

19205

Виктор Асенов

Shell Scripting



Съдържание

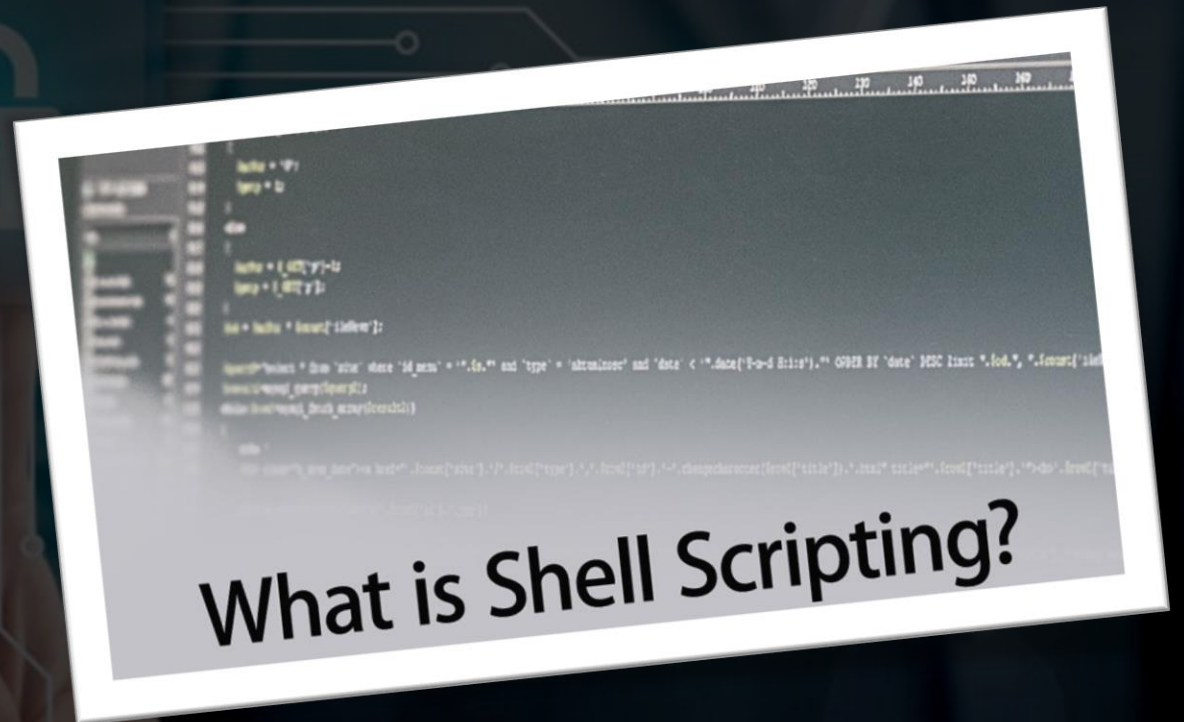
- Основи на Bash scripting
- Променливи, условни оператори, цикли, масиви, функции.
 - Аргументи
- Примерни програми и практически примери

```
1  #!/bin/bash
2  #INPUT_SAMPLE_LIST=$1
3  cd /Volumes/PhilDrive_EMS/TestDec7/snv_postprocess/
4  ...
11 . paths.txt
12 ...
30
31 echo "Debug level set for $DEBUG_LEVEL"
32 echo "log found in scripts directory"
33 ...
50 cp $HIGH_SNP_OUT ./
51 cp $LOW_SNP_OUT ./
52 cp $GERM_SNP_OUT ./
53 # echo "${SCRIPT_DIR}/run_somatic_mutation_analysis ${i} no_false_snp"
54 if [ $DEBUG_LEVEL -gt 0 ]
55 then
56 echo "INFO: ${SCRIPT_DIR}run_somatic_mutation_analysis.sh $SAMPLE no_false_snp
57 `basename ${LOW_SNP_OUT}` `basename ${GERM_SNP_OUT}` `basename ${HIGH_SNP_OUT}`
58 ${D_BAM_FILE} ${G_BAM_FILE}\n">${LOG}
59
60 fi
61 ${SCRIPT_DIR}run_somatic_mutation_analysis.sh
62
63 echo "End of somatic mutation analysis">> $LOG
```

Основи на Bash scripting

1. Shell scripting:

- Намира голямо приложение в автоматизацията на процеси и влияние върху операционната система;
- Представява създаване на файл, съдържащ серия от команди/инструкции, които могат да бъдат изпълнени едновременно;
- Bash е команден интерфейс и интерпретатор за shell scripting;
- Важен инструмент за администратори и разработчици;





Основи на Bash scripting

2. Предимства на Bash scripting:

