UNIX

Lecture 8 : Shell Programming (bash) (Part 2)

Filipe Vasconcelos[1]
[1]ESME, Lille, filipe.vasconcelo@esme.fr

ESME
PARIS•BORDEAUX•LILLE•LYON

Outline

Intended Learning Outcomes

ILO4

Produce, test, and verify the results of Shell scripts (in Bash) to perform tasks ranging from simple to complex algorithms.

Outline

Exit Status (or Return Code)

The `exit` command is used to terminate a script, which is equivalent to `return` in a C program. A script or program always returns an integer between 0 and 255. A script or program that succeeds without errors returns 0. The exit status of the last executed command is given by `$`.
Example (exit status 0):

```
$ ls
$ echo $?
```

```
0
```

Example (exit status "command not found"):

```
$ lss
$ echo $?
```

The `return` Statement in C

```
$ ./return_en_C.sh
```

```
-------------------------------------
Code:
-------------------------------------
#include <stdio.h>
int main(){
    return 0;
}
-------------------------------------

gcc returnC0.c (compilation)
./a.out        (execution)
return $? = 0  ($? of execution)
```

```
-------------------------------------
Code:
-------------------------------------
#include <stdio.h>
int main(){
    return 1;
}
-------------------------------------

gcc returnC1.c (compilation)
./a.out        (execution)
return $? = 1  ($? of execution)
```

Note: The `C` language is beyond the scope of this course, but the connection between this language and Unix, Linux, and where the bash interpreter is very important!

Outline

Testing in Bash

In Bash, there are two equivalent commands to test a condition: `[]` and `test`. (see `man [`). The value of the exit status (`$?`) indicates whether the condition is true (`0`) or false (`1`). Using the `[` command:

```
[ 1 = 0 ] ; echo $?
```

```
1
```

Using the `test` command:

```
test 0 = 0; echo $?
```

```
0
```

Outline

Conditional Structures

Conditional structures involve testing the exit status of the `[]` or `test` command.

if-then-fi

The `if-then` structure:
Example:

```
if [ condition ]
then
    echo "the condition is true"
fi
```

- The `fi` keyword terminates the statement.
- No spaces before or after the condition.
- The syntax `if [ condition ];then` is also allowed (i.e., on the same line).

Conditional Structures

if-then-else-fi

The `if-then-else` structure:
Example:

```
if [ condition ]
then
    echo "the condition is true"
else
    echo "the condition is false"
fi
```

Conditional Structures

if-then-elif-then-else-fi

The if-then-elif-then-else structure:
Example:

```
if [ condition1 ]
then
    echo "condition1 is true"
elif [ condition2 ]
then
    echo "condition2 is true"
else
    echo "all conditions are false"
fi
```

Condition Syntax

Different Types of Conditions

- on strings of characters;
- on numbers;
- on files;
- on command execution.

Boolean Operators

- AND: `&&` or `-a`
- OR: `||` or `-o`
- NOT: `!` NOT

String Conditions

In bash, all variables are considered as strings of characters. It is, therefore, straightforward to test the value of a string.

Condition	Meaning	Syntax
`$string1 = $string2` `$string1 == $string2`	Checks if the two strings are identical.	`[ "$var" = "Monday" ]`
`$string1 != $string2`	Checks if the two strings are different.	`[ "$var" != "Monday" ]`
`-z $string`	Checks if the string is empty.	`[ -z "$var" ]`
`-n $string`	Checks if the string is not empty.	`[ -n "$var" ]`

Example:

```
var="Thursday"
if [ "$var" == "Monday" ]
then
    echo "It's Monday"
else
    echo "It's not Monday"
fi
```

Numeric Conditions

Condition	Meaning	Syntax
`$num1 -eq $num2`	Checks if the numbers are equal (equal). Not to be confused with "=" which compares two strings.	`[ "$num" -eq 5 ]`
`$num1 -ne $num2`	Checks if the numbers are different (not equal). Not to be confused with "!="	`[ "$num" -ne 5 ]`
`$num1 -lt $num2`	Checks if num1 is less than num2 (lower than).	`[ "$num" -lt 5 ]`
`$num1 -gt $num2`	Checks if num1 is greater than num2 (greater than).	`[ "$num" -gt 5 ]`
`$num1 -le $num2`	Checks if num1 is less than or equal to num2 (lower or equal).	`[ "$num" -le 5 ]`
`$num1 -ge $num2`	Checks if num1 is greater than or equal to num2 (greater or equal).	`[ "$num" -ge 5 ]`

