Create the script: create-and-launch-rocket.sh

# Step 0 – define the mission name as a variable

mission\_name=lunar-mission

#Step 1 – create directory for mission

mkdir [mission-name] OR $mission\_name

#Step 2 – do the mission

rocket-add [mission name] OR $mission\_name

rocket-start-power [mission name] OR $mission\_name

rocket-internal-power [mission name] OR $mission\_name

rocket-start-sequence [mission name] OR $mission\_name

rocket-start-engine [mission name] OR $mission\_name

rocket-lift-off [mission name] OR $mission\_name

#Step 3 – verify the mission

rocket-status [mission name] OR $mission\_name

rocket-ls – list all rockets

Execute the script:

A. bash create-and-launch-rocket.sh

B. create-and-launch-rocket

For B to work we have to configure the script as a command you have to add it as a command:

a) echo $PATH, which has the output: /usr/local..................:/bin

b) export PATH=/usr/local..................:/bin:/home/michael

c) export PATH=$PATH:/home/michael

To see the location of a command: which create-and-launch-rocket

Make sure that you have the correctly permissions to EXECUTE: ls –l /[path]/[file]

Grant permissions to execute: chmod +x /home/michael/create-and-launch-rocket

[Shell-Scripting-kodekloud.pdf](https://kodekloud.com/wp-content/uploads/2022/01/Shell-Scripting-kodekloud.pdf)

1. Use the deck for referring to concepts and syntax only
2. Do not copy and paste commands or script from the deck as it may have hidden characters.

For copying commands, code snippets and answers to lab questions please refer to this Github Repository:

<https://github.com/kodekloudhub/shell-scripting-for-beginners-course>

(Note: this is still in the works and will be completed soon)

Join the KodeKloud Community and post your queries regarding courses at this channel:

[https://community.kodekloud.com/c/shell-scripting](https://community.kodekloud.com/c/shell-scripting?_gl=1*1kmenud*_ga*MTM1OTMxNTM4NS4xNzAzNjgwOTQy*_ga_LYL47LCHPW*MTcwNDg3Nzg1MC4zMy4xLjE3MDQ4ODIxODAuNTkuMC4w#_ga=2.219589649.968433884.1704465314-1359315385.1703680942)

Second script example:

# Step 1 - Create the following directories under /home/bob/countries - USA, UK, India

mkdir /home/bob/countries

cd /home/bob/countries

mkdir USA UK India

# Step 2 - Create a file under each directory by the name capital.txt and add the capital cities name in the file - Washington, D.C, London, New Delhi

echo "Washington, D.C" > USA/capital.txt

echo "London" > UK/capital.txt

echo "New Delhi" > India/capital.txt

# Step 3 - Print uptime of the system

uptime

You could use variable to store the result of another command

Print the status of the launch as launching, success or failed: rocket-status lunar-mission

Store the output of the rocket-status command in a variable:

rocket\_status=$(rocket-status $mission\_name)

Print a message to the screen using the echo command and replace the variable to display the status of the launch:

echo „Status of launch: $rocket\_status”

Ways of passing in input to the script:

A. By using a command line argument of $1: If you have more missions (mars-mission/jupyter-mission etc) to run the script for them would be easier to do directly from the command itself

create-and-launch-rocket(this is argument $0) mars-mission(this is argument $1)

you have to modify in the variable declaration in the script: mission\_name=$1

The script would look like:

# Step 0 – define the mission name as a variable

mission\_name=$1

#Step 1 – create directory for mission

mkdir $mission\_name

#Step 2 – do the mission

rocket-add $mission\_name

rocket-start-power $mission\_name

rocket-internal-power $mission\_name

rocket-start-sequence $mission\_name

rocket-start-engine $mission\_name

rocket-lift-off $mission\_name

#Step 3 – verify the mission

rocket-status $mission\_name

echo „Status of launch: $rocket\_status”