- Автоматизация на повторяеми задачи и процеси, спестявайки време и редуциране на риска от грешки при ръчното изпълнение;
- Портативност – могат да бъдат изпълнени на всички платформи;
- Гъвкавост – лесно могат да бъдат модифицирани според нуждите и съчетани с други програмни езици;
- Достъпност – не са ни необходими специални инструменти и софтуери, за да пишем bash script-ове, необходим е само един текстов редактор;
- Интеграция – може да бъде интегриран с други инструменти и приложения (напр. дата-бази, уеб сървъри и др.);

Основи на Bash scripting:

Setup-ване и изпълняване на Bash script

- Има 3 основни стъпки, които се изпълняват, за да работи успешно всеки скрипт:
 1. Създаваме файл с разширение **.sh** – идентифицира файла като shell script - фиг. 1
 2. Добавяне на "**shebang**„ - задава кой интерпретатор да се стартира за скрипта ни. Слага се на първия ред (първа инструкция в скрипта) - фиг. 2 и 3
 3. Добавяне на права на скрипта - фиг. 4
- За да изпълним скрипт пишем: **bash scriptName.sh** или **./scriptName.sh**

```
rokyuto@rokyuto:~$ bash script.sh
Hello UKTC
rokyuto@rokyuto:~$ ./script.sh
Hello UKTC
rokyuto@rokyuto:~$
```

Основи на Bash scripting:

Setup-ване и изпълняване на Bash script

```
rokyuto@rokyuto:~$ gedit script.sh &  
[1] 4000  
rokyuto@rokyuto:~$
```

Фиг. 1

* Команда **gedit** – чрез аргумента **&** отваряме във фонов режим текстовият редактор **gedit** и създаваме файл по наше усмотрение

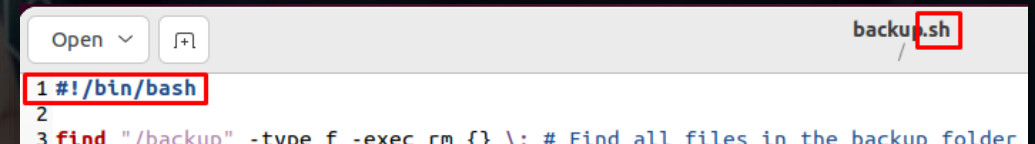
```
rokyuto@rokyuto:~$ sudo chmod a+x script.sh  
[sudo] password for rokyuto:
```

Фиг. 4

* Команда **chmod** – променяме правата на подаденият като аргумент файл или директория
chmod [REFERENCE] [OPERATOR] [MODE] FILE

Shebang	Interpreter
<code>#!/bin/bash</code>	Bash
<code>#!/bin/sh</code>	Bourne shell
<code>#!/usr/bin/env <interpreter></code>	Uses the <code>env</code> program to locate the interpreter. Use this shebang for other scripting languages, such as Perl , Python, etc.
<code>#!/usr/bin/pwsh</code>	Powershell

Фиг. 2



```
Open  backup.sh  
1 #!/bin/bash  
2  
3 find "/backup" -type f -exec rm {} \; # Find all files in the backup folder
```

Фиг. 3

Променливи, условни оператори, цикли, масиви, функции: Променливи

```
variable_name = <variable data>
```

- Името на променливата може да съдържа:
 - Букви: **a-z, A-Z**
 - Цифри: **0-9**
 - Специален символ „_“
- Като името трябва да започва с буква или „_“
- 1. Дефиниране на променлива:
 - Името на променливата, символът „=“ и стойността, която ѝ задаваме трябва да са слепени;

```
Open  ▾  [icon]  scr1.sh  ~/
1  #!/bin/bash
2
3  name="Viktor"
4  age=18
5
6  |
```

• Valid Variable Names

```
ABC
_AV_3
AV232
```

• Invalid variable names

```
2_AN
!ABD
$ABC
&QAID
```

Променливи, условни оператори, цикли, масиви, функции:

Променливи

2. Достъпване на променлива:

- Пред името на променливата се слага символът **\$**

```
Open ▾ [⌘] *scr1.sh ~/
1 #!/bin/bash
2
3 name="Viktor"
4 age=18
5
6 echo $name
7 echo $age
8
9
```

```
rokyuto@rokyuto:~$ ./scr1.sh
Viktor
18
rokyuto@rokyuto:~$
```

3. Изключване на променлива:

- Става чрез командата **unset**
- Представява изтриване на променливата и нейната стойност
- Синтаксис: **unset variableName**
- * При подаване на променливата НЕ се слага символът **\$**

```
Open ▾ [⌘] *scr1.sh ~/
1 #!/bin/bash
2
3 name="Viktor"
4 age=18
5
6 echo $name $age
7
8 unset age
9
10 echo $name $age
11
```

```
rokyuto@rokyuto:~$ ./scr1.sh
Viktor 18
Viktor
rokyuto@rokyuto:~$
```


