Bash Shell Script
Part 1

How to create and run a script in Linux?

Note: You can get all the files created and used in this lesson in our Github repository

```
# mkdir bash_scripting
# cd bash_scripting
# git clone https://github.com/utrains/utrains_shell_script.git
# cd utrains_shell_script
```

This is an introduction to Scripts in Linux

This is the first part of a series of 2 lessons. Make sure you go through both successively to better understand the concepts.

Let's get started!

Table of content

1

What is a script?

Definition and usage

What is a script?

- In **Linux**, you can **run commands or execute tasks manually,** but you can also use **scripts**
- A **script** is an executable file that contains a set of commands to accomplish a specific task
- It is a way to **automate tasks in Linux**
- The various commands used in a script are based on what they do to help us reach the **expected result**

Let's take some examples

What is a script?

Example 1: Create a script that will **display Hello World** on the screen

Solution: In other to solve the problem, there are few steps we need to follow:

- Open the terminal in your Visual studio code and connect to a centos 7 server launched in Lightsail
- Create the script: **# touch hello.sh**
- Give execute permission to the file: **# chmod +x hello.sh**
- Get into the file: **# vi hello.sh**
- Insert the commands to display **Hello World** and **save the file**
  - **echo "Hello World"** then **Esc**, **save and quit** the vim (**:wq**)
- Run the script: **./hello.sh**

What is a script?

Example 2: Create a script to **install some packages in Linux**

Solution: To solve this problem, you need to:

- **Know the system on which you are** (to be able to determine the command you will use to install packages: CentOS version, Ubuntu etc.)
- **Know the names of the various packages** you want to install

Now you can follow the steps to write your script

- Create the script: **# touch pkg.sh**
- Give execute permission to the file: **# chmod +x pkg.sh**
- Get into the file: **# vi pkg.sh**

What is a script?

- **Insert the commands** to install the various packages

  yum install finger -y

  yum install curl -y

  yum install zip -y

  yum install vim -y

- **Save and quit the vim** (**esc :wq**)
- **Run the script**: **./pkg.sh**

The packages will start installing successively till the end of the script (Remember you must have root access to install packages)

2

Useful tips in Scripting

The Shell used and the description in the script

The shell used

- In **Linux**, we have many shells and the users might use differents shells.
- To **avoid some errors** while running our scripts, **we need to tell the script in which shell it should run**
- That is done by **precising the shell at the first line of the script** as follows: *(examples for the **bash** and the **korn** shells)*

```
#!/bin/bash

#!/bin/ksh
```

Description in the script

- It is also very useful to put a small **description** at the beginning of the script to:
  - Explain the **task it accomplishes,**
  - Specify the **Author**
  - And the **creation date** of the script
- This is done in case in the future, **someone else might want to discuss, make a suggestion or understand the script**
- To do that, you just need to **add a # sign in front of those lines**
- When you put a **#** sign in front of a line, that **line is not interpreted as a command** but, the system considers it as a **comment in the script**

Description in the script

Let's modify the beginning of our **pkg.sh** script:

 # vi pkg.sh then go to the INSERT mode

```
#!/bin/bash


# Description: Script to install some packages
# Author: serge
# Date: January 2022
```

Save and Quit

Note:

The first line (#!/bin/bash) start with the # sign but it will be interpreted as a command

That is the only line starting with the # sign that will be interpreted as a command!!

3

Variables in scripting

What is a variable?

user defined variables

Variables in scripting

- A **variable** is a **string that carries a specific value.**
- To declare or define a variable, you give it a name and a value:

  VariableName=variableValue

Example: **a=serge**

- We can get the value just by calling the variable with the $ sign:

  $variableName or ${variableName}

Example: In the Terminal, declare a variable called **NAME** with the value **john** then **print its value on the screen**

Solution: **NAME=john** then print with the command **$ echo $NAME**

Variables in scripting

Let's implement the variable concept in a script

 # vim variables.sh

```
#!/bin/bash
# Description:
# Author:
# Date:
echo "Serge is a very hardworking guy"
echo "The manager says, he will hire Serge in his new team wherever he goes"
echo "Serge likes to show off because the manager loves him "
echo "Serge will always be available when they need him"
```

