Aula P03
Shell Programming

Carlos Coutinho
Carlos.Eduardo.Coutinho@iscte-iul.pt

Topics

- Linux Shell Programming
- Variables
- Command execution and Logic
- Shell Script: “” vs ‘’ vs ``
- Execute a Shell script
- “to source or not to source”
- Script Arguments
- Operators – Advice

Linux Shell Programming

- A shell script is actually a program that can be executed
- It features variables – Environment variables
- It features control structures such as if, while, for, case
- It features functions and calling of external executables and scripts
- It features passing of arguments to scripts and functions
- It features exit codes

Variables

```
₴ a=1
₴ echo a      # prints a
a
₴ echo $a     # prints 1
1
```

- Be careful with the spaces on the syntax:

```
₴ a=1         # OK
₴ a = 1       # NOK: Execute “a” with arguments “=” and “1”
-bash: a: command not found
₴ a= 1        # NOK: Set a=“” and tries to execute “1”
-bash: 1: command not found
₴ a =1        # NOK: Execute “a” with argument “=1”
-bash: a: command not found
```

```
₴ echo $HOME # prints environment variable HOME
/home/a12345
₴ echo $PATH # prints environment variable PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

Command execution and Logic

exit 0: success → true
exit 1: error → false

```
$ echo a
a
$ echo b
b
```

- **";"** allows running multiple commands on the same line and **always executes all commands**, independently of their exit code/condition value

```
$ echo a; true; echo b; false; echo c  # ";" runs multiple commands on a single line and always executes all
a
b
c
```

- **"&&"** allows running multiple commands on the same line and executes them **while their exit code/condition value is 0/true**

```
$ echo a && echo b && false && echo c  # "&&" runs multiple commands on a single line and executes while true
a
b
```

- **"||"** allows running multiple commands on the same line and executes them **while their exit code/condition value is 1/false**

```
$ false || false || false || echo a || echo b   # "||" runs multiple commands on a single line and executes while false
a                                                # In this example, "b" is not displayed because "echo a" returns true
```

- This allows some interesting and complex flows of commands (on command line, but also on scripts!):

```
$ echo a && true && echo b && false && echo c || echo d && echo e
  runs      runs      runs      does not run      runs      runs
a
b
d
e
```

Shell Script: "" vs '' vs ``

- Script helloworld.sh:

```
#!/bin/bash -x
# This is a comment!
echo "1Hello World"       # This is a comment, too!
echo "2Hello      World" # 1 argument
echo  3Hello      World  # 2 arguments
echo  4Hello "   " World  # 3 arguments
echo "5Hello * World"
echo  6Hello * World      # Hello <List all files> World
echo "7Hello" * "World"   # Hello <List all files> World
echo "8Hello "*" World"
echo "9Hello$HOME" World # The use of "" echoes the environment variable HOME
echo '0hello$HOME' World # The use of '' echoes the text $HOME
echo `date` World         # `date` executes “date” and echoes its value - OLD
echo $(date) World        # $(date) executes “date” and echoes its value - NEW
```

Shell Script: "" vs '' vs `` : Testing

```
₴  ./helloworld.sh
+ echo "1Hello World"      # This is a comment, too!
1Hello World
+ echo "2Hello      World" # 1 argument
2Hello      World
+ echo  3Hello      World  # 2 arguments
3Hello World
+ echo  4Hello "  " World  # 3 arguments
4Hello    World
+ echo "5Hello * World"
5Hello * World
+ echo  6Hello * World     # Hello <List all files> World
6Hello arguments.sh b.sh helloworld.sh World
+ echo "7Hello" * "World"  # Hello <List all files> World
7Hello arguments.sh b.sh helloworld.sh World
+ echo "8Hello "*" World"
8Hello * World
+ echo "9Hello$HOME" World # The use of "" echoes the environment variable HOME
9Hello/home/a12345 World
+ echo ‘0Hello$HOME’ World # The use of '' echoes the text $HOME
0Hello$HOME World
+ echo `date` World        # `date` executes “date” and echoes its value – OLD SYNTAX
Thu Oct 08 02:57:48 WEST 2020 World
+ echo $(date) World       # $(date) executes “date” and echoes its value – NEW SYNTAX
Thu Oct 08 02:57:48 WEST 2020 World
```

Execute a Shell script

- Ensure that +x attribute is on the shell file
- ./script # why ./ ?

Execute a Shell script

- Ensure that +x attribute is on the shell file
- ./script # why ./ ?
- Linux does NOT search automatically on current folder (like Win)
- Linux searches on the directories specified on special variable PATH

Execute a Shell script

- Ensure that +x attribute is on the shell file
- ./script # why ./ ?
- Linux does NOT search automatically on current folder (like Win)
- Linux searches on the directories specified on special variable PATH