Променливи, условни оператори, цикли, масиви, функции:

Променливи

```
read <options> <arguments>
```

4. Четене на input (стандартен вход) от терминала и записване като променлива:
- Използва се командата **read**
 - Променливата се дефинира при **read** командата

```
1 #!/bin/bash
2
3 read -p "Hello UKTC , from " name
4
5 echo $name
```

```
rokyuto@rokyuto:~$ ./scr1.sh
Hello UKTC , from Viktor
Viktor
```

Резултат: Принтим променливата **name**, на която сме задали стойност – въведеното от нас в терминала чрез **read** командата

Option	Description
-a <array>	Assigns the provided word sequence to a variable named <array>.
-d <delimiter>	Reads a line until the provided <delimiter> instead of a new line.
-e	Starts an interactive shell session to obtain the line to read.
-i <prefix>	Adds initial text before reading a line as a prefix.
-n <number>	Returns after reading the specified number of characters while honoring the delimiter to terminate early.
-N <number>	Returns after reading the specified number of chars, ignoring the delimiter.
-p <prompt>	Outputs the prompt string before reading user input.
-r	Disable backslashes to escape characters.
-s	Does not echo the user's input.
-t <time>	The command times out after the specified time in seconds.
-u <file descriptor>	Read from file descriptor instead of standard input.

Променливи, условни оператори, цикли, масиви, функции: Условни оператори

1. Условие (If – else if – else)

- Често срещан случай в програмирането;
- Използва се когато искаме да оценим дадено условие и на база резултата да изпълним един набор от между 2 или повече набора оператори;
- За да работи проверката, условието трябва да е отделено с интервали вътре в скобите, които са квадратни [], а ключовата дума if също е отделена с интервал от тях (фиг. 1);
- При BASH програмирането се използват и ключовите думи **then** и **fi**
- При BASH програмирането else if се записва с **elif**

```
if [condition]
then
    statement1
else
    statement2
fi
```

space style 1 space

```
if [ condition1 ]
then
    statements
elif [ condition2 ]
then
    statements
else
    statements
fi
```

style 2

```
if [ condition1 ]; then
    statements
elif [ condition2 ]; then
    statements
else
    statements
fi
```

Фиг. 1

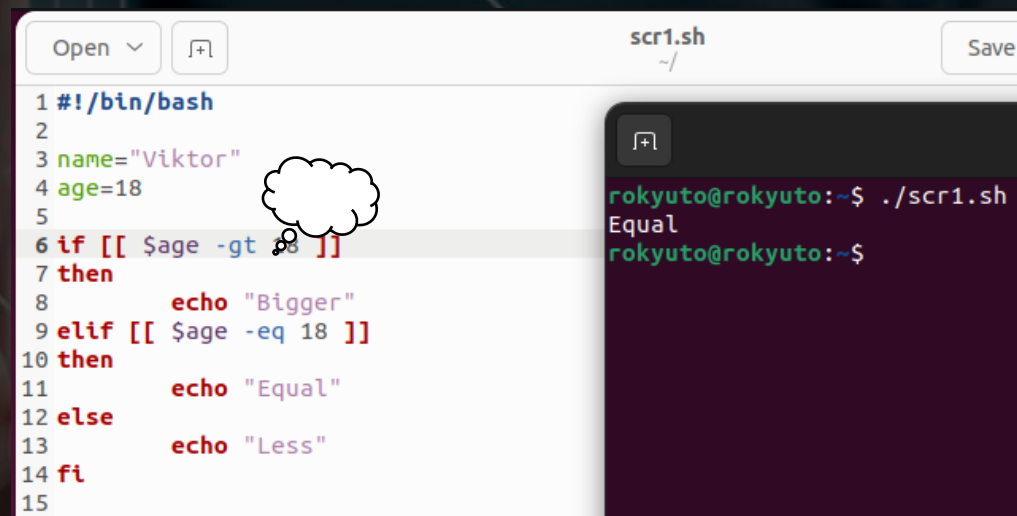
Променливи, условни оператори, цикли, масиви, функции:

Условни оператори

2. Структура на условието (if – else)

- 1) If + условие
- 2) Then – Ако условието върне TRUE, ще се изпълни кода в блока then
- ❑ Elif + условие2
- ❑ Then – Ако условието върне TRUE, ще се изпълни кода в блока then
- 3) Else – Ако условието/а върнат FALSE, ще се изпълни кода в блока else
- 4) Fi – Слага край на условието

