

Shell script part 3

Loops

1- for loop:

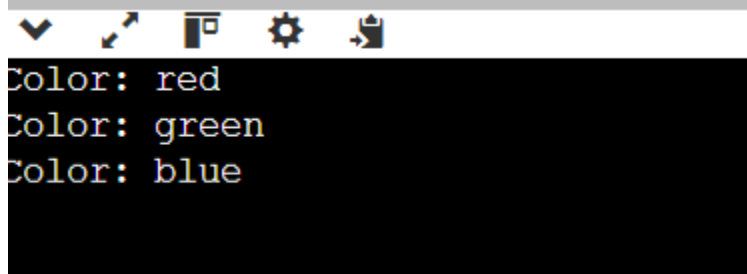
- Syntax:

A- Iterating Over a List

```
for var in word1 word2 ... wordn
do
    command
    command
    ...
done
```

Example:

```
# # Using a list
for color in red green blue
do
    echo "Color: $color"
done
```



A terminal window with a dark background and a toolbar at the top. The toolbar contains icons for a dropdown menu, a cursor, a window, a gear (settings), and a clipboard. The terminal output shows the execution of the script from the example above, with the word 'Color:' in red and the color names in blue.

```
Color: red
Color: green
Color: blue
```

```
#!/bin/sh
echo Number of arguments passed is $#
for arg in $*
do
echo $arg
done
```

```
ahed@DESKTOP-OK5G6FV:~/ahed$ ./test.sh 1 2 3 mohammad
Number of arguments passed is 4
1
2
3
mohammad
ahed@DESKTOP-OK5G6FV:~/ahed$
```

B- Iterating Over a Range of Numbers (with **bash** shell)

```
for i in {start..end..step}
do
    # Commands to execute for each number
done
```

Example :

```
# Using brace expansion
for i in {1..5}
do
    echo "Number: $i"
done
```

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

```
# Using brace expansion
for i in {1..5..2}
do
    echo "Number: $i"
done
```

```
Number: 1
Number: 3
Number: 5
```

C- The for Without the List

- اذا ما اعطيتها list رح تعمل loop على arguments :

```
echo "the number of args = $#"  
for i  
do  
    echo "$i"  
done
```

```
ahed@DESKTOP-OK5G6FV:~/ahed$ ./test.sh 1 2 3  
the number of args = 3  
1  
2  
3  
ahed@DESKTOP-OK5G6FV:~/ahed$
```

- في شكل ثاني ل for loop الي هو :

```
#!/bin/bash  
  
for ((i=1; i<=5; i+=1))  
do  
    echo "Number: $i"  
done
```

- لكن هذا الشكل لا يعمل الا مع bash shell ولا يعمل مع sh shell
- عشان احدد نوع shell بستخدم شي اسمه shebang (!#) وهاي بتكون اول سطر في shell script :

```
#!/bin/bash  
  
for ((i=1; i<=5; i+=1))  
do  
    echo "Number: $i"  
done
```

- اذا غيرتها ل sh مش رح تشتغل لانه هاذ الشكل (syntax) مش معروف ب sh :

```
#!/bin/sh

for ((i=1; i<=5; i+=1))
do
    echo "Number: $i"
done
```

● طبعا ال default shell هي bash

2- while

```
i=1
while [ "$i"-le 5 ]
do
    echo $i
    i=$((i + 1))
done
```

```
1
2
3
4
5
```

3- until:

● هي عكس while يعني ما دام الشرط خطأ (برجع exit status < 0) بضل تشتغل:

```
until [CONDITION]
do
    # Commands to execute
done
```

```
8 # Initialize a variable
9 count=1
10
11 # Until loop that runs until count is greater than
12 # 5 | ضلك نفذ ب لوب لحتى يصير المتغير اكبر من 5
13 until [ $count -gt 5 ]
14 do
15     echo "Count: $count"
16     count=$((count+1)) # Increment the count
17 done
18
19 echo "Loop finished!"
```

```
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
Loop finished!
```

4- break and continue:

- break:

```
7
8 for i in {1..10}
9 do
10     if [ $i -eq 5 ]
11     then
12         echo "Breaking the loop at $i"
13         break
14     fi
15     echo "Number: $i"
16 done
17
18 echo "Loop exited."
19
```

Number: 1
Number: 2
Number: 3
Number: 4
Breaking the loop at 5
Loop exited.