B. By input when prompted – using the read statemend – not that preferred – only for programs that require manual intervention (before deleting a file or rebooting a server)

create-and-launch-rocket(it will prompt you to enter a value)

For this:

# Step 0 – define the mission name as a variable

read –p “Enter mission name:” mission\_name

#Step 1 – create directory for mission

mkdir $mission\_name

#Step 2 – do the mission

rocket-add $mission\_name

rocket-start-power $mission\_name

rocket-internal-power $mission\_name

rocket-start-sequence $mission\_name

rocket-start-engine $mission\_name

rocket-lift-off $mission\_name

#Step 3 – verify the mission

rocket-status $mission\_name

echo „Status of launch: $rocket\_status”

Script example:

# This script creates a backup of a given file by creating a copy as bkp

# For example some-file is backed up as some-file\_bkp

set -e

file\_name="some-file"

cp $file\_name ${file\_name}\_bkp

echo "Done"

**- Arithmetic Operations**

> Through the expression command: expr 3 + 6 ; expr 6 – 3; expr 6 / 3;

expr \\* 3(inmultire) – due to the fact that \* symbol is reserved regex character in shell – it typically means everything – so we must escape the star symbol using a back slash \

It could be used with variables as well:

A=6

B=3

expr $A + $B; and so on.....

> Through double paranthesses: echo $(( A + B )); ...; echo $(( A\*B ))

This method does not necessarily needs spaces like the expression command and it can be used with \* without \

There could be used program C style manipulation of variables:

echo $(( ++A )) – increment a variable

echo $(( --A )) – decrement a variable

echo $(( A++) – value of the variable is returned first and then incremented

echo $(( A--)) – value of the variable is returned first and then decremented

> Through bc – basic calculator – used when you have float values:

A=10

B=3

echo $A / $B | bc –l

===Another example script===

price=$(( $1 \* $2 ))

echo "The total price for items is ${price} dollars"

===Another example script which accepts exactly 3 command-line arguments and calculate the average and prints the result as float===

**num1=$1**

**num2=$2**

**num3=$3**

**sum=$(( num1 + num2 + num3 ))**

**average=$(echo "$sum / 3" | bc -l)**

**echo $average**

**Conditional logic:**

# Step 0 – define the mission name as a variable

mission\_name=$1

#Step 1 – create directory for mission

mkdir $mission\_name

#Step 2 – do the mission

rocket-add $mission\_name

rocket-start-power $mission\_name

rocket-internal-power $mission\_name

rocket-start-sequence $mission\_name

rocket-start-engine $mission\_name

rocket-lift-off $mission\_name

#Step 3 – verify the mission

rocket\_status=$(rocket-status $mission\_name)

echo „Status of launch: $rocket\_status”

**# Step 4 – if the launch has failed, check why it failed:**

**if [ $rocket\_status = „failed” ]**

**then**

**rocket-debug $mission\_name**

**elif [ $rocket\_status = „success” ]**

**echo „This is successful”**

**else**

**echo „The state is not failer or successfull”**

**fi**

Logica:

daca(if) [ conditia asta este indeplinita]

atunci(then)

fa asta

daca nu e indeplinita prima conditie din if, poate e a doua(elif) [ conditia 2 este indeplinita ]

