Bash scripting

Ce este shell-ul?

SHELL este un program care asigură interfața dintre utilizator și un sistem de operare. Când utilizatorul se conectează, sistemul de operare pornește un shell pentru utilizator .

Shell-ul este de mai multe tipuri:
The C Shell (csh) /bin/csh
The Bourne Shell (sh) /bin/sh
The Korn Shell (ksh) /bin/ksh
GNU Bourne-Again Shell (bash) /bin/bash

GNU Bourne-Again Shell (BASH)

Ce este bash? Bash (Bourne Again Shell) este versiunea gratuită și îmbunătățită a shell-ului Bourne distribuită cu sistemele de operare Linux și GNU .

Exemplu de script în bash:

```
#!/bin/bash
echo “hello world”
```

Prin convenție primul rând va avea **#!** la care adăugăm numele shell-ului pe care dorim să îl executăm. Astfel, îi spunem sistemului de operare cum să execute scriptul nostru .

GNU Bourne-Again Shell (BASH)

Să presupunem că acest text este într-un fișier cu numele **myscript .**
Putem executa acest fișier cu comanda**: bash myscript**

```
demo@Ubuntu:/tmp/bash$ bash script
hello world
demo@Ubuntu:/tmp/bash$
```

Se obișnuiește ca atunci când scriem un script în bash, să aibă extensia **.sh**

De asemenea, ca să nu fim nevoiți să dăm comanda “bash” în față, putem să îi dăm acelui fișier drepturi de execuție cu comanda:
chmod +x nume_script.sh

```
demo@Ubuntu:/tmp/bash$ ./script.sh
hello world
demo@Ubuntu:/tmp/bash$
```

IT School
FROM ZERO TO HERO!

Tipuri de date

Cele mai cunoscute tipuri de variabile folosite sunt:

integer – Care reprezintă un numar întreg . Ex: 4,-10,54,1000
float – Matematic reprezintă numerele raționale: Ex: 4,1 sau -3,43
string – Este o combinație de seturi de caractere. Ex: “THIS IS A STRING” sau “ab”
boolean – Este un tip de date cu două valori posibile: adevărat sau fals (de obicei corespondente cu 1 sau 0)

Ce este o variabilă ?

Gandiți-vă la o variabilă ca “o cutie” în care putem pune diferite lucruri. Putem pune de ex un număr întreg sau putem pune un șir de caractere (string) sau chiar o valoare booleană (False).

Dacă dorim ca variabila x să aibă valoarea 5, trebuie să îi atribuim acea valoare cu comanda **x=5 .**

Ca să afișăm valoarea (conținutul) pe care o are variabila x, adica pe exemplul dat mai sus ce este in “cutie”, rulăm comanda: **echo $x**

IT School®
FROM ZERO TO HERO!

Exemplu

```
#!/bin/bash
x=3
y=-4.4
z="this is my string"
echo "in variabila x am pus un numar intreg" $x
echo "###################################"
echo "in variabila y am pus un numar rational" $y
echo "###################################"
echo "in variabila z am pus un string” $z
```

```
demo@Ubuntu:/tmp/scripting$ ./display.sh
in variabila x am pus un numar intreg 3
###################################
in variabila y am pus un numar rational -4.4
###################################
in variabila z am pus un string this is my string
demo@Ubuntu:/tmp/scripting$
```

IT School
FROM ZERO TO HERO!

Exemplu

```
#!/bin/bash
user=$(whoami)
echo "outputul de la comanda whoami este: " $user
user="myuser"
echo “valoarea variabilei user este acum: " $user
```

În aceeași variabilă putem pune mai multe valori, dar mereu ultima variantă atribuită va fi cea actuală. În cazul de mai sus, prima dată variabila user are o valoare (cea data de comanda whoami) și în al doilea caz, variabila ‘user’ va avea valoarea ‘myuser’ .

Pentru a putea avea valorile setate în scriptul de mai sus în shell-ul curent, este nevoie ca acel script să fie rulat cu punct înainte. De exemplu **. ./script.sh**

Prin aceasta metoda ne putem seta ușor variabilele de environment cu un singur script fara a fi nevoie sa exportăm fiecare variabilă în parte .

