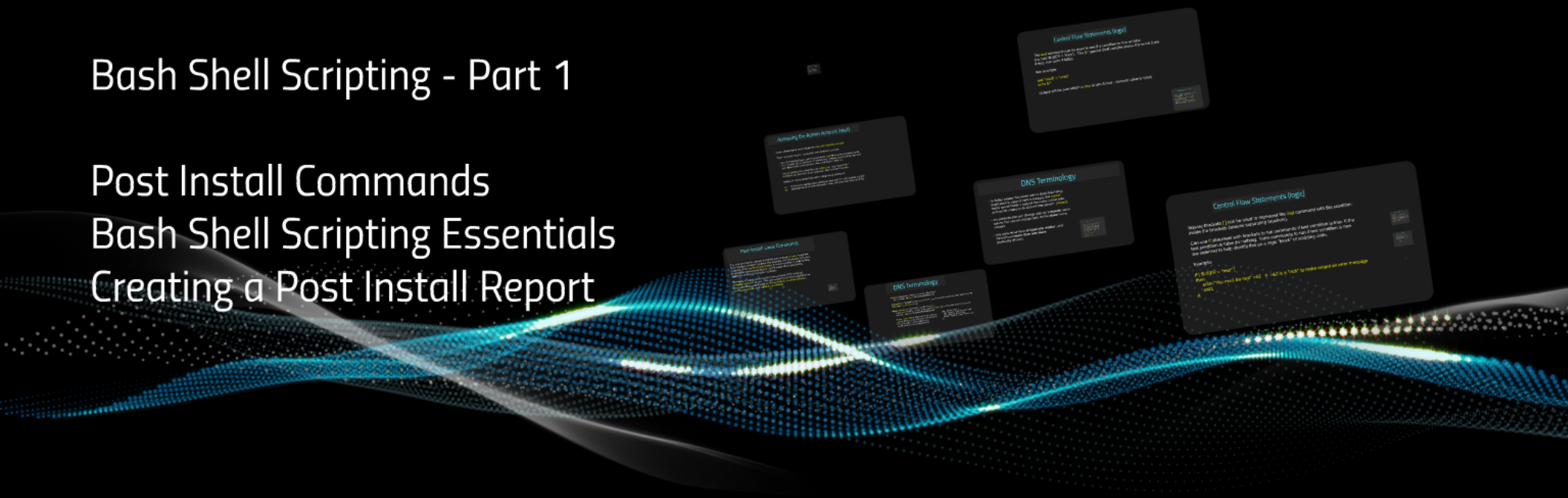
OPS235

Bash Shell Scripting - Part 1

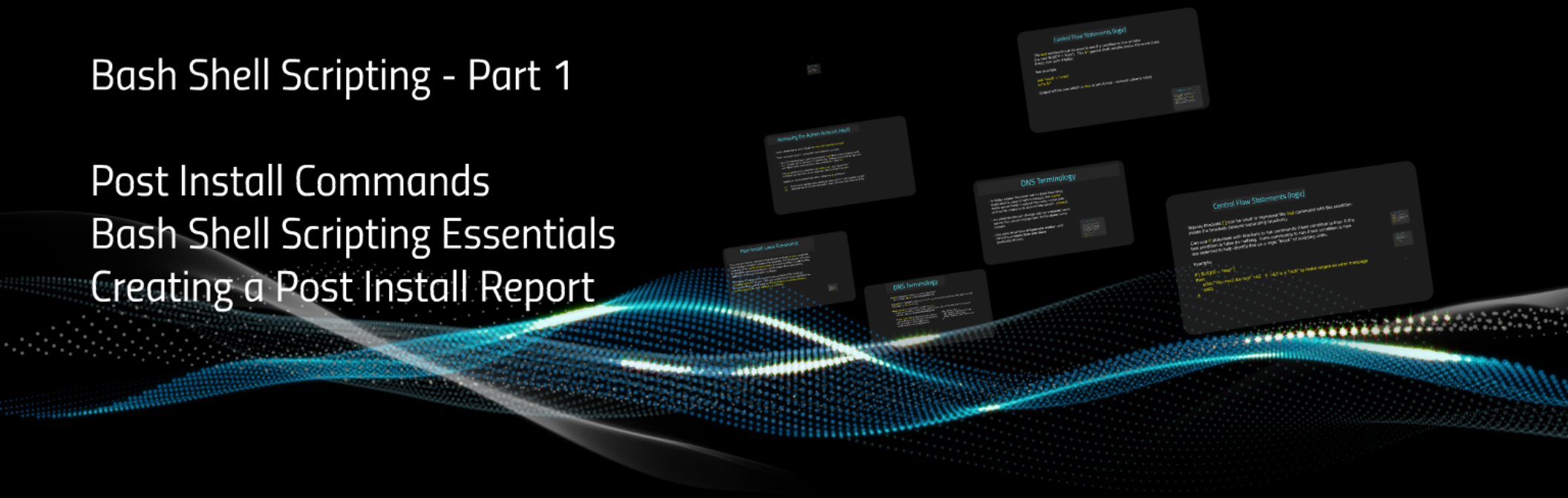
Post Install Commands
Bash Shell Scripting Essentials
Creating a Post Install Report