Example:

```
num=5
if [ "$num" -eq 5 ]
then
    echo "The number is equal to 5"
else
    echo "The number is different from 5"
fi
```

File Conditions

Condition	Meaning	Syntax
`-e filename`	Checks if the file exists.	`[ -e "$file" ]`
`-d filename`	Checks if the file is a "directory".	`[ -d "$file" ]`
`-f filename`	Checks if the file is a regular file (not a directory).	`[ -f "$file" ]`
`-L filename`	Checks if the file is a symbolic link (shortcut).	`[ -L "$file" ]`
`-r filename`	Checks if the file is readable (r).	`[ -r "$file" ]`
`-w filename`	Checks if the file is writable (w).	`[ -w "$file" ]`
`-x filename`	Checks if the file is executable (x).	`[ -x "$file" ]`
`$file1 -nt $file2`	Checks if file1 is newer than file2 (newer than).	`[ "$file1" -nt "$file2" ]`
`$file1 -ot $file2`	Checks if file1 is older than file2 (older than).	`[ "$file1" -ot "$file2" ]`

Example:

```
file="test.txt"
if [ -e $file ]
then
    echo "The file $file exists"
else
    echo "The file $file does not exist"
fi
```

Using Booleans

Condition	Meaning	Syntax
`!condition`	true if condition is false	`[!("$var"="abc")]`
`cond1 && cond2`	true if cond1 AND cond2 are both true	`[-e $f] && [-w $f]`
`cond1 || cond2`	true if cond1 OR cond2 is true	`[$n -ge 2] || [ $n -le 4 ]`

Example:

```
if [ -e $file ] && [ -w $file ]
then
    echo "The file $file exists and is writable"
fi
if [ $num -ge 2 ] || [ $num -le 6 ]
then
    echo " 2 <= n <= 6"
fi
```

Condition Based on Command Result

- To test a command, you can use the variable $?.
- If the command succeeds ($? equals 0), then the condition is met, and the corresponding action is executed.
- All other return values are considered equivalent to false (execution error).

Example:

```
grep $USER /etc/passwd
if [ $? -eq 0 ]
then
    echo "You are mentioned in /etc/passwd"
else
    echo "You are not mentioned in /etc/passwd"
fi
```

Here, grep returns 0 if the pattern is found in the file, otherwise, it returns 1.

Outline

Iterative Structures

Iterative structures allow you to repeat a part of the code (a loop) as many times as necessary. There are three ways to implement loops:

- `for`
- `while`
- `until`

for Loop

In a **for** loop, instructions are executed n times.

The "for-do" Loop

```
for var in "value1" "value2" "value3"
do
    echo "the variable is $var"
done
```

Defining a list from the result of a command:

```
for file in $(ls)
do
    echo "file $file found"
    <instruction on file>
done
```

`for` Loop with Numeric Values

Using the seq Command

```
for i in $(seq 1 10)
do
    echo ${i}
done
```

To change the step (refer to `man seq`): Example (every 2 steps):

```
for i in $(seq 1 2 10)
do
    echo ${i}
done
```

`for`: Using a "C-Like" Syntax

A "C-like" syntax can be used:

```
for ((i=1;i<=5;i++))
do
    echo ${i}
done
```

`while` Loop

In a **while** loop, instructions are executed as long as the condition is true.
Example: Infinite Loop

```
while true
do
    echo "I am an infinite loop"
done
```

Common Usage: Reading a File Line by Line

```
n=1
while read line
do
    echo "$n : $line"
    n=$((n+1))
done < fichier.txt
```

while Loop

Example: Interacting with the User

```
while [ "$choice" != "q" ]
do
    echo "Type q to exit the loop"
    read choice
done
```

`until` Loop

In the **until** loop, instructions are executed until a condition becomes true.
Example:

```
until false
do
    echo "I am an infinite loop"
done
```