Cum exportăm o variabilă de environment?
export my_key=12341dvd14313341341

Aceste variabile se pot verifica cu comanda **env**
sau cu **echo $my_key**

Variabila PATH

Variabila PATH este o variabilă de mediu care conține o listă ordonată de căi pe care Linux le va căuta pentru a le executa atunci când se rulează o comandă .

Cum verificam PATH-ul curent? Cu comanda:
echo $PATH -> /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games

Cum modificăm PATH-ul curent? Cu comanda:
export PATH=/some/new/path:$PATH

Vom face un folder cu numele **TEST** în home-ul userului curent și vom pune un script care afișează “test path”. Vom adăuga
export PATH=/HOME/YOUR_USER/TEST:$PATH

IT School®
FROM ZERO TO HERO!

Variabila PATH

Momentan ce am setat este temporar. Daca vom da comanda de switch user nu vom mai vedea path-ul setat de noi mai devreme.

Pentru a face permanente modificările făcute pentru utilizatorul curent, trebuie adaugată comanda de export și în ~/.profile urmată de încarcarea acestui fișier shell-ul folosit: **source ~/.profile**

Operatori matematici

OPERATOR	MOD DE FOLOSIRE	DESCRIERE
+	a + b	Suma a două numere/concatenează string-uri
-	a – b	Face scaderea între a și b
*	a*b	Produsul între a și b
/	a/b	Divide a cu b
%	a%b	Restul împărțirii lui a la b

Operații matematice în bash

```
#!/bin/bash
x=4
y=5
suma=$((x+y))
diferenta=$((x-y))

echo "suma este: $suma"
echo "diferenta este: $diferenta"
```

```
#!/bin/bash
x=15
y=5
P=$((x*y))
Q=$((x/y))

echo "rezultatul inmultirii numerelor este: $P"
echo "rezultatul impartirii numerelor este: $Q"
```

Operatii matematice în bash

```
#!/bin/bash
x=10
y=3
rest=$((x%y))

echo "restul impartirii este: $rest"
```

```
#!/bin/bash
x=20
y=2
rest=$((x%y))

echo "Daca restul impartirii este 0, numarul x
este par, altfel numarul, este impar"
echo "restul este: $rest"
```

Această operație este foarte utilă pentru a afla dacă un numar este par sau impar .

Argumente într-un script

Argumentele transmise unui script sunt procesate în aceeași ordine în care sunt trimise. Indexarea argumentelor începe de la unu, iar primul argument poate fi accesat în interiorul scriptului folosind $1 .

```
#!/bin/bash
echo "my name is: $1“
```

Cum se apelează?

```
demo@Ubuntu:/tmp$ ./mynameis.sh George
my name is: George
demo@Ubuntu:/tmp$
```

IT School
FROM ZERO TO HERO!

Argumente într-un script

Se pot da mai multe argumente în același script .

```
#!/bin/bash
echo "name of the script is: $0"
echo "my name is: $1 $2"
```

$0 ne va returna numele scriptului care se execută.
Numele “George” va fi asociat argumentului $1
Numele “Radu” va fi asociat argumentului $2

```
demo@Ubuntu:/tmp$ ./mynameis.sh George Radu
name of the script is: ./mynameis.sh
my name is: George Radu
demo@Ubuntu:/tmp$
```

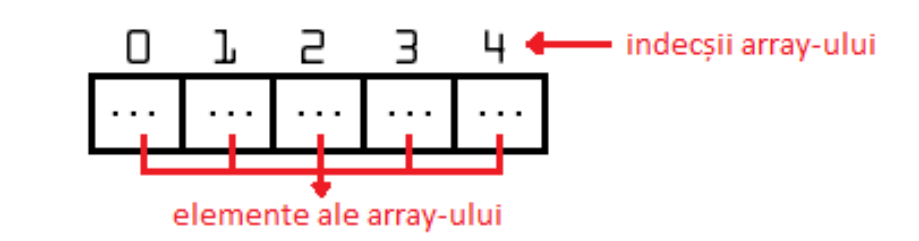
IT School
FROM ZERO TO HERO!

Arrays

Ce este un array? Un array este o structură de date constând dintr-o colecție de elemente, fiecare identificată de un index .

Un array se definește astfel: nume_array=(element1 element2 …)

Primul element se poate apela astfel ${nume_array[0]} și al doilea ${nume_array[1]}. Primul index va fi mereu 0, urmat de 1,2,.. etc



IT School
FROM ZERO TO HERO!

