conditional logic for multi-OS support
✔️ Secure file copy with SCP
✔️ Remote script execution
✔️ Cleanup for professional finish
✔️ Foundation for tools like Ansible

7️⃣ Troubleshooting

🔧 SSH Fails?

- Check key-based login with `ssh devops@web01`
- Run `ssh-copy-id devops@web01` to install public key

🔧 Permission Denied for File Copy?

- Non-root users can't write to `/root/`
- Use `/tmp/` or `/home/devops/` instead

🔧 Webpage Doesn't Load?

- Check firewall:

bash
CopyEdit

```
sudo firewall-cmd --permanent --add-service=http
sudo firewall-cmd --reload
```

8️⃣ Final Words

If you completed this project:
✅ You automated mixed-OS web server deployment
✅ You simulated how tools like Ansible/Puppet operate
✅ You mastered variables, conditions, remote execution, and scripting logic

Congratulations — you're now ready to confidently approach any real-world DevOps automation tool.