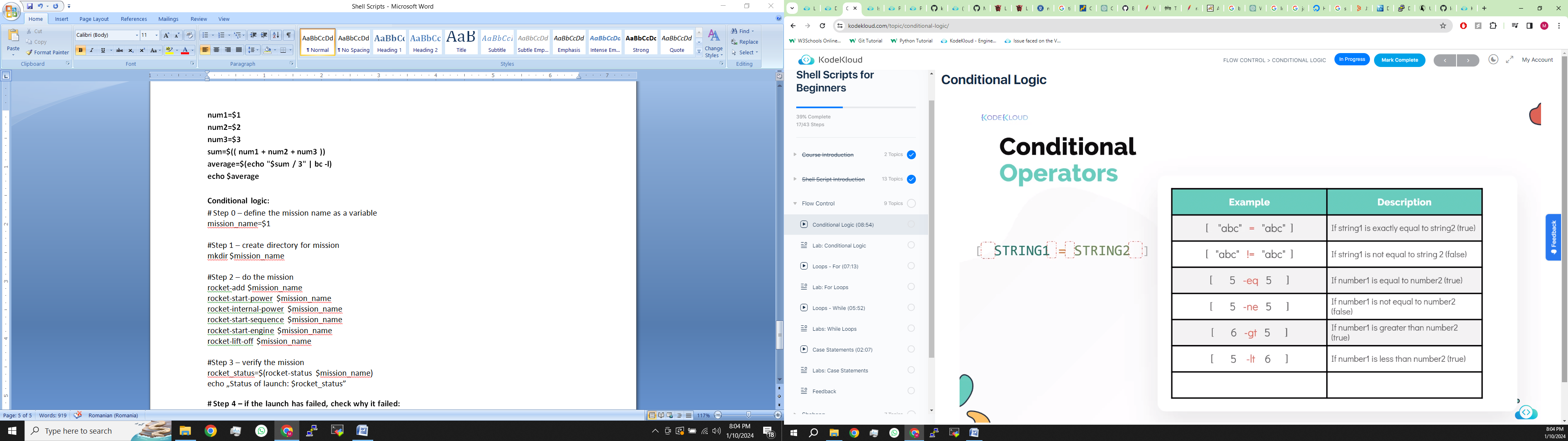
atunci(then)

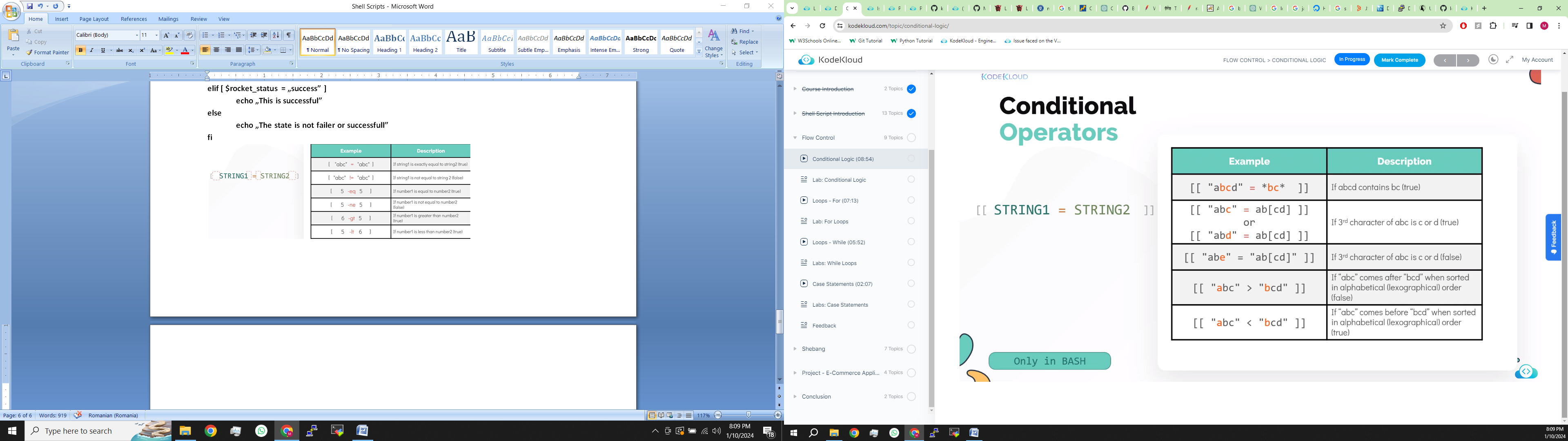
fa asta

daca nici a nici b conditii nu sunt indeplinite (else)