Exemplu

```
#!/bin/bash
lista=(12 32 4 54 65 76 23 54)
echo "Elementul cu index-ul 0 este: ${lista[0]}"
echo "Elementul cu index-ul 1 este: ${lista[1]}"
echo "Elementul cu index-ul 2 este: ${lista[2]}"
echo "Elementul cu index-ul 3 este: ${lista[3]}"
echo "Elementul cu index-ul 4 este: ${lista[4]}"
echo "Elementul cu index-ul 5 este: ${lista[5]}"
echo "Elementul cu index-ul 6 este: ${lista[6]}"
echo "Elementul cu index-ul 7 este: ${lista[7]}"
echo "Elementul cu index-ul 8 este: ${lista[8]}"
echo "lista comepleta de numere este: "
echo "${lista[@]}“
```

La ${lista[8]} putem observa că nu se afișează nimic pentru că acea valoare nu există, adică pe indexul corespondent nu avem atribuită nicio valoare . Pentru a afișa toată lista cu o singură comandă, este nevoie să scriem ${nume_array[@]} .

Exemplu

```
demo@Ubuntu:~/Scripts$ ./myscript.sh
Elementul cu index-ul 0 este: 12
Elementul cu index-ul 1 este: 32
Elementul cu index-ul 2 este: 4
Elementul cu index-ul 3 este: 54
Elementul cu index-ul 4 este: 65
Elementul cu index-ul 5 este: 76
Elementul cu index-ul 6 este: 23
Elementul cu index-ul 7 este: 54
Elementul cu index-ul 8 este:
lista comepleta de numere este:
12 32 4 54 65 76 23 54
demo@Ubuntu:~/Scripts$
```

Exemplu

```
#!/bin/bash
lista=("aaaa" "bbbb" "cccc")
echo -e "Elementul cu index-ul 0 este:\n${lista[0]}"
echo -e "Elementul cu index-ul 1 este:\n${lista[1]}"
echo -e "Elementul cu index-ul 2 este:\n${lista[2]}”
```

Un array poate contine si elemente de tip string. In exemplul de mai sus am utilizat pe langa comanda echo si “-e”, care daca citim documentatia comenzii echo (man echo) vom vedea ca nu face altceva decat sa ia in considerare in comanda data “\n” adica new line.

```
demo@Ubuntu:~/Scripts$ ./myscript.sh
Elementul cu index-ul 0 este:
aaaa
Elementul cu index-ul 1 este:
bbbb
Elementul cu index-ul 2 este:
cccc
demo@Ubuntu:~/Scripts$
```

```
demo@Ubuntu:~/Scripts$ ./myscript.sh
Elementul cu index-ul 0 este:aaaa
Elementul cu index-ul 1 este:bbbb
Elementul cu index-ul 2 este:cccc
```

Comanda read

Un script pe langă argumentele prezentate mai sus ($1 $2 $3 .. etc) se mai pot atribui valori de intrare unei variabile definite de utilizator folosind comanda de citire (read) .

```
#!/bin/bash

read nume
echo "Hello, $nume !"
```

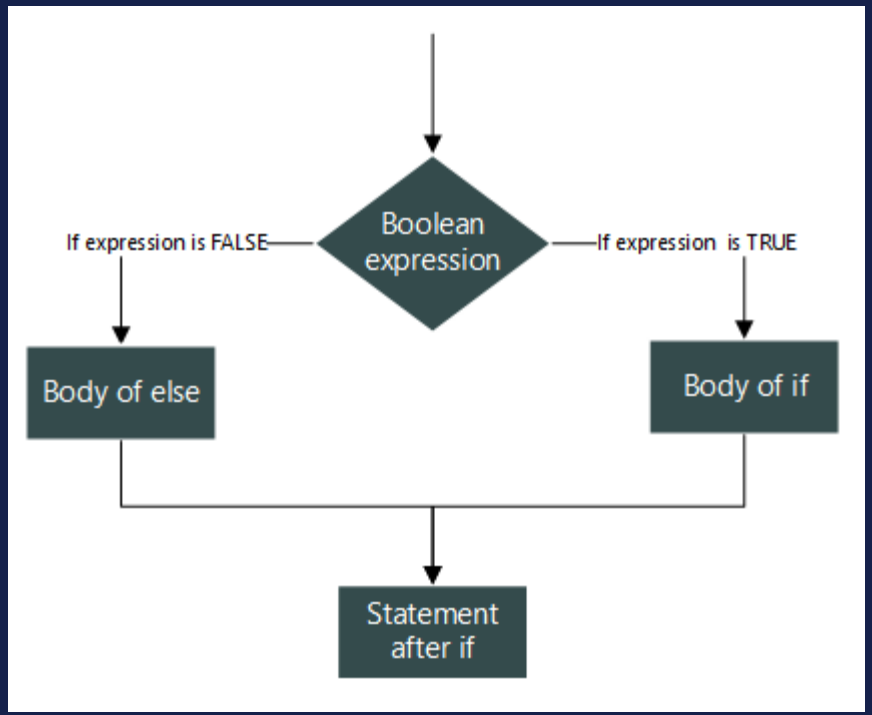
```
#!/bin/bash

read -p "numele tau este: " nume
echo "Hello, $nume !"
```

