Laboratorul 5

Programarea in shell
Shell scripting

Shell-ul poate fi folosit si pentru a interpreta continutul unor fisiere care contin comenzi si a le executa. Aceste fisiere de comenzi poarta numele de *script*-uri. Ele pot fi vazute ca programe executabile in format text. Pentru acest lucru, este necesar ca fisierul script, *myscript* in exemplul de mai jos, sa aiba permisiuni de executie setate cel putin pentru proprietarul fisierului:

```
$ chmod u+x myscript
```

Ulterior adaugarii permisiunilor de executie, fisierul script se poate executa ca orice fisier executabil compilat, de ex, considerand fisierul *myscript* de mai sus:

```
$ ./myscript
```

In general, pentru a preciza foarte exact pentru ce tip de shell (sau de interpretor, in sensul cel mai general) este potrivit scriptul, in mod uzual prima linie din fisierul script are o sintaxa speciala prin care specifica programul care va interpreta scriptul. Aceasta prima linie instruieste shell-ul ce interpretor de comenzi sa lanseze in executie pentru a interpreta si executa comenzile din script. De exemplu, un shell *bash* poate lansa in executie un fisier script caruia i s-au dat drepturi de executie ca mai sus si care contine urmatoarele comenzi:

```
#!/bin/bash

echo "Hello scripting world !"
exit 0
```

Prima linie este un comentariu (incepe cu caracterul #) special (imediat dupa # urmeaza !) si care specifica interpretorul de comenzi (*/bin/bash*) care este folosit pentru a interpreta si executa comenzile care urmeaza. Ultima linie care apeleaza comanda *exit* cu parametrul 0 evidentiaza o buna practica in programarea Unix in general (si shell scripting-ul nu face exceptie) prin care se

comunica procesului parinte (shell-ul care a lansat in executie scriptul de mai sus in cazul nostru) codul de terminare al programului (al scriptului in cazul de mai sus). Conventia Unix spune ca orice program care se termina fara eroare intoarce un cod 0. Orice alta valoare de retur reprezinta a priori o terminare eronata a programului.

In cazul cel mai general, prima linie din script poate desemna orice tip de interpretor care desigur va fi folosit pentru a interpreta comenzile care urmeaza in script. Iata un script *awk*, care poate fi lansat si el in executie de catre orice alt interpretor de comenzi, inclusiv bash, si care tipareste argumentele primite de script in linia de comanda:

```
#!/usr/bin/awk -f

BEGIN   {
        for(i = 0; i < ARGC; i++)
                printf "ARGV[%d] = %s\n", i, ARGV[i]
        exit
}
```

Dupa cum se observa, *awk* este practic un limbaj de programare cu structura apropiata de limbajele de programare compilate.

In fapt, toate interpretoarele de comenzi au si propriul limbaj de programare, cu propria sintaxa si propriile comenzi. In cele ce urmeaza, vor fi prezentate principalele "instructiuni" pe care le intelege Bourne Again Shell: loop-uri (while/for), instructiuni conditionale (if/case) si functii.

1 Test

Operatia de test uzual folosita in instructiuni conditionale sau iterative este implementata de *bash* cu ajutorul comenzii interne *test* sau *[*. Pentru a verifica ca *[* este de fapt o comanda puteti folosi comanda interna *type*:

```
$ type test
test is a shell builtin
$ type [
[ is a shell builtin
$ type type
type is a shell builtin
$
```

Fiind un program ca oricare altul, *[* are nevoie de spatii in jurul sau. De pilda, expresia *[$myvar = "somestring"]* va genera o eroare de sintaxa fiind interpretata drept *test$myvar = "somestring"]*. Versiunea corecta a comenzii anterioare este *[$myvar = "somestring"]*.