* Команда **echo** – Принти стойността на подаденият аргумент (променлива / hardcode-нато нещо)



```
1 #!/bin/bash
2
3 name="Viktor"
4 age=18
5
6 if [[ $age -gt 18 ]]
7 then
8     echo "Bigger"
9 elif [[ $age -eq 18 ]]
10 then
11     echo "Equal"
12 else
13     echo "Less"
14 fi
15
```

rokyuto@rokyuto:~\$./scr1.sh
Equal
rokyuto@rokyuto:~\$

Променливи, условни оператори, цикли, масиви, функции: Условни оператори

2. Условни оператори

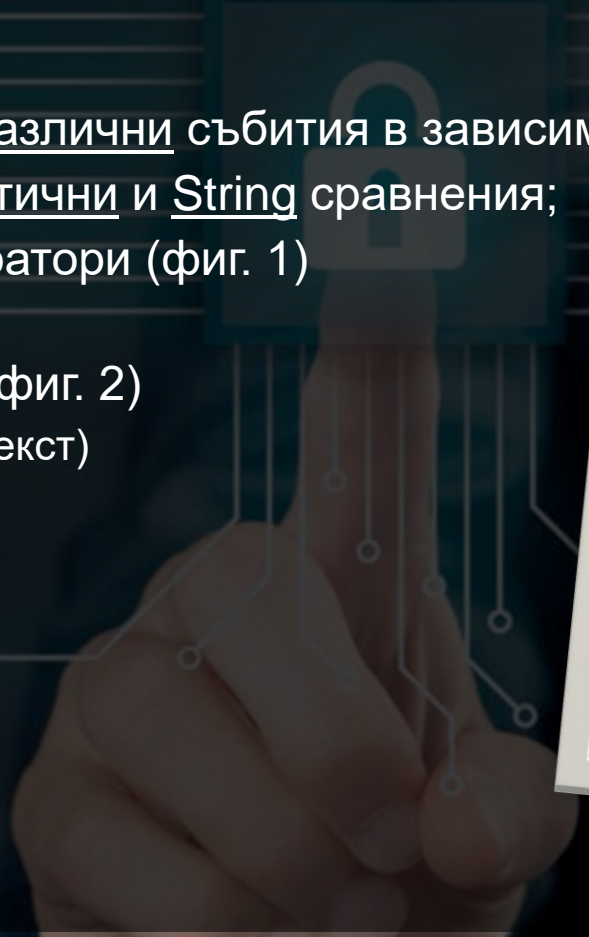
- Постоянно се използват;
- Служи за изпълняване на различни събития в зависимост от резултата на условието;
- Делят се на 2 вида: Аритметични и String сравнения;

а) Аритметични условни оператори (фиг. 1)

- Сравняват се цифри

б) String условни оператори (фиг. 2)

- Сравняват се String-ове (текст)



```
3 name="Viktor"  
4 age=18  
5  
6 if [[ $age -gt 18 ]]
```

A hand is pointing at a screen. On the screen, there is a code snippet in a light blue box. The code is: `3 name="Viktor"`, `4 age=18`, `5`, and `6 if [[$age -gt 18]]`. To the right of the code, there is a white thought bubble with a question mark inside.

Променливи, условни оператори, цикли, масиви, функции: Условни оператори

-lt (less than)	<
-gt (greater than)	>
-le (less-equal)	<=
-ge (greater-equal)	>=
-eq (equal)	==
-ne (not equal)	!=

Фиг. 1

```
re="^[0-9]+"  
  
if [[ ! $2 =~ $re ]]  
then  
    echo "$2 is not a number!" >&2  
    exit 1  
fi
```

=	equal
!=	not equal
<	less than
>	greater than
-n s1	string s1 is not empty
-z s1	string s1 is empty

Фиг. 2

* Оператор `=~` се използва за сравняване на подадена променлива с регулярен израз

* Този код проверява дали аргумент 2, който е подаден при извикване на bash script-а, като команда, е число, използвайки регулярен израз

Променливи, условни оператори, цикли, масиви, функции:

Цикли

- Цикълът представлява итерация на блок от код/команди N брой пъти;
- Цикълът се изпълнява докато условието, което е зададено е изпълнено (връща TRUE);
- В Bash програмирането има 3 вида цикли:
 - For Loop (фиг. 1 и 2);
 - While Loop (фиг. 3);
 - Until Loop (фиг. 4)

```
for ((initialize ; condition ; increment)); do  
[COMMANDS]  
done
```