```
$echo $PATH  # PATH has format dir1:dir2:dir3:…
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

“to source or not to source”

- Script b.sh (see man builtins):

```
#!/bin/bash -x
b=2
```

- Testing:

```
ƨ echo $b     # prints ""

ƨ ./b.sh
+ b=2
ƨ echo $b     # prints ""

ƨ . ./b.sh      # ". ./b.sh" is the same as "source ./b.sh"

ƨ echo $b     # prints 2
2
```

Script arguments

- Script arguments.sh (see man bash # Special Parameters):

```
#!/bin/bash -x
echo $0  # Script name
echo $#  # Nr Arguments
echo $1  # 1st argument  $1 to $99999… : nth argument
echo $3  # 3rd argument  $1 to $99999… : nth argument
echo $$  # Running Shell PID
echo $-  # Set Options (see man builtins # set)
echo $?  # Exit code of the last foreground command
echo $@  # Array of arguments
```

Script arguments: Test

```
₴ ./arguments a b c d e f g h
+ echo $0  # Script name
./arguments.sh
+ echo $#  # Nr Arguments
8
+ echo $1  # 1st argument  $1 to $99999… : nth argument
a
+ echo $3  # 3rd argument  $1 to $99999… : nth argument
c
+ echo $$  # Running Shell PID (Process ID)
32334
+ echo $-  # Set Options (see man builtins # set): -h –x -B
hxB
+ echo $?  # Exit code of the last foreground command
0
+ echo $@  # Array of arguments
a b c d e f g h
```

Script arguments: $?

Results: = 0: OK <> 0: NOK (error)

```
ƨ [ abc = abc ]; echo $?
0
ƨ [ abc = def ]; echo $?
1

ƨ [ 1 = 1 ]; echo $?
0
ƨ [ 1 = 2 ]; echo $?
1

ƨ [ abc = *bc ]; echo $?
1
ƨ [[ abc = *bc ]]; echo $?
0
```

Script arguments: Shift and Set commands

- Script arguments2.sh

```
#!/bin/bash -x
echo $@    # Array of arguments
echo $1    # 1st argument

shift      # Removes $1, shifts $2 to $1 and so on
echo $@    # Array of arguments after shift
echo $1    # 1st argument

shift 2    # Removes $1 and $2, shifts all others 2
echo $@    # Array of arguments after shift 2
echo $1    # 1st argument

set z y x # Redefines new arguments
echo $@    # Array of arguments after set arguments
echo $1    # 1st argument
```

Script arguments: Shift and Set commands: test

```
₴ ./arguments2 a b c d e f g h
+ echo $@    # Array of arguments
a b c d e f g h
+ echo $1    # 1st argument
a
+ shift      # Removes $1, shifts $2 to $1 and so on
echo $@      # Array of arguments after shift
b c d e f g h
+ echo $1    # 1st argument
b
+ shift 2    # Removes $1 and $2, shifts all others 2
+ echo $@    # Array of arguments after shift 2
d e f g h
+ echo $1    # 1st argument
d
+ set z y x # Redefines new arguments
+ echo $@    # Array of arguments after set arguments
z y x
+ echo $1    # 1st argument
z
```

