جامعة البعث

الكلية التطبيقية-السنة الثالثة -تقنيات حاسوب

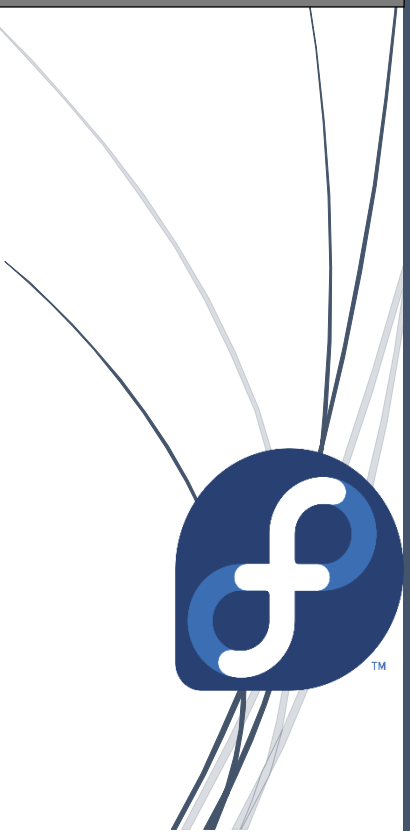
الفصل الأول – القسم العملي

Network Operating Systems 1

Lecture4

2022-2021

Supervised by



البرمجة بلغة مفسّر العمليات Shell
(Programming Shell)

➢ البرمجة النصيّة من نوع Bash shell scripting

- **Bash** هو مُفسّر لغة أوامر. متاح على نطاق واسع في مختلف أنظمة التشغيل وهو مترجم أوامر افتراضيّ في معظم أنظمة Linux. الاسم هو اختصار لـ "**B**ourne-**A**gain **SH**ell"، حيث أن Bash هو تحديث لـBourne Shell القديمة التي تم استخدامها في أنظمة UNIX.
- **Shell** هو معالج صغري أو ببساطة عبارة عن برنامج يسمح بتنفيذ أمر معيّن بشكل تفاعلي أو غير تفاعلي، وإنّ Bash هو ال Shell الافتراضية في توزيعة Fedora.

لمعرفة المُفسّر الافتراضي نكتب التعليمة التالية ضمن ال Terminal:

```
echo $SHELL

/bin/bash
```

أو التعليمة **which bash**

- تسمح البرمجة النصيّة(**Scripting**) بتنفيذ الأوامر تلقائيًا التي كان من الممكن تنفيذها بشكل تفاعلي واحدًا تلو الآخر على محرر الأوامر (Command line).

باستخدام البرمجة النصيّة، يمكن أتمتة أي تفاعل مع shell وبرمجته.

- تُعد Bash Scripting مهارة ضرورية لأي وظيفة إداريّة في نظام Linux على الرغم من أنه قد لا يطلبها صاحب العمل صراحةً.

Note:

Shell scripting is scripting in any shell, whereas Bash scripting is scripting specifically for Bash.

➢ Hello World Bash Shell Script

- نحتاج أولاً إلى معرفة مكان وجود المُفسّر Bash وهو كما رأينا سابقاً **/bin/bash**
- نفتح محرر نصوص (وليكن VI Editor) وننشئ ملفًا يسمى hello_world**.sh**.

```
[ali@Ali Desktop]$ vi hello-world.sh
```

- يبدأ كل نص برمجي (bash shell script) بالرمز shebang: **"#!"** الذي لا يُقرأ كتعليق حيث يتم في السطر الأول كتابة مكان وجود المُفسّر وهو في هذه الحالة المسار /bin/bash/ وتحته نكتب ال scriptالذي نريد:

```
#!/bin/bash

echo "Hello World"

~
```

نحفظ بعدها الملف (wq : أو Shift+zz).

- بعدها لا بد أن نجعل البرنامج النصي (script) قابلاً للتنفيذ باستخدام الأمر chmod مع الخيار +x وبعدها نقوم بتنفيذه باستخدام مسار نسبي **./hello-world.sh**

```
[ali@Ali Desktop]$ chmod +x hello-world.sh
[ali@Ali Desktop]$ ./hello-world.sh
Hello World
```

ملاحظة: أي سطر مسبوق بالرمز # فقط سيكون عبارة عن تعليق.

