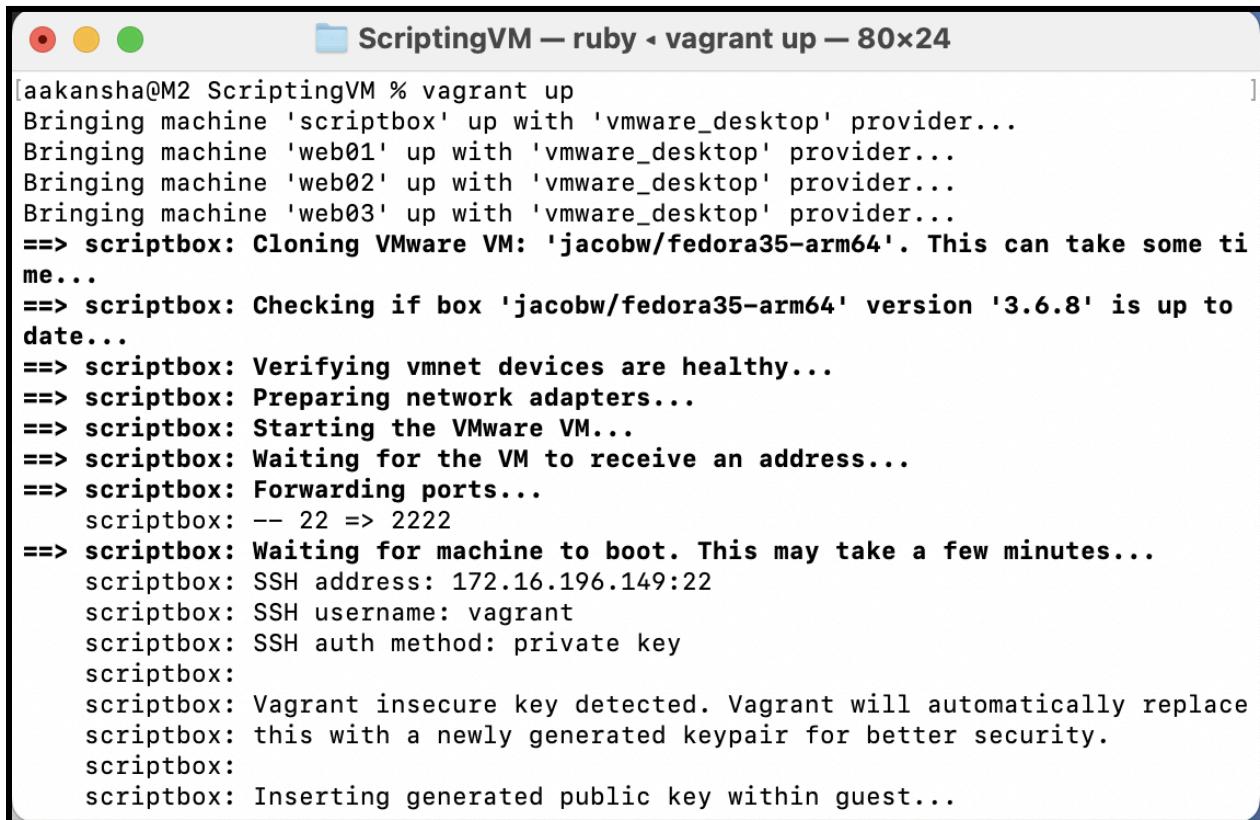


# BASH SCRIPTING

## # Bring up the VMs

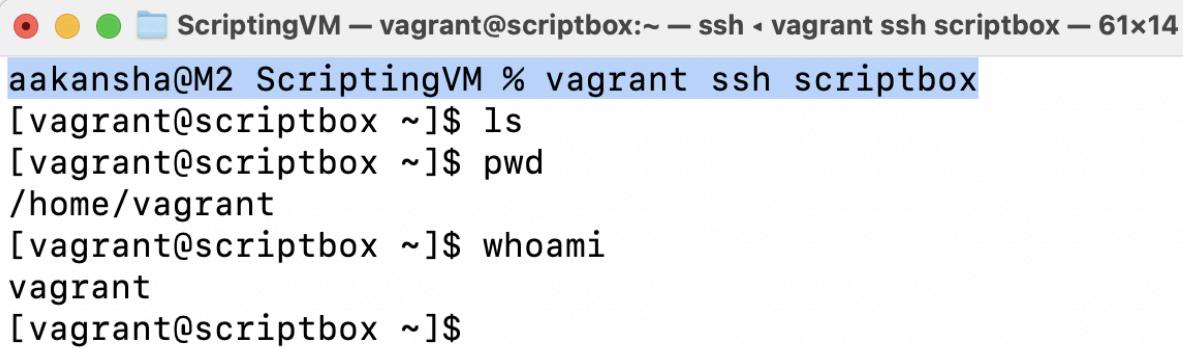
```
aakansha@M2 ScriptingVM % vagrant up
```



```
[aakansha@M2 ScriptingVM % vagrant up
Bringing machine 'scriptbox' up with 'vmware_desktop' provider...
Bringing machine 'web01' up with 'vmware_desktop' provider...
Bringing machine 'web02' up with 'vmware_desktop' provider...
Bringing machine 'web03' up with 'vmware_desktop' provider...
==> scriptbox: Cloning VMware VM: 'jacobw/fedora35-arm64'. This can take some time...
==> scriptbox: Checking if box 'jacobw/fedora35-arm64' version '3.6.8' is up to date...
==> scriptbox: Verifying vmnet devices are healthy...
==> scriptbox: Preparing network adapters...
==> scriptbox: Starting the VMware VM...
==> scriptbox: Waiting for the VM to receive an address...
==> scriptbox: Forwarding ports...
  scriptbox: -- 22 => 2222
==> scriptbox: Waiting for machine to boot. This may take a few minutes...
  scriptbox: SSH address: 172.16.196.149:22
  scriptbox: SSH username: vagrant
  scriptbox: SSH auth method: private key
  scriptbox:
  scriptbox: Vagrant insecure key detected. Vagrant will automatically replace
  scriptbox: this with a newly generated keypair for better security.
  scriptbox:
  scriptbox: Inserting generated public key within guest...
```

## #Enter into scriptbox vm

```
aakansha@M2 ScriptingVM % vagrant ssh scriptbox
```



```
aakansha@M2 ScriptingVM % vagrant ssh scriptbox
[vagrant@scriptbox ~]$ ls
[vagrant@scriptbox ~]$ pwd
/home/vagrant
[vagrant@scriptbox ~]$ whoami
vagrant
[vagrant@scriptbox ~]$
```

## #Become root

```
[vagrant@scriptbox ~]$ sudo -i
```

```
ScriptingVM — root@scriptbox:~ — ssh -> vagrant ssh scriptbox — 61x19
[vagrant@scriptbox ~]$ sudo -i
[root@scriptbox ~]# cat /etc/hostname
scriptbox
[root@scriptbox ~]#
[root@scriptbox ~]#
```

## #Writing the first script

#Create a new folder and cd into that

#Check for vim editor

```
ScriptingVM — root@scriptbox:/opt/scripts — ssh -> vagrant ssh scriptbox — 79x19
[root@scriptbox ~]# mkdir /opt/scripts
[root@scriptbox ~]# cd /opt/scripts/
[root@scriptbox scripts]# ls
[root@scriptbox scripts]# vim
[root@scriptbox scripts]# yum install vim -y
Last metadata expiration check: 0:10:24 ago on Tue 10 Sep 2024 03:20:15 AM PDT.
Package vim-enhanced-2:8.2.4428-1.fc35.aarch64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[root@scriptbox scripts]#
```

## # Write the script : A script to print system info

```
ScriptingVM — root@scriptbox:/opt/scripts — ssh ▾ vagrant ssh scriptbox — 65x26
[[root@scriptbox scripts]# vim firstscript.sh
[[root@scriptbox scripts]# cd /opt/
[[root@scriptbox opt]# ls
scripts
[[root@scriptbox opt]# cd scripts/
[[root@scriptbox scripts]# ls
firstscript.sh
[[root@scriptbox scripts]# cat firstscript.sh
#!/bin/bash

echo "Hi, I am Aakansha. Welcome to Bash Script."
echo

echo "The uptime of the system is:"
uptime

echo "Memory Utilization"
free -m

echo "Disk Utilization"
df -h

[root@scriptbox scripts]#
```

## #Execute the script

```
ScriptingVM — root@scriptbox:/opt/scripts — ssh -v vagrant ssh scriptbox — 81x29
[[root@scriptbox scripts]# ./firstscript.sh
-bash: ./firstscript.sh: Permission denied
[[root@scriptbox scripts]# ls -l firstscript.sh
-rw-r--r--. 1 root root 180 Sep 10 03:39 firstscript.sh
[[root@scriptbox scripts]# chmod 754 firstscript.sh
[[root@scriptbox scripts]# ls -l
total 4
-rwxr-xr--. 1 root root 180 Sep 10 03:39 firstscript.sh
[[root@scriptbox scripts]# ./firstscript.sh
Hi, I am Aakansha. Welcome to Bash Script.

The uptime of the system is:
03:46:06 up 27 min, 1 user, load average: 0.05, 0.02, 0.00
Memory Utilization
              total        used        free      shared  buff/cache   available
Mem:       1950          203       1338            0         407      1721
Swap:      1949            0       1949
Disk Utilization
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M    0  4.0M  0% /dev
tmpfs           976M    0  976M  0% /dev/shm
tmpfs           391M  708K  390M  1% /run
/dev/mapper/fedora_fedora35-root  15G  2.0G   14G  13% /
/dev/nvme0n1p2     1014M 148M  867M  15% /boot
/dev/nvme0n1p1      599M  5.9M  593M  1% /boot/efi
tmpfs            196M    0  196M  0% /run/user/1000
[root@scriptbox scripts]#
```

## Commands:

```
[root@scriptbox scripts]# history
 1 cat /etc/hostname
 2 clear
 3 mkdir /opt/scripts
 4 cd /opt/scripts/
 5 ls
 6 vim
 7 yum install vim -y
 8 clear
 9 vim firstscript.sh
10 cd /opt/
11 ls
12 cd scripts/
13 ls
14 cat firstscript.sh
15 clear
16 ./firstscript.sh
17 ls -l firstscript.sh
18 chmod 754 firstscript.sh
19 ls -l
20 ./firstscript.sh
21 clear
[root@scriptbox scripts]#
```

## #Writing script for website setup

```
[root@scriptbox scripts]# vim websetup.sh
[root@scriptbox scripts]# cat websetup.sh
#!/bin/bash

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

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

# Stop & Disable Service
echo "#####
echo "Stop & Disable Firewalld Service"
echo "#####
sudo systemctl stop firewalld
sudo systemctl disable firewalld
echo
```

