BASH Scripting

-By Mabrouki Malik

- Introduction
- Variables && Parameters
- Decision && Repetition (if-then, while, …)
- Arithmetic Operations
- String Manipulation
- Special Features
- Arrays && Functions

Introduction

What Is BASH ?

The Bourne-Again Shell is a command interpreter and a high-level programming language. As a command interpreter, it processes commands you enter on the command line in response to a prompt. And as a programming language, it processes commands stored in files called *shell scripts*.

Why Script With BASH ?

> Automate repetitive/time-consuming tasks
> Preferred tool for system administration
> Powerful utility when it comes to input/output communication between programs
> It is basically the *glue logic* that makes small general purpose tools work together

Introduction

```
# How To Run A Shell Script ?
    $ chmod +x yourscript.sh
    $ ./yourscript.sh
```

Variables

> Assign a value to a variable :
  $ *varname=value* (No spaces in between !)

> Print value of variable :
  $ echo $*varname*

> Environment variables :
  $USER : current logged in user
  $HOME : home directory of user
  $PATH : directory list where to find programs

> Read user input :
  $ read -p “*prompt*” *varname* [*more vars*]

Variables

Special Shell Variables

Parameter	Meaning
$0	Name of the current shell/shell script
$1-$9	Positional parameters 1 through 9
$#	The number of positional parameters
$*	All positional parameters, “$*” is one string
$@	All positional parameters, “$@” is a set of strings
$?	Return status of most recently executed command
$$	Process id of current shell/shell script
$(*command*)	Output of ***command*** after execution

Positional Parameters

```
$ ./myscript.sh apple banana orange
```

$0 $1 $2 $3

> $# = 3
> values contained in "$*" :
"apple banana orange"
> values contained in "$@" :
"apple"
"banana"
"orange"

Decisions

```
# if-then
    if command
    then
        statements
    fi
```

> ***statements*** are executed only if ***command*** succeeds, i.e has return status 0

```
# if-then-else
    if command; then
        statements-1
    else
        statements-2
    fi
```

Decisions

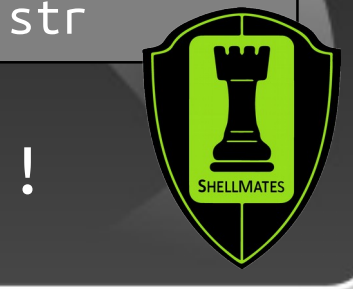
```
# test command
    test expression ⇔ [ expression ] ⇔ [[ expression ]]

    > Evaluates expression and returns true or false

    Example :
        if [ -e “$file”  ]; then
            echo “file $file exists !”
        fi

# if-then-elif
    if [ condition-1 ]; then
        statements-1
    elif [ condition-2 ]; then
        statements-2
    |
    |
    else
        statements-n
    fi
```

Decisions

Relational Operators (test command)

Meaning	Numeric	String
Greater than	-gt	
Greater than or equal	-ge	
Less than	-lt	
Less than or equal	-le	
Equal	-eq	= or ==
Not equal	-ne	!=
str1 is less than str2		[[str1 < str2]]
str1 is greater str2		[[str1 > str2]]
String length is greater than zero		-n str
String length is zero		-z str

> Look in the man page, there are even more !



Decisions

```
# case statement
    case $varname in
        pattern-1)
            statements-1
        ;;
        pattern-2)
            statements-2
        ;;
        pattern-n)
            statements-n
        ;;
    esac
```

Wildcards are allowed in ***pattern*** :

- `*` matches any character **0 or more times**
- `?` matches any **single** character
- `[sequence]` matches any character in ***sequence***
- `p1|p2` matches ***p1*** or ***p2***

Challenge N°01

Task : Write a script that asks for a user name, if it is the same as the current user print “Access granted for <username>” else print “Access denied for <username>”.

Expected output : (current user == shellmates)

```
###
Username : shellmates
Access granted for shellmates

###
Username : intruder
Access denied for intruder
```

Repetition

```
# for loop
   for varname in argument-list
   do
      statements
   done

   or

   for (( varname=start; condition; inc/dec ))
   do
      statements
   done
```

Repetition

```
# while loop
   while [ condition ] (or command)
   do
      statements
   done

# until loop
   until [ condition ] (or command)
   do
      statements
   done
```

Arithmetic Operations

To use arithmetic expressions on integers :
   let or (())

```
$ let n=10
$ (( n += 2 ))
$ echo $n
   12
$ let n++
$ echo $n
   13
$ echo $((n*2))
   26
```

Challenge N°02

Task : The file **archive.tar.gz** has been targz-ed a 100 times, can you write a script to see what's actually hiding in there ?

To decompress a tar.gz archive :

```
tar xzf archive.tar.gz
```

String Manipulation

Syntax	Meaning
${#*varname*}	Length of string
${*varname*:pos[:count]}	Substring beginning at *pos* and ending after *count* characters (or until the end if count is omitted)
${*varname*#[#]*pattern*}	Remove minimal/maximal matching prefix matching *pattern*
${*varname*%[%]*pattern*}	Remove minimal/maximal matching suffix matching *pattern*
${varname/[/]*pattern*/*replace*}	Replace first occurence of *pattern* with *replace* (all occurences if //)

Challenge N°03

Task : The folder contains some jpg images and some png images, we need you to convert thoses jpg images to png and vice-versa. Write a script that performs this task

To convert an image from jpg to png, use this command :

```
convert image.jpg image.png
```

And vice-versa :

```
convert image.png image.jpg
```

Special Features

Syntax	Expands to
`file_{1,2,3}.txt`	file_1.txt file_2.txt file_3.txt (brace expansion)
`{1..5}`	1 2 3 4 5 (brace sequence expansion)
`{a..d}`	a b c d (brace sequence expansion)
`${!`*varname*`}`	Replaces {!*varname*} with the value contained in *varname*, so it becomes $value (variable indirection)
`<(`*command*`)`	Replaces the output of *command* with a temporary special file (named pipe, or FIFO) (process substitution)

+ Many many more features...

Arrays

```
# Syntax
   Declaration: name=(element1 element2 …)
   Access by index: ${name[n]}
   Length: ${#name[@]} (or ${#name[*]})

# Examples
   $ arr=(apple banana orange)
   $ echo ${arr[0]}
   > apple
   $ echo ${arr[-1]}
   > orange
   $ echo ${#arr[@]}
   > 3
```

Functions

 # Syntax

Declaration:

```
func_name () {
    # code
}
```

Usage: ***func_name*** *param1 param2 …*

Note: variables in functions use global scope !!
Use “**local** *varname=value*” to declare variables used only in the function

 # Examples

```
say_hello() {
    echo “Hello $1 !”
}

$ say_hello mate
> Hello mate !
```

Resources

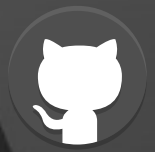
- A Practical Guide to Linux Commands, Editors, and Shell Programming 4th Edition - Mark G. Sobell
- Luke Smith
- Google

Contact

Malik MMML#8501 (@shellmates)

Laid Malik

im_mabrouki@esi.dz



https://github.com/malikDaCoda/

Thank You Mates !