➢ المتحولات والبارامترات (Variables & Parameters)

- المتحوّل(variable) عبارة عن Parameter يُشار إليه باسم معيّن، ويمكن أن يكون للمتحوّل قيمة أو صفر أو مكوّن من أكثر من واصفة (Attribute). يتم إسناد الواصفات باستخدام أمر التصريح **declare**.
يتم تعيين Parameter إذا تم تعيين قيمة له.
- لنقم بإنشاء برنامج نصيّ جديد welcome.sh بالمحتوى التالي (حيث نجد كيفية استخدام المتحولات والحصول على قيم منها بوضع الرمز $ قبل اسم المتحول):

```
#!/bin/bash

greeting="Welcome"
user=$(whoami)
date=$(date)

echo "$greeting back $user! Today is $day, which is the best day of the entire
week!"
echo "Your Bash shell version is: $BASH_VERSION. Enjoy!
```

فيكون الخرج كالتالي:

```
[ali@Ali Desktop]$ chmod +x welcome.sh
[ali@Ali Desktop]$ ./welcome.sh
 Welcome back ali . Date is: Sun Nov  1 22:46:07 EET 2020 .
Your current version of Bash Shell is: 4.2.10(1)-release . Enjoy!
```

حيث أن المتحوّل BASH_VERSION هو متحوّل داخليّ مُعرَّف كجزء من ال Shell.

ملاحظة:

لا تقم أبدًا بتسمية متحولاتك الخاصة باستخدام الأحرف الكبيرة. هذا لأن أسماء المتحولات الكبيرة محجوزة لمتغيرات shell الداخلية، وأنت تخاطر بالكتابة عليها. قد يؤدي هذا إلى نص برمجي معطّل أو خاطئ النتيجة.

- يمكن أيضًا استخدام المتحوّلات مباشرةً في سطر أوامر ال Terminal. يوضح المثال التالي المتحولين a وb ولهما قيم عددية صحيحة، حيث وباستخدام أمر echo يمكننا طباعة قيمها أو حتى إجراء عملية حسابية كما هو موضح في المثال التالي:

```
[ali@Ali Desktop]$ a=5
[ali@Ali Desktop]$ b=3
[ali@Ali Desktop]$ echo $a
5
[ali@Ali Desktop]$ echo $b
3
[ali@Ali Desktop]$ echo $[$b+$b]
6
[ali@Ali Desktop]$ echo $[$a+$b]
8
```

- تكون الأقواس المتعرجة {} مطلوبة في حال كان المتحوّل متبوعاً بأحرف ليست جزءًا من اسم هذا المتحوّل. سنوضح ذلك باستخدام المثال التالي عن النسخ الاحتياطي لل home directory لمستخدم معيّن:

```
- #!/bin/bash
-
- # This bash script is used to backup a user's home directory to /tmp/.
-
- user=$(whoami)
- input=/home/$user
- output=/tmp/${user}_home_backup_file$(date).tar
-
- tar -cf $output $input
- echo "Backup of $input completed! Details about the output backup file:"
- ls -l $output
```

ونحصل على الخرج التالي:

```
[ali@Ali Desktop]$ chmod +x backup.sh
[ali@Ali Desktop]$ ./backup.sh
tar: Removing leading `/' from member names
Backup of /home/ali Completed. Details about the output backup file:
-rw-rw-r--. 1 ali ali 32819200 Nov  2 00:13 /tmp/ali_home.tar
```

➢ تمرير الوسطاء إلى bash script

يمكن تمرير وسطاء للبرنامج والتعامل معها كأرقام حيث $1 يرمز للوسيط الأول و$2 للوسيط الثاني وهكذا، أو يمكن تخين جميع الوسطاء في مصفوفة خاصة (“$@”)=args والتعامل مع الوسطاء عن طريق الدليل حيث ${args[0]} ترمز للوسيط الأول و ${args[1]} للوسيط الثاني وهكذا.
كما يمكن معرفة عدد هذه الوسطاء عن طريق طباعة $#.

```
#!/bin/bash
# use predefined variables to access passed arguments
#echo arguments to the shell
echo $1 $2 $3 $4
```

```
# We can also store arguments from bash command line in special array
args=("$@")
#echo arguments to the shell
echo ${args[0]} ${args[1]} ${args[2]} ${args[3]}

#use $@ to print out all arguments at once
echo $@

# use $# variable to print out
# number of arguments passed to the bash script
echo Number of arguments passed: $#
```

والناتج كالتالي:

```
[ali@Ali Desktop]$ chmod +x arguments.sh
[ali@Ali Desktop]$ ./arguments.sh hello every one here!
hello every one here!
hello every one here!
hello every one here!
Number of arguments passed: 4
```

➢ تنفيذ أوامر ال Shell باستخدام ال Bash script

يتم ذلك بإحاطة الأمر بعلامة التنصيص من الشكل ` `

```
#!/bin/bash
# use back ticks " ` ` " to execute shell command
echo ` whoami `
# executing bash command without backticks
echo whoami
```

ويكون الناتج كالتالي:

```
[ali@Ali Desktop]$ chmod +x backticks.sh
[ali@Ali Desktop]$ ./backticks.sh
ali
whoami
```

➢ إعادة توجيه الدخل والخرج والأخطاء Input, Output and Error redirection

يمكن إعادة توجيه الخرج الصحيح (stdout) والخرج الذي يحوي أخطاء (stderr) إلى خرجين منفصلين حيث يتم تخزين كل منهما ضمن ملف معيّن.