fa asta

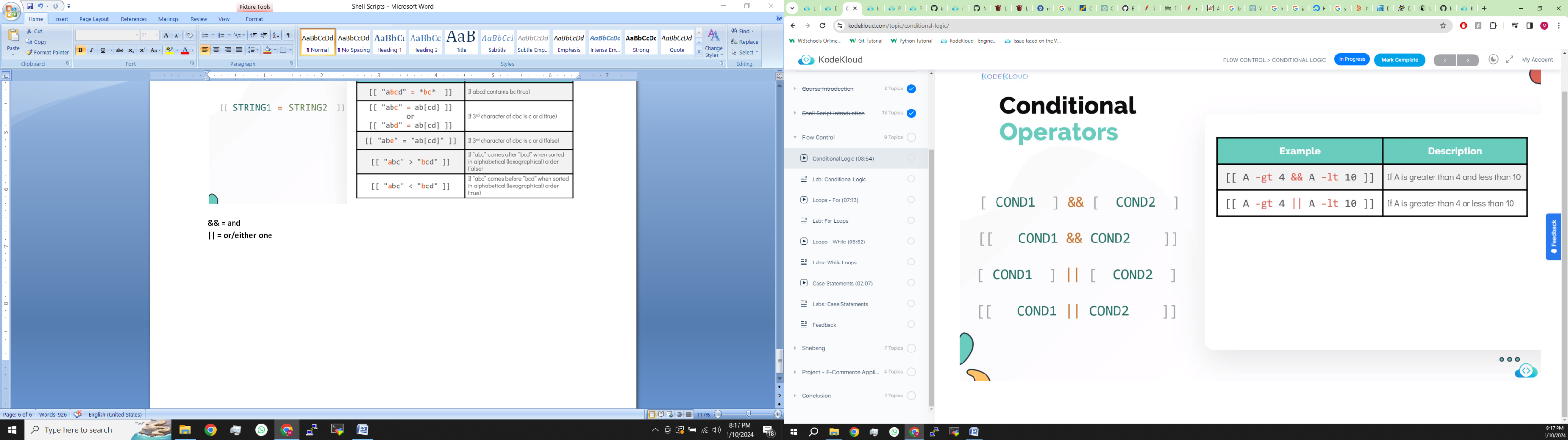
fi

****

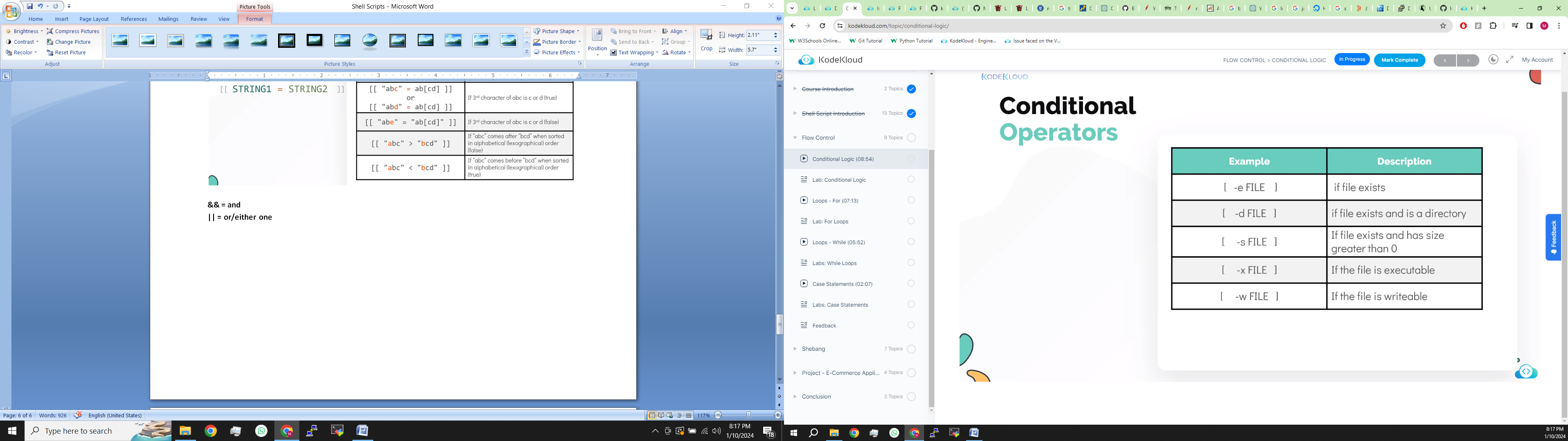
****

**&& = and**

**|| = or/either one**

****

**File level operators:**

****

**Example:**print the line Directory exists if the directory /home/bob/caleston exists. If not, it should print Directory not found

if [ -d "/home/bob/caleston" ]

then

echo "Directory exists"

else

echo "Directory not found"

fi

**Example:** Develop a shell script /home/bob/print-month-name.sh that accepts the number of a month as input and prints the name of the respective month. eg ./print-month-name.sh 1 should print January and ./print-month-name.sh 5 should print May. Also keep these in mind.

1. The script must accept a month number as a command line argument.

2. If a month number is not provided as command line argument, the script must exit with the message No month number given.

3. The script must not accept a value other than 1 to 12. If not the script must exit with the error Invalid month number given.

Script:

month\_number=$1

if [ -z $month\_number ]

then

echo "No month number given. Please enter a month number as a command line argument."

echo "eg: ./print-month-number 5"

exit

fi

if [[ $month\_number -lt 1 ]] || [[ $month\_number -gt 12 ]]

then

echo "Invalid month number given. Please enter a valin number - 1 to 12."

exit

fi

if [ $month\_number -eq 1 ]

then

echo "January"

...................................

elif [ $month\_number -eq 12]

then

echo "December"

fi

**For loop, use:**

**- Execute a command or a set o commands many times**

**- Iterate through files**

**- Iterage through lines within a file**

**- Iterage through lines the output of a command**

**– if you would like to run the same command multiple times**

**For example, we want the create-and-launch-rocket command to be executed for more than one mission:**

**for mission in lunar-mission earth-mission