## #Executing the script

```
ScriptingVM — root@scriptbox:/opt/scripts — ssh + vagrant ssh scriptbox — 115x49

[root@scriptbox scripts]# ls -l websetup.sh
-rw-r--r--. 1 root root 1405 Sep 10 04:02 websetup.sh
[root@scriptbox scripts]# chmod 754 websetup.sh
[root@scriptbox scripts]# ls -l | grep websetup.sh
-rwxr-xr--. 1 root root 1405 Sep 10 04:02 websetup.sh
[root@scriptbox scripts]# ./websetup.sh
#####
Installing packages.
#####

#####
Start & Enable HTTPD Service
#####
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd/system/httpd.service.
#####
Stop & Disable Firewalld Service
#####
Removed /etc/systemd/system/multi-user.target.wants/firewalld.service.
Removed /etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service.

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

--2024-09-10 04:08:10-- https://www.tooplate.com/zip-templates/2098_health.zip
Resolving www.tooplate.com (www.tooplate.com)... 72.52.176.250
Connecting to www.tooplate.com (www.tooplate.com)|72.52.176.250|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1521593 (1.5M) [application/zip]
Saving to: '2098_health.zip'

2098_health.zip 100%[=====] 1.45M 952KB/s in 1.6s

2024-09-10 04:08:14 (952 KB/s) - '2098_health.zip' saved [1521593/1521593]

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

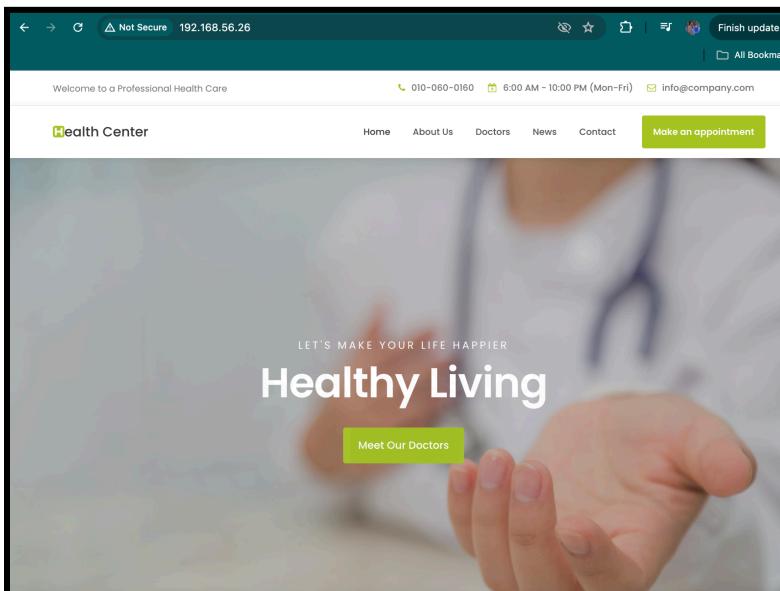
#####
Removing Temporary Files
#####

● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; )
   Active: active (running) since Tue 2024-09-10 04:08:15 PDT; 18ms
     Docs: man:httpd.service(8)
    Main PID: 17026 (httpd)
```

## #Let's test our website on browser using the VM IP

```
ScriptingVM — root@scriptbox:/opt/scripts — ssh -v vagrant ssh scriptbox — 64x21
[root@scriptbox scripts]# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:bd:54:93 brd ff:ff:ff:ff:ff:ff
    altname enp1s1
    inet 172.16.196.149/24 brd 172.16.196.255 scope global dynamic noprefixroute eth0
        valid_lft 1385sec preferred_lft 1385sec
        inet6 fe80::bd64:53c8:b0ba:4243/64 scope link noprefixroute
            valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:bd:54:9d brd ff:ff:ff:ff:ff:ff
    altname enp1s4
    inet 192.168.56.26/24 brd 192.168.56.255 scope global noprefixroute eth1
        valid_lft 1385sec preferred_lft 1385sec
```

## # Open the Browser and paste the IP



## **Commands:**

```
[root@scriptbox scripts]# history
```

1. clear
2. vim websetup.sh
3. cat websetup.sh
4. clear
5. ls -l websetup.sh
6. chmod 754 websetup.sh
7. ls -l | grep websetup.sh
8. ./websetup.sh
9. clear
10. ip addr show

```
[root@scriptbox scripts]#
```

## #Variables

In Bash scripting, **variables** are used to store and manipulate data. They act as placeholders for values, allowing you to reuse and manage information efficiently within a script.

### Key points:

**Declaring a Variable:** Variables in Bash are created by assigning a value without using the `=` sign. For example:

```
name="John"
```

- Here, `name` is the variable holding the value "John". Note that there should be no spaces around the `=` sign.

**Accessing a Variable:** To access the value of a variable, you prefix it with a `$` symbol:

```
echo $name
```

- This will output `John`.

**No Data Types:** Bash variables are untyped, meaning they can hold strings, numbers, or even the output of commands without specifying a type.

Variables are essential for creating flexible and dynamic scripts, enabling better control over script behavior.

## #Another Example

```
● ● ● ScriptingVM — root@scriptbox:/opt/scripts — ssh - vagrant ssh scriptbox — 79x21
[root@scriptbox scripts]# ls
firstscript.sh websetup.sh
[root@scriptbox scripts]# SKILL="DevOps"
[root@scriptbox scripts]# echo $SKILL
DevOps
[root@scriptbox scripts]# echo SKILL
SKILL
[root@scriptbox scripts]# PACKAGE="httpd wget unzip"
[root@scriptbox scripts]# yum install $PACKAGE -
Last metadata expiration check: 1:11:08 ago on Tue 10 Sep 2024 03:35:19 AM PDT.
Package httpd-2.4.51-2.fc35.aarch64 is already installed.
Package wget-1.21.2-2.fc35.aarch64 is already installed.
Package unzip-6.0-53.fc35.aarch64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[root@scriptbox scripts]#
```

## #Let's use our website script with variables

```
● ● ● ScriptingVM — root@scriptbox:/opt/scripts — ssh - vagrant ssh scriptbox — 79x26
[root@scriptbox scripts]# ls
firstscript.sh websetup.sh
[root@scriptbox scripts]# mv firstscript.sh 1_firshscript.sh
[root@scriptbox scripts]# mv websetup.sh 2_website.sh
[root@scriptbox scripts]# ls
1_firshscript.sh 2_website.sh
[root@scriptbox scripts]# cp 2_website.sh 3_vars_website.sh
[root@scriptbox scripts]# ls
1_firshscript.sh 2_website.sh 3_vars_website.sh
[root@scriptbox scripts]# vi 3_vars_website.sh
[root@scriptbox scripts]# cat 3_vars_website.sh
#!/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
```

## #Let's remove all the packages from 2\_website.sh to test 3\_vars\_website.sh website script with variables

```
● ● ● ScriptingVM - root@scriptbox:/opt/scripts - ssh - vagrant ssh scriptbox - 79x26
[[root@scriptbox scripts]# vim dismantle.sh
[[root@scriptbox scripts]# chmod +x dismantle.sh
[[root@scriptbox scripts]# ls
1_firshscript.sh 2_website.sh 3_vars_website.sh dismantle.sh
[[root@scriptbox scripts]# cat dismantle.sh
#!/bin/bash
sudo systemctl stop httpd
sudo rm -rf /var/www/html/*
sudo yum remove httpd wget unzip -y

[[root@scriptbox scripts]# ./dismantle.sh
Dependencies resolved.

=====
Package           Arch    Version      Repo      Size
=====
Removing:
httpd            aarch64 2.4.51-2.fc35   @fedora   9.0 M
unzip            aarch64 6.0-53.fc35    @fedora   469 k
wget             aarch64 1.21.2-2.fc35   @updates  3.2 M
Removing dependent packages:
perl              aarch64 4:5.34.0-482.fc35  @updates   0
Removing unused dependencies:
annobin-docs      noarch  9.87-3.fc35    @updates  85 k
annobin-plugin-gcc aarch64 9.87-3.fc35    @updates  68 k
apr               aarch64 1.7.0-14.fc35   @fedora   297 k
apr-util          aarch64 1.6.1-17.fc35   @fedora   224 k
```

## #Let's test 3\_vars\_website.sh

```
● ● ● ScriptingVM - root@scriptbox:/opt/scripts - ssh - vagrant ssh scriptbox - 81x19
[[root@scriptbox scripts]# ls
1_firshscript.sh 2_website.sh 3_vars_website.sh dismantle.sh
[[root@scriptbox scripts]# ./3_vars_website.sh
#####
Installing packages.
#####

#####
Start & Enable HTTPD Service
#####
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd/system/httpd.service.
#####
Stop & Disable Firewalld Service
#####

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

#Working Fine.

## #Command line arguments

In Bash scripting, command-line arguments allow you to pass data or parameters to your script when you run it from the terminal. These arguments make your script more dynamic, as you can provide different inputs without modifying the script itself.

### Key Concepts:

- \$0: Refers to the name of the script itself.
- \$1, \$2, ...: These represent the first, second, and subsequent command-line arguments passed to the script.
- \$#: Represents the total number of command-line arguments passed.
- \$@: A list of all the arguments.
- \$\*: Similar to \$@, but handles arguments slightly differently in special cases.

### Example:

If you run a script like this:

```
./script.sh arg1 arg2
```

- \$0 would be `./script.sh`
- \$1 would be `arg1`
- \$2 would be `arg2`
- \$# would be `2`, indicating two arguments were passed.

This allows for flexible scripts that can process different inputs based on the provided arguments.

## #Let's start the practice

```
[root@scriptbox scripts]# ls
1_firshscript.sh 2_website.sh 3_vars_website.sh dismantle.sh
[[root@scriptbox scripts]# vim 4_args.sh
[[root@scriptbox scripts]# chmod +x 4_args.sh
[[root@scriptbox scripts]# ./4_args.sh
*****
The value of 0 is
./4_args.sh
The value of 1 is

The value of 2 is

The value of 3 is

*****
[[root@scriptbox scripts]# ./4_args.sh Aakansha
*****
The value of 0 is
./4_args.sh
The value of 1 is
Aakansha
The value of 2 is