- break n :

معناها اطلع من loop n يعني اذا في 3 loops في جوا بعض nested
وحطيت break 2 رح يطلع من inner loop 2

```

7 for i in {1..3}
8 do
9     echo "Outer loop: $i"
10    for j in {1..5}
11    do
12        if [ $j -eq 3 ]; then
13            echo "Breaking out of the outer loop from inner loop when j is $j"
14            break 2 # Break out of both loops
15        fi
16        echo "Inner loop: $j"
17    done
18 done
19
20 echo "Exited all loops."
21

```

input

```

er loop: 1
er loop: 1
er loop: 2
aking out of the outer loop from inner loop when j is 3
ted all loops.

```

- continue:


```
8 for i in {1..10}
9 do
10     if [ $i -eq 5 ]; then
11         echo "Skipping number $i"
12         continue
13     fi
14     echo "Number: $i"
15 done
16
17 echo "Loop finished."
18
```

Number: 2
Number: 3
Number: 4
Skipping number 5
Number: 6
Number: 7
Number: 8
Number: 9
Number: 10
Loop finished.

- `continue n` : break نفس مبدأ
- Summary:
 - **break n**: Exits from the **n**-th enclosing loop.
 - **continue n**: Skips to the next iteration of the **n**-th enclosing loop.

5- getopt :

- هو shell command يستخدمه عشان لعمل option ل shell script
- طبعا بعطي الاوبشن من خلال arguments
- لازم استخدمه مع loop

- syntax :

`getopts optstring variable`

- optstring:

بستخدمها عشان اعرف ال option ل shell script
 يكون عبارة عن string ، كل char في string يعبر عن option ، واذا
 بدي اخلي option يوخذ argument بحط وراه (:)
 مثلا "a:b" معناها انه في 2 options الي هم a و b ، و a بتوخذ
 argument لانه وراها (:)

- variable:

رح يكون فيه قيم option الحالي الذي يتم معالجته

- في كمان 2 variables الي هم :

OPTIND: بتكون فيه قيمة الاندكس ل الاوبشن التالي (with **bash** shell)

OPTARG: ل الاوبشن الحالي arg بتكون فيه قيمة

Example :

```
while getopts "abc:" flag
do
echo "$flag" $OPTIND $OPTARG
done
```

- في هاد المثال بحكيه في 3 options الي هم (a,b,c) وكمان c بتوخذ argument

- في كل لفه في اللوب بمسكهم اوبشن اوبشن ف ببلش ب a ، اذا انا معطيه اوبشن a رح ينفذ السطر التالي :

`echo "$flag" $OPTIND $OPTARG`

- طبعا flag الي هو بكون في الاوبشن a
- و OPTIND هو الاندكس ل الاوبشن الي بعد a ،

- و OPTARG هو قيم argument ل الاوبشن a لكن اوبشن a ما بوخذ argument ف مش رح يطبع شي
- خلينا نشغل script ونشوف الاوتبوت :

```
ahed@DESKTOP-OK5G6FV:~/ahed$ ./test.sh -abc "soso"
a 1
b 1
c 3 soso
ahed@DESKTOP-OK5G6FV:~/ahed$
```

- لما شغلت script مررت الاوبشن هيـك -abc ويمكن تحطو كل وحده لحال
- We can use the options -a, -b, and -c with an argument like this: -a -b -c "soso". However, since options -a and -b do not require arguments, we can also combine them and write it as -abc "soso"
- كيف ينفذ الكود؟ اول شي بفحص بالكود اوبشن a وبشوف هل انا مررت اوبشن a
- اه انا مررت اوبشن a لما شغلت script ف بروح بدخل بجمله الطباعة echo
- ف بطبع flag الي بكون فيه قيمة الاوبشن الحالي وهو a
- وبعدين بطبع قيمة OPTIND الي هو الاندكس للاوبشن الي بعد a وهو اوبشن b ، وطبعاً اوبشن b موجود ب اندكس 1 ، ليش؟

```
ahed@DESKTOP-OK5G6FV:~/ahed$ ./test.sh -abc "soso"
a 1
b 1
c 3 soso
ahed@DESKTOP-OK5G6FV:~/ahed$
```