Comanda *test* este extrem de complexa si poate testa stringuri, numere, fisiere. Pentru o imagine comprehensiva, consultati pagina de manual. In cele ce urmeaza vom evidentia cateva dintre utilizarile des intalnite ale comenzii,

apeland la comenzile conditionale puse la dispozitie de catre shell. In sectiunile urmatoare vom vedea cum poate fi folosita comanda *test* impreuna cu instructiuni conditionale si iterative.

Pentru exemplele care urmeaza vom folosi o variabila shell *X* careia i se vor asigna valori diverse si o vom folosi impreuna cu comanda *test*. Pentru lizibilitate vom folosi caracterul \care permite extensia unei singure linii de comanda pe linia urmatoare (atat interactiv, la promptul shell-ului, cat si in scripturi). In mod interactiv, utilizarea \implica automat aparitia promptului de continuare ">" stocat in variabila de mediu PS2.

```
$ X=4
$ [ "$X" -lt "0" ] && echo "X is less than zero" \
> || echo "X is greater than zero"
X is greater than zero

$ X=-1
$  [ "$X" -lt "0" ] && echo "X is less than zero" \
> || echo "X is greater than zero"
X is less than zero

$ X=0
$ [ "$X" = "0" ] && \
>       echo "X is the string or number \"0\""
X is the string or number "0"

$ X=hello
$ [ "$X" = "hello" ] && \
>       echo "X matches the string \"hello\""
X matches the string "hello"

$ X="not hello"
$ [ "$X" != "hello" ] && echo "X is not the string \"hello\""
X is not the string "hello"

$ echo $X
not hello
$ [ -n "$X" ] && echo "X is of nonzero length"
X is of nonzero length

$ X=somefile
$ [ -f "$X" ] && \
>       echo "X is the path of a real file" || \
>       echo "No such file: $X"
No such file: somefile

$ X=/etc/passwd
```

```
$ [ -f "$X" ] && \
>       echo "X is the path of a real file" || \
>       echo "No such file: $X"
X is the path of a real file

$ X=/bin/ls
$ [ -x "$X" ] && echo "X is the path of an executable file"
X is the path of an executable file

$ X=.
$ [ "$X" -nt "/etc/passwd" ] && \
>       echo "X is a file which is newer than /etc/passwd"
X is a file which is newer than /etc/passwd
$

$ X=hello
$  [ -n "$X" ] && echo "$X is a non-zero string" \
> || echo "$X is the null string"
hello is a non-zero string

$ Y=bye
$ [ $X != $Y ] && echo "$X is not equal to $Y" \
> || echo "$X is equal to $Y"
hello is not equal to bye
$
```

Expresiile supuse comenzii *test* pot fi compuse cu ajutorul operatorilor logici: negatie, conjunctie si disjunctie.

```
$ echo $X
hello
$ [ ! "$X" = "" ] && echo "$X is not an empty string" \
> || echo "$X is the empty string"
hello is not an empty string

$ echo $X $Y
hello bye
$ [ "$X" = "hello" -a "$Y" = "bye" ] && \
> echo "Proper greetings have been made" \
> || echo "This is not quite polite"
Proper greetings have been made

$ Y=notbye
$ echo $X $Y
hello notbye
$ [ "$X" != "hello" -o "$Y" != "bye" ] && \
> echo "This is not quite polite" \
```

```
> || echo "Politeness has been satisfied"
This is not quite polite
$
```

2 Instructiunile de tip If si Case

Instructiunile conditionale de tip *if* din shell arata in felul urmator:

```
if [ ... ]
then
  # if-code
else
  # else-code
fi
```

Observati pozitia cuvintelor cheie *then*, *else* si respectiv *fi*. Ele trebuie sa apara singure pe o linie chiar la inceputul ei. Pentru a face economie de spatiu, se poate folosi caracterul ";" care permite unirea a doua linii consecutive pe aceeasi linie:

```
if [ ... ]; then
  # if-code
else
  # else-code
fi
```

