Lab 2

Introducere

Exemplu : *Hello World*

Se deschide un editor de text (*gedit*), in acest fisier scriem comanda:

```
echo Hello, World
```

Se salveaza sub numele "**hw**" in directorul **lab2** (**~**/lab2). Navigam catre directorul nou creat si verificam existenta fisierului folosind comanda **ls**.

Incercati sa rulati acest script tastand "hw" in terminal.

Eroarea primita ("*Command not found*") este cauzata de faptul ca sistemul de operare are o lista definita de directorare prin care sa se uite pentru a gasi o comanda. Aceasta lista este salvata in variabila de sistem **PATH**. Putem printa aceasta lista tastand in terminal:

```
echo $PATH
```

Putem scapa de eroarea primita prin:

- Modificarea variabilei **PATH**, tastand in terminal:

  ```
  export PATH=$PATH:~/lab2
  ```

  In comanda exportam in instanta curenta de terminal o variabila denumita PATH, initializam aceasta varibila cu continul vechii variabile PATH la care concatenam textul ":~/lab2". *Aceasta schimbare este valabila numai in instanta curenta de terminal.*
- Specificand exact calea catre scriptul nostru

  ```
  # daca programul este in acelasi director
  ./hw
  # daca programul este intr-un director diferit
  # EX: noi suntem in ~ dar programul este in subdirectorul lab2
  ~/lab2/hw
  ```

Cauza pentru noua eroare primita ("*Permission denied*"), se poate vedea daca rulam comanda **ls -l**: *Creatorul fisierului nu are drept de executie.*

Aceasta problema se poate rezolva folosind comanda **chmod**.

```
chmod u+x hw
```

Incercati sa rulati acest script tastand "hw" in terminal. (sau **./hw**, daca variabila PATH nu a fost modificata)

Ce se intampla in fundal? Sistemul de operare incearca sa execute programul "**hw**", vede ca este un fisier text si din aceasta cauza foloseste interpretorul default **bash**. Acesta executa, linie cu linie, comenzile scrise in fisier.

Exemplu: *Aplicatie de luat notite*

Vrem sa avem o aplicatie de luat notite, numita **tn**, pe care putem sa o chemam din terminal. Aceasta va lua textul primit ca si parametru, si-l va scrie intr-un fisier *notes.txt* ce va exista in folderul root al userului.

Deschidem un editor si scriem urmatorul cod, il salvam in directorul *lab2* sub numele *tn*:

```
#!/bin/bash
# simple note-taking application
echo $1 >> ~/notes.txt
```

- Prima linie: Contine operatorul special "#!" (shebang), specifica ce interpretor trebuie folosit pentru executarea programului, in cazul de fata "bash", dar poate fi "python", "perl", etc. . **Aceasta ar trebui sa fie intodeauna prima linie dintr-un script**. Daca aceasta prima linie lipseste si scriem un script bash si este rulat intr-un sistem in care interpretorul default nu este bash, scriptul nostru nu va poate fi executat.
- A doua linie: este un comentariu explicativ al aplicatiei
- A treia linie:
  - Folosim comanda *echo* pentru a printa.
  - *$1* este o variabila speciala in bash, specifica primul parametru primit de catre program
  - Prin >> redirectam outputul programului *echo* catre fisierul dorit, *notes.txt*

Daca directorul "*lab2*" nu a fost adaugat in **PATH**, il adaugam acum. Modificam permisiuniile fisierului astfel incat sa putem sa il executam si rulam urmatoarea comanda:

```
tn this is a test
```

Daca deschidem fisierul *~/notes.txt* observam ca numai primul cuvant a fost salvat. De ce? Paramatrii sunt separati dupa spatiu (" "). Scriptul nostru nu vede un parametru in textul dat, vede patru.

Solutia este sa modifcam scriptul nostru ca in loc sa folosim variabila *$1*, sa folosim variabila *$**, ce contine toti parametri primiti de programul nostru.