mars-mission**

**do**

**create-and-launch-rocket $mission**

**done**

**Or if we hae a large amount of missions build a file mission-names.txt where you write a list of the missions and:**

**for mission in `cat mission-names.txt`/$(cat mission-names.txt)**

**do**

**create-and-launch-rocket $mission**

**done**

**Or if we want to launch a limited number of missions:**

**for mission in 1 2 3 4 5 6/{0..100}**

**do**

**create-and-launch-rocket mission-$mission**

**done**

**Real example:**

**1.**

**for file in $(ls)**

**do**

**echo Line count of $file is $(cat $file | wc-l)**

**done**

**2.**

**for package in $(cat install-packages.txt)**

**do**

**sudo apt-get –y install $package**

**done**

**3.**

**for server in $(cat servers.txt)**

**do**

**ssh $server „uptime”**

**done**

**Real script example: We have a few different applications running on this system. The list of applications are stored at /tmp/assets/apps.txt file. Each application has it's logs stored in /var/log/apps directory under a file with its name. Check it out!**

**A simple version of the script has been developed for you /home/bob/count-requests.sh that inspects the log file of an application and prints the number of GET, POST, and DELETE requests. Update the script to use a for loop to read the list of applications from the apps.txt file and count number of requests for each application and display it in a tabular format like this.**

echo -e " Log name \t GET \t POST \t DELETE "

echo -e "------------------------------------------------------------"

**for** app **in** $(cat /tmp/assets/apps.txt)

**do**

get\_requests=$(cat /var/log/apps/${app}\_app.log | grep "GET" | wc -l)

post\_requests=$(cat /var/log/apps/${app}\_app.log | grep "POST" | wc -l)

delete\_requests=$(cat /var/log/apps/${app}\_app.log | grep "DELETE" | wc -l)

echo -e " ${app} \t ${get\_requests} \t ${post\_requests} \t ${delete\_requests}"