The value of 3 is

*****
[[root@scriptbox scripts]# ./4_args.sh Aakansha AWS DevOps
*****
The value of 0 is
./4_args.sh
The value of 1 is
Aakansha
The value of 2 is
AWS
The value of 3 is
DevOps
*****
[root@scriptbox scripts]#
```

This Bash script prints the values of the script name and the first three command-line arguments passed to it:

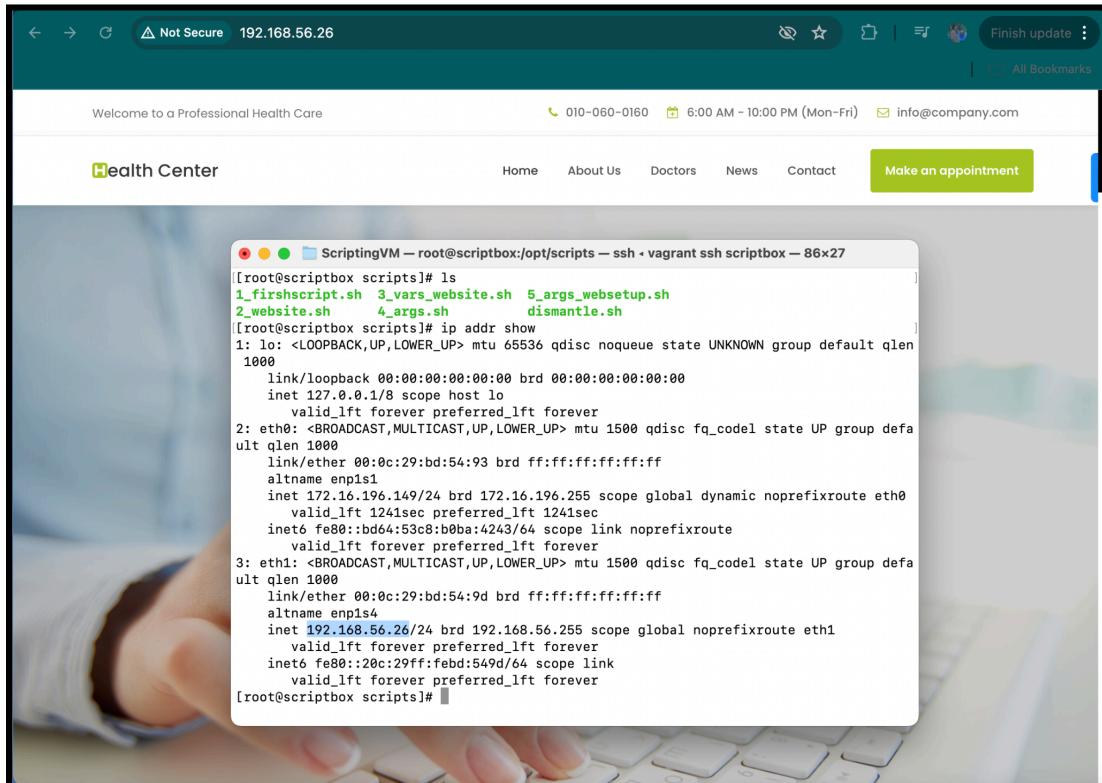
- \$0: Prints the name of the script.
- \$1, \$2, \$3: Prints the first, second, and third arguments provided when running the script.
- When you run the script with arguments, It will output the script name and the values of `arg1`, `arg2`, and `arg3` accordingly.

## #Let's see the application of command line arguments in our website setup script

```
[root@scriptbox scripts]# ls
1_firshscript.sh 2_website.sh 3_vars_website.sh 4_args.sh dismantle.sh
[root@scriptbox scripts]# cp 3_vars_website.sh 5_args_websetup.sh
[root@scriptbox scripts]# vim 5_args_websetup.sh
[root@scriptbox scripts]# ./dismantle.sh > /dev/null 2>&1
[root@scriptbox scripts]# vim 3_vars_website.sh
[root@scriptbox scripts]# chmod +x 5_args_websetup.sh
[root@scriptbox scripts]# ./5_args_websetup.sh https://www.tooplate.com/zip-templates/
2098_health.zip 2098_health
#####
Installing packages.
#####

#####
Start & Enable HTTPD Service
#####
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/lib/s
ystemd/system/httpd.service.
#####
Stop & Disable Firewalld Service
#####
```

## #Output



#Let's see the application of command line arguments in our website setup script with another url:

[https://www.tooplate.com/zip-templates/2091\\_ziggy.zip](https://www.tooplate.com/zip-templates/2091_ziggy.zip)

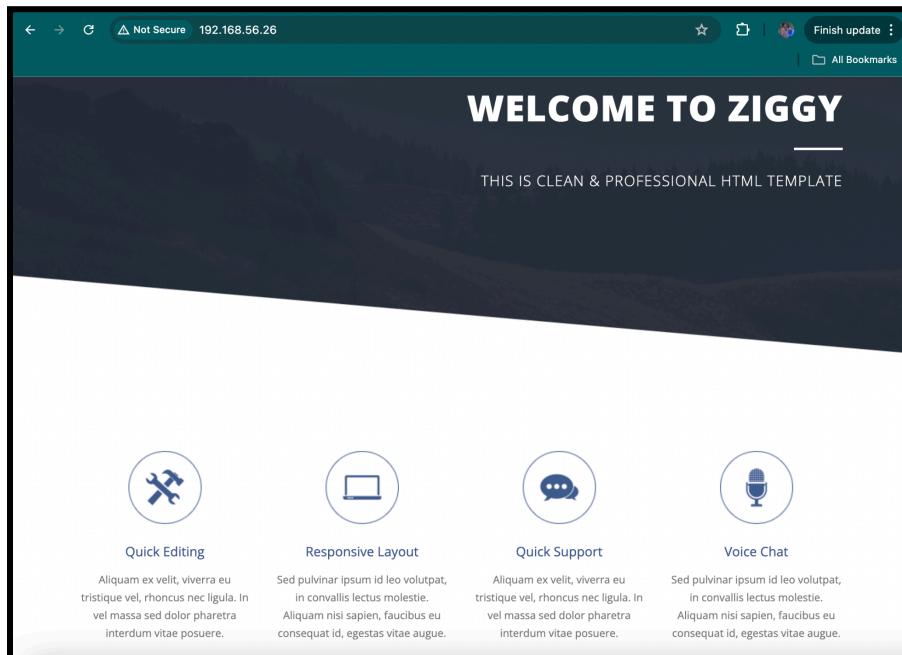
```
[root@scriptbox scripts]# ls
1_firstscript.sh 2_websetup.sh 3_vars_websetup.sh 4_args.sh 5_args_websetup.sh dismantle.sh
[root@scriptbox scripts]# ./5_args_websetup.sh https://www.tooplate.com/zip-templates/2091_ziggy.zip 2091_ziggy > /dev/null
--2024-09-10 22:29:35-- https://www.tooplate.com/zip-templates/2091_ziggy.zip
Resolving www.tooplate.com (www.tooplate.com)... 72.52.176.250
Connecting to www.tooplate.com (www.tooplate.com)|72.52.176.250|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4252660 (4.1M) [application/zip]
Saving to: '2091_ziggy.zip'

2091_ziggy.zip          100%[=====] 4.05M  453KB/s   in 11s

2024-09-10 22:29:48 (378 KB/s) - '2091_ziggy.zip' saved [4252660/4252660]

[root@scriptbox scripts]# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:bd:54:93 brd ff:ff:ff:ff:ff:ff
    altname enp1s1
    inet 172.16.196.149/24 brd 172.16.196.255 scope global dynamic noprefixroute eth0
        valid_lft 1623sec preferred_lft 1623sec
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:bd:54:9d brd ff:ff:ff:ff:ff:ff
    altname enp1s4
    inet 192.168.56.26/24 brd 192.168.56.255 scope global noprefixroute eth1
        valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fedb:549d/64 scope link
        valid_lft forever preferred_lft forever
[root@scriptbox scripts]#
```

## #Output



## # System variables

System variables in Bash scripting are predefined variables that provide information about the system and the environment. These variables are automatically set by the shell and can be accessed within scripts.

### Common system variables:

- **\$HOME**: The home directory of the current user.
- **\$USER**: The username of the current user.
- **\$PATH**: A list of directories where the system looks for executable files.
- **\$SHELL**: The path to the current shell.
- **\$PWD**: The current working directory.

**System variables are useful for writing scripts that interact with the environment or require system-specific data.**

```
ScriptingVM — root@scriptbox:~/scripts — ssh -> vagrant ssh scriptbox — 82x24
[[root@scriptbox scripts]]# free -m
      total        used        free      shared  buff/cache   available
Mem:       1950         211       1322          1        416       1713
Swap:      1949          0       1949
[[root@scriptbox scripts]]# echo $?
0
[[root@scriptbox scripts]]# freeeee -m
-bash: freeeee: command not found
[[root@scriptbox scripts]]# echo $?
127
[[root@scriptbox scripts]]# free -msdafsdd
free: seconds argument failed: 'dafssdd': Invalid argument
[[root@scriptbox scripts]]# echo $?
1
[[root@scriptbox scripts]]# echo $USER
root
```

This example demonstrates the exit status in Bash. After running a command, you can check its success or failure using the special variable **\$?**.

- **free -m** shows memory usage and returns an exit status of **0**, indicating success.
- **freeeee -m** gives "command not found" and returns **127**, meaning the command doesn't exist.
- **free -msdafsdd** gives an invalid argument error and returns **1**, indicating the command ran but failed due to incorrect input.

The exit status helps in determining whether a command executed successfully (0) or encountered an error (non-zero).

## # Quotes

In Bash scripting, quotes control how variables and special characters are interpreted.

```
ScriptingVM — root@scriptbox:~/scripts — ssh -v vagrant ssh scriptbox — 82x24
[[root@scriptbox scripts]]# NAME="Aakansha"
[[root@scriptbox scripts]]# echo "The girl $NAME have lost $5 million"
The girl Aakansha have lost $5 million
[[root@scriptbox scripts]]# echo 'The girl $NAME have lost $5 million'
The girl $NAME have lost $5 million
[[root@scriptbox scripts]]# echo "The girl $NAME have lost \$5 million"
The girl Aakansha have lost $5 million
[root@scriptbox scripts]#
```

- Double quotes ("'): Variables like \$NAME are expanded, but special characters like \$ can still be escaped with a backslash (\). In the above example, "The girl \$NAME" correctly expands to "Aakansha", but \\$5 is shown as "\$5" when escaped.
- Single quotes (''): Everything inside is treated as literal text, so \$NAME and \$5 are not expanded or interpreted.

Quotes allows precise control over how strings and variables are handled.

## # Command substitution

Command substitution in Bash allows you to run a command and use its output as part of another command or assign it to a variable.

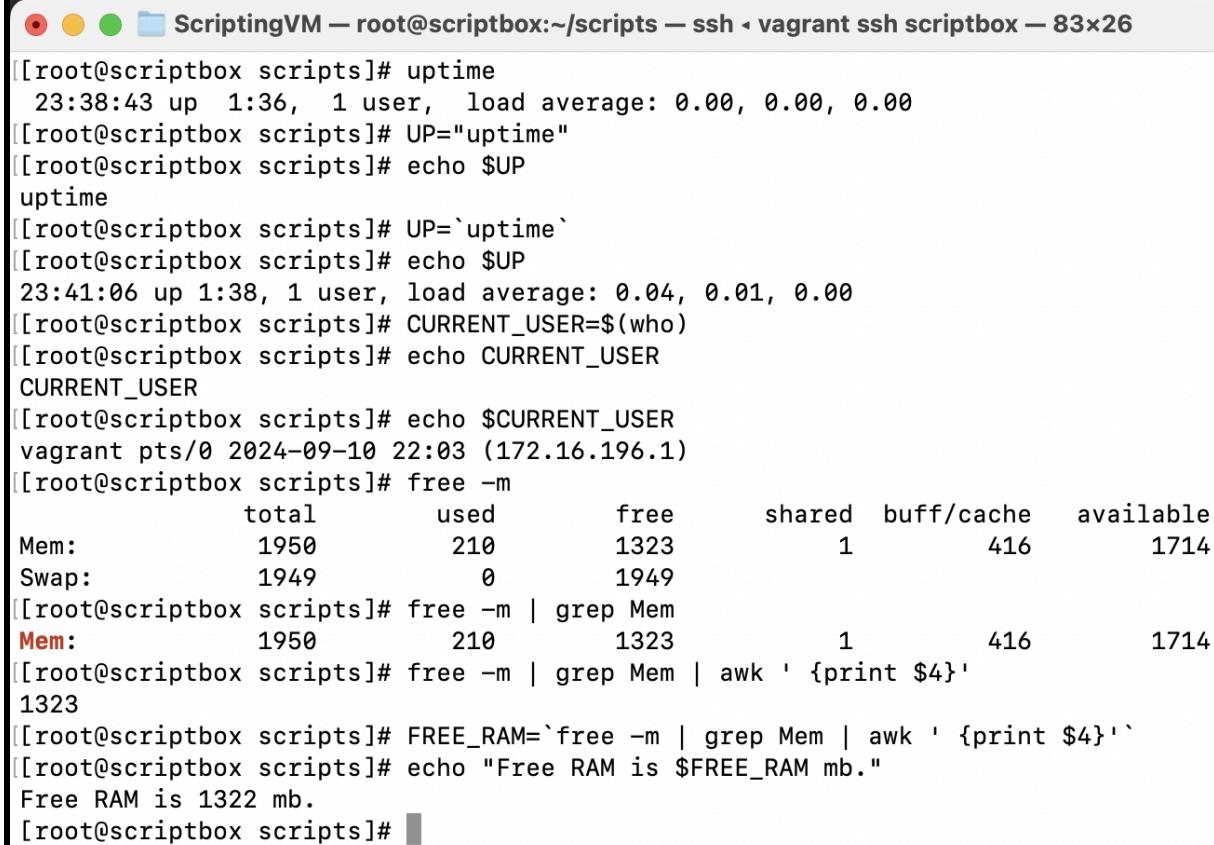
Syntax:

- Using backticks: ``command``
- Using `\$(command)` (preferred).