Save and Quit
Note: Type the text to avoid errors with "

Variables in scripting

- Give the **execution permission**: **# chmod +x variables.sh**
- Run the script: **# ./variables.sh**
- Now if we want to change the name **Serge** to another name in this script, we will:
  - Either **look everywhere where there is Serge and replace it** with the new name (this can be very difficult if you are working with a massive code and you wanna change an item in there)
  - Or **store the name in a variable and call it whenever needed** (if that name has to change, only one line will be affected)

Variables in scripting

Let's modify our code:

```
# vim variables.sh
    #!/bin/bash
    # Description:
    # Author:
    # Date:
    NAME=Serge
    echo "$NAME is a very hardworking guy"
    echo "The manager says, he will hire $NAME in his new team wherever he goes"
    echo "$NAME likes to show off because the manager loves him "
    echo "$NAME will always be available when they need him"
```

Save and Quit
Run the Script and you will get the same output as before

Variables in scripting

Now you can change the value of the variable **NAME** as you wish.

 # vim variables.sh

```
#!/bin/bash
# Description:
# Author:
# Date:
NAME=Ellys
echo "$NAME is a very hardworking guy"
echo "The manager says, he will hire $NAME in his new team wherever he goes"
echo "$NAME likes to show off because the manager loves him "
echo "$NAME will always be available when they need him"
```

Save and Quit
Run the Script

Variables in scripting

- Variable names are **case sensitive**
- Thus, when calling a variable, you must **write it exactly the same as when you declared it.**
- If you **call a variable without declaring or defining it**, **nothing will be displayed**. It does not exist!
- Let's practice that on the following script: **variable.sh**

Variables in scripting

vim variables.sh

```
#!/bin/bash
# Description:
# Author:
# Date:
FIRST_NAME=Dianna
LAST_NAME=Kamgang
AGE=20
COLOR=PURPLE
```

```
echo "${FIRS_NAME} is a very hardworking girl"
echo "Her last name is ${LAST_NAME} and her favorite color is ${COLOR}"
echo "${FIRST_NAME} is ${AGE} years old "
echo "$NAME is not defined"
```

Variables in scripting

- When you run the script, you can notice that:
  - The **first name did not display at the beginning** (The T was missing when calling the variable **FIRS_NAME** instead of **FIRST_NAME**)
  - The **$NAME** is not printing anything (**that variable was not declared: it does not exist)**
- **Modify the script and correct the errors**

Environment or system variables

Environment variables

- There are some variables that are already set in the system for each user called **Environment variables**
- When they are called, the **system prints their current value depending on the logged in user.**
- You can run the command **# env** or **# printenv** in the Terminal to check those variables

Example: Run the command **# env** and let's check some important environment variables we will use frequently

Environment variables

Variable name	Description
HOME	The user's Home directory
PWD	The present working directory
SHELL	The shell of the user
USER	The current username
LOGNAME	The logged in user

Environment variables

Let's implement that in the **variable.sh** script:

 # vi variables.sh

```
#!/bin/bash
# Description:
# Author:
# Date:
echo "Hello there, your username is ${USER}"
echo"Your User ID is ${UID}"
echo "Your Shell is ${SHELL} "
echo "Your home directory is ${HOME}"
echo "This server's Hostname is  ${HOSTNAME}"
```

Save and Quit
Run the script

Environment variables

```
echo " Hello there your username is ${USER}"
echo " Your uid is  ${UID} "
echo " Your shell is ${SHELL} "
echo " Your home directory is ${HOME} "
echo " this serve's hostname is ${HOSTNAME} "
```

Save and Quit
Run the script

```
[root@ip-172-26-3-92 git-test]# ./variable.sh
Hello there, your username is root
Your User ID is 0
Your Shell is /bin/bash
Your home directory is /root
This server's Hostname is  ip-172-26-3-92.ec2.internal
```

Environment variables

- The **environment variables** are not that different from the **regular variables** we used earlier in the previous lesson.
- Just that, **you don't need to declare them, you can call them anywhere in your script**
- The **system already knows them and has their various values stored at any time**

Environment variables

Let's try to **run this script as a regular user**. (we were running as the root since)