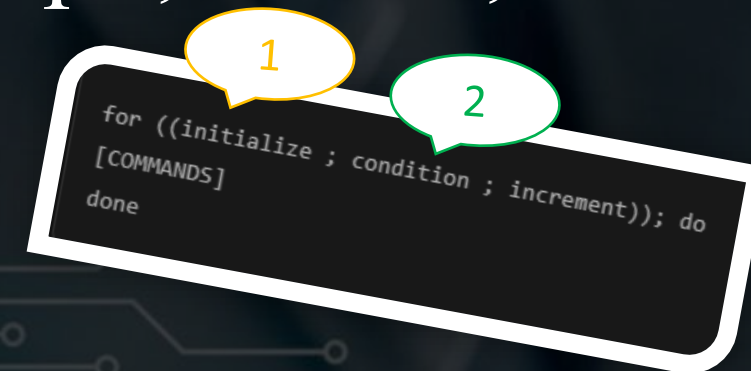
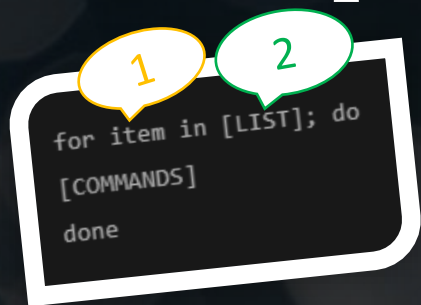
```
for item in [LIST]; do  
[COMMANDS]  
done
```

```
while [ condition ]; do  
[COMMANDS]  
done
```

```
until [ condition ]; do  
[COMMANDS]  
Done
```


Променливи, условни оператори, цикли, масиви, функции:

Цикли



• For Loop

- Имаме условие - колко пъти да се изпълни кода (командите) в body блока на цикъла.
- Това става чрез вградени start point и end point променливи, дефинирани при „създаването“ на цикъла

* И 2-та кода правят едно и също – изписват **Hello UKTC** 10 пъти в конзолата

```
3 for i in {0..10}
4 do
5     echo "Hello UKTC"
6 done
7
```

```
rokyuto@rokyuto:~$ ./scr1.sh
0
1
2
3
4
5
6
7
8
9
10
```

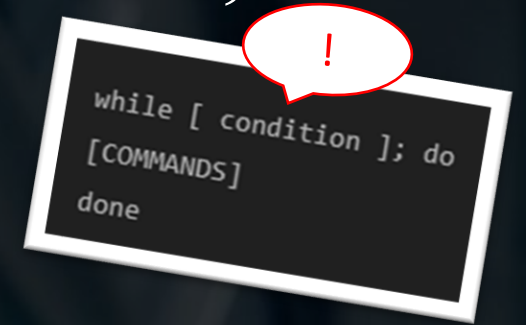
```
1 #!/bin/bash
2
3 for ((i = 0 ; i < 10 ; i++))
4 do
5     echo "Hello UKTC"
6 done
7
```

❖ **Do** – отваря body блока на цикъла.

❖ **Done** – слага край на цикъла.

Променливи, условни оператори, цикли, масиви, функции:

Цикли



• While Loop

- Имаме условие, което докато е изпълнено, кода (командите) в body блока на цикъла се вика;
- Този цикъл често се използва за infinite loop - кода (командите) в body блока на цикъла се изпълнява постоянно, докато програмата работи, постига се чрез **while true** (фиг. 1 и 2);
- Тук също присъстват ключовите думи **do** и **done**;

```
1 #!/bin/bash  
2  
3 i=0  
4  
5 while [ $i -lt 10 ]  
6 do  
7     echo $i  
8     ((i++))  
9 done
```

* Този код изписва стойността на променливата **i** в конзолата, след това инкрементира стойността ѝ с 1. Цикъла се повтаря докато стойността на променливата **i** е по-малка от 10

❖ **i++** – Увеличава (increment) стойността на променливата **i** с 1 (стъпка) . Ако искаме по-голяма стъпка записът е: **i+=step**

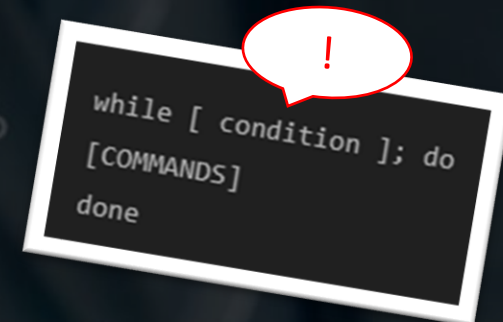
8

((i+=2))

```
rokyuto@rokyuto:~$ ./scr1.sh  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
rokyuto@rokyuto:~$
```

Променливи, условни оператори, цикли, масиви, функции:

Цикли



```
while [ condition ]; do  
[COMMANDS]  
done
```

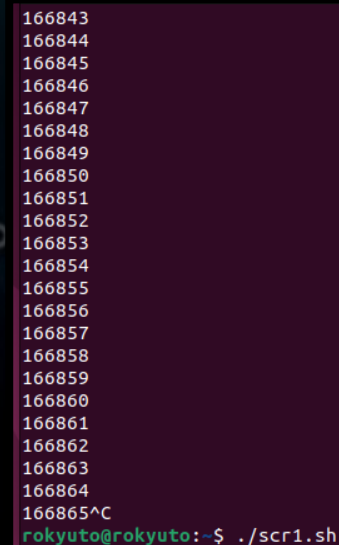
- While Loop

- При условие **while true** (фиг. 1), кода (командите) в body блока на цикъла се изпълнява всяка милисекунда (фиг. 2)
- Този код изписва стойността на променливата **i** в конзолата, след това инкрементира стойността ѝ с 1. Цикъла е безкраен и се повтаря докато програмата не спре

Фиг. 1

```
1 #!/bin/bash  
2  
3 i=0  
4  
5 while true  
6 do  
7     echo $i  
8     ((i++))  
9 done  
10
```

Резултат от кода



```
166843  
166844  
166845  
166846  
166847  
166848  
166849  
166850  
166851  
166852  
166853  
166854  
166855  
166856  
166857  
166858  
166859  
166860  
166861  
166862  
166863  
166864  
166865^C  
rokyuto@rokyuto:~$ ./scr1.sh
```

Фиг. 2

Променливи, условни оператори, цикли, масиви, функции:

Цикли

- Until Loop

- „Изпълни докато“
- Тук условието представлява кога цикълът да спре да изпълнява кода (командите) в body блока;
- Обърнатата логика на **while** цикъла;
- Отново присъстват **do** и **done**

```
until [ condition ]; do  
[COMMANDS]  
Done
```

```
1 #!/bin/bash  
2  
3 i=0  
4  
5 until [ $i -gt 10 ]  
6 do  
7     echo $i  
8     ((i++))  
9 done  
10
```

* Този код изписва стойността на променливата **i** в конзолата, след това инкрементира стойността ѝ с 1. Цикъла се повтаря докато стойността на променливата **i** не е по-голяма от 10

```
rokyuto@rokyuto:~$ ./scr1.sh  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

Променливи, условни оператори, цикли, масиви, функции:

Масиви

- Структура от данни, в която може да „складираме“ информация (данни) – фиг. 1);
- Масивът може да съдържа различни типове данни в себе си (напр. едновременно да държи **Integer** и **String** типове данни – фиг. 2);
- Елементите в масива се отделят с **ИНТЕРВАЛ**, няма запетай;
- Елементите в масива имат индекс (позиция в масива), който е уникален (даден индекс отговаря само на 1 позиция). Броенето започва от 0; и завършва с размера на масива – 1 (N-1);

```
array_name=(value1 value2 value3 ... )
```

Фиг. 1

```
user=( "john" 122 "sudo,developers" "bash" )
```

Фиг. 2

Променливи, условни оператори, цикли, масиви, функции: Масиви

- Елемент от масива може да бъде достъпен чрез неговият индекс;
- Индексът се поставя в [] скоби, които са слепени за името на масива, целият израз е поставен в { } скоби пред които има знакът \$, който пояснява на интерпретаторът, че това е променлива;

```
1 #!/bin/bash
2
3 #      fName      lName      number yearsOld
4 info=("Viktor" "Asenov" 19205 18)
5
6 echo "First Element: " ${info[0]}
7 echo "Third Element: " ${info[2]}
8 echo "Last Element: "  ${info[3]}
9 |
```

```
rokyuto@rokyuto:~$ ./scr1.sh
First Element:  Viktor
Third Element:  19205
Last Element:   18
rokyuto@rokyuto:~$
```


Променливи, условни оператори, цикли, масиви, функции: Масиви

```
1 #!/bin/bash
2
3 #      fName    lName    number yearsOld
4 info=("Viktor" "Asenov" 19205 18)
5
6 echo "Array: " ${info[*]}
```

`${info[*]}`

Връщат всички елементи на
масива

```
rokyuto@rokyuto:~$ ./scr1.sh
Array: Viktor Asenov 19205 18
```

```
1 #!/bin/bash
2
3 #      fName    lName    number yearsOld
4 info=("Viktor" "Asenov" 19205 18)
5
6 echo "Array: " ${info[@]}
7
```

`${info[@]}`

```
1 #!/bin/bash
2
3 #      fName    lName    number yearsOld
4 info=("Viktor" "Asenov" 19205 18)
5
6 echo "Array Size: " ${#info[@]}
7
```

`${#info[@]}`

Връща размера на
масива