```
today=$(date)
```

This assigns the output of the `date` command to the `today` variable.

Command substitution is useful for dynamically capturing command output and using it within scripts.



The screenshot shows a terminal window titled "ScriptingVM — root@scriptbox:~/scripts — ssh ↵ vagrant ssh scriptbox — 83x26". The terminal displays the following commands and their outputs:

```
[[root@scriptbox scripts]]# uptime
23:38:43 up 1:36, 1 user, load average: 0.00, 0.00, 0.00
[[root@scriptbox scripts]]# UP="uptime"
[[root@scriptbox scripts]]# echo $UP
uptime
[[root@scriptbox scripts]]# UP=`uptime`
[[root@scriptbox scripts]]# echo $UP
23:41:06 up 1:38, 1 user, load average: 0.04, 0.01, 0.00
[[root@scriptbox scripts]]# CURRENT_USER=$(who)
[[root@scriptbox scripts]]# echo CURRENT_USER
CURRENT_USER
[[root@scriptbox scripts]]# echo $CURRENT_USER
vagrant pts/0 2024-09-10 22:03 (172.16.196.1)
[[root@scriptbox scripts]]# free -m
      total        used        free      shared  buff/cache   available
Mem:       1950         210       1323           1         416        1714
Swap:      1949          0       1949
[[root@scriptbox scripts]]# free -m | grep Mem
Mem:       1950         210       1323           1         416        1714
[[root@scriptbox scripts]]# free -m | grep Mem | awk '{print $4}'
1323
[[root@scriptbox scripts]]# FREE_RAM=`free -m | grep Mem | awk '{print $4}'` 
[[root@scriptbox scripts]]# echo "Free RAM is $FREE_RAM mb."
Free RAM is 1322 mb.
[[root@scriptbox scripts]]#
```

This example demonstrates command substitution and pipelining:

**1. Command substitution:** The output of a command is assigned to a variable, like using `uptime` or \$(who) to store the output in UP and CURRENT\_USER respectively.

- o UP=`uptime` captures the system uptime and stores it in UP`.
- o CURRENT\_USER=\$(who) stores the current user information.

**2. Pipelining:** Commands are combined using pipes (|) to process output in stages.

- o free -m | grep Mem | awk '{print \$4}' extracts the available memory in MB.
- o This value is stored in FREE\_RAM and printed as "Free RAM is \$FREE\_RAM mb."

This illustrates how command substitution and pipelines can be used together to process and store command outputs dynamically.

## #Let's create a script to print the health of the system using the concept of command substitution and pipelining.

```
● ● ● ScriptingVM — root@scriptbox:~/scripts — ssh -> vagrant ssh scriptbox — 83x32
[root@scriptbox scripts]# vim 6_command_subs.sh
[root@scriptbox scripts]# chmod +x 6_command_subs.sh
[root@scriptbox scripts]# cat 6_command_subs.sh
#!/bin/bash

echo "Welcome $USER on $HOSTNAME."
echo "#####
FREERAM=$(free -m | grep Mem | awk '{print $4}')
LOAD=`uptime | awk '{print $9}'`
ROOTFREE=$(df -h | grep '/' | awk '{print $4}')

echo #####
echo "Available free RAM is $FREERAM MB"
echo #####
echo "Current Load Average $LOAD"
echo #####
echo "Free ROOT partition size is $ROOTFREE"
echo #####
[root@scriptbox scripts]#
[root@scriptbox scripts]# ./6_command_subs.sh
Welcome root on scriptbox.
#####
Available free RAM is 1322 MB
#####
Current Load Average 0.02,
#####
Free ROOT partition size is 14G
#####
[root@scriptbox scripts]#
```

## # Exporting variables

In Bash, exporting variables allows them to be accessible to child processes or scripts spawned from the current shell. Without exporting, variables are only available in the current shell session.

### To export a variable:

```
export VAR_NAME=value
```

Once a variable is exported, it can be accessed by any scripts or commands run in the same shell session or any subshells.

### Example:

```
NAME="Aakansha"  
export NAME
```

Here, `NAME` is exported and can be used by any child processes, like another script run from the terminal.

**Exporting is essential when you need to share environment variables between scripts or commands.**

**The `.bashrc` file is commonly used to export variables that need to be available in every Bash session automatically.**

By adding `export` commands in `.bashrc`, you ensure that those variables are set and exported each time you open a new terminal or start a new session.

The `.bashrc` file is a script that runs automatically whenever a new Bash session (non-login) is started, such as opening a terminal. It is used to configure your environment by defining aliases, functions, and environment variables that persist in each session.

## Key points:

- Located in your home directory (~), it's hidden (indicated by the . before the name).
- You can customize it to set environment variables or add aliases that you want available in every terminal session.
- After editing `.bashrc`, you need to run `source .bashrc` to apply changes immediately without restarting the session.

```
[root@scriptbox scripts]# ls
1_firstscript.sh  3_vars_websetup.sh  5_args_websetup.sh  dismantle.sh
2_websetup.sh    4_args.sh          6_command_subs.sh
[root@scriptbox scripts]# cd
[root@scriptbox ~]# ls
anaconda-ks.cfg  original-ks.cfg  scripts
[root@scriptbox ~]# ls -a
.  anaconda-ks.cfg  .bash_logout   .bashrc      .lessht      scripts  .viminfo
..  .bash_history   .bash_profile  .cshrc      original-ks.cfg .tcshrc
[root@scriptbox ~]# source .bashrc
[root@scriptbox ~]# vi .bashrc
[root@scriptbox ~]#
```

## Current Bashrc File:

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

# User specific environment
if ! [[ "$PATH" =~ "$HOME/.local/bin:$HOME/bin:" ]]
then
    PATH="$HOME/.local/bin:$HOME/bin:$PATH"
fi
export PATH

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions

alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
~

:d|
```

## After updating the bashrc file with export command:

```
ScriptingVM — root@scriptbox:~ — ssh -v vagrant ssh scriptbox — 82x30
[[root@scriptbox ~]# ls -a
. anaconda-ks.cfg .bash_logout .bashrc .lesshst      scripts .viminfo
.. .bash_history   .bash_profile .cshrc  original-ks.cfg .tcshrc
[[root@scriptbox ~]# cat .bashrc
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

# User specific environment
if ! [[ "$PATH" =~ "$HOME/.local/bin:$HOME/bin:" ]]
then
    PATH="$HOME/.local/bin:$HOME/bin:$PATH"
fi
export PATH

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions

alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

export NAME="Aakansha"

[[root@scriptbox ~]#
```

## Output:

```
ScriptingVM — vagrant@scriptbox:~ — ssh -v vagrant ssh scriptbox — 82x22
[[root@scriptbox ~]# vi .bashrc
[[root@scriptbox ~]# echo $NAME
Aakansha
[[root@scriptbox ~]# exit
logout
[[vagrant@scriptbox ~]$ sudo -i
[[root@scriptbox ~]# ls
anaconda-ks.cfg  original-ks.cfg  scripts
[[root@scriptbox ~]# echo $NAME
Aakansha
```

## # User Input

In Bash scripting, you can get user input using the `read` command. It's simple and effective for interactive scripts.

```
[root@scriptbox scripts]# ls
1_firstscript.sh  3_vars_websetup.sh  5_args_websetup.sh  7_userInput.sh
2_websetup.sh    4_args.sh          6_command_subs.sh  dismantle.sh
[root@scriptbox scripts]# chmod +x 7_userInput.sh
[root@scriptbox scripts]# ./7_userInput.sh
Enter your skill:
[DevOps
Your DevOps skill is in high demand in IT Industry.
[Username: Aakansha
>Password:
Login Successfull: Welcome USER Aakansha
[root@scriptbox scripts]# cat 7_userInput.sh
#!/bin/bash

echo "Enter your skill:"
read SKILL

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

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

echo

echo "Login Successfull: Welcome USER $USR"

[root@scriptbox scripts]#
```

This script prompts the user for input and displays personalized messages based on the input.

1. It asks for the user's skill using `read SKILL` and then prints a message that their skill is in high demand.
2. It prompts for a **username** and **password**:
  - o `read -p 'Username: '` takes the username input.
  - o `read -sp 'Password: '` takes the password input silently (without showing on the screen).
3. After the input, it prints a welcome message using the entered username.

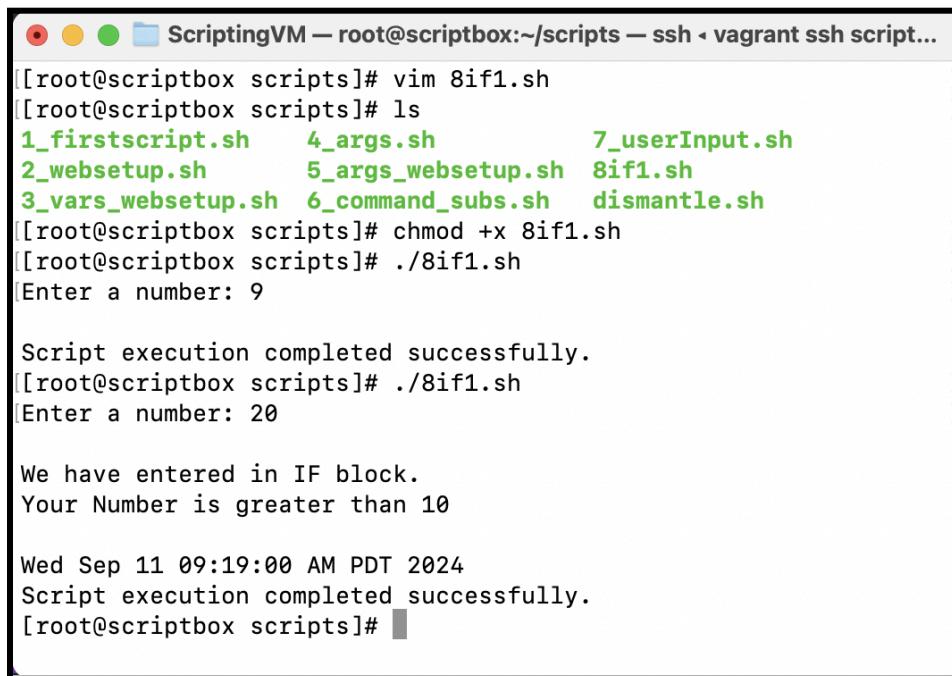
**This is an interactive script that handles user input dynamically.**

## # Decision-making

In Bash scripting, decision-making allows you to control the flow of the script based on conditions.

