



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
<code>\$0</code>	Name of the current shell/shell script
<code>\$1-\$9</code>	Positional parameters 1 through 9
<code>\$#</code>	The number of positional parameters
<code>\$*</code>	All positional parameters, “\$*” is one string
<code>\$@</code>	All positional parameters, “\$@” is a set of strings
<code>\$?</code>	Return status of most recently executed command
<code>\$\$</code>	Process id of current shell/shell script
<code>\$(command)</code>	Output of <i>command</i> 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` \Leftrightarrow `[expression]` \Leftrightarrow `[[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	<code>-gt</code>	
Greater than or equal	<code>-ge</code>	
Less than	<code>-lt</code>	
Less than or equal	<code>-le</code>	
Equal	<code>-eq</code>	<code>=</code> or <code>==</code>
Not equal	<code>-ne</code>	<code>!=</code>
<code>str1</code> is less than <code>str2</code>		<code>[[str1 < str2]]</code>
<code>str1</code> is greater <code>str2</code>		<code>[[str1 > str2]]</code>
String length is greater than zero		<code>-n str</code>
String length is zero		<code>-z str</code>

> 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
<code>\${#varname}</code>	Length of string
<code>\${varname:pos[:count]}</code>	Substring beginning at <i>pos</i> and ending after <i>count</i> characters (or until the end if count is omitted)
<code>\${varname#[#]pattern}</code>	Remove minimal/maximal matching prefix matching <i>pattern</i>
<code>\${varname%[%]pattern}</code>	Remove minimal/maximal matching suffix matching <i>pattern</i>
<code>\${varname//[/]pattern/ replace}</code>	Replace first occurrence of <i>pattern</i> with <i>replace</i> (all occurrences if <code>//</code>)





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
<code>file_{1,2,3}.txt</code>	<code>file_1.txt file_2.txt file_3.txt</code> (brace expansion)
<code>{1..5}</code>	<code>1 2 3 4 5</code> (sequence expansion)
<code>{a..d}</code>	<code>a b c d</code> (sequence expansion)
<code>\${!varname}</code>	Replaces <code>{!varname}</code> with the value contained in <i>varname</i> , so it becomes <code>\$value</code> (variable indirection)
<code><(command)</code>	Replaces the output of <i>command</i> 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}`

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 !