```
rokyuto@rokyuto:~$ ./scr1.sh
Array Size: 4
```

Променливи, условни оператори, цикли, масиви, функции: Масиви

```
1 #!/bin/bash
2
3 #      fName      lName      number yearsOld
4 info=("Viktor" "Asenov" 19205 18)
5
6 echo "Years Before: " ${info[3]}
7
8 info[3]=17
9
10 echo "Years After: " ${info[3]}
11 |
```

info[index]=newValue

Променя стойността на
елемента от масива

```
rokyuto@rokyuto:~$ ./scr1.sh
Years Before: 18
Years After: 17
```

```
1 #!/bin/bash
2
3 #      fName      lName      number yearsOld
4 info=("Viktor" "Asenov" 19205 18)
5
6 echo "Array Before: " ${info[*]}
7
8 info+=("Pleven")
9
10 echo "Array After: " ${info[*]}
11 |
```

Info+=newElement

Добавя нов елемент към
масива

```
rokyuto@rokyuto:~$ ./scr1.sh
Array Before: Viktor Asenov 19205 18
Array After: Viktor Asenov 19205 18 Pleven
```

Променливи, условни оператори, цикли, масиви, функции: Масиви

```
1 #!/bin/bash
2
3 #      fName    lName    number yearsOld
4 info=("Viktor" "Asenov" 19205 18 "Pleven")
5
6 echo "Array Before: " ${info[*]}
7
8 unset info[4]
9
10 echo "Array After: " ${info[*]}
```

unset info[index]

Изтрива елемент от
масива по подаден индекс

```
rokyuto@rokyuto:~$ ./scr1.sh
Array Before:  Viktor Asenov 19205 18 Pleven
Array After:   Viktor Asenov 19205 18
```

```
1 #!/bin/bash
2
3 #      fName    lName    number yearsOld
4 info=("Viktor" "Asenov" 19205 18 "Pleven")
5
6 echo "Array Before: " ${info[*]}
7
8 unset info
9
10 echo "Array After: " ${info[*]}
```

unset info

Изтрива целия масив

```
rokyuto@rokyuto:~$ ./scr1.sh
Array Before:  Viktor Asenov 19205 18 Pleven
Array After:
```


Променливи, условни оператори, цикли, масиви, функции: Функции

- Блок/парче код, който може да бъде извикан множество пъти;
- Функциите раздробяват кода, целта е една функция да е специализирана и да върши 1 основно нещо;
- В bash скриптовете функциите нямат дефиниран тип (int/void/String), идентично с езика Python;

Syntax:

```
# for defining
function_name(){
    commands
    .....
}

function_name # for calling
```

Дефиниране на функция

```
1 #!/bin/bash
2
3 testFunction(){
4     echo "Hello UKTC"
5 }
6
```



Извикване на функция

```
6
7 testFunction
8
```



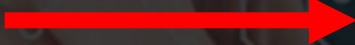
Резултат при изпълнение на скрипта

```
rokyuto@rokyuto:~$ ./scr1.sh
Hello UKTC
```

Променливи, условни оператори, цикли, масиви, функции: Функции

- При извикване на функция можем да подаваме **параметри/аргументи**.
- Те имат индекс, чрез който могат да бъдат достъпени вътре във функцията.
- Броенето започва от 1 тъй като името на функцията е 0;

```
1 #!/bin/bash
2
3 testFunction(){
4     echo "Hello $1"
5 }
6
7 testFunction "UKTC"
8 |
```

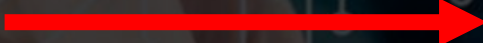


```
rokyuto@rokyuto:~$ ./scr1.sh
Hello UKTC
```

Променливи, условни оператори, цикли, масиви, функции: Функции

- Функцията може да връща информация;
- Става чрез ключовата дума **return**;

```
1 #!/bin/bash
2
3 testFunction(){
4     return $(( $1*$2 ))
5 }
6
7 testFunction 5 2
8 echo Figure Area: $?
```



```
rokyuto@rokyuto:~$ ./scr1.sh
Figure Area: 10
```


Аргументи

- Както при функциите, така и при извикване на bash скрипт имаме аргументи – фиг. 1;
- Броят на аргументите може да е до безкрай;
- Броенето започва от 0 като първият аргумент е името на скрипта – фиг. 2;
- Аргументите се извикват чрез: **\$argument_Index** - фиг. 3;
- Разделянето на един аргумент от друг става чрез space (разстояние) – фиг. 1;



Фиг. 1