### 1. If command:

The **if** command checks a condition, and if it's true, it executes the following commands.



The screenshot shows a terminal window titled "ScriptingVM — root@scriptbox:~/scripts — ssh - vagrant ssh script...". The terminal displays the following session:

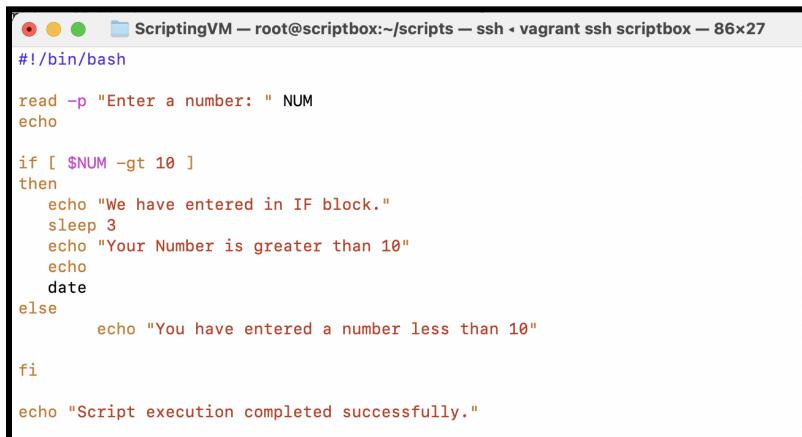
```
[root@scriptbox scripts]# vim 8if1.sh
[[root@scriptbox scripts]# ls
1_firstscript.sh    4_args.sh        7_userInput.sh
2_websetup.sh       5_args_websetup.sh 8if1.sh
3_vars_websetup.sh 6_command_subs.sh dismantle.sh
[[root@scriptbox scripts]# chmod +x 8if1.sh
[[root@scriptbox scripts]# ./8if1.sh
[Enter a number: 9

Script execution completed successfully.
[[root@scriptbox scripts]# ./8if1.sh
[Enter a number: 20

We have entered in IF block.
Your Number is greater than 10

Wed Sep 11 09:19:00 AM PDT 2024
Script execution completed successfully.
[root@scriptbox scripts]# ]]
```

The **else** statement is used to define a block of code that will execute if none of the **if** or **elif** conditions are met. It's the final fallback option when all other conditions are false.



The screenshot shows a terminal window titled "ScriptingVM — root@scriptbox:~/scripts — ssh - vagrant ssh scriptbox — 86x27". The terminal displays the following session:

```
#!/bin/bash

read -p "Enter a number: " NUM
echo

if [ $NUM -gt 10 ]
then
    echo "We have entered in IF block."
    sleep 3
    echo "Your Number is greater than 10"
    echo
    date
else
    echo "You have entered a number less than 10"
fi

echo "Script execution completed successfully."
```

## OUTPUT:

```
ScriptingVM — root@scriptbox:~/scripts — ssh -vagrant ssh scriptbox — 86x27
[[root@scriptbox scripts]# ls
1_firstscript.sh 4_args.sh      7_userInput.sh
2_websetup.sh    5_args_websetup.sh 8if1.sh
3_vars_websetup.sh 6_command_subs.sh dismantle.sh
[[root@scriptbox scripts]# cp 8if1.sh 9_if1.sh
[[root@scriptbox scripts]# vim 9_if1.sh
[[root@scriptbox scripts]# chmod +x 9_if1.sh
[[root@scriptbox scripts]# ./9_if1.sh
[Enter a number: 4

./9_if1.sh: line 18: unexpected EOF while looking for matching `"'
./9_if1.sh: line 19: syntax error: unexpected end of file
[[root@scriptbox scripts]# vim 9_if1.sh
[[root@scriptbox scripts]# ./9_if1.sh
[Enter a number: 5

You have entered a number less than 10
Script execution completed successfully.
[[root@scriptbox scripts]# ./9_if1.sh
[Enter a number: 90

We have entered in IF block.
Your Number is greater than 10

Wed Sep 11 09:26:39 AM PDT 2024
Script execution completed successfully.
[[root@scriptbox scripts]# ]]
```

## 2. Elif command:

`elif` stands for "else if," and it allows you to check multiple conditions.

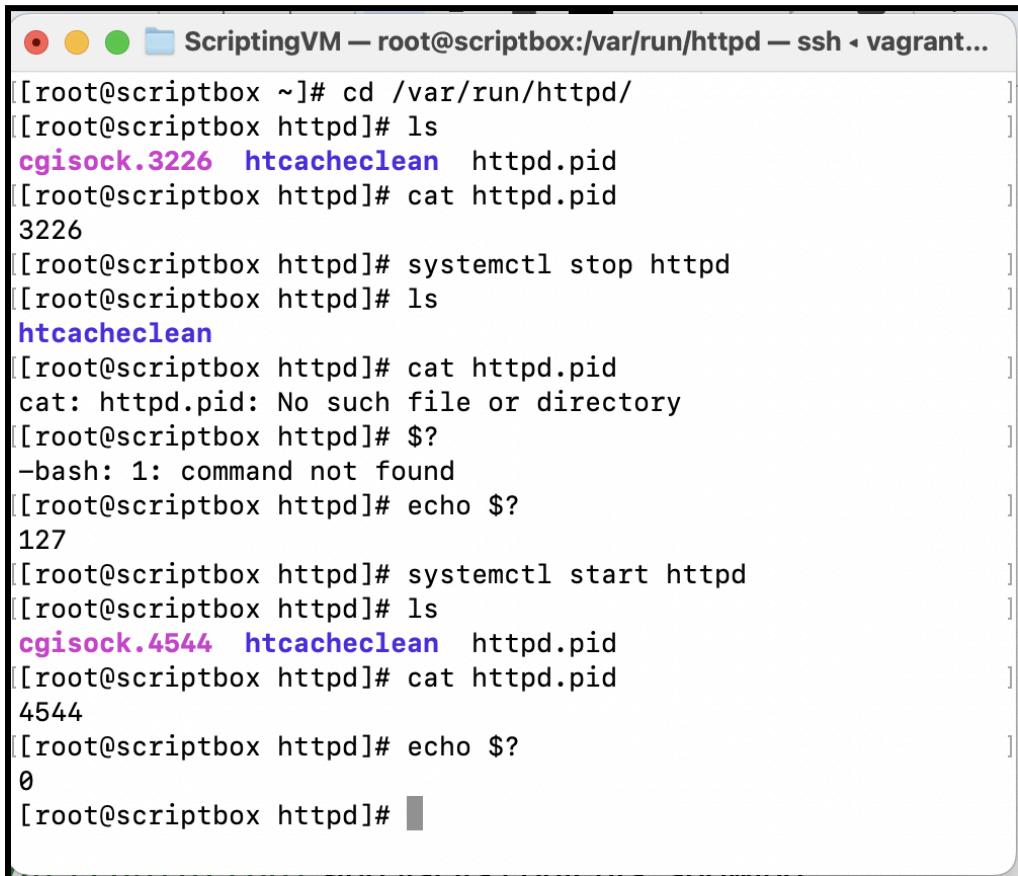
```
ScriptingVM — root@scriptbox:~/scripts — ssh -vagrant ssh scriptbox — 86x27
[[root@scriptbox scripts]# ls
1_firstscript.sh 4_args.sh      7_userInput.sh  dismantle.sh
2_websetup.sh    5_args_websetup.sh 8if1.sh
3_vars_websetup.sh 6_command_subs.sh 9_if1.sh
[[root@scriptbox scripts]# vim 10_ifelif.sh
[[root@scriptbox scripts]# chmod +x 10_ifelif.sh
[[root@scriptbox scripts]# ls
10_ifelif.sh     3_vars_websetup.sh 6_command_subs.sh 9_if1.sh
1_firstscript.sh 4_args.sh        7_userInput.sh  dismantle.sh
2_websetup.sh    5_args_websetup.sh 8if1.sh
[[root@scriptbox scripts]# ./10_ifelif.sh
Found Multiple Active Interface.
[[root@scriptbox scripts]# ]]
```

## # Bash operators

Operator	Description	Example
`-eq`	Equal to (integer comparison)	`[ 5 -eq 5 ]` (true)
`-ne`	Not equal to (integer comparison)	`[ 5 -ne 3 ]` (true)
`-gt`	Greater than	`[ 5 -gt 3 ]` (true)
`-lt`	Less than	`[ 3 -lt 5 ]` (true)
`-ge`	Greater than or equal to	`[ 5 -ge 5 ]` (true)
`-le`	Less than or equal to	`[ 5 -le 5 ]` (true)
`==`	String equality	`[ "abc" == "abc" ]` (true)
`!=`	String inequality	`[ "abc" != "xyz" ]` (true)
`-z`	String is empty	`[ -z "\$str" ]`
`-n`	String is not empty	`[ -n "\$str" ]`
`&&`	Logical AND	`[ 5 -gt 3 ] && [ 5 -lt 10 ]` (true)
` `		` `
`!`	Logical NOT	`[ ! 5 -eq 3 ]` (true)
`-d`	Checks if directory exists	`[ -d /path/to/dir ]`
`-f`	Checks if file exists	`[ -f /path/to/file ]`
`-r`	File is readable	`[ -r /path/to/file ]`
`-w`	File is writable	`[ -w /path/to/file ]`
`-x`	File is executable	`[ -x /path/to/file ]`

## # Process ID

Whenever there is a process running there will be a PID file created.



The screenshot shows a terminal window titled "ScriptingVM — root@scriptbox:/var/run/httpd — ssh - vagrant...". The terminal session demonstrates the creation and destruction of an `httpd.pid` file:

```
[root@scriptbox ~]# cd /var/run/httpd/
[root@scriptbox httpd]# ls
cgisock.3226  htcacheclen  httpd.pid
[root@scriptbox httpd]# cat httpd.pid
3226
[root@scriptbox httpd]# systemctl stop httpd
[root@scriptbox httpd]# ls
htcacheclen
[root@scriptbox httpd]# cat httpd.pid
cat: httpd.pid: No such file or directory
[root@scriptbox httpd]# $?
-bash: 1: command not found
[root@scriptbox httpd]# echo $?
127
[root@scriptbox httpd]# systemctl start httpd
[root@scriptbox httpd]# ls
cgisock.4544  htcacheclen  httpd.pid
[root@scriptbox httpd]# cat httpd.pid
4544
[root@scriptbox httpd]# echo $?
0
[root@scriptbox httpd]#
```

1. We navigate to `/var/run/httpd/` and list its contents, showing files related to the HTTP server, including `httpd.pid`, which contains the process ID (PID) of the running `httpd`.
2. After stopping, the `httpd.pid` file is gone because the process has terminated.
3. The usage (`echo $?`) returns `127`, indicating an error.
4. After restarting the `httpd` service, a new PID (`4544`) appears in `httpd.pid`, and the command `echo $?` returns `0`, showing success.