Expresia booleană

O expresie booleană este o declarație logică care este fie **TRUE**, fie **FALS**.

De obicei în programare:
cifra 0 este corespondentă pentru FALS
cifra 1 este corespondentă pentru ADEVAR



IT School
FROM ZERO TO HERO!

Tabela de adevăr

0 -> FALS
1 -> ADEVĂR

AND

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

IT School®
FROM ZERO TO HERO!

Instrucțiunea if

Reprezintă o structură de calcul ce permite executarea (sau nu) a unei instrucţiuni sau serie de instrucţiuni în funcţie de anumite condiţii. În programare, astfel de intrucțiune arată în felul urmator:

IF expression
THEN
<commands>

check.sh

```
#!/bin/bash
if [ $1 -ge 3 ];
 then
  echo "Numarul tau este mai mare ca 3"
 fi
```

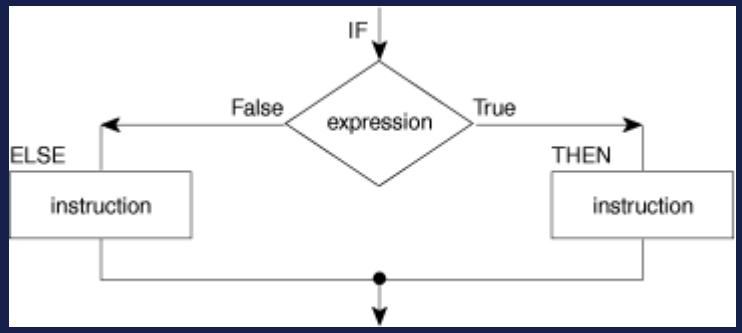
check2.sh

```
#!/bin/bash
if (("$1" <= "$2"))
 then
  echo "Numarul $1 este mai mic sau egal
decat $2"
 fi
```

IT School
FROM ZERO TO HERO!

Exemplu

```
demo@Ubuntu:~/ifcondition$ ./check.sh 2
demo@Ubuntu:~/ifcondition$ ./check.sh 20
Numarul tau este mai mare ca 3
demo@Ubuntu:~/ifcondition$ ./check.sh 1
demo@Ubuntu:~/ifcondition$ ./check2.sh 1 10
Numarul 1 este mai mic sau egal decat 10
demo@Ubuntu:~/ifcondition$ ./check2.sh 11 3
demo@Ubuntu:~/ifcondition$
```



Intrucțiunea if-else

Uneori dorim să efectuăm un anumit set de acțiuni dacă o afirmație este adevărată și un alt set de acțiuni dacă este falsă. Putem face acest lucru prin utilizarea instructiunii “if” cu cea de “else”.

if expression
then
<commands>
else
<other commands>
fi

```
#!/bin/bash
if [ $1 -eq 10 ]
then
    echo "ai pus numarul 10, raspuns corect"
else
    echo "ai pus lat numar, raspuns gresit"
fi
```

```
demo@Ubuntu:~/ifcondition$ ./check3.sh 11
ai pus lat numar, raspuns gresit
demo@Ubuntu:~/ifcondition$ ./check3.sh 10
ai pus numarul 10, raspuns corect
```

IT School
FROM ZERO TO HERO!