```
rokyuto@rokyuto:~$ ./scr1.sh Viktor Asenov 19205 18
```

Фиг. 2

Фиг. 3

The terminal window shows the execution of the script with arguments: `rokyuto@rokyuto:~$./scr1.sh Viktor Asenov 19205 18`. The output shows the first argument, `Viktor`, followed by a prompt. To the right of the terminal window, a small inset shows the script file `scr1.sh` with the third line changed to `3 echo $2|`, indicating that the second argument (`Asenov`) is being accessed.

Аргументи

- Има 2 полезни функционалности относно аргументите при извикване на скрипт:
 - Общ брой на въведените аргументи – фиг. 1
 - Връщане на всички аргументи – фиг. 2

```
rokyuto@rokyuto:~$ ./scr1.sh
0
rokyuto@rokyuto:~$ ./scr1.sh 1 2 3 5 6
5
rokyuto@rokyuto:~$
```

1	#!/bin/bash
2	
3	echo \$#

Фиг. 1

```
rokyuto@rokyuto:~$ ./scr1.sh 1 2 3 5 6
1 2 3 5 6
rokyuto@rokyuto:~$
```

1	#!/bin/bash
2	
3	echo \$@

Фиг. 2

Примерни програми и практически примери: Базова програма за изчисляване на лице на фигура

```
1 #!/bin/bash
2
3 read -p "Enter Side: " side
4 read -p "Enter Height: " height
5
6 calcArea(){
7     return $((($1*$2))
8 }
9
10 calcArea $side $height
11 echo Figure Area: $?
12
```

```
rokyuto@rokyuto:~$ ./scr1.sh
Enter Side: 10
Enter Height: 3
Figure Area: 30
```

Програмата изчаква потребителят да въведе **дължина на страната** и **дължина на височината**, които се подават като аргументи при извикване на функцията **calcArea**, която връща лицето на фигурата и накрая я принтим

Примерни програми и практически примери: Автоматизиране на архивиране (backup) с bash скрипт

```
1 #!/bin/bash
2
3 find "/backup" -type f -exec rm {} \; # Find all files in the
  with the Course Project) and delete them
4
5 # tar -> Archive tool for compression
6 tar -cvf /backup/backup.tar /var/www/html/192_WEB_Project
7 #c = Creation of tar file
8 #v = Verbose the creation of the tar file
9 #f = To grant access to all the files in the specified path
```

```
rokyuto@rokyuto:/$ sudo crontab -e
```

```
rokyuto@rokyuto:/$ sudo ./backup.sh
[sudo] password for rokyuto:
tar: Removing leading '/' from member names
/var/www/html/192_WEB_Project/
/var/www/html/192_WEB_Project/WEB/
/var/www/html/192_WEB_Project/WEB/Footer.css
/var/www/html/192_WEB_Project/WEB/AboutUsPage/
/var/www/html/192_WEB_Project/WEB/AboutUsPage/AboutUs.php
/var/www/html/192_WEB_Project/WEB/AboutUsPage/AboutUs.css
/var/www/html/192_WEB_Project/WEB/BookFlightPages/
/var/www/html/192_WEB_Project/WEB/BookFlightPages/Style.css
/var/www/html/192_WEB_Project/WEB/BookFlightPages/Scripts/
/var/www/html/192_WEB_Project/WEB/BookFlightPages/Scripts/Script_AirportMenu.js
/var/www/html/192_WEB_Project/WEB/BookFlightPages/Scripts/Script_Slideshow.js
/var/www/html/192_WEB_Project/WEB/BookFlightPages/Images/
/var/www/html/192_WEB_Project/WEB/BookFlightPages/Images/MainPage_Banner_Full.png
/var/www/html/192_WEB_Project/WEB/BookFlightPages/Images/BALKAN-Bulgarian-Airlines-2.png
/var/www/html/192_WEB_Project/WEB/BookFlightPages/Images/SwapButton.png
/var/www/html/192_WEB_Project/WEB/BookFlightPages/Images/BALKAN-Bulgarian-Airlines-2.png
```

```
# m h dom mon dow   command
0 4 28 * * /backup.sh
0 4 * * 1 /deleteTMPFiles.sh
```

Програмата намира дали има файлове в папката backup, изтрива ги и след това създава нов архив в папката backup на папката 192_WEB_Project, в която се намира целият сайт и SQL скриптове. За да се автоматизира този процес се използва CRON, като тук е зададено backup скрипта да се изпълнява на всяко 28-мо число от месеца в 04:00 часа

Примерни програми и практически примери

Bash скриптовете намират голямо приложение в ежедневието на фирмите. Ето още примери:

- Мониторинг на свободно място;
- Автоматизирано менажиране на потребители;
- Автоматизиран процес на сканиране за физическо приложение;

Заключение: Bash scripting-ът е полезен инструмент, който може да върши черната работа и да спестява време и грешки. Познанията в тази сфера няма да са от вреда



Благодаря за вниманието!