**# Let's use the above concept to make a monitoring script that will detect if the process is running and if it's not running it will start the process.**

```
[root@scriptbox scripts]# vim 11_monit.sh
[root@scriptbox scripts]# chmod +x 11_monit.sh
[root@scriptbox scripts]# ls
10_ifelif.sh      2_websetup.sh      5_args_websetup.sh  8if1.sh
11_monit.sh       3_vars_websetup.sh  6_command_subs.sh  9_if1.sh
1_firstscript.sh  4_args.sh         7_userInput.sh     dismantle.sh
[root@scriptbox scripts]# ./11_monit.sh
#####
Wed Sep 11 09:39:30 PM PDT 2024
Httpd process is running.
#####

[root@scriptbox scripts]# systemctl stop httpd
[root@scriptbox scripts]# ./11_monit.sh
#####
Wed Sep 11 09:39:57 PM PDT 2024
Httpd process is NOT Running.
Starting the process
Process started successfully.
#####

[root@scriptbox scripts]#
```

**# Let's automate this process for every minute by scheduling it. First edit the crontab and schedule the job.**

```
[root@scriptbox scripts]# crontab -e
no crontab for root - using an empty one
crontab: installing new crontab
```

**We are scheduling it for every day of the month, every minute.**

```
# MM HH DOM mm DOW
#Example:
# 30 20 * * 1-5 COMMAND

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

:wq
```

## # Let's test the script and check the log file

```
ScriptingVM — root@scriptbox:~/scripts — ssh -v vagrant ssh scriptbox — 84x29
[[root@scriptbox scripts]# systemctl stop httpd
[[root@scriptbox scripts]# ls /var/log/monit_httpd.log
/var/log/monit_httpd.log
[[root@scriptbox scripts]# cat /var/log/monit_httpd.log
#####
Wed Sep 11 09:54:01 PM PDT 2024
Httpd process is running.
#####

#####
Wed Sep 11 09:55:01 PM PDT 2024
Httpd process is running.
#####

#####
Wed Sep 11 09:56:01 PM PDT 2024
Httpd process is running.
#####
```

## # Let's add some creativity to the script using bash operator (-f)

```
ScriptingVM — root@scriptbox:~/scripts — ssh -v vagrant ssh scriptbox — 84x29
[[root@scriptbox scripts]# cp 11_monit.sh 12_monit.sh
[[root@scriptbox scripts]# vim 12_monit.sh
[[root@scriptbox scripts]# chmod +x 12_monit.sh
[[root@scriptbox scripts]# ./12_monit.sh
#####
Wed Sep 11 10:08:42 PM PDT 2024
Httpd process is running.
#####

[[root@scriptbox scripts]# systemctl stop httpd
[[root@scriptbox scripts]# ./12_monit.sh
#####
Wed Sep 11 10:09:31 PM PDT 2024
Httpd process is NOT Running.
Starting the process
Process started successfully.
#####

[root@scriptbox scripts]#
```

## # Loops in Bash

In Bash scripting, loops allow you to execute a block of code repeatedly, depending on a condition. The most common types of loops are:

### 1. For Loop

Executes a block of code for each item in a list or range.

```
for item in 1 2 3 4 5; do
    echo $item
done
```

### 2. While Loop

Runs as long as the condition remains true.

```
count=1
while [ $count -le 5 ]; do
    echo "Count is $count"
    ((count++))
done
```

### 3. Until Loop

Runs until the condition becomes true.

```
count=1
until [ $count -gt 5 ]; do
    echo "Count is $count"
    ((count++))
done
```

## #Let's start the practice using scripts

### Example-1 :

```
ScriptingVM — root@scriptbox:~/scripts — ssh - vagrant ssh scriptbox — 84x29
[root@scriptbox scripts]# vim 13_for.sh
[root@scriptbox scripts]# chmod +x 13_for.sh
[root@scriptbox scripts]# ./13_for.sh
Looping...
#####
Value of VARI is java.
#####
Wed Sep 11 10:23:08 PM PDT 2024
Looping...
#####
Value of VARI is .net.
#####
Wed Sep 11 10:23:09 PM PDT 2024
Looping...
#####
Value of VARI is python.
#####
Wed Sep 11 10:23:10 PM PDT 2024
Looping...
#####
Value of VARI is ruby.
#####
Wed Sep 11 10:23:11 PM PDT 2024
Looping...
#####
Value of VARI is php.
#####
Wed Sep 11 10:23:12 PM PDT 2024
[root@scriptbox scripts]#
```

### Example-2 :

```
ScriptingVM — root@scriptbox:~/scripts — ssh - vagrant ssh scriptbox — 84x29
[root@scriptbox scripts]# vim 14_for.sh
[root@scriptbox scripts]# chmod +x 14_for.sh
[root@scriptbox scripts]# ./14_for.sh
Adding user Aakansha.
uid=1001(Aakansha) gid=1001(Aakansha) groups=1001(Aakansha)
#####
Adding user Aman.
uid=1002(Aman) gid=1002(Aman) groups=1002(Aman)
#####
Adding user Ankita.
uid=1003(Ankita) gid=1003(Ankita) groups=1003(Ankita)
#####
[root@scriptbox scripts]#
```

## While Loops in Bash

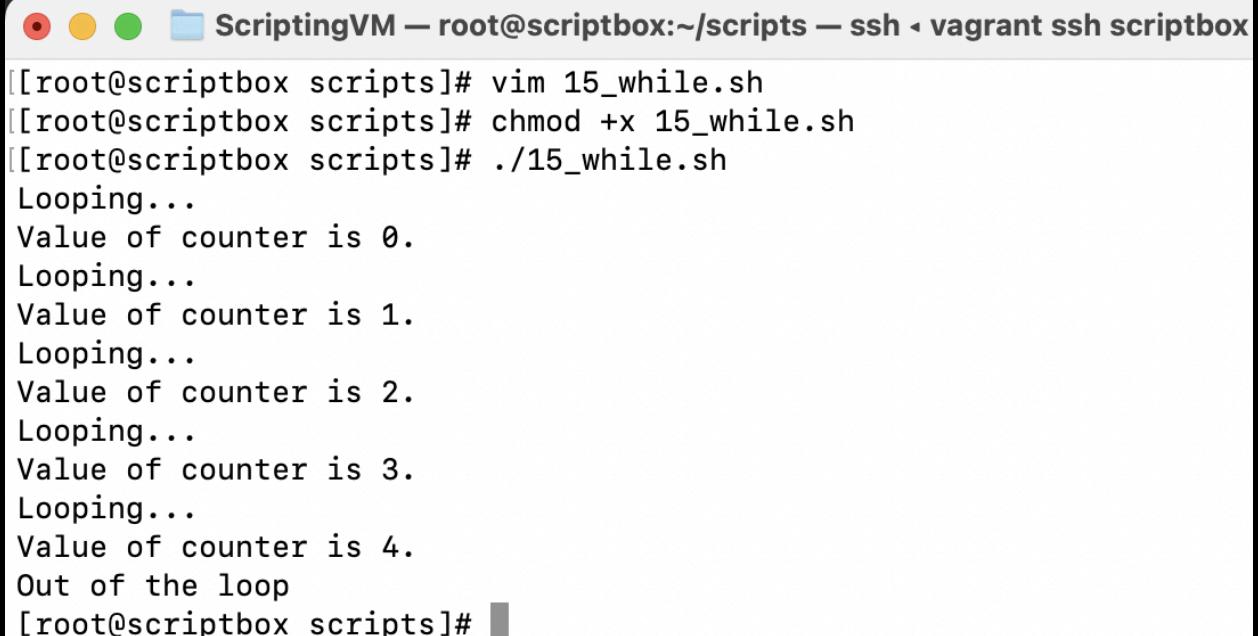
A **while** loop in Bash repeatedly executes a block of code as long as the given condition is true. The condition is evaluated before each iteration, and if it returns true, the loop continues.

### Syntax:

```
while [ condition ]; do
    # Code to be executed
done
```

### Let's Start the practice

#### Example-1: Simple while loop



The screenshot shows a terminal window titled "ScriptingVM — root@scriptbox:~/scripts — ssh - vagrant ssh scriptbox". The terminal output is as follows:

```
[root@scriptbox scripts]# vim 15_while.sh
[root@scriptbox scripts]# chmod +x 15_while.sh
[root@scriptbox scripts]# ./15_while.sh
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
[root@scriptbox scripts]#
```

## Infinite **while** loop (until manually stopped)

```
while true; do
    echo "This will run forever"
done
```

## Example-2: Infinite while loop

```
ScriptingVM — root@scriptbox:~/scripts — ssh -v vagrant ssh scriptbox — 84x29
[[root@scriptbox scripts]]# ls
10_ifelif.sh 14_for.sh      3_vars_websetup.sh 7_userInput.sh
11_monit.sh   15_while.sh    4_args.sh          8if1.sh
12_monit.sh   1_firstscript.sh 5_args_websetup.sh 9_if1.sh
13_for.sh     2_websetup.sh   6_command_subs.sh  dismantle.sh
[[root@scriptbox scripts]]# cp 15_while.sh 16_while.sh
[[root@scriptbox scripts]]# vim 16_while.sh
[[root@scriptbox scripts]]# chmod +x 16_while.sh
[[root@scriptbox scripts]]# ./16_while.sh
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.
Looping...
Value of counter is 64.
Looping...
Value of counter is 128.
Looping...
Value of counter is 256.
^C
[[root@scriptbox scripts]]#
```



## ...PROJECT TIME...

### # Remote Command Execution

Remote Command Execution (RCE) allows a user to execute commands on a remote machine from a local system. This is commonly done using tools like [ssh](#) (Secure Shell) to securely connect and manage remote servers or computers.

#### How It Works:

- You connect to a remote server using SSH.
- Once authenticated, you can execute commands on the remote machine as if you were physically present at the system.

#### Benefits of Remote Command Execution:

1. **Convenience:**  
Allows system administrators and users to manage multiple machines from a single location, saving time and effort.
2. **Automation:**  
Scripts can be written to execute commands remotely on multiple servers, making it easy to perform tasks like updates, backups, and monitoring across machines.
3. **Security:**  
SSH encrypts all communications, making it a secure way to manage remote systems and execute commands.
4. **Resource Management:**  
Remote command execution helps in monitoring and managing remote system resources without needing to access them physically.
5. **Efficiency:**  
Quick troubleshooting, software installation, and configuration changes can be performed remotely, improving the efficiency of IT management.