نستخدم علامة الأكبر > لإعادة توجيه الخرج الصحيح إلى ملف معيّن، ونستخدم الرمز 2> لإعادة توجيه الخرج الذي يحوي أخطاء أو تنبيهات، أما الرمز &> لإعادة توجيه كلا الخرجين إلى نفس الملف.

مثال: إن الرسالة التالية الناتجة عن تنفيذ الأمر tar تُرسل إلى الخرج stderr:

```
tar: Removing leading `/' from member names
```

وسنقوم بإعادة توجيهها إلى المسار /dev/null وهذا يعني تجاهلها **وعدم عرضها:**

```
#!/bin/bash

# This bash script is used to backup a user's home directory to /tmp/.

user=$(whoami)
input=/home/$user
output=/tmp/${user}_home_$(date +%Y-%m-%d_%H%M%S).tar

tar -cf $output $input 2> /dev/null
echo "Backup of $input completed! Details about the output backup file:"
ls -l $output
```

والخرج كالتالي:

```
[ali@Ali Desktop]$ chmod +x redirect.sh
[ali@Ali Desktop]$ ./redirect.sh
Backup of /home/ali completed! Details about the output backup file:
-rw-rw-r--. 1 ali ali 32798720 Nov  2 02:47 /tmp/ali_home_2020-11-02_024701.tar
```

أيضاً يمكن إعادة توجيه الدخل stdin ليكون مثلاً من ملف بدلاً من الدخل الافتراضيّ (وهو لوحة المفاتيح) , ويكون ذلك باستخدام علامة الأصغر <.

في المثال التالي: يتم في أول تعليمة بأخذ الدخل من لوحة المفاتيح (وينتهي الإدخال بالضغط على Ctrl+D) وتوجيه الخرج إلى الملف، أما في التعليمة الثانية يتم إعادة توجيه الدخل ليصبح من الملف file1.txt:

```
~$ cat > file1.txt
 I am using keyboard to input text.
Cat command reads my keyboard input, converts it to stdout which is instantly redirected to file1.txt
That is, until I press CTRL+D
~$ cat < file1.txt
I am using keyboard to input text.
Cat command reads my keyboard input, converts it to stdout which is instantly redirected to file1.txt
That is, until I press CTRL+D
```

➢ قراءة دخل المستخدم

يتم كتابة نص معيّن للتفاعل مع المستخدم باستخدام الأمر echo مع الخيار e– من أجل انتظار دخل من المستخدم أو الخيار a– لقراءة الدخل كمصفوفة، وبعدها وباستخدام الأمر **read** تتم قراءة دخل المستخدم من لوحة المفاتيح.

```
#!/bin/bash

echo -e "Hi, please type the word: \c "
read  word
echo "The word you entered is: $word"
echo -e "Can you please enter two words? "
read word1 word2
echo "Here is your input: \"$word1\" \"$word2\""
echo -e "How do you feel about bash scripting? "
# read command now stores a reply into the default build-in variable $REPLY
read
echo "You said $REPLY, I'm glad to hear that! "
echo -e "What are your favorite colours ? "
# -a makes read command to read into an array
read -a colours
echo "My favorite colours are also ${colours[0]}, ${colours[1]} and ${colours[2]}:-)"
```

فيكون الخرج التفاعليّ كالتالي:

```
[ali@Ali Desktop]$ chmod +x read.sh
[ali@Ali Desktop]$ ./read.sh
Hi, please type the word: hello
The word you entered is: hello
Can you please enter two words?
hi everyone
Here is your input: "hi" "everyone"
How do you feel about bash scripting?
veryGood
You said veryGood, I'm glad to hear that!
What are your favorite colours(Enter 3 Colours. Please) ?
Black Green Blue
My favorite colours are also Black, Green and Blue:-)
```

➢ التوابع Functions

يتم التعريف عن تابع باستخدام الكلمة المفتاحية function.

```
!/bin/bash
# BASH FUNCTIONS CAN BE DECLARED IN ANY ORDER
function function_B {
        echo Function B.
```

```
}
function function_A {
        echo $1
}
function function_D {
        echo Function D.
}
function function_C {
        echo $1
}
# FUNCTION CALLS
# Pass parameter to function A
function_A "Function A."
function_B
# Pass parameter to function C
function_C "Function C."
function_D
```

الخرج:

```
[ali@Ali Desktop]$ chmod +x functions.sh
[ali@Ali Desktop]$ ./functions.sh
Function A.
Function B.
Function C.
Function D.
./functions.sh: line 23: /home/ali: Is a directory
```