Condiții multiple

Pe langă condiția **if-else**, în caz de avem variante multiple de alegere, să spunem compărarea unui număr. Astfel, putem verifica 3 variante, dacă x este mai mic decât 10, x este egal cu 10 și ultima variantă x este mai mare decât 10 .

```
#!/bin/bash
n=$1
if (( $n > "10" ))
then
   echo "numarul este mai mare ca 10"
 elif (( $n == "10" ))
then
   echo "numarul este este egal cu 10"
 else
   echo "numarul este mai mic ca 10"
fi
```

Exemplu

```
#!/bin/bash
n=$1
if (( $n == "10" ))
then
   n=$((n+1))
 else
   n=$((n*10))
fi

echo "numarul final este: $n“
```

Exemplu

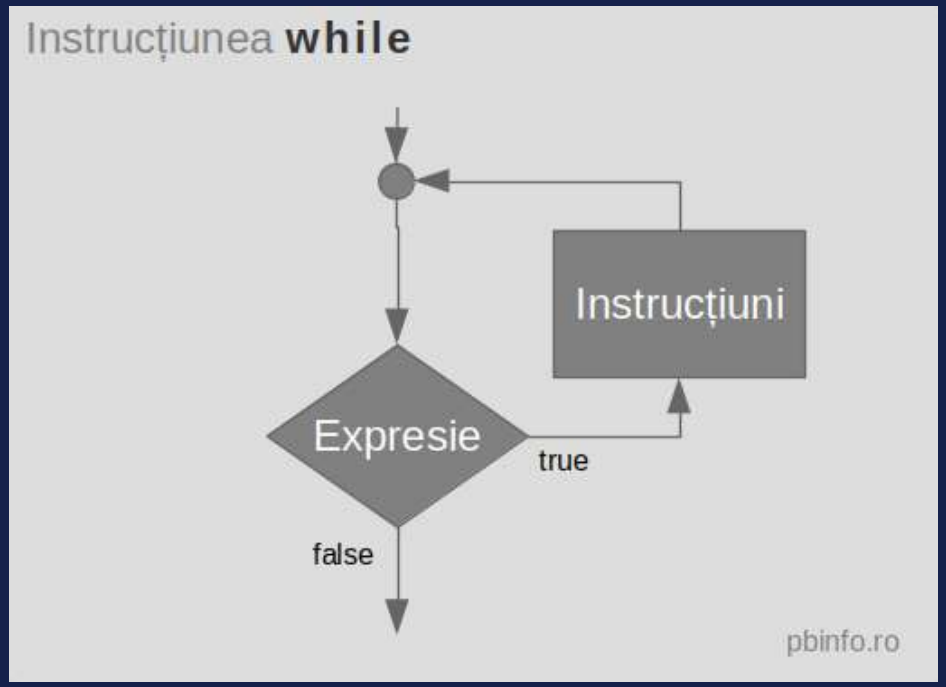
```
#!/bin/bash
age=$1
if [ $age -ge 18 ] && [ $age -le 100 ]
then
  echo "varsta ta este intre 18 si 100 de ani"
elif [ $age -gt 100 ]
then
  echo "ai peste 100 de ani"
else
  echo "ai sub 18 ani"
fi

echo "varsta ta este: $age“
```

-ge -> greater or equal than
-le -> lower or equal than
-gt -> greater than
-lt -> lower than

Instrucțiunea while

O buclă “while” este folosită pentru a repeta un anumit bloc de cod de un număr necunoscut de ori, până când este îndeplinită o condiție precizată. De exemplu, să presupunem că inițial, în variabila “i” avem salvată valoarea 1. Vom afișa pe ecran “hello world” și valoarea numarului “i” ori de câte ori este indeplinită condiția ca numarul salvat în “i” < 10. După fiecare pas, i va fi incrementat cu 1.



IT School
FROM ZERO TO HERO!