- لانه b موجود في argument الاولى
- لما يوصل عند c ، ايش الاندكس للاوبشن الي بعدها ، اكيد مش 2 لانه الاندكس 2 هو عبارة عن argument لاوبشن c ف عشان هيـك هو بتوقع يجي كمان اوبشن في اندكس 3 ف عشان هيـك عند c بطبع 3

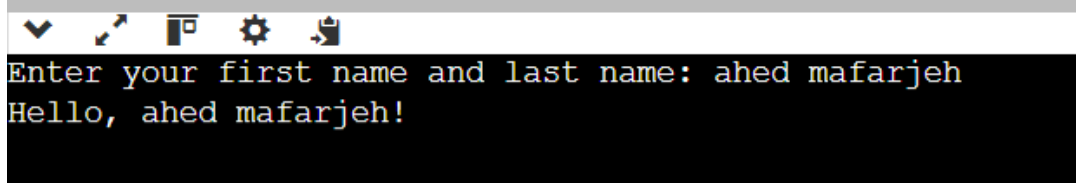
6- read : يستخدمها عشان اقرأ قيمة من اليوزر او من ملف

```
5
6 # Prompt the user for their name
7 echo -n "Enter your name: "
8 read name
9
10 # Prompt the user for their age
11 echo -n "Enter your age: "
12 read age
13
14 # Display the information back to the user
15 echo "Hello, $name! You are $age years old."
16
```

```
Enter your name: ahmad
Enter your age: 20
Hello, ahmad! You are 20 years old.
```

- في هاد المثال قرأنا من اليوزر الاسم والعمر وخرناهم في name, age وبعدين طبعناهم
- و ممكن في نفس السطر اقرأ اكثر من قيمة واحطهم في variables :

```
7
8 # Prompt for multiple inputs
9 echo -n "Enter your first name and last name: "
10 read first last
11
12 # Display the full name
13 echo "Hello, $first $last!"
14
```



Enter your first name and last name: ahed mafarjeh
Hello, ahed mafarjeh!

- ويمكن تخفي الانبوت الي بكتبه اليوزر في حال مثلا بده يكتب باسورد
read -s) (with **bash** shell):)

Prompt the user for a password

```
-n "Enter your password: "  
-s password
```

Optional: Ask for confirmation

```
-n "Confirm your password: "  
-s confirm_password
```

Check if the passwords match

```
"$password" = "$confirm_password" ]
```

```
echo -e "\nPassword successfully set."
```

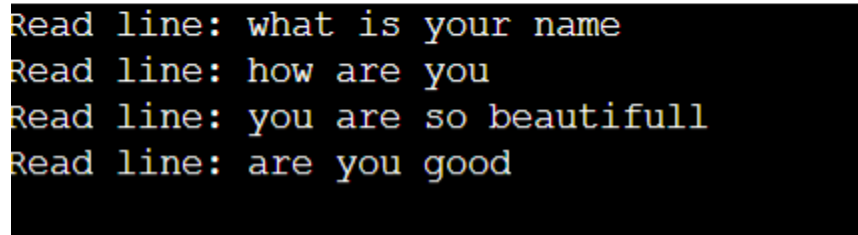
```
echo -e "\nPasswords do not match. Please try again."
```



```
password:  
ur password:  
uccessfully set.
```