- مثال النسخ الاحتياطي لل home directory الخاص بالمستخدم باستخدام مفهوم التوابع مع حساب عدد المجلدات(directories) و الملفات:

```
#!/bin/bash

# This bash script is used to backup a user's home directory to /tmp/.

user=$(whoami)
input=/home/$user
output=/tmp/${user}_home_$(date +%Y-%m-%d_%H%M%S).tar

# The function total_files reports a total number of files for a given directory.
function total_files {
        find $1 -type f | wc -l
}

# The function total_directories reports a total number of directories
# for a given directory.
function total_directories {
        find $1 -type d | wc -l
```

```
}

tar -cf $output $input 2> /dev/null

echo -n "Files to be included:"
total_files $input
echo -n "Directories to be included:"
total_directories $input

echo "Backup of $input completed!"

echo "Details about the output backup file:"
ls -l $output
```

إن الخيار n– بعد الأمر echo من أجل عدم إضافة سطر جديد.
الخرج:

```
[ali@Ali Desktop]$ chmod +x backup2.sh
[ali@Ali Desktop]$ ./backup2.sh
Files to be included:249
Directories to be included:131
Backup of /home/ali completed!
Details about the output backup file:
-rw-rw-r--. 1 ali ali 32870400 Nov  2 07:11 /tmp/ali_home_2020-11-02_071143.tar
```

- **المتحوّلات العامة والمحلية (Global and Local variables)**

إنّ التصريح عن متحوّل عام يكون فقط بكتابة اسم المتحوّل صراحةً حيث سيُعتبر وبشكل افتراضيّ على أنه متحوّل عام، بينما نستخدم الكلمة المفتاحية **local** للتصريح عن المتحوّل المحليّ.
كما هو موضّح بالمثال التالي:

```
#!/bin/bash
#Define bash global variable
#This variable is global and can be used anywhere in this bash script
VAR="global variable"
function bash {
#Define bash local variable
#This variable is local to bash function only
local VAR="local variable"
echo $VAR
}
echo $VAR
bash
# Note the bash global variable did not change
# "local" is bash reserved word
echo $VAR
```

ويكون الخرج كالتالي:

```
[ali@Ali Desktop]$ ./variables.sh
global variable
local variable
global variable
```

➢ Bash Trap Command

يتم تنفيذ تابع bash trap بمجرّد الضغط على Ctrl+C (أي لمقاطعة تنفيذ البرنامج):

```
#!/bin/bash
# bash trap command
trap bashtrap INT



# bash clear screen command
clear;
# bash trap function is executed when CTRL-C is pressed:
# bash prints message => Executing bash trap subrutine !
bashtrap()
{
    echo "CTRL+C Detected !...executing bash trap !"
}
# for loop from 1/10 to 10/10
for a in `seq 1 10`; do
    echo "$a/10 to Exit."
    sleep 1;
done
echo "Exit Bash Trap Example!!!"
```

أثناء التنفيذ نضغط Ctrl+C ويكون الخرج كالتالي:

```
1/10 to Exit.
2/10 to Exit.
3/10 to Exit.
4/10 to Exit.
5/10 to Exit.
^CCTRL+C Detected !...executing bash trap !
6/10 to Exit.
7/10 to Exit.
8/10 to Exit.
9/10 to Exit.
^CCTRL+C Detected !...executing bash trap !
10/10 to Exit.
Exit Bash Trap Example!!!
./trap.sh: line 18: /home/ali: Is a directory
```

حيث وفي كل مرة يتم بها الضغط على Ctrl+C يتم تنفيذ التابع bashtrap() وطباعة العبارة

CTRL+C Detected!...executing bash trap !

➢ expr Command

يتيح لنا استخدام الأمر expr إجراء عملية حسابية حتى بدون إرفاق تعبيرنا الرياضي بين أقواس أو علامات اقتباس. ومع ذلك، لا ننسى الهروب من الضرب بعلامة النجمة لتجنب الخطأ: expr: syntax error وذلك باستخدام backslash.

نوضّح ذلك كالتالي:

```
[ali@Ali Desktop]$ expr 2+2
2+2
[ali@Ali Desktop]$ expr 2 + 2
4
[ali@Ali Desktop]$ expr 5 * 5
expr: syntax error
[ali@Ali Desktop]$ expr 5 \* 5
25
[ali@Ali Desktop]$ expr 6 / 3
2
[ali@Ali Desktop]$ expr 5 - 3
2
```

ملاحظات:

- يتم تجاهل المحارف الخاصة واعتبارها كنص عادي وطباعتها باستخدام علامة التنصيص المفردة Single quotes.

```
#!/bin/bash

 #Declare bash string variable
 BASH_VAR="Bash Script"

 # echo variable BASH_VAR
 echo $BASH_VAR

 # meta characters special meaning in bash is suppressed when  using single quotes
 echo '$BASH_VAR  "$BASH_VAR"'
```

الخرج:

```
[ali@Ali Desktop]$ chmod +x singleQuotes.sh
[ali@Ali Desktop]$ ./singleQuotes.sh
Bash Script
$BASH_VAR  "$BASH_VAR"
```