if and test

```
ƨ if true; then echo 1; else echo 0; fi
1
ƨ if false; then echo 1; else echo 0; fi
0
ƨ if 1 = 1; then echo 1; else echo 0; fi  # bash will try to run “1 = 1”. Solution? test
-bash: 1: command not found
ƨ if test 1 = 1; then echo 1; else echo 0; fi
1
ƨ if [ 1 = 1 ]; then echo 1; else echo 0; fi
1
ƨ if [ 1 = 2 ]; then echo 1; else echo 0; fi
0
```

- Be careful with the spaces on the syntax:

```
ƨ if [ condition ]; then  # OK
ƨ if [condition]; then    # NOK (error)
ƨ if [condition ]; then   # NOK (error)
ƨ if [ condition]; then   # NOK (error)
```

Operators – [ ] vs [[ ]] vs (( )) vs ( )

exit 0: success → true
exit 1: error → false

- **if [condition]**

```
if [ -f $file ] && [ true ]   # Checks if $file is the name of an existing file and another condition
if [ -f $file -a true ]       # Same as previous (another syntax), not easily readable
if [ "$answer" = y -o "$answer" = yes ] # Checks if answer is “y” or if it is “yes”
```

- **if [[condition]]** (aligned with high-level languages, this accepts &&, ||, ==, != and =~ operators)

```
if [[ -f $file && true ]]     # Same as [ ] but allows “&&”, “||”, “!”, “!=” in the test brackets
if [[ $answer =~ ^y(es)?$ ]]  # Same as [ ] but way simpler (using RegExp)
if [[ $answer = y* ]]         # Even simpler than the above, matches anything starting with “y”
```

- **if ((condition))**

This performs arithmetic. As the result of the arithmetic, an exit code is set and the if statement acts accordingly. It returns an exit code of zero (true) if the result of the arithmetic calculation is nonzero.

- **if (command)**

This runs command in a subshell. When command completes, it sets an exit code and the if statement acts accordingly.

- **if command**

command is executed and the if statement acts according to its exit code.

Functions: Operator ( ) and operator { }

- Script function1.sh

```
#!/bin/bash -x
get_tmp_filenames() (
    cd /tmp;
    files=*;
    echo "The files in /tmp are: $files";
)
```

- Testing Script function1.sh:

```
₴ source ./function1.sh
₴ pwd; get_tmp_filenames; pwd
/home/a12345
The files in /tmp are: hsperfdata_a83032
/home/a12345
₴ echo $files
```

Functions: Operator ( ) and operator { }

- Script function2.sh

```
#!/bin/bash -x
get_tmp_filenames() {
    cd /tmp;
    files=*;
    echo "The files in /tmp are: $files";
}
```

- Testing Script function1.sh:

```
₴ source ./function2.sh
₴ pwd; get_tmp_filenames; pwd
/home/a12345
The files in /tmp are: hsperfdata_a83032
/tmp
₴ echo $files
hsperfdata_a83032
```

References

- https://www.shellscript.sh/
- https://zwischenzugs.com/2018/01/06/ten-things-i-wish-id-known-about-bash/
- http://mywiki.wooledge.org/BashFAQ/082
- https://www.gnu.org/software/bash/manual/html_node/Positional-Parameters.html#Positional-Parameters
- https://www.gnu.org/software/bash/manual/html_node/Special-Parameters.html#Special-Parameters
- https://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_01.html
- https://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_02.html
- https://www.gnu.org/software/bash/manual/html_node/Bash-Conditional-Expressions.html
- https://ryanstutorials.net/bash-scripting-tutorial/bash-if-statements.php
- https://stackoverflow.com/questions/3427872/whats-the-difference-between-and-in-bash/3427931
- http://mywiki.wooledge.org/BashFAQ/031
- https://unix.stackexchange.com/questions/306111/what-is-the-difference-between-the-bash-operators-vs-vs-vs