Pentru a extinde functionalitatea scriptului nostru, astfel incat sa printeze si data la care notita a fost luata, putem sa folosim comanda **date**.

```
#!/bin/bash
# simple note-taking application
echo $(date) : $1 >> ~/notes.txt
```

Folosim operatorul de substitutie de comanda **$()** pentru a specifica faptul ca este o comanda externa, daca nu l-am folosi, textul "date" ar fi fost printat.

Variabile

In bash, creare variabilelor se face prin asignare. Acestea pot fi accesate prin operatorul **$**. Variabile pot fi rescrise.

```
# assigns a numeric value
x=10
# prints 10
echo $x
# assigns a string value
x="something"
# prints something
echo $x
# assigns a string value containing spaces
x="something else"
# prints something else
echo $x
# assigns a boolean value
x=true
# prints true
echo $x
```

Important: nu se foloseste whitespace la asignare

Exemplu: *Aplicatie de luat notite* 2

Se cere modificarea scriptului tn astfel incat userul sa fie prompat cu un mesaj si ca notitele sa poata fi grupate in functie de categorie (todo, shopping list ,etc.) .

Pentru a putea extinde functionalitatea, ne folosim de functia **read -p** (read with prompt).

```
#!/bin/bash
# simple note-taking application

# get the date
date=$(date)
```

```
# get the topic
# of no argument exists, topic will be empty
topic=$1

# filename to write to
filename=~/${topic}notes.txt

# prompt the user for input
read -p "Your note: " note

# print note in file
echo $date: $note >> $filename

echo "Note '$note' saved to $filename"
```

In acest script, folosim variabilele *date*, *topic*, *note* si *filename*. In *date* salvam data curenta, in *topic* primim ca si parametru categoria din care face parte notita, iar in *note* salvam ce a scris userul.

Nu am putea folosi *$topicnotes.txt* deoarece sistemul s-ar uita dupa o variabila denumita *$topicnotes.txt*, asa ca separam variabila de restul textului folosind *{}*.

Structuri de control

If, else.

Mod de folosinta:

```
# simple if structure
if condition; then
# code to execute if true
fi
# if-else structure
if condition; then
# code to execute if true
else
# code to execute if false
fi
```

Cuvintele cheie, **if** (testeaza conditia), **then** (codul ce va fi executat daca conditia este evaluata ca find adevarata) **else** (codul ce va fi executat daca conditia este evaluata ca find falsa) si **fi** (sfarsitul structurii if), trebuiesc scrise pe un rand nou, in codul de mai sus **;** forteaza interpretorul sa trateze orice dupa el ca si o linie noua. O reprezentare alternativa, ce poate fi scrisa in linia de comanda a unui if ar fi:

```
if condition ; then code to execute if true ; else code to execute if false
; fi
```

Expresii conditionale

In bash, expresiile conditonale pot verifica:

- teste pe fisiere si directoare
- teste string-uri
- teste aritmetice

Structura:

```
# the structure of conditional expression is:
# [[ expression ]]

# str is not empty
[[ $str ]]
# str equals string "something". Whitespace is important
[[ $str = "something" ]]
# true if filename exists
[[ -e $filename ]]
# true if directory exists
[[ -e $dirname ]]
```

Important: Intodeauna folositi spatiu intre semnul "=", daca nu, bash va iterpreta totul ca si o singura variabila, ex: $str="something", exista text, deci conditia va fi intodeauna adevarata

Exemplu: *Aplicatie de luat notite 2*

Daca vrem sa tratam cazul in care utilizatorul nu a introdus nici un input dupa ce a fost promptat, modificam codul astfel:

```
#!/bin/bash
# simple note-taking application

# get the date
date=$(date)

# get the topic
# of no argument exists, topic will be empty
topic=$1

# filename to write to
filename=~/${topic}notes.txt

# prompt the user for input
read -p "Your note: " note

if [[ $note ]]; then
    # print note in file
    echo $date: $note >> $filename
```

```
    echo "Note '$note' saved to $filename"
else
    # print note in file
    echo "No input, note was not saved"
fi
```

In conditia if-ului, evaluam continutul variabilei *$note*, daca aceasta contine un text, executia continua ca si pana acum, daca variabila nu contine nimic, printam textul "No input, note was not saved" si nu salvam nimic in fisier.