- لطباعة double quotes كما هي نسبقها ب \ (backslash)

```
-   #!/bin/bash

    #Declare bash string variable
    BASH_VAR="Bash Script"

    # echo variable BASH_VAR
    echo $BASH_VAR

    # meta characters and its special meaning in bash is
    # suppressed when using double quotes except "$", "\" and "`"

    echo "It's $BASH_VAR  and \"$BASH_VAR\" using backticks: `date`"
```

الخرج:

```
[ali@Ali Desktop]$ chmod +x doubleQuotes.sh
[ali@Ali Desktop]$ ./doubleQuotes.sh
Bash Script
It's Bash Script  and "Bash Script" using backticks: Mon Nov  2 05:53:15 EET 2020
```

➢ الحلقات until, for, while

- الحلقات مع for

يمكن تنفيذ الحلقة إما على سطر الأوامر أو وضعها ضمن ملف script.

ونوضح طريقة كتابتها بالمثال التالي:

```
#!/bin/bash

# bash for loop
for f in $( ls /var/ ); do
       echo $f
done
```

يُقرأ كالتالي: من أجل كل عنصر f ضمن العناصر الموجودة ضمن المسار /var/ قم (do) بطباعة هذا العنصر. **لا ننسى الفاصلة المنقوطة بعد شرط ال for، والكلمة done لإنهاء الحلقة.**

ولو أردنا تنفيذها على سطر الأوامر، نكتب كما يلي:

```
$ for f in $( ls /var/ ); do echo $f; done
```

ويكون الخرج:

```
[ali@Ali Desktop]$ chmod +x loop.sh
[ali@Ali Desktop]$ ./loop.sh
account
cache
cvs
db
empty
ftp
games
gdm
lib
```

مثال آخر:

```
#!/bin/bash

for i in 1 2 3; do
    echo $i
done
```

كما يمكن كتابة حلقة for السابقة بشكل مكافئ كما يلي: ;for i in `seq 1 3`

- **الحلقات مع while:**

تشبه الحلقة for لكن مع تحديد شرط التوقف ضمن [] بعد الكلمة while.
مثال: العد التنازليّ بدءاً من رقم محدد.

```
#!/bin/bash
COUNT=5
# bash while loop
while [ $COUNT -gt 0 ]; do
       echo Value of count is: $COUNT
       let COUNT=COUNT-1
done
```

الخرج:

```
[ali@Ali Desktop]$ chmod +x count.sh
[ali@Ali Desktop]$ ./count.sh
Value of count is: 5
Value of count is: 4
Value of count is: 3
Value of count is: 2
Value of count is: 1
```

- **الحلقات مع until:**

```
-   #!/bin/bash
    COUNT=0
```

```
# bash until loop
until [ $COUNT -gt 5 ]; do
        echo Value of count is: $COUNT
        let COUNT=COUNT+1
done
```

الخرج:

```
[ali@Ali Desktop]$ chmod +x until.sh
[ali@Ali Desktop]$ ./until.sh
Value of count is: 0
Value of count is: 1
Value of count is: 2
Value of count is: 3
Value of count is: 4
Value of count is: 5
./until.sh: line 9: /home/ali: Is a directory
```

➢ الجمل الشرطية (case, if)

- Case:

تمكننا من تنفيذ أمر أو جزء برمجي معين بناءً على إدخال المستخدم. بعد تحديد الخيارات المتاحة ضمنها، يتم إنهاؤها بمعكوس الكلمة case وهو esac.

يوضّح المثال التالي استخدام case حيث يدخل المستخدم رقم ويقوم البرنامج بطباعة النص الموافق:

```
#!/bin/bash
echo "What is your preferred programming /scripting language?"
echo "1) bash"
echo "2) perl"
echo "3) phyton"
echo "4) c++"
echo "5) I do not know !"
read case;
#simple case bash structure
# note in this case $case is variable and does not have to
# be named case this is just an example
case $case in
```

```
    1) echo "You selected bash";;
    2) echo "You selected perl";;
    3) echo "You selected phyton";;
    4) echo "You selected c++";;
    5) exit
esac
```

الخرج:

```
[ali@Ali Desktop]$ chmod +x case.sh
[ali@Ali Desktop]$ ./case.sh
What is your preferred programming /scripting language?
1) bash
2) perl
3) phyton
4) c++
5) I do not know !
1
You selected bash
./case.sh: line 20: /home/ali: Is a directory
```

- **if**

الصيغة العامة كالتالي

if [Condition]; then

    Some code

else

    Another code

fi

يجب وجود فراغات بين الشرط وكلا القوسين [] وإلا سيعطي خطأ.

يتم إغلاق كل if بالكلمة المفتاحية fi.