Instrucțiunea while

```
while [ condition]
do
Command1
Command2
Command3
done
```

```
#!/bin/bash
x=1
while (( $x < "10" ))
do
  echo "Aceasta fraza se va scrie pana cand x
va fi mai mic ca 10,x = $x"
  x=$(( $x + 1 ))
done
```

Exemplu

Cum aflăm numarul de elemente dintr-un arrray?

```
#!/bin/bash
ar=(32 10 132 32 5 7 43 104)
length=${#ar[@]}
echo $length
```

```
#!/bin/bash
ar=(32 10 132 32 5 7 43 104)
i=0
length=${#ar[@]}

echo "Elementele din array sunt:"
while [ $i -lt $length ];
do
  echo ${ar[$i]}
  i=$((i+1))
done
```

Bonus: dacă vrem să afișăm elementele pe aceeași linie vom scrie : echo -n "${ar[$i]} "

Comanda sleep

Sleep este o comandă folosită pentru a întârzia programul pentru o anumită perioadă de timp. Putem spune programului sa nu faca nimic timp de n secunde și poate fi utilă pentru a avea timp să vizualizăm ce ne afișează la output .
Comanda sleep va primi la input numarul de secunde pe care dorim să îl așteptăm. **Exemplu**: sleep 3 -> ne va pune să așteptăm timp de 3 secunde între execuția instrucțiunilor .

```
#!/bin/bash
n=0
while (( $n < "5" ))
do
  echo "vom astepta 2 secunde intre afisarea
numerelor, n = $n"
  sleep 2
  n=$((n+1))
done
```

Break

Instrucțiunea break termină structura repetitivă din care face parte. De obicei se utilizează atunci când se dorește întreruperea unui loop la îndeplinirea unei condiții .

În exemplul de mai jos avem un while infinit, el nu are condiție să se oprească și se poate opri doar prin comanda CTRL+C (care după cum știm, întrerupe (kill) procesul curent din prim-plan care rulează în terminal.

```
#!/bin/bash
while :
do
        echo "Press [CTRL+C] to stop.."
        sleep 1
done
```

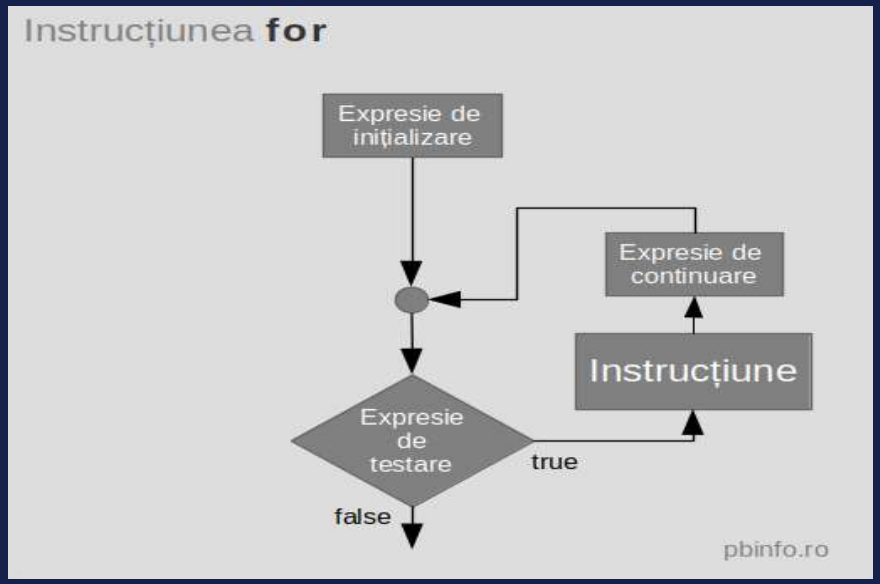
Break

```
#!/bin/bash

while :
do
  read -p "scrie un numar: " x
  if (( x == 10 ))
  then
     echo "numarul introdus de tine este corect"; break
  else
     echo "mai incearca"
  fi
done
```

Instucțiunea for

Instrucțiunea **for** este similară cu cea **while**, se vor executa anumite seturi de instrucțiuni într-o buclă atât timp cât se îndeplinesc anumite condiții definite .




```
for VARIABLE in 1 2 3 4 5 .. N
do
        command1
        command2
        ..................
        commandN
done
```

IT School
FROM ZERO TO HERO!

Exemplu

Afișarea numerelor pare

```
#!/bin/bash
echo "numerele pare sunt: "
for (( i=0; i<=10; i++ ))
do
    if (( $i%2==0 ))
    then
      echo "$i  "
    fi
done
```

Afișarea numerelor 1,2,3,4,5

```
#!/bin/bash
for i in 1 2 3 4 5
do
  echo "for interatii: $i"
done
```

Sfârșit