OPS235

Bash Shell Scripting - Part 1

Post Install Commands
Bash Shell Scripting Essentials
Creating a Post Install Report



Post-Install Linux Commands

It is very common for System Administrators to keep records regarding their installed computer systems. For example, it is necessary to have a record of all the hardware information for each machine in order to help fix computer hardware problems, and to assist when purchasing additional consistent computer hardware.

Therefore, it makes sense to also have a record of the installed computer software as well. This can contain information regarding the Linux operating system such as kernel version, installed software, running processes, and network connectivity.

Post Install Linux Commands

Below are some common LInux commands to provide information for a newly install Linux server.

Basic Linux OS information such as kernel version, host-name of Linux server, and all processes that are running on the system after installation:
uname -rv, hostname, ps -ef

Obtaining number of installed packages in the rpm database:
rpm -q -a | wc -l, rpm -q -a -l | wc -l, rpm -q -l gedit | wc -l
(Obtain number of installed packages in the rpm database. Option -q is to "query" information, option -a means for all installed packages, option -l means all files installed as opposed to just the application)

Obtain network connectivity confirmation including: IP ADDRESS, Netmask, routing (default gateway), and the default Domain Name Server:
ifconfig, route -n, nslookup

Recording Installed System Information

It is important when issuing this commands to record information (for example IP ADDRESS) for your c7host machine, and and other virtual machines that you will be creating in lab2.

Why?!?

One reason is that throughout the labs to test network connectivity or perform tasks on your VMs remotely from your host via SSH you will need to know your machines IP ADDRESSES for at least labs 1 - 5.

Another reason is that you will be required to issue or note the IP Address of any machine in order to run a lab-checking script. You will be using the same regular user IDs and passwords for your hostmachine and your VMs.

Accessing the Admin Account (root)

Many administrative tasks require the root administrative account.

There are many ways to access this administration account:

- At a CLI terminal login, enter the username root (then enter root password)
- You can also run a command to "switch user" temporarily and when you exit, you return to the previous user. The command is called: su

  The su command is referred to as switch user (not "superuser")
  If there is no username as an argument, then it defaults to root.

  Below are some subtle tricks when using the su command:

  su Remains in regular user's directory, does not run root's startup script(s).
  su - Changes to root's home directory (/root) and runs root's start script(s).

Shell Scripting

You may have learned about creating and running Bash Shell Scripts in your ULI101 course. Shell scripts help Linux users and system administrators to automate repetitive tasks in order to become more efficient and to help them save time. You will be reviewing and building a basic Bash Shell script to generate information reports for your newly-installed Linux host machine.