مثال 1:

```
#!/bin/bash

num_a=400
num_b=200

if [ $num_a -lt $num_b ]; then
    echo "$num_a is less than $num_b!"
else
    echo "$num_a is greater than $num_b!"
fi
```

الخرج:

```
[ali@Ali Desktop]$ vim if.sh
[ali@Ali Desktop]$ chmod +x  if.sh
[ali@Ali Desktop]$ ./if.sh
400 is greater than 200!
```

مثال 2: فحص إذا كانت Directory معينة موجودة أم لا.

```
#!/bin/bash
directory="./BashScriptingDirectory"

# bash check if directory exists
if [ -d $directory ]; then
      echo "Directory exists"
else
      echo "Directory does not exists"
fi
```

مثال 3:

يمكن إعطاء منطق أكثر لبرنامج النسخ الاحتياطي(backup2.sh) في الفقرات السابقة كالتالي:

IF the number of files between the source and destination target is equal THEN print the OK message, ELSE, print ERROR.

كما هو موضّح بال script التالي:

```
#!/bin/bash

user=$(whoami)
input=/home/$user
output=/tmp/${user}_home_$(date +%Y-%m-%d_%H%M%S).tar.gz

function total_files {
        find $1 -type f | wc -l
}

function total_directories {
        find $1 -type d | wc -l
}

function total_archived_directories {
        tar -tzf $1 | grep  /$ | wc -l
```

```
}

function total_archived_files {
        tar -tzf $1 | grep -v /$ | wc -l
}

tar -czf $output $input 2> /dev/null

src_files=$( total_files $input )
src_directories=$( total_directories $input )

arch_files=$( total_archived_files $output )
arch_directories=$( total_archived_directories $output )

echo "Files to be included: $src_files"
echo "Directories to be included: $src_directories"
echo "Files archived: $arch_files"
echo "Directories archived: $arch_directories"

if [ $src_files -eq $arch_files ]; then
        echo "Backup of $input completed!"
        echo "Details about the output backup file:"
        ls -l $output
else
        echo "Backup of $input failed!"
fi
```

الخرج:

```
[ali@Ali Desktop]$ chmod +x backup3.sh
[ali@Ali Desktop]$ ./backup3.sh
Files to be included: 252
Directories to be included: 131
Files archived: 256
Directories archived: 131
Backup of /home/ali failed!
```

Nested if/else -

سنرى في ال script التالي مثال عن عبارات if الشرطية المتداخلة ببعضها البعض:

```
#!/bin/bash

# Declare variable choice and assign value 4
choice=4
# Print to stdout
 echo "1. Bash"
 echo "2. Scripting"
 echo "3. Tutorial"
```

```
 echo -n "Please choose a word [1, 2 or 3]?”
# Loop while the variable choice is equal 4
# bash while loop
while [ $choice -eq 4 ]; do

# read user input
read choice
# bash nested if/else
if [ $choice -eq 1 ] ; then

        echo "You have chosen word: Bash"

else

        if [ $choice -eq 2 ] ; then
                 echo "You have chosen word: Scripting"
        else

                if [ $choice -eq 3 ] ; then
                        echo "You have chosen word: Tutorial"
                else
                        echo "Please make a choice between 1-3 !"
                        echo "1. Bash"
                        echo "2. Scripting"
                        echo "3. Tutorial"
                        echo -n "Please choose a word [1,2 or 3]? "
                        choice=4
                fi
        fi
fi
done
```

الخرج:

```
[ali@Ali Desktop]$ chmod +x nestedif.sh
[ali@Ali Desktop]$ ./nestedif.sh
1. Bash
2. Scripting
3. Tutorial
Please choose a word [1,2 or 3]? 7
Please make a choice between 1-3 !
1. Bash
2. Scripting
3. Tutorial
Please choose a word [1,2 or 3]? 1
You have chosen word: Bash
```

➢ مقارنة الأرقام والسلاسل (Numeric and String Comparisons)

- **مقارنة الأرقام:**

Symbol	Meaning
-lt	<
-gt	>
-le	<=
-ge	>=
-eq	==
-ne	!=

مثال 1:

```
#!/bin/bash
# declare integers
NUM1=2
NUM2=2
if [ $NUM1 -eq $NUM2 ]; then
      echo "Both Values are equal"
else
      echo "Values are NOT equal"
fi
```

مثال 2:

```
#!/bin/bash
# declare integers
NUM1=2
NUM2=1
if    [ $NUM1 -eq $NUM2 ]; then
      echo "Both Values are equal"
elif [ $NUM1 -gt $NUM2 ]; then
      echo "NUM1 is greater than NUM2"
else
      echo "NUM2 is greater than NUM1"
fi
```

إن elif اختصار ل else+if ونحدد ضمنها الشرط المراد تحقيقه.