Mai multe *if-uri* se pot imbrica folosind instructiunea *elif*, ca in scriptul urmator, pe care il puteti salva in fisierul *if.sh* (atat pentru acest script cat si pentru toate cele care urmeaza, nu uitati sa setati permisiunea de executie inainte de a rula scriptul):

```
#!/bin/bash

read filename
if  [ ! -e "$filename" ]; then
 echo "$filename does not exist"
 elif [ -f "$filename" ]; then
   echo "$filename is a regular file"
 else
   echo "$filename exists, but we don't know what kind of file it is"
fi
```

Rularea scriptului evidentiaza functionalitatea instructiunii *if*:

```
$ ./if.sh
somefile
somefile does not exist
$ ./if.sh
```

```
/etc/passwd
/etc/passwd is a regular file
$ ./if.sh
/etc
/etc exists, but we don't know what kind of file it is
$
```

Pentru a face scriptul de mai sus ceva mai complet se poate folosi instructiunea *case*, ca in scriptul de mai jos pe care il vom denumi *case.sh*:

```
#!/bin/bash

echo -n "Please input a filename: "
read filename
longformat=`ls -ld $filename 2> /dev/null`
case "${longformat:0:1}" in
        -)
                echo "$filename is a regular file"
                ;;
        d)
                echo "$filename is a directory"
                ;;
        b)
                echo "$filename is a block file"
                ;;
        c)
                echo "$filename is a character file"
                ;;
        *)
                echo "Sorry, I don't know anything about $filename"
                ;;
esac
```

Rememorati din laboratoarele anterioare ca orice comanda care apare intre backquotes este inlocuita cu un string care reprezinta rezultatul executiei comenzii dintre backquotes. Redirectarea *stderr* reprezentata prin descriptorul de fisier 2 este necesara pentru a trata situatia in care numele fisierului introdus de la tastatura nu corespunde unui fisier existent. */dev/null* este un fisier de tip caracter cu un comportament special in sistem: orice scriere la */dev/null* se pierde, iar de la */dev/null* nu se poate citi nimic. Pe cale de consecinta, redirectarea *stderr* la */dev/null* are ca efect eliminarea mesajului de eroare generat de comanda *ls -l* atunci cand numele de fisier citit de la tastatura nu corespunde unui fisier existent in sistem.

Asa cum am mentionat la curs, listarea in format lung a atributelor unui fisier are ca efect, printre altele, tiparirea pe ecran ca prim caracter a tipului fisierului. Pentru a izola acest caracter in cadrul stringului generat de comanda *ls -l* se foloseste sintaxa *${nume_variabila:offset:length}* care permite obtinerea

unui substring al variabilei *nume_variabila* care incepe la pozitia *offset* si are lungimea *length*.

Rezultatele rularii scriptului de mai sus arata de maniera urmatoare:

```
$ ./case.sh
Please input a filename: /etc
/etc is a directory
$ ./case.sh
Please input a filename: /etc/passwd
/etc/passwd is a regular file
$ ./case.sh
Please input a filename: /dev/tty1
/dev/tty1 is a character file
$ ./case.sh
Please input a filename: /dev/sda1
/dev/sda1 is a block file
$ ./case.sh
Please input a filename: somefile
Sorry, I don't know anything about somefile
$
```

3 Instructiuni iterative

Limbajele interpretate ofera si functionalitate de tipul instructiunilor iterative, a instructiunilor executate repetitiv intr-o bucla. Principalele constructii sintactice pe care le ofera bash pentru implementarea buclelor sunt instructiunile *for* si *while*.