If you require additional practice in creating shell scripts , run the following command in your Matrix account:

/home/murray.saul/scripting-1

She-Bang Line: **#!/bin/bash**

Shell scripts have evolved over the past 40 years. To avoid running a newer shell script on an older shell, it is recommended to force running the shell script in the correct shell.

In order to do this, on the first line at the beginning of the shell script, you add the #! special command followed by the pathname of a shell to run the shell script.

In other words, # as in "shhhh" (i.e. a comment) and ! is referred to as "bang" (i.e. run a commmand). To see how ! works, run the history command and then enter ! following by a command number from the output of the history command and press ENTER to see what happens.

If there is no shell on that machine, the shell script will not run as a precaution. The Linux administrator should know how to make a fix to the shell script if required).

Variables

There are 3 types of variables that can be used in shell scripting:

ENVIRONMENT (eg. $USER). Useful for using system information

User-defined ($varName). Can create and use your own custom variables
Can be set with equal sign (eg. var="Murray Saul"), or can use the read command to store variables (eg. read -p "enter number" userNumber)

Positional parameters (eg. $1, $2... containing arguments after shell script or by using set command (eg. set $(ls)).

Using a dollar sign ($) in front of variable expands the value assigned.
Example:

```
read -p "enter your age (in years): " yourAge
echo "You are $yourAge years old"
```

Command Substitution

Command substitution is a useful method to expand output from a command to be used as an argument for another command.

Here is an example: If your issued the command file $(ls), the shell would notice $(ls) and IMMEDIATELY expand it - which in this case, would issue the ls command and list all non-hidden filenames in the current directory. But in this case, those file names would be come arguments (each separated by a space) AFTER the file command which would describe the file-type of each of those files.

Examples (try to guess what each one does):

```
set $(ls);echo $#;echo $*
```

```
echo "hostname: $(hostname)"
```

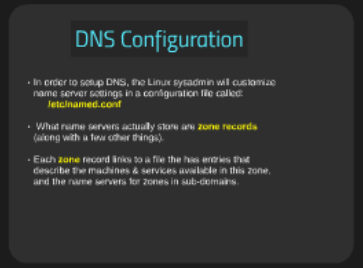
Control Flow Statements (logic)

The test command can be used to see if a condition is true or false (i.e. test $USER = "root") . The $? special shell variable stores the result (zero if true, non-zero if false).

For example:

```
test "word" = "word"
echo $?
```

(output will be zero which is true in Unix/Linux - nonzero value is false)



Control Flow Statements (logic)

Square brackets [ ] can be used to represent the test command with the condition inside the brackets (spaces separating brackets).

Can use if statement with brackets to run commands if test condition is true. If the test condition is false do nothing. Note commands to run if test condition is true are indented to help identify this as a logic "block" of scripting code.

Example:

```
if [ $USER = "root" ]
then
    echo "You must be root" >&2   #  >&2 is a "trick" to make output an error message
    exit1
fi
```

Control Flow Statements (logic)

The if-else statement runs commands if the condition tested is true and runs a different set of commands if the condition tested is false. The if-elif-else statement tests a condition and keeps retesting a series of conditions if the previous tests were false. If all tests are false, then the else statement is used to run the last alternate set of commands.

For number comparison when testing conditions: use: -gt , -ge , -lt , -le , -eq , -ne

Examples:

```
if [ $age -gt 65 ]
then
 echo "retire"
else
 echo "don't retire"
fi
```

```
if [ $grade -gt 79 ]
then
   echo "You get Good Mark"
elif [ $grade -gt 49 ]
then
   echo "You pass"
else
   echo "You fail"
fi
```

Control Flow Statements (logic)

For testing for file information, you can use the -d option to test if directory pathname exists, and -f option if the file pathname exists. You can use ! for negation.

Examples:

```
if [ -d directory-pathname ]
then
echo "directory exists"
fi

if [ ! - f file-pathname ]
then
echo "File does not exist"
fi
```