- **مقارنة السلاسل:**

Symbol	Meaning
=	**Equal**
!=	**not equal**
<	**less then**
>	**greater then**
-n s1	**string s1 is not empty**
-z s1	**string s1 is empty**

مثال 1:

```
#!/bin/bash
#Declare string S1
S1="Bash"
#Declare string S2
S2="Scripting"
if [ $S1 = $S2 ]; then
      echo "Both Strings are equal"
else
      echo "Strings are NOT equal"
fi
```

سواءً كانت المقارنة بين سلسلتين أو رقمين فإن الناتج 0 يدل على True والناتج 1 يدل على False.

مثال 2:

```
#!/bin/bash

string_a="UNIX"
string_b="Linux"

echo "Are $string_a and $string_b strings equal?"
[ $string_a = $string_b ]
echo $?

num_a=100
num_b=100

echo "Is $num_a equal to $num_b ?"
[ $num_a -eq $num_b ]
echo $?
```

نلاحظ استخدام الأمر $? echo دائماً لعرض ناتج المقارنة.

الخرج:

```
[ali@Ali Desktop]$ chmod +x compare.sh
[ali@Ali Desktop]$ ./compare.sh
Are UNIX and GNU strings equal?
1
Is 100 equal to 100 ?
0
```

➢ اختبار الملفات (Bash File Testing)

Symbol	Meaning
-b filename	**Block special file**
-c filename	**Special character file**
-d directoryname	**Check for directory existence**
-e filename	**Check for file existence**
-f filename	**Check for regular file existence not a directory**
-G filename	**Check if file exists and is owned by effective group ID.**
-g filename	**True if file exists and is set-group-id.**
-O filename	**True if file exists and is owned by the effective user id.**
-r filename	**Check if file is a readable**
-s filename	**Check if file is nonzero size**
-w filename	**Check if file is writable**
-x filename	**Check if file is executable**

مثال 1:

```
#!/bin/bash
file="./file"
if [ -e $file ]; then
      echo "File exists"
else
      echo "File does not exists"
fi
```

مثال 2:

يمكننا استخدام حلقة while للتحقّق مما إذا كان الملف غير موجود. سيبقى هذا البرنامج في وضع sleep حتى يصبح الملف موجوداً.

```
#!/bin/bash

while [ ! -e myfile ]; do
# Sleep until file does exists/is created
sleep 1
done
```

➢ العمليات الحسابية

ملاحظة: - لتعريف متحول من نمط Integer **بشكل صريح** نكتب كالتالي: declare **–i** *variable_name*

-يقوم الأمر let بتنفيذ العملية الرياضية وتخزين الناتج ضمن متحوّل محدد.

- **الجمع**

```
-   #!/bin/bash

    let RESULT1=$1+$2
    echo $1+$2=$RESULT1 ' -> # let RESULT1=$1+$2'
    declare -i RESULT2
    RESULT2=$1+$2
    echo $1+$2=$RESULT2 ' -> # declare -i RESULT2; RESULT2=$1+$2'
    echo $1+$2=$(($1 + $2)) ' -> # $(($1 + $2))'
```

الخرج:

```
[ali@Ali Desktop]$ chmod +x add.sh
[ali@Ali Desktop]$ ./add.sh 10 20
10+20=30 -> # let RESULT1=$1+$2
10+20=30  -> # declare -i RESULT2; RESULT2=$1+$2
10+20=30  -> # $(($1 + $2))
```

- **فيما يلي مثال كامل للتعامل مع العمليات والرموز الحسابية:**

```
-   #!/bin/bash

    echo '### let ###'
    # bash addition
    let ADDITION=3+5
    echo "3 + 5 =" $ADDITION

    # bash subtraction
    let SUBTRACTION=7-8
    echo "7 - 8 =" $SUBTRACTION

    # bash multiplication
    let MULTIPLICATION=5*8
    echo "5 * 8 =" $MULTIPLICATION

    # bash division
    let DIVISION=4/2
    echo "4 / 2 =" $DIVISION

    # bash modulus
    let MODULUS=9%4
    echo "9 % 4 =" $MODULUS

    # bash power of two
```

```
let POWEROFTWO=2**2
echo "2 ^ 2 =" $POWEROFTWO


echo '### Bash Arithmetic Expansion ###'
# There are two formats for arithmetic expansion: $[ expression ]
# and $(( expression #)) its your choice which you use

echo 4 + 5 = $((4 + 5))
echo 7 - 7 = $[ 7 - 7 ]
echo 4 x 6 = $((3 * 2))
echo 6 / 3 = $((6 / 3))
echo 8 % 7 = $((8 % 7))
echo 2 ^ 8 = $[ 2 ** 8 ]

echo '### Declare ###'
echo -e "Please enter two numbers \c"
# read user input
read num1 num2
declare -i result
result=$num1+$num2
echo "Result is:$result "

# bash convert binary number 10001
result=2#10001
echo $result

# bash convert octal number 16
result=8#16
echo $result

# bash convert hex number 0xE6A
result=16#E6A
echo $result
```