**RCE is widely used in system administration, DevOps, and cloud management for effective remote control of resources.**

**In this project we will see how we can execute commands from script box to the other VMS which were created using this vagrant file.**

**Let's start with checking their hostnames.**

```
ScriptingVM — vagrant@web03: ~ — ssh - vagrant ssh web03 — 84x46

[[root@scriptbox scripts]# exit
logout
[[vagrant@scriptbox ~]$ exit
logout
Connection to 172.16.196.149 closed.
[aakansha@M2 ScriptingVM % vagrant ssh web01
[[vagrant@web01 ~]$ hostname
web01
[[vagrant@web01 ~]$ exit
logout
Connection to 172.16.196.150 closed.
[aakansha@M2 ScriptingVM % vagrant ssh web02
[[vagrant@web02 ~]$ hostname
web02
[[vagrant@web02 ~]$ exit
logout
Connection to 172.16.196.151 closed.
[aakansha@M2 ScriptingVM % vagrant ssh web03
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.4.0-89-generic aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 System information as of Thu 12 Sep 2024 06:08:55 AM UTC

 System load:  0.0          Processes:      202
 Usage of /:   21.0% of 19.56GB  Users logged in:    0
 Memory usage: 30%
 Swap usage:   0%          IPv4 address for eth0: 172.16.196.135
                           IPv4 address for eth1: 192.168.56.29

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
 just raised the bar for easy, resilient and secure K8s cluster deployment.

 https://ubuntu.com/engage/secure-kubernetes-at-the-edge

276 updates can be applied immediately.
212 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Last login: Thu Oct 28 16:32:51 2021
[vagrant@web03:~$ hostname
web03
vagrant@web03:~$
```

**Now let's take note of the IP addresses of these machines to map in the scriptbox**

```
aakansha@M2 ScriptingVM % ls
Vagrantfile
aakansha@M2 ScriptingVM % cat Vagrantfile
Vagrant.configure("2") do |config|
  config.vm.define "scriptbox" do |scriptbox|
    scriptbox.vm.box = "jacobw/fedora35-arm64"
    scriptbox.vm.network "private_network", ip: "192.168.56.26"
    scriptbox.vm.hostname = "scriptbox"
    scriptbox.vm.provider "vmware_desktop" do |v|
      v.allowlist_verified = true
      v.ssh_info_public = true
      v.gui = true
    end
    scriptbox.vm.provision "shell", inline: <<-SHELL
      mv /etc/yum.repos.d/fedora-updates.repo /tmp/
      mv /etc/yum.repos.d/fedora-updates-modular.repo /tmp/
      yum clean all
      yum update
      systemctl stop firewalld
    SHELL
  end
  config.vm.define "web01" do |web01|
    web01.vm.box = "jacobw/fedora35-arm64"
    web01.vm.network "private_network", ip: "192.168.56.27"
    web01.vm.hostname = "web01"
    web01.vm.provider "vmware_desktop" do |v|
      v.allowlist_verified = true
      v.ssh_info_public = true
      v.gui = true
    end
    web01.vm.provision "shell", inline: <<-SHELL
      mv /etc/yum.repos.d/fedora-updates.repo /tmp/
      mv /etc/yum.repos.d/fedora-updates-modular.repo /tmp/
      yum clean all
      yum update
      systemctl stop firewalld
    SHELL
  end
  config.vm.define "web02" do |web02|
    web02.vm.box = "jacobw/fedora35-arm64"
    web02.vm.network "private_network", ip: "192.168.56.28"
    web02.vm.hostname = "web02"
    web02.vm.provider "vmware_desktop" do |v|
      v.allowlist_verified = true
      v.ssh_info_public = true
      v.gui = true
    end
    web02.vm.provision "shell", inline: <<-SHELL
      mv /etc/yum.repos.d/fedora-updates.repo /tmp/
      mv /etc/yum.repos.d/fedora-updates-modular.repo /tmp/
      yum clean all
      yum update
      systemctl stop firewalld
    SHELL
  end
  config.vm.define "web03" do |web03|
    web03.vm.box = "spox/ubuntu-arm"
    web03.vm.box_version = "1.0.0"
    web03.vm.network "private_network", ip: "192.168.56.29"
    web03.vm.hostname = "web03"
    web03.vm.provider "vmware_desktop" do |v|
      v.allowlist_verified = true
    end
  end
end
```

**Now Let's save these into the host file of script box**

```
[aakansha@M2 ScriptingVM % vagrant ssh scriptbox
Last login: Tue Sep 10 22:03:00 2024 from 172.16.196.1
[vagrant@scriptbox ~]$ sudo -i
[root@scriptbox ~]# vim /etc/hosts
[root@scriptbox ~]# ping -c 1 web01
PING web01 (192.168.56.27) 56(84) bytes of data.
64 bytes from web01 (192.168.56.27): icmp_seq=1 ttl=64 time=17.2 ms

--- web01 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 17.156/17.156/17.156/0.000 ms
[root@scriptbox ~]# ping -c 1 web02
PING web02 (192.168.56.28) 56(84) bytes of data.
64 bytes from web02 (192.168.56.28): icmp_seq=1 ttl=64 time=3.34 ms

--- web02 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 3.338/3.338/3.338/0.000 ms
[root@scriptbox ~]# ping -c 1 web03
PING web03 (192.168.56.29) 56(84) bytes of data.
64 bytes from web03 (192.168.56.29): icmp_seq=1 ttl=64 time=3.18 ms

--- web03 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 3.179/3.179/3.179/0.000 ms
[root@scriptbox ~]# ]]
```

**The hostname is resolving successfully to IP addresses.**

**Now let's test the connection and enter the other VMS using ssh**

**Note: The password for a vagrant user is vagrant**

```
ScriptingVM — root@scriptbox:~ — ssh -vagrant ssh scriptbox — 79x27
[root@scriptbox ~]# ssh vagrant@web01
The authenticity of host 'web01 (192.168.56.27)' can't be established.
ED25519 key fingerprint is SHA256:vwfKXUjWMPyTaSQG5jWAY4XPnR3yiS0Dzy89G1m/kxs.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'web01' (ED25519) to the list of known hosts.
[vagrant@web01's password:
Last login: Wed Sep 11 23:07:51 2024 from 172.16.196.1
[[vagrant@web01 ~]$ hostname
web01
[[vagrant@web01 ~]$ logout
Connection to web01 closed.
[root@scriptbox ~]#
```

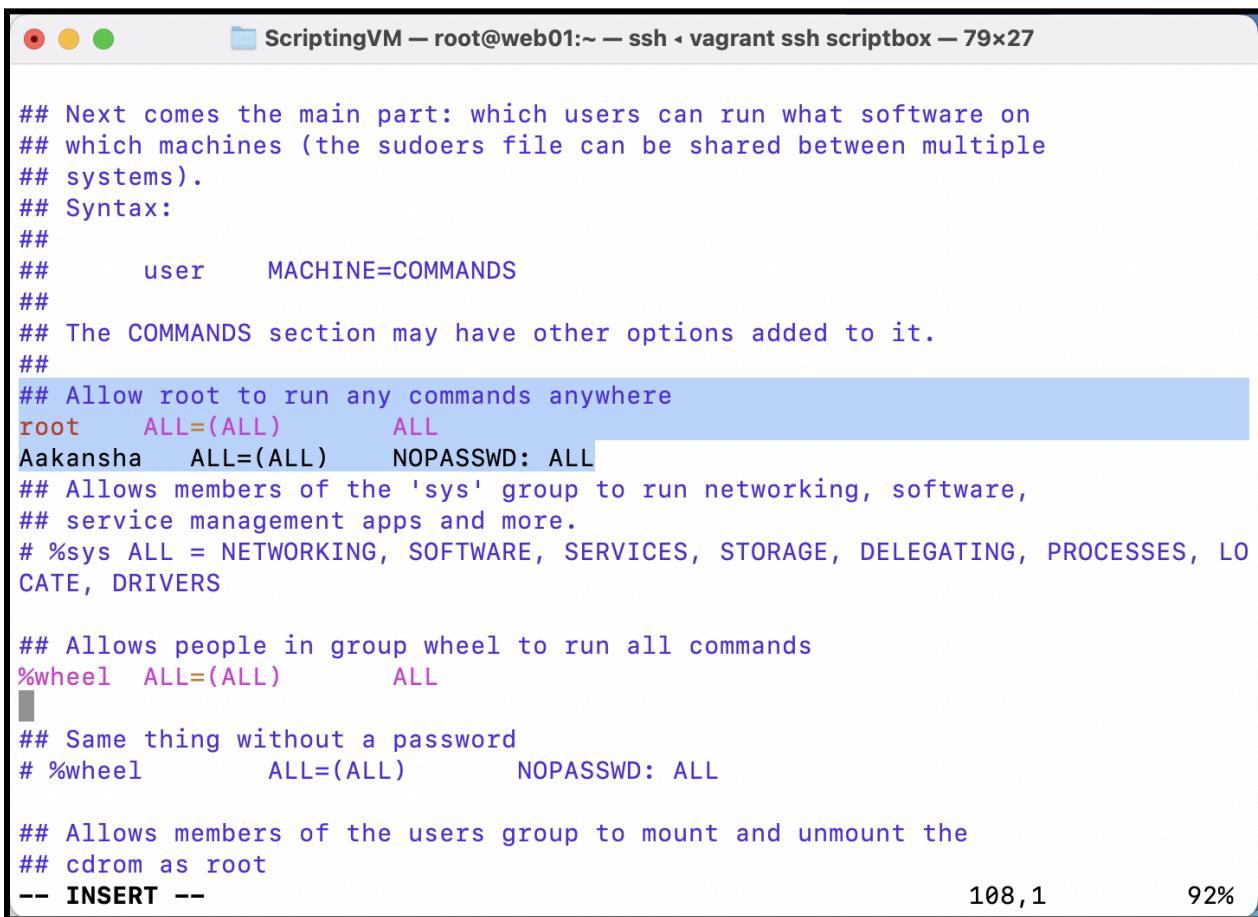
**The test was successful. Let's move to the next part.**

Now we will not be using the vagrant user. We will create our own user one by one in all the machines.

Also we will be using some system commands and install packages so we will be adding this user to the sudoers file.

```
ScriptingVM — root@scriptbox:~ — ssh -vagrant ssh scriptbox — 79x27
[root@scriptbox ~]# ssh web01
[root@web01's password:
[[root@web01 ~]$ hostname
web01
[[root@web01 ~]$ useradd Aakansha
[[root@web01 ~]$ passwd Aakansha
Changing password for user Aakansha.
[New password:
BAD PASSWORD: The password contains the user name in some form
[Retype new password:
passwd: all authentication tokens updated successfully.
[[root@web01 ~]$ visudo
[[root@web01 ~]$ exit
logout
Connection to web01 closed.
[root@scriptbox ~]#
```

## Make the change in the file and save and quit



```
## Next comes the main part: which users can run what software on
## which machines (the sudoers file can be shared between multiple
## systems).
## Syntax:
##
##       user      MACHINE=COMMANDS
##
## The COMMANDS section may have other options added to it.
##
## Allow root to run any commands anywhere
root    ALL=(ALL)      ALL
Aakansha  ALL=(ALL)      NOPASSWD: ALL
## Allows members of the 'sys' group to run networking, software,
## service management apps and more.
# %sys  ALL = NETWORKING, SOFTWARE, SERVICES, STORAGE, DELEGATING, PROCESSES, LOCATE, DRIVERS

## Allows people in group wheel to run all commands
%wheel  ALL=(ALL)      ALL
##
## Same thing without a password
# %wheel      ALL=(ALL)      NOPASSWD: ALL

## Allows members of the users group to mount and umount the
## cdrom as root
-- INSERT --                                         108,1          92%
```

Repeat the same for the other VMS:

- Web02
- Web03

**Note: If you are not able to enter web03 from scriptbox then exit out of it. Then:**

- **Do vagrant ssh web03**
- **After getting into the vm open vim /etc/ssh/sshd\_config**
- **Find PasswordAuthentication**
- **If it's no change it to yes**
  - **PasswordAuthentication yes**

## Now let's execute our command remotely

```
[vagrant@scriptbox ~]$ ssh Aakansha@web01 uptime
The authenticity of host 'web01 (192.168.56.27)' can't be established.
ED25519 key fingerprint is SHA256:vwfKXUjWMPyTaSQG5jWAY4XPnR3yiSODzy89G1m/kxs.
This key is not known by any other names
[Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'web01' (ED25519) to the list of known hosts.
[Aakansha@web01's password:
 00:15:07 up  9:44,  0 users,  load average: 0.00, 0.00, 0.00
[vagrant@scriptbox ~]$
```

We have executed the command from our scriptbox in web01.

