Bash Shell Script
Part 2

utrain

BASH

THE BOURNE-AGAIN SHELL

How to create and run a script in Linux?

u train

This is an introduction to Scripts in Linux

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

Launch and connect to your **centos 7** server

Let's get started!

utrain

Table of content

1

The Read statement

What is it used for?

The read statement

- When you need some informations from a user in order to run a script, you need to use a **read statement**
- This statement is **used to take input from the user**

Example: Let's write a script that takes as input the **name, the birth year, the city, the reason why the user came to the store** and **displays those informations.**

Solution: To solve this, we need to **create a script** and use some variables that will contain the various informations:

na for the name,	**y** for the year of birth
CITY for the city	**R** for the reason

The read statement

```
# vim read.sh

#!/bin/bash

# Description
# Author
# Date

echo “What is your name? ”
read na
echo “What year were you born? ”
read y
echo “What city are you from?
read CITY
echo “What brought you to the store today? ”
read R

echo “Hello ${na}, you were born
in ${y}, you live in ${CITY} and the
reason for coming here is : ${R}  ”
```

- **Save and Quit**
- Give the execute permission on the script: **# chmod +x read.sh**
- Run the script: **./read.sh**
  - na: **serge**
  - y: **1950**
  - CITY: **Bafoussam**
  - R: **I am running out of water**

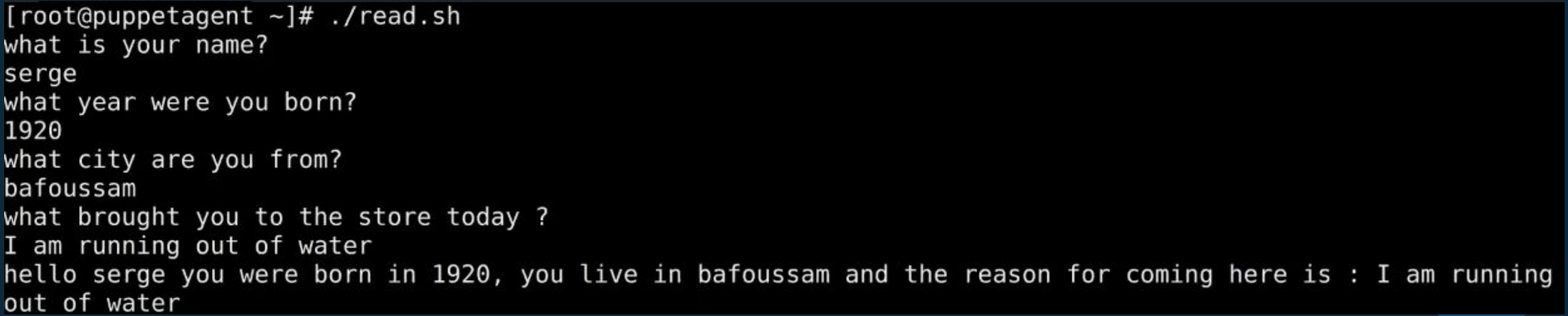
The ‘“’ error is a common mistake with the **echo command**.

If you encounter this error, just go back to the script and check where the “ is missing!

When you run a script and it throws and error, just read the line where the error appears and try to fix it!

The read statement

You can see the script ran successfully and displayed the informations as expected



```
[root@puppetagent ~]# ./read.sh
what is your name?
serge
what year were you born?
1920
what city are you from?
bafoussam
what brought you to the store today ?
I am running out of water
hello serge you were born in 1920, you live in bafoussam and the reason for coming here is : I am running
out of water
```

The read statement

- We can **add some conditions on variables in there.**
- For example, let's say **if a user skips the name** (ie he does not enter his name) we display a message that **asks him to enter a valid name**
- You can do that by adding the **if statement:**

```
if [ -z ${na} ]
then
echo "Please enter a valid name"
exit 2
fi
```

- You can do the same for all the variables. You can even **nest the if statements** using **elif (ie else if)**

The read statement

You can also link many conditions in the **if statement** using:

- || for **or**

  if [condition 1] || [condition 2] || […] || [condition n]

- && for **and**

  if [condition 1] && [condition 2] && […] && [condition n]

- You can even mix || and &&

2

The for loop

The for loop structure

The for loop

- The **for loop** is used to **iterate through a list of items to perform repetitive tasks**
- Its **structure** is as follows:

```
for item in (list);
do
command 1
command 2
…
command n
done
```

For each item, the various commands will be executed
The loop will exit when the list is exhausted

The for loop

Example: Write a for loop to **delete a list of users** with the following usernames: **username1, username2, username3**

Solution:

```
for item in username1 username2 username3;
do
userdel -r ${item}
done
```

Let's practise this in the terminal

The for loop

Example: Write a script to **create 4 regular users on the system**: u6bt, u7bt, u8bt, u9bt

Solution: create a new script with **# vim for.sh** and go to the **INSERT mode**

