IT Scripting and Automation

Bash Scripting
-Parameters, STDIN and ‘exit’ status

Lecturer: Art Ó Coileáin

Positional Parameters – Recap

Remember from last week if we used a sample script which is capable of using positional parameters e.g.:

```
./script.sh parameter1parameter2 parameter3
```

Then:

- $0 contains “script.sh”
- $1 contains ”parameter1”
- $2contains”parameter2”
- $3contains”parameter3”
- **$@** refers to the array of all the positional parameters {$1, $2, $3, …}. In the example above **$@**contains a list: {**paramter1, paramter2, paramter1**}
- **$#** refers to the number of positional parameters.

Exercise - Positional Parameters

Task:

- Create a script to lock a particular user account and archive the home directory

Criteria:

- The account name must be passed as a parameter to the script.
  - Use passwd command to lock the account.
  - Use tar command to archive the home directory.

Hint:

  - Check man command for options in each command.

Exercise -Solution

```
#!/bin/bash
echo “Locking user account: $1”
echo “Archiving $1 user home directory”
#Locking user account
passwd -l $1

#Archive of the home directory.
#!/bin/bash
tar -czvf /tmp/${1}.tar.gz /home/${1}
echo “Process finished”
```

Exercise -Positional Parameters

- Extend the previous script to accept one or more parameters, i.e., allow to pass multiple users to the script.
- Sample of execution
- **./lockAccounts.sh** user1 user2 user3 user4

User Input (STDIN) - Recap

- The read command accepts STDIN.
- Syntax:
  - read -p “Message to display” VARIABLE

- Example:

```
#!/bin/bash
read –p “Enter a user name: ” USER
echo “Archiving user: $USER”
```

‘Exit’ Status / Return Code

Every time a command is executed it returns an exit status.

Return code**:**

- The **exit status** is sometimes called **return code** or **exit code** is an integer number ranging from 0 to 255.

Success:

- By convention command that execute **successfully** return a **0** exit status.

‘Exit’ Status / Return Code

The special variable **$?** contains the return code of the previously executed command.

Error:

- If some sort of error is encountered then a non-zero exit status is returned.
- Example

  ls /directory/non-existent-file

  echo “$?”

```
student@ITSA-Server:~$ ls test1.txt
ls: cannot access test1.txt: No such file or directory
student@ITSA-Server:~$ echo $?
2
student@ITSA-Server:~$ ls test.txt
test.txt
student@ITSA-Server:~$ echo $?
0
student@ITSA-Server:~$ _
```

The exit command

Not only normal commands return and exit status but shell scripts do as well.

- You can control the exit status by using the **exit** command. Explicitly define the return code:
  - exit 0
  - exit 1
  - exit 255
- If you don't specify the return exit command in a script. The default value is that of the last command executed.

The exit command

- The shell scripts can be **called by another scripts** and its **exit status** can be examined just like any other command.
- It is possible to make the **return codes have a significant meaning**, i.e., return code of 1 for one type of error occurred or return code of 2 for a different type of error occurred.
- The scripts can make decisions based on the exit statuses of various commands.

‘Exit’ Status / Return Code Example

```
#!/bin/bash
HOST=“google.com”

# Send one package to google.com
ping –c 1 $HOST
RETURN_CODE=$?

If [ “$RETURN_CODE” –eq “0” ]
then
        echo “$HOST reachable.”
else
        echo “$HOST unreachable.”
fi
```

Logical Operators

You can chain multiple command with either **AND**'s or **OR**'s.

- **&& = AND**
- The command following a double ampersand (**&&**) will only execute if previous command **succeeds**.

E.g.:

```
mkdir /tmp/bkup && cptest.txt /tmp/bkup
```

Logical Operators and Commands

- **||=OR**
- The command following a double pipe (**||**) will only execute if the previous command **fails**

E.g.:

```
cp test.txt /tmp/bak/ || cp test.txt /tmp
```

- In other words if the first command returns a **non-zero exit status**, then the next command is executed

Logical Operators and Commands

Explain the difference of the following scripts:

```
#!/bin/bash
HOST="google.com"
ping -c 1 $HOST && echo "$HOST reachable."
```

---------- And ----------

```
#!/bin/bash
HOST="google.com"
ping -c 1 $HOST || echo "$HOST unreachable."
```

The Semicolon ";"

One the command line you can run multiple separate commands separated with ";" to execute them sequentially:

- `cp file.txt /tmp/bkup/; cp file.txt /tmp`

This is the same as:

```
cp file.txt /tmp/bkup/
cp file.txt /tmp
```

- The command following a semicolon will always get executed, no matter if the previous command failed or succeeded.

Exercise

- Modify script from ping example slide to return the correspondent exit status based on if the ping command succeeded or not.