**Problem:** Asking us to authenticate every time using a password.

**Solution:** SSH Key Exchange

```
[root@scriptbox ~]# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa
Your public key has been saved in /root/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:oYmwZLmX08gahkSLj9N6/SXWSDt4NIPz0MjTsJmZrH4 root@scriptbox
The key's randomart image is:
+---[RSA 3072]---+
| . .
| . = . .
| oo.* # o .
| .oo ^ X S
| o= + X *
| =.*.+ O o
| ..*. E+ +
| o.... ...
+---[SHA256]---+
[root@scriptbox ~]#
```

This is the private key: [/root/.ssh/id\\_rsa](/root/.ssh/id_rsa)

This is the public key: [/root/.ssh/id\\_rsa.pub](/root/.ssh/id_rsa.pub)

( This is like a lock and key, so the public key acts as the lock and private key as the key to the lock.

We are going to put this lock on web01, web02, web03 and we will have this key to open them. )

## Now let's lock the VMs with the public key

```
[root@scriptbox ~]# ssh-copy-id Aakansha@web01
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
Aakansha@web01's password:

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'Aakansha@web01'"
and check to make sure that only the key(s) you wanted were added.
```

Note: Repeat the same for the other VMs

## Now let's test it

```
ScriptingVM — root@scriptbox:~ — ssh -vagrant ssh scriptbox — 75x27

[vagrant@scriptbox ~]$ ssh Aakansha@web01 uptime
01:07:53 up 10:35, 0 users, load average: 0.04, 0.01, 0.00
[vagrant@scriptbox ~]$ pwd
/home/vagrant
[vagrant@scriptbox ~]$ sudo -i
[root@scriptbox ~]# pwd
/root
[root@scriptbox ~]# ls -a
.              .bash_history  .bashrc   original-ks.cfg  .tcshrc
..             .bash_logout    .cshrc    scripts        .viminfo
anaconda-ks.cfg .bash_profile .lessht  .ssh
[root@scriptbox ~]# ls .ssh/
id_rsa  id_rsa.pub  known_hosts  known_hosts.old
[root@scriptbox ~]#
```

It's not asking for the password now.

What's Happening is everytime we login it's using the **id\_rsa** key to login and execute our command.



## Practical Application of RCE

We were running our scripts in the scriptbox, now let's make a framework through which we will run our scripts on our other VMs.

```
ScriptingVM — root@scriptbox:~/scripts/remote_websetup — ssh -vagrant ssh scriptbox — 96x27
[[root@scriptbox ~]]# ls
anaconda-ks.cfg original-ks.cfg scripts
[[root@scriptbox ~]]# cd scripts/
[[root@scriptbox scripts]]# ls
10_ifelif.sh 15_while.sh      4_args.sh      9_if1.sh
11_monit.sh 16_while.sh      5_args_websetup.sh dismantl.e.sh
12_monit.sh 1_firstscript.sh  6_command_subs.sh
13_for.sh    2_websetup.sh   7_userInput.sh
14_for.sh    3_vars_websetup.sh 8if1.sh
[[root@scriptbox scripts]]# mkdir remote_websetup
[[root@scriptbox scripts]]# cd remote_websetup/
[[root@scriptbox remote_websetup]]# ls
[[root@scriptbox remote_websetup]]# vim remhosts
```

Now it's time to check the power of RCE

```
ScriptingVM — root@scriptbox:~/scripts/remote_websetup — ssh -vagrant ssh scriptbox — 101x30
[[root@scriptbox remote_websetup]]# ls
remhosts
[[root@scriptbox remote_websetup]]# for host in `cat remhosts`; do ssh Aakansha@$host hostname;done
web01
web02
[Aakansha@web03's password:
web03
[[root@scriptbox remote_websetup]]# for host in `cat remhosts`; do ssh Aakansha@$host uptime;done
 10:00:15 up 13:07,  0 users,  load average: 0.00, 0.00, 0.00
 10:00:15 up 13:07,  0 users,  load average: 0.01, 0.01, 0.00
[Aakansha@web03's password:
 17:00:20 up 13:06,  0 users,  load average: 0.00, 0.00, 0.00
[[root@scriptbox remote_websetup]]# for host in `cat remhosts`; do ssh Aakansha@$host pwd;done
/home/Aakansha
/home/Aakansha
[Aakansha@web03's password:
/home/Aakansha
[[root@scriptbox remote_websetup]]# for host in `cat remhosts`; do ssh Aakansha@$host free -m;done
              total        used         free      shared  buff/cache   available
Mem:       1950         221        1360          0        368       1704
Swap:      1949          0        1949
              total        used         free      shared  buff/cache   available
Mem:       1950         222        1356          0        372       1703
Swap:      1949          0        1949
[Aakansha@web03's password:
              total        used         free      shared  buff/cache   available
Mem:        719         166         100          1        453       458
Swap:      1437          17        1420
[[root@scriptbox remote_websetup]]#
```

We are able to check the hostname, uptime, working directory, and available memory on all servers remotely without needing to physically log into each server individually.

**Let's modify our websetup script which was applicable to centos.  
Now we will make it applicable to other OS.**

```
ScriptingVM — root@scriptbox:~/scripts/remote_websetup — ssh -v vagrant ssh scriptbox — 101x32
[[root@scriptbox remote_websetup]# ls ../
10_ifelif.sh 13_for.sh 16_while.sh 3_vars_websetup.sh 6_command_subs.sh 9_if1.sh
11_monit.sh 14_for.sh 1_firstscript.sh 4_args.sh 7_userInput.sh dismantle.sh
12_monit.sh 15_while.sh 2_websetup.sh 5_args_websetup.sh 8if1.sh remote_websetup
[[root@scriptbox remote_websetup]# cp ..../3_vars_websetup.sh multios_websetup.sh
[[root@scriptbox remote_websetup]# ls
multios_websetup.sh remhosts
[[root@scriptbox remote_websetup]# vim multios_websetup.sh
[[root@scriptbox remote_websetup]# ./multios_websetup.sh
Running Setup on CentOS
#####
Installing packages.
#####
Start & Enable HTTPD Service
#####
Starting Artifact Deployment
#####

--2024-09-12 10:22:53-- https://www.tooplate.com/zip-templates/2098_health.zip
Resolving www.tooplate.com (www.tooplate.com)... 72.52.176.250
Connecting to www.tooplate.com (www.tooplate.com)|72.52.176.250|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1521593 (1.5M) [application/zip]
Saving to: '2098_health.zip'

2098_health.zip      100%[=====] 1.45M 363KB/s in 4.1s
2024-09-12 10:23:00 (363 KB/s) - '2098_health.zip' saved [1521593/1521593]
```

**We have checked the script locally and it's working well.**

**Now let's push it to our VM's.**

```
ScriptingVM — root@scriptbox:~/scripts/remote_websetup — ssh -v vagrant ssh scriptbox — 101x32
[[root@scriptbox remote_websetup]# ls
multios_websetup.sh remhosts
[[root@scriptbox remote_websetup]# echo "testfile" > testfile.txt
[[root@scriptbox remote_websetup]# ls
multios_websetup.sh remhosts testfile.txt
[[root@scriptbox remote_websetup]# scp testfile.txt Aakansha@web01:/tmp/
testfile.txt                                         100%   9    10.8KB/s  00:00
[[root@scriptbox remote_websetup]#
[[root@scriptbox remote_websetup]#
```

**The `scp` (Secure Copy Protocol) command is used to securely transfer files between a local host and a remote host, or between two remote hosts. It uses SSH to ensure that data is transferred securely.**

In the above example, **scp** is used to securely copy a file (**testfile.txt**) from the local machine to a remote host (**web01**), specifically to the **/tmp/** directory on that host.

Now, we will create a script to deploy our multios websetup script to our VMs

```
ScriptingVM — root@scriptbox:~/scripts/remote_websetup — ssh - vagrant ssh scriptbox — 80x23
[root@scriptbox remote_websetup]# vim webdeploy.sh
[root@scriptbox remote_websetup]# chmod +x webdeploy.sh
[root@scriptbox remote_websetup]# ls
multios_websetup.sh  remhosts  testfile.txt  webdeploy.sh
[root@scriptbox remote_websetup]# ./webdeploy.sh

#####
Connecting to web01
Pushing Script to web01
multios_websetup.sh          100% 3143      4.2MB/s   00:00
Executing Script on web01
Running Setup on CentOS
#####
Installing packages.
#####

#####
Start & Enable HTTPD Service
#####
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr
/lib/systemd/system/httpd.service.

#####

```

```
ScriptingVM — root@scriptbox:~/scripts/remote_websetup — ssh - vagrant ssh scriptbox — 80x23
js
news-detail.html
#####

#####
Connecting to web02
Pushing Script to web02
multios_websetup.sh          100% 3143      2.8MB/s   00:00
Executing Script on web02
Running Setup on CentOS
#####
Installing packages.
#####


```

Working well with all the VMs. Have doubts ...Go check for yourself 😊

## PROJECT CONCLUSION

**Remote Command Execution (RCE) demonstrated the power of managing and controlling multiple systems from a single machine.**

In the above examples, we have executed commands on multiple remote servers (`web01`, `web02`, `web03`) from the local machine using SSH.

**Let's Summarize the Key Powers of RCE:**

- 1. Centralized Control:** The admin can log into multiple machines (`web01`, `web02`, `web03`) using a single script and perform administrative tasks like checking hostnames, uptime, or available memory.
- 2. Automation:** With a simple loop, commands like `hostname`, `uptime`, and `free -m` are executed across multiple servers in sequence. This allows for quick monitoring and management, saving time compared to manually logging into each system.
- 3. Consistency:** Commands executed via RCE ensure consistent results across systems, helping administrators standardize configurations and collect data uniformly.
- 4. Security:** Though RCE provides great power, it requires proper authentication. Each time you connect to a server, SSH ensures a secure connection, prompting for a password if necessary.



😊 ...HAPPY LEARNING... 😊