3.1 For loops

Instructiunile de tip *for* itereaza printr-o lista de valori. Executati urmatorul script pe care-l salvati intr-un fisier *myfor-loop.sh*:

```
$ cat -> myfor-loop.sh
#!/bin/bash
for i in 1 2 3 4 5
do
  echo "Iteratia cu numarul $i"
done
$ chmod u+x myfor-loop.sh
$ ./myfor-loop.sh
Iteratia cu numarul 1
Iteratia cu numarul 2
Iteratia cu numarul 3
Iteratia cu numarul 4
Iteratia cu numarul 5
```

```
$
```

Observati ca *do* si *done*, cuvinte cheie ale instructiunii *for* trebuie sa apara pe o linie separata chiar la inceputul ei. Daca modificati scriptul si mutati cuvantul cheie *do* pe aceeasi linie cu *for* la executia scriptului veti obtine o eroare de sintaxa. Puteti repara aceasta eroare folosind caracterul special *;* care uneste doua linii:

```
#!/bin/bash
for i in 1 2 3 4 5; do
  echo "Iteratia cu numarul $i"
done
```

Valorile din lista pot fi practic orice. De pilda, incercati sa executati urmatorul script, pe care-l puteti salva in fisierul *myfor-loop2.sh*:

```
#!/bin/bash
for i in math 1.5 * 2 done
do
  echo "Valoarea lui i este $i"
done
```

Ca sa intelegeti comportamentul instructiunii *for* in acest caz este important sa va reamintiti de interpretarea wildcard-urilor in bash prezentata la curs. Incercati sa rulati scriptul de mai sus cu si fara *. Dar daca folositi escape character * ? Si daca folositi *, ce se intampla daca in corpul instructiunii *for* afisati valoarea variabilei *i* fara sa o includeti in ghilimele (ca mai jos)?

```
echo Valoarea lui i este $i
```

3.2 While loops

Instructiunea *while* este cel mai adesea folosita impreuna cu comanda *test*, ca in exemplul de mai jos pe care il puteti salva in fisierul *while.sh*:

```
#!/bin/bash
while [ "${INPUT_STRING:-hello}" != "bye" ]
do
  echo "Introduceti date (bye pentru a iesi din bucla)"
  read INPUT_STRING
  echo "Ati introdus: $INPUT_STRING"
done
```

Scriptul de mai sus citeste siruri de caractere de la tastatura in bucla pana cand se tipareste "bye". Ce se intampla daca variabila INPUT_STRING nu are valoare intiala?

Scriptul de mai sus se poate modifica in urmatoarea varianta (*while2.sh*) care nu foloseste comanda test ci caracterul ":" (echivalent cu comanda *true*) care intoarce permanent valoarea de adevar:

```
#!/bin/bash
while :
do
  echo "Introduceti date (Ctrl-C pentru a iesi din bucla)"
  read INPUT_STRING
  echo "Ati introdus: $INPUT_STRING"
done
```

De asemenea, instructiunea *while* se foloseste des impreuna cu *read*. Ca exemplu aveti scriptul de mai jos (*while3.sh*), care modifica scriptul anterior *case.sh* pentru a permite introducerea de nume de fisiere in mod repetitiv:

```
#!/bin/bash

echo -n "Please input a filename: "
while read filename
do
        longformat=`ls -ld $filename 2> /dev/null`
        case "${longformat:0:1}" in
                -)
                        echo "$filename is a regular file"
                        ;;
                d)
                        echo "$filename is a directory"
                        ;;
                b)
                        echo "$filename is a block file"
                        ;;
                c)
                        echo "$filename is a character file"
                        ;;
                *)
                        echo "Sorry, I don't know anything about $filename"
                        ;;
        esac
        echo -n "Please input a filename: "
done
```

Puteti incheia introducerea datelor cu Ctrl-d sau sa terminati programul cu Ctrl-c.

4 Mai multe despre variabile

Pe langa variabilele shell-ului (variabile interne si variabile de mediu) exista un set de variabile speciale care de cele mai multe ori nu pot fi modificate.