- To do that, we first of all need to **copy the script to a directory that can be accessible to any user on the system**. Let's take the **/tmp** directory: **# cp ./variable.sh /tmp**
- Now, let's check a **regular user** on the system and **login** to that user's account (or create a user account)
- **# tail -3 /etc/passwd** (here I picked the user **centos**)

```
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
chrony:x:998:995::/var/lib/chrony:/sbin/nologin
centos:x:1000:1000:Cloud User:/home/centos:/bin/bash
```

Environment variables

- **# sudo su - centos** (eventually enter the password)
- Now run the script with: **/tmp/variables.sh**

```
[centos@ip-172-26-3-92 git-test]$ /tmp/variable.sh
Hello there, your username is centos
Your User ID is 1000
Your Shell is /bin/bash
Your home directory is /home/centos
This server's Hostname is  ip-172-26-3-92.ec2.internal
```

Environment variables

- You can use the environment variable **to restrict access to a script**
- That is, **you can put a condition to define which user can run it**, with an if statement inside the script

Let's use or **pkg.sh** script:

- Remember this script was used to **install some packages** on the system.
- Now we want **only the root to be able to run that script successfully** (since we know only the root can run the **yum** command)

Environment variables

Switch back to the root user (**$ exit**) and modify the **pkg.sh** script

 # vi pkg.sh then go to the INSERT mode

```
#!/bin/bash

# Description: Script to install some packages
# Author: serge
# Date: January 2022

yum install finger -y
yum install curl -y
yum install zip -y
yum install vim -y
```

Environment variables

Now, let's put our condition:

```
#!/bin/bash

# Description: Script to install some packages
# Author: serge
# Date: January 2022

if [ ${USER} != root ]
then
echo "you need root access to run this"
exit 1
fi
```

```
yum install finger -y
yum install curl -y
yum install zip -y
yum install vim -y
```

Here, we need to **exit** if the condition is not satisfied. If not the script will continue its execution.

You give an exit code that is different from 0, for the system to know that the **script did not execute successfully till the end**. (**Example: exit 99**)

Let's check how the script will run for the **root user** and for **a regular user**

Environment variables

- Here we are going to run the script as the **root** user of the system and check what will be displayed
- We are trying to check if our **if statement is working perfectly**
- make sure you are the root user: **# echo $USER** (If it does not display root then you need to switch to the root user: **# sudo su - root** then **# id** to check)
- Run : **chmod a+x pkg.sh** to make your script executable then run the script: **./pkg.sh**
- **The script will run successfully till the end**

Another condition that could be used is if [${UID} -ne 0] since we know only the root has the UID = 0

Environment variables

- In **Terminal 1**, make sure you are the root user:
  - **# echo $USER** (If it does not display root then you need to switch to the root user: **# sudo su - root** then **# id** to check)
  - Run : **chmod a+x pkg.sh** to make your script executable
  - Now run the script: **./pkg.sh**
- **The script will run successfully till the end**
- Another condition that could be used is **if [${UID} -ne 0] since we know only the root has the UID = 0**

Environment variables

- Secondly, we are going to run the script as a regular user (centos)
- For any user to access the script, let's copy it in the **/tmp** directory with: **# cp pkg.sh /tmp**
  - **# echo $USER** (to make sure it displays your username (**centos for me)**
  - Now run the script: **/tmp/pkg.sh**
- **The script will exit**

Create an environment variable

How can we create an environment variable?

Environment variables

- To create an environment variable, we use:

  # export VarName=Value

Example: Let's create an environment variable called **TMPDIR** that will carry the **/tmp** directory path: **# export TMPDIR=/tmp**

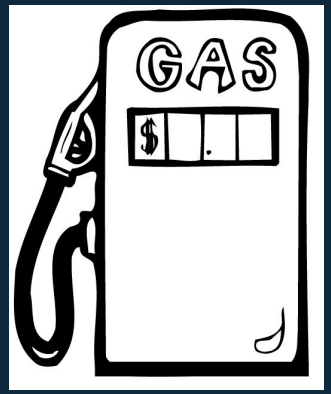
- Check with **# env** and you will see **a new variable created.** You can filter the output with grep: # **env | grep TMPDIR**
- Now, even **if the system is rebooted, the variable will still be there.**
- To **unset** it, you can use: **# unset TMPDIR**
- Check back in **# env**: It is no more there!