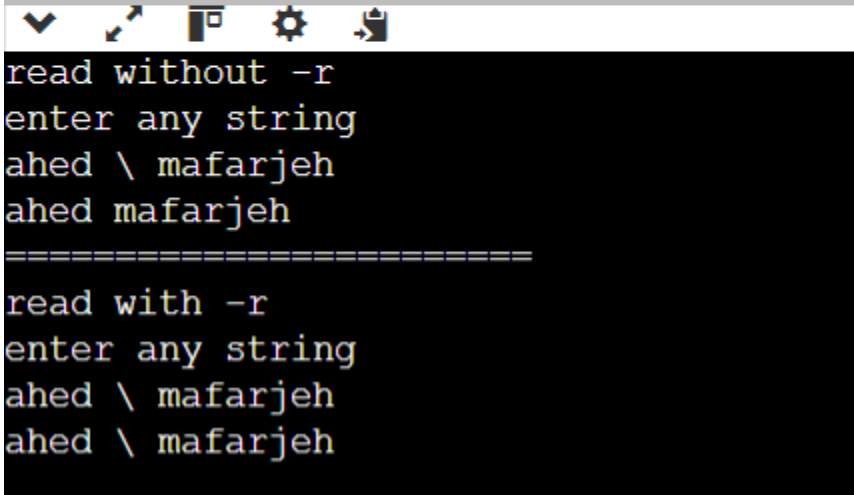
● منقدر نستخدمها عشان نقرأ ملف كمان :

```
6 filename="file.txt"
7
8 while read -r line; do
9     echo "Read line: $line"
10 done < "$filename"
```



- رح يشوفها ك حرف escape يعني اذا لقي \ مش رح يعتبرها : -r عادي

```
5 echo "read without -r"
6 echo "enter any string"
7 read str
8 echo $str
9 echo "=====
10 echo "read with -r"
11 echo "enter any string"
12 read -r str
13 echo $str
```



```
read without -r
enter any string
ahed \ mafarjeh
ahed mafarjeh
=====
read with -r
enter any string
ahed \ mafarjeh
ahed \ mafarjeh
```

7- sleep :

- من خلالها بقدر اعمل delay او يعني رح يوقف تنفيذ code لفترة زمنية معينة
- Examples :

```
echo "Sleeping for 5 seconds..."
sleep 5
echo "Done!"
```



```
echo "Sleeping for 1 minute..."  
sleep 1m  
echo "Done!"
```

```
echo "Sleeping for 2 hours..."  
sleep 2h  
echo "Done!"
```

```
echo "Sleeping for 1 day..."  
sleep 1d  
echo "Done!"
```

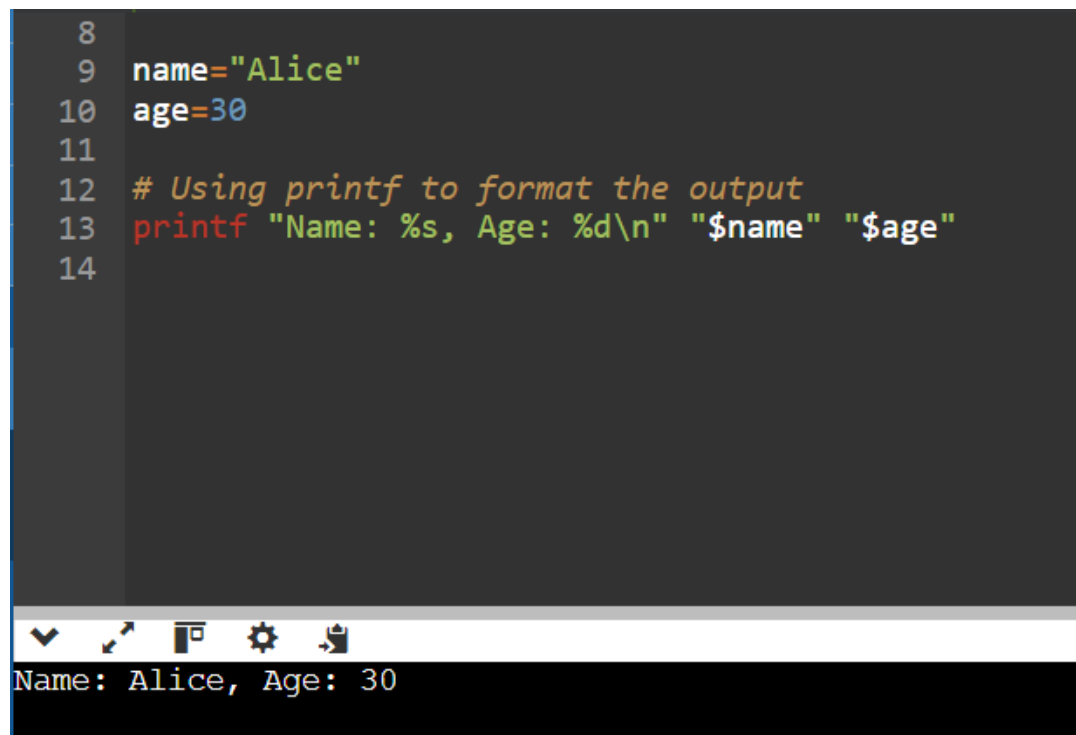
8- printf :