**done**

**Real example 2 : We have some images under the directory /home/bob/images. Develop a script /home/bob/rename-images.sh to rename all files within the images folder that has extension jpeg to jpg. A file with any other extension should remain the same.**

**Tip: Use a for loop to iterate over the files within /home/bob/images**

**Tip: Use an if conditional to check if the file extension is jpeg.**

**Tip: Use mv to rename a file.**

**To replace jpeg to jpg in a filename use echo user1.jpeg | sed 's/jpeg/jpg/g'.**

**for** file **in** $(ls images)

**do**

**if** [[ $file = \*.jpeg ]]

**then**

new\_name=$(echo $file| sed 's/jpeg/jpg/g')

mv images/$file images/$new\_name

**fi**

**done**

**While loops usage:**

**- Execute a comand or set of commands multiple times but you are not sure how many times (like in the for loop)**

**- Execute a command or a set of commands until a specific condition occurs (if the launch is successful or failed)**

**- Create infinite loops**

**- Menu drive programs**

**The while loop takes a conditional statement and a set of commands, works like the for loop but executes the loop as long as the condition is true.**

**Example: check the status of the rocket – until nou we used if which is a conditional, but if the state is blocked into launching while is better to use**

The script would look like:

mission\_name=$1

mkdir $mission\_name

rocket-add $mission\_name

rocket-start-power $mission\_name

rocket-internal-power $mission\_name

rocket-start-sequence $mission\_name

rocket-start-engine $mission\_name

rocket-lift-off $mission\_name

rocket\_status=$(rocket-status $mission\_name)

echo "The status of launch is $rocket\_status"

while [ $rocket\_status = "launching" ]

do

sleep 2

rocket\_status=$(rocket-status $mission\_name)

done

if [ $rocket\_status = "failed" ]

then

rocket-debug

fi

Real life use cases:

while true

do

echo „1. Shutdown”

echo „2. Restart”

echo „3. Exit Menu”

read –p „Enter your choice: „ choice

if [ $choice –eq 1 ]

then

shutdown now

elif [ $choice –eq 2 ]

then

shutdown –r now

elif [ $choice –eq 3]

then

break

else

continue

fi

done

**Case statements - instead of using the if conditional like above:**

while true

do

echo „1. Shutdown”

echo „2. Restart”

echo „3. Exit Menu”

read –p „Enter your choice: „ choice

case $choice in

1) shutdown now ;;

2) shutdown –r now ;;

3) break ;;

\*) continue ;;

esac

done

**Shebang – used in order to gain compatibility among other shells: #!/bin/bash**

**Exit codes(return code)**

By default: EXIT STATUS = 0 -> SUCCESS

By default: EXIST STATUS > 0 -> FAILURE

To see the exit code of a command write after that command is given: echo $?

Functions are used to break up large cript that performs many different tasks”0

- Installing packages

- Adding users

- Configuring firewalls

- Perfor Mathematical calculations

The script would look like:

**# The function definition block:**

function launch-rocket () {

mission\_name=$1

mkdir $mission\_name

rocket-add $mission\_name

rocket-start-power $mission\_name

rocket-internal-power $mission\_name

rocket-start-sequence $mission\_name

rocket-start-engine $mission\_name

rocket-lift-off $mission\_name

rocket\_status=$(rocket-status $mission\_name)

echo "The status of launch is $rocket\_status"

while [ $rocket\_status = "launching" ]

do

sleep 2

rocket\_status=$(rocket-status $mission\_name)

done

if [ $rocket\_status = "failed" ]

then

rocket-debug $mission\_name

return 1 # if failed the exit code = 1 (there is another way – exit – but it exist the script)

fi

}

**# Main script that calls that function:**

launch-rocket lunar-mission

LUNAR\_STATUS\_CODE=$?

launch-rocket mars-mission

LUNAR\_STATUS\_CODE=$?