Expresii conditionale aritmetice

Structura:

```
[[ arg1 OP arg2 ]]
```

Unde OP poate fi:

- -eq *egal*
- -ne *diferit*
- -lt *mai mic decat*
- -gt *mai mare decat*

Exemplu: Comparare a doua numere

```
#!/bin/bash

arg1=$1
arg2=$2

if [[ $arg1 -gt $arg2 ]] ; then
echo "$arg1 is bigger than $arg2"
else
echo "$arg2 is bigger than $arg1"
fi
```

And, Or, Not

Intr-o expresie conditionala:

```
# use ! to negate a test
# true if file does not exist
[[ ! -e $file ]]

# use && for "and"
# true if file does not exists and we have a parameter given
[[ ! -e $file && $1 ]]

# use || for "or"
# true if file does not exists or we have a parameter given
[[ ! -e $file || $1 ]]
```

While and Until

Structurile while si until sunt asemanatoare cu structura if, diferenta este ca, orice cod este inclus in ele se va repeta pana cand:

- conditia evaluata nu mai este adevarata (while)
- conditia evaluata este adevarata (until)

Structura generala:

```
# while structure
# runs until condition is false
while condition; do
# code to run
done

# until structure
# runs until condition is true
until condition; do
# code to run
done
```

Exemplu: Ghiceste numarul corect

```
#!/bin/bash

# generate target number between 0 and 99
target=$(($RANDOM % 100))
# initialize the user's guess
guess=
```

```
until [[ $guess -eq $target ]] ; do
    # read in user's guess
    read -p "Take a guess: " guess
    if [[ $guess -lt $target ]] ; then
        echo "Higher!"
    elif [[ $guess -gt $target ]] ; then
        echo "Lower!"
    elif [[ $guess -eq $target ]] ; then
        echo "You found it!"
    fi
done
```

In scriptul de mai sus **$RANDOM** reprezinta o variabila de sistem ce contine un numar aleator, de asemenea este introdus si operatorul de operatii aritmetice **$(())**. In interiorul acestui operator ne folosim de operatio modulo, pentru a genera un numar intre 0 si 99. (Restul impartiiri a oricarui numar la 100 va fi intre 0 si 99)

For

Asemanator structuriilor while si until, structura for repeta un cod pana cand o conditie este evaluata ca si fiind adevarata, diferenta este ca ne permite initializarea de variabile si operatii aritmetice.

Structura generala:

```
for (( INIT; CONDITION; UPDATE )) ; do
# code to run
done
```

Exemplu: Afisare numere intre 0 si un numar primit ca si parametru

```
#/bin/bash

# verify argument was given
if [[ ! $1 ]] ; then
    echo "one argument is needed"
    exit
fi

#print numbers between 0 and $1
for (( i=0; i<$1; ++i  )) ; do
    echo $i
done
```

In interiorul parantezelor (()) codul se interpreteaza ca si **operatie aritmetica**.

Cerinte laborator:

- ◯ Sa se scrie un script shell care aduna numerele date ca parametrii in linia de comanda
- ◯ Sa se scrie un script shell care scrie numerele in ordine descrescatoare incepand de la n dat ca parametru folosind while.
- ◯ Sa se inverseze cifrele unui numar (254 -> 452).
- ◯ Scrieti un script care afiseaza data curenta, ora, numele utilizatorului si disrectorul curent.
- ◯ Scrieti un script care determina daca o comanda contine caracterul "*".
- ◯ Scrieti un script care afiseaza numerele lui Fibonacci.
- ◯ Scrieti un script care transforma litere mari in litere mici pentru nume de fisiere primite ca parametru.