الخرج:

```
[ali@Ali Desktop]$ chmod +x calculation.sh
[ali@Ali Desktop]$ ./calculation.sh
### let ###
3 + 5 = 8
7 - 8 = -1
5 * 8 = 40
4 / 2 = 2
9 % 4 = 1
2 ^ 2 = 4
### Bash Arithmetic Expansion ###
4 + 5 = 9
7 - 7 = 0
4 x 6 = 6
6 / 3 = 2
8 % 7 = 1
2 ^ 8 = 256
### Declare ###
Please enter two numbers 5 3
Result is:8
17
14
3690
```

➢ التعامل مع المصفوفات

- **إنشاء مصفوفة:**

يمكننا صراحةً إنشاء مصفوفة باستخدام الأمر declare:

```
declare -a my_array
```

حيث يدل الخيار a– لإنشاء مصفوفة مفهرسة (indexed array).

يمكننا إنشاء مصفوفات مفهرسة بصيغة أكثر إيجازًا، ويتم ذلك ببساطة عن طريق إسناد بعض القيم لها مباشرةً:

```
my_array= (value1 value2)
```

في الحالة السابقة، قمنا بإسناد عدة عناصر دفعة واحدة للمصفوفة، ولكن يمكننا أيضًا إدراج قيمة واحدة في كل مرة، مع تحديد دليلها:

```
my_array[0] =value1
```

- **طباعة قيم مصفوفة:**

لعرض جميع قيم المصفوفة، يمكننا استخدام الصيغة التالية:

```
echo ${my_array[@]}
```

أو بالصيغة التالية:

```
echo ${my_array[*]}
```

ويمكن استخدام حلقة for ضمن سطر الأوامر لطباعة كل عناصر المصفوفة كالتالي:

```
for i in "${my_array[@]}"; do echo "$i"; done
```

```
[ali@Ali Desktop]$ my_array=(aa bb)
[ali@Ali Desktop]$ for i in "${my_array[@]}"; do echo "$i"; done
aa
bb
[ali@Ali Desktop]$ for i in ${my_array[@]}; do echo $i; done
aa
bb
```

عند استخدام *، ويكون المتحوّل الخاص بالمصفوفة محاطاً بإشارات اقتباس، سيتم إعطاء "نتيجة" واحدة، تحتوي على جميع عناصر المصفوفة (أي عرض جميع عناصر المصفوفة ضمن سطر واحد):

```
[ali@Ali Desktop]$ for i in ${my_array[*]}; do echo $i; done
aa
bb
[ali@Ali Desktop]$ for i in "${my_array[*]}"; do echo "$i"; done
aa bb
```

- **طباعة الأدلة ضمن مصفوفة:**

الطريقة مشابهة لطريقة طباعة قيم عناصر المصفوفة ولكن هنا يتم استخدام الرمز **!** قبل متحوّل المصفوفة:

```
[ali@Ali Desktop]$ my_array=(A B C)
[ali@Ali Desktop]$ for index in "${!my_array[@]}"; do echo "$index"; done
0
1
2
```

- **حساب حجم مصفوفة:**

يتم باستخدام الإشارة # قبل متحوّل المصفوفة:

```
[ali@Ali Desktop]$ my_array=(A B C)
[ali@Ali Desktop]$ echo "The array contains ${#my_array[@]} elements"
```

- **إضافة عناصر إلى مصفوفة:**

تتم باستخدام المُعامل =+.

```
[ali@Ali Desktop]$ my_array=(A B C)
[ali@Ali Desktop]$ my_array+=(D)
```

كما يمكن إضافة عدة عناصر سويةً كالتالي:

```
[ali@Ali Desktop]$ my_array=(A B C)
[ali@Ali Desktop]$ my_array+=(D E F)
[ali@Ali Desktop]$ echo "${my_array[@]}"
A B C D E F
```

- **حذف عنصر من المصفوفة:**

يجب معرفة دليل العنصر المُراد حذفه أولاً ثمّ نستخدم الأمر unset:

```
[ali@Ali Desktop]$ my_array=(A B C D E F)
[ali@Ali Desktop]$ unset my_array[1]
[ali@Ali Desktop]$ echo "${my_array[@]}"
A C D E F
[ali@Ali Desktop]$ echo "${!my_array[@]}"
0 2 3 4 5
```

- **حذف مصفوفة:**

فقط نمرر اسم المصفوفة كوسيط للأمر unset:

```
[ali@Ali Desktop]$ unset my_array
[ali@Ali Desktop]$ echo "${!my_array[@]}"

[ali@Ali Desktop]$ echo "${my_array[@]}"

```

ونلاحظ حصولنا على نتيجة فارغة عند طلب أدلة المصفوفة أو قيم عناصرها.

Questions???