Primul set de variabile de interes este cel al variabilelor *0, 1, ... 9*. Variabila *$0* de pilda reprezinta numele programului, numit in general *basename*, in vreme

ce restul variabilelor pana la 9, *$1, $2,..., $9* reprezinta parametrii cu care a fost apelat scriptul. Variabila @ reprezinta acesti parametri, al caror numar exact este stocat in variabila #. Pentru intelegera mai buna a acestor aspecte, rulati urmatorul script *param.sh*:

```
#!/bin/sh

echo "I was called with $# parameters"
echo "My name is $0"
echo "My nicer name is `basename $0`"
echo "My first parameter is $1"
echo "My second parameter is $2"
echo "All parameters are $@"
```

Comanda externa *basename* este folosita pentru a elimina calea din numele scriptului. Iata aici rezultatele unei posibile rulari:

```
$ ./param.sh
I was called with 0 parameters
My name is ./param.sh
My nicer name is param.sh
My first parameter is
My second parameter is
All parameters are
$ ./param.sh first second third fourth
I was called with 4 parameters
My name is ./param.sh
My nicer name is param.sh
My first parameter is first
My second parameter is second
All parameters are first second third fourth
$
```

Cu ajutorul comenzii *shift* se pot folosi mai mult de 9 parametri in linia de comanda. Scriptul urmator, *param2.sh*, evidentiaza functia comenzii *shift* care, atunci cand e apelata, itereaza printre parametrii de apel ai scriptului ("shifteaza" parametrii la stanga). Pe masura ce se shifteaza parametrii, numarul lor, continut in variabila $#, scade.

```
#!/bin/sh
while [ "$#" -gt "0" ]
do
  echo "\$1 is $1"
  shift
done
```

Rezultatul executiei comenzii demonstreaza felul in care se pot accesa toti parametrii de apel, indiferent de numarul lor:

```
$ ./param2.sh 1 2 3 4
$1 (first parameter) is 1
$1 (first parameter) is 2
$1 (first parameter) is 3
$1 (first parameter) is 4
$
```

In final, asa cum am discutat la curs si in laboratoarele anterioare, variabila *?* contine valoarea codului de retur cu care s-a incheiat comanda anterioara. Valoarea acestui cod de retur se poate folosi pentru a notifica utilizatorul in privinta rezultatului executiei unei comenzi ca in scriptul de mai jos (*retur.sh*):

```
#!/bin/sh

read cmd
eval "$cmd" 2> /dev/null
if [ "$?" -ne "0" ]; then
  echo "$cmd has failed!"
fi
```

Comanda interna shell *eval* evalueaza stringul furnizat ca argument drept comanda si intoarce codul de return al acestei comenzi. Iata cateva exemple de rulare:

```
$ ./retur.sh
somecmd
somecmd has failed!
$ ./retur.sh
[ "0" -gt "1" ]
[ "0" -gt "1" ] has failed!
$
```

5 Sarcini de laborator

1. Executați toate comenzile prezentate în acest laborator.
2. Modificati unele din scripturile de la laborator pentru a primi datele necesare ca parametri in linie de comanda, in loc sa fie citite de la tastatura (eg, scripturile care identifica tipuri de fisiere).
3. Scrieti propria versiune a comenzii *ls* fara nici un parametru (adica pentru a lista continutul directorului curent).
4. Scrieti un script care foloseste o comanda de tip pipeline pentru a afisa utilizatorii si PID-urile proceselor lor asa cum sunt afisate de comanda *ps auxw*. Scriptul trebuie sa captureze intr-un pipe outputul comenzii *ps auxw* si sa itereze prin fiecare linie tiparind primele doua campuri ale outputului, cele care corespund utilizatorului si PID-ului.

5. Scrieti un script *myfind* care emuleaza comportamentul simplificat al comenzii *find* cu flagurile *-name*, *-type* si *-exec*. Comanda primeste ca prim parametru un director si nu functioneaza recursiv (i.e., maxdepth = 1). Trebuie sa fie capabila sa gaseasca un fisier dupa nume si/sau tip si odata identificat sa poata executa o comanda asupra lui. Numele, tipul si comanda de executie sunt furnizate ca parametrii in linia de comanda, exact ca pentru comanda *find (man find)*.