- بتستخدمها عشان اطبع اي شي نفس echo
- لكن الي بميزها عن echo انه بقدر اتحكم في شكل output

Common Format Specifiers

- ``%s``: String
 - ``%d``: Integer
 - ``%f``: Floating-point number
 - ``%x``: Hexadecimal number
 - ``%c``: Character
- Examples :

```
8
9  name="Alice"
10 age=30
11
12 # Using printf to format the output
13 printf "Name: %s, Age: %d\n" "$name" "$age"
14
```



Name: Alice, Age: 30

```
10 price=19|
11 quantity=3
12
13 # Calculate total price
14 total=$(( $price * $quantity ))
15
16 # Print formatted output
17 printf "Price: \$.2f, Quantity: %d, Total: \$.2f\n"
18
```

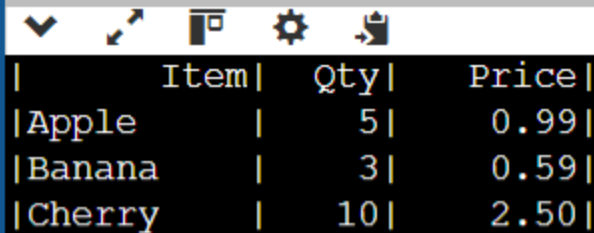


```
Price: $19.00, Quantity: 3, Total: $57.00
```

```

8 # Print header with specified formatting
9 printf "|%10s|%5s|%8s|\n" "Item" "Qty" "Price"
10
11 # Print items with formatted quantities and prices
12 printf "|%-10s|%5d|%8.2f|\n" "Apple" 5 0.99
13 printf "|%-10s|%5d|%8.2f|\n" "Banana" 3 0.59
14 printf "|%-10s|%5d|%8.2f|\n" "Cherry" 10 2.50
15

```



Item	Qty	Price
Apple	5	0.99
Banana	3	0.59
Cherry	10	2.50

Tasks)

1. Write a shell script that accepts an unknown number of integer arguments and calculates their average.
2. Write a shell script that checks the current minutes every 3 seconds. If the minutes exceed 30, it will print the current time in the format (hour:min:sec).

3. Write a shell script that accepts an unknown number of inputs from the user and calculates their average. The script will print the average when the user enters 0.
4. Write a shell script that accepts four options: ``-a``, ``-s``, ``-n1``, and ``-n2``. If the ``-a`` option is specified, the script will print the sum of the values for ``n1`` and ``n2``. If the ``-s`` option is specified, the script will print the subtraction of ``n1`` from ``n2``.
5. Write a shell script that reads a list of products along with their prices and quantities from the user, then prints a formatted table displaying this information along with the total price for each product. The script should also print the grand total at the end.
6. Write a shell script that takes a string input from the user and splits it into individual characters.
7. Develop a shell script to output the odd lines from a particular file (without using the read command).
8. Write a shell script that creates a backup of a specified directory. The backup should be stored in a time stamped directory, and it should include all subdirectories and files.