4

The if statement

How do we use this statement in scripting?

The if statement



Do you want car wash?

Yes → Which option? → ...

No → Thanks, Goodbye

The if statement

The if statement is used **to check a specific condition in scripting** before executing a set of instructions. Its **structure** is as follows:

Simple structure

```
if [condition]
then
command 1
command 2
...
command n
fi
```

Complete structure

```
if [condition]
then
command 1
command 2
...
command n
else
command 1
command 2
...
command n
fi
```

The if statement

With the if statement:

- If the condition is **True**, all the commands written under the **then** **will be executed** and the ones under the **else** **will be ignored**
- If the condition is not True (**False**), all the commands written under the **then** **will be ignored** and the ones under the **else** **will be executed**

Example: Write an if statement to check the **exit code** of a command

- Remember the **echo $?** command helps you **to check the exit code of a command.**
- **Inside the script**, you don't need to put the **echo**, just use **$?**

The if statement

Solution: In the **pkg.sh** script we previously created

 # vim pkg.sh then go into the **INSERT mode**

```
yum install finger -y
if [ $? -eq 0 ]
then
echo "finger installed successfully"
else
echo "finger did not install"
fi
```

Save and Quit

Run the script: ./pkg.sh

The if statement

Solution: Now let's put an **error** in the command to install finger in other to check if the **else part** works correctly

 # vim pkg.sh then go into the **INSERT mode**

```
yuminstall finger -y
if [ $? -eq 0 ]
then
echo "finger installed successfully"
else
echo "finger did not install"
fi
```

Save and Quit

Run the script: ./pkg.sh

Set the if condition

How do we set the if statement condition in our script?

Set the if condition

Question: Where do **=, !=, -eq, -ne** come from? Where do we get the reference to know how to write the **if condition**?

Answer: from the test manual with the command # man test

- When you use options that are in that test manual, the bash knows exactly what to do

Let's take a look at some options here

Set the if condition

With expressions

Condition	Meaning
(EXPRESSION)	EXPRESSION is **True**
!EXPRESSION	EXPRESSION is **False**
EXPRESSION1 -a EXPRESSION2	EXPRESSION1 and EXPRESSION2 are **True**
EXPRESSION1 -o EXPRESSION2	EXPRESSION1 or EXPRESSION2 is **True**

Set the if condition

To compare Strings

Condition	Meaning
-n STRING	The STRING length is **not zero**
-z STRING	The STRING length is **zero**
STRING1 = STRING2	The two strings are **equal**
STRING1 != STRING2	The strings are **not equal**

Example: *if [-n serge]*
Then ...

Set the if condition

To compare integers

Condition	Meaning
INTEGER1 -eq INTEGER2	The 2 integers are **equal**
INTEGER1 -gt INTEGER2	INTEGER1 is **greater than** INTEGER2
INTEGER1 -le INTEGER2	INTEGER1 is **less or equal to** INTEGER2
INTEGER1 -lt INTEGER2	INTEGER1 is **less than** INTEGER2
INTEGER1 -ne INTEGER2	INTEGER1 is **not equal to** INTEGER2

Set the if condition

To compare files

Condition	Meaning
FILE1 -ef FILE2	The 2 files have the **same device and inode number**
FILE1 -nt FILE2	FILE1 **newer than** FILE2
FILE1 -ot FILE2	FILE1 **older than** FILE2
-e FILE	FILE **exists**
-f FILE	FILE **exists and is a regular file**

Set the if condition

Example on files: Write a script that creates a file called **success** in the current directory **only if it does not exist in there**.

 # vim file.sh

```
If [ -f success ]
then
echo “the file already exist”
else
touch success
fi
```

Run the script with **bash file.sh**

Note:

When you don't give the execute permission to a script, you can still run it with **bash scriptname.sh**

Start getting yourself use to **variables, conditions in scripting, when and how to use them**

Play around with the examples and do some exercises on your own.

If you encounter some issues, do some research before posting your questions in the group.

See you guys in the next Part!

Thanks!

Any questions?

You can find us at:

website: https://utrains.org/

Phone: +1 (302) 689 3440

Email: contact@utrains.org