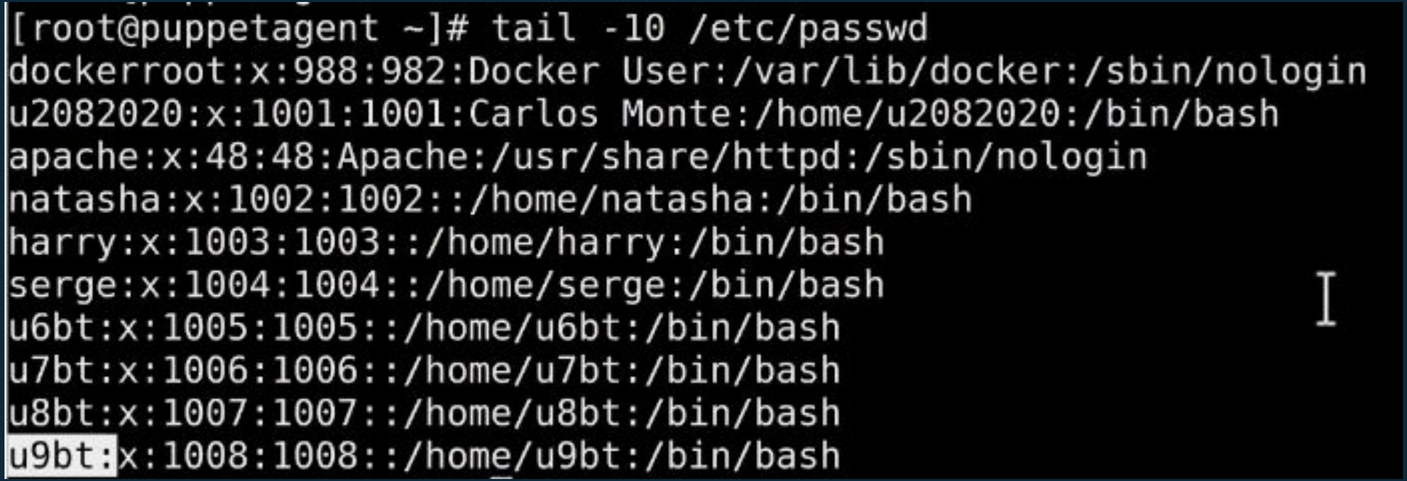
```
for i in u6bt u7bt u8bt u9bt;
do
useradd ${i}
echo “user $i is successfully created”
sleep 3
done
```

Now, let’s give the execute permission to the script: **# chmod +x for.sh**

Run the script: **./for.sh**

The for loop

You can check if the users were successfully created in the **/etc/passwd** file with: **# touch -10 /etc/passwd**



```
[root@puppetagent ~]# tail -10 /etc/passwd
dockerroot:x:988:982:Docker User:/var/lib/docker:/sbin/nologin
u2082020:x:1001:1001:Carlos Monte:/home/u2082020:/bin/bash
apache:x:48:48:Apache:/usr/share/httpd:/sbin/nologin
natasha:x:1002:1002::/home/natasha:/bin/bash
harry:x:1003:1003::/home/harry:/bin/bash
serge:x:1004:1004::/home/serge:/bin/bash
u6bt:x:1005:1005::/home/u6bt:/bin/bash
u7bt:x:1006:1006::/home/u7bt:/bin/bash
u8bt:x:1007:1007::/home/u8bt:/bin/bash
u9bt:x:1008:1008::/home/u9bt:/bin/bash
```

The for loop

- Now let's **delete the accounts we previously created.**
- To do that, you just need to replace the **useradd** command in the script with the **userdel -r** command!

```
for i in u6bt u7bt u8bt u9bt;
do
Userdel -r ${i}
echo “user $i is successfully deleted”
sleep 3
done
```

Run the script: **./for.sh**

It may be difficult to list 100 or more users in the script.

A way to avoid this is to **create a file that will contain the list of users** and just **cat** that file in the script with: **for i in $(cat filename)**

The for loop

Example: create a new file in the **/tmp** directory and write the the usernames in there.

# touch /tmp/username

# vim /tmp/username and go to the **INSERT mode**

```
u6bt
u7bt
u8bt
u9bt
...
```

You can modify the **for.sh** script **to create the users first and delete them after.**

```
for i in $(cat /tmp/username)
do
...
```

3

The while loop

How do we use loops in scripting?

The while loop

- Just like the for loop, the **while loop is used to iterate thru a list of items to execute some commands**
- Its structure is as follows:

```
while [condition]
do
command 1
command 2
command 3
...
command n
done
```

While the condition is **true**, run the commands
When the condition becomes **false,** exit the loop

The while loop

Example: create a new script to practice the while loop

# vim while.sh

```
#!/bin/bash
while [ 2 -eq 2 ]
do
echo ”This is a while loop”
sleep 2
echo “success”
done
```

What will happen here?

The **loop will run indefinitely because the condition is always True**: 2 is always equal to 2

Give the execute permission to the script: **# chmod +x while.sh**

Run the script: **./while.sh**

Since the script will continue running, **you can kill it with Ctrl-C**

The while loop

Now, let’s put some logic in there **for it to stop at some point**:

# vim while.sh

```
COUNT=0
while [ ${COUNT} -lt 6 ]
do
echo “This is a while loop”
sleep 2
echo “success”
COUNT=$(( $COUNT + 1 ))
done
```

Here, we are applying these on simple examples for you to understand the notions. But soon, we will get into more serious usage, then you will see how useful this is in the company environment

If you don't understand it now, don't get frustrated, **we will do a lot of practice in class!**

Play around with the **conditions on the various variables**

Make more research and practice on this concept.

See you guys in the next part!

Thanks!

Any questions?

You can find us at:

website: http://utrains.org/

Phone: +1 (302) 689 3440

Email: contact@utrains.org

utrain

Click on the link below to contact the support team for any issue!

utrains.org/support/

Create a ticket for your problem and we will get back to you soon!