Laborator 13

Scripting

1. Generalitati

Un shell este un macro-procesor ce executa comenzi. Scripturile sunt fisiere text ce contin insiruiri de comenzi de comenzi de shell. Ele pot avea acelasi status ca si comenzile standard oferite de shell, permitand usurarea lucrului si personalizarea mediului oferit de sistemul de operare. Exemple de shell-uri de Unix/Linux sunt bash, csh, ksh, zsh; in mediul Windows sunt folosite fisierele batch.

2. Bash

Bash este un shell Unix ce ofera atat un interpretor de comenzi, cat si un limbaj de programare.
Documentatia standard este accesibila prin info **`bash.`**

3. Apel, variabile, parametrii

Un script poate fi apelat fie prin „bash myscript" fie prin ./myscript. In al doilea caz, fisierul trebuie sa fie executabil (chmod u+x myscript).
Scriptul trebuie sa aiba pe primul rand secventa #!/bin/bash. Primele doua caractere reprezinta magic-ul pentru fisierele script.
http://en.wikipedia.org/wiki/Magic_number_%28programming%29
Scripturile pot primi oricati parametrii. Acestia sunt accesibili cu ajutorul variabilelor *$1 $2 $3*...
Variabilele au numele case-sesitive. Nu trebuiesc definite, ci pot fi folosite direct (nu au tipuri).
Variabile utile:

- *$#* numarul de parametrii
- *$@* toti parametrii
- *$0* numele scriptului
- *$$* PID-ul scriptului
- *$?* exit-code anterior

Cand este atribuita o variabila, se foloseste semnul ‚=' FARA SPATII.

4. Substitutii (quoting)

In bash exista patru interpretari ale valorii unei variabile, un functie de felul de incadrare in care se afla (poziti si tipul „ghilimelelor"):

1. nu este incadrata intre ghilimele ($var) – sirul de caractere atribuit variabilei ramane la fel, cu exceptia spatiilor albe consecutive ce apar in interiorul sirului cu valorile lor
2. intre ghilimele simple ('$var') – variabila isi pastreaza forma literala (ea nu este interpretata)
3. intre ghilimele duble („$var") - variabila este interpretata si sunt permise caractere speciale (\n, \t, etc.); in plus, spatierea in cadrul sirului se pastreaza, spre deosebire de primul caz (variabila se comporta ca un singur cuvant)
4. intre backquotes (`$var`) - valoarea expresiei `$var` este output-ul obtinut in urma executiei efective a sirului de caractere „$var".

Exemplu:

```
#!/bin/bash

PATH='/mnt/Windows/Documents and Settings/mirc/My Documents'

# acesta e un comentariu
echo "primul parametru este $1"
echo 'el se gaseste in variabila $1'

TEXT="acesta este  un text  care  are mult prea  multe  spatii"
echo "$TEXT"; echo $TEXT
echo "toti parametrii sunt: $*"
echo "numele scriptului este $0, iar numarul de parametrii este $#"

VAR1=scr; VAR2=ipt
echo "acesta este un exemplu de ${VAR1}${VAR2}" #concatenare

VAR1=scr${VAR2} #concatenare si schimare de variabila
echo "Numele utilizatorului care a lansat scriptul este $USER"

echo \z      # z
echo \\z     # \z
echo '\\z'   # \\z
```

5. Substitutii folosind „wildcard”-uri (Globbing)

Substitutia este modalitatea prin care shell-ul expandeaza un string care contine wildcard-uri (caractere speciale), intr-o lista de string-uri posibile.
Trei dintre cele mai cunoscute metode de globbing sunt:

- * - semnifica aparitia oricarui caracter(e) de zero sau mai multe ori
- ? - seminifca aparitia singulara a oricarui caracter
- [caractere] – seminifica aparitia singulara a oricarui caracter din sirul „caractere”

Folosirea * (Exemple)

```
# Afisarea tuturor fisierelor si directoarelor care incep cu rc
        ls /home/student/file*                  # file1 file2 file3
# Afisarea tuturor fisierelor care se termina in tar.gz
        ls /home/student/*.tar.gz   # poze.tar.gz arhiva.tar.gz scripting.tar.gz
```

Folosirea ? (Exemple)

```
# Afisarea tuturor fisierelor cu numele din 3 caractere
        ls $HOME/???
# Schimbarea permisiunilor pentru toate fisierele din 4 caractere, avand caracterele 2 si 3 gz
        chmod u+rx ./?gz?
```

Folosirea [abc] (Exemple)

```
#  Afisarea tuturor fisierelor care incep cu majuscula
        ls [A-Z]*
# Mutarea tuturor fisierelor care NU se termina intr-o cifra sau litera
        mv *[!a-zA-Z0-9]
```

6. Operatii aritmetice

Putem folosi variabilele pentru a efectua calcule cu numere intregi (bash nu suporta numere reale). Operatiile posibile sunt cele uzuale: +, -, *, /, % (modulo), ** (ridicare la putere). De asemenea, exista operatori pentru operatii la nivel de bit si pentru calcul boolean (aceiasi ca in C), interpretarea in diferite bazi etc. Exista multe metode insa vom prezenta numai una.

```
a=3             # valoarea lui a
a=$[$a%2]       # 1
a=$[$a<<1]      # 2       #left shift (operatie la nivel de bit)
b=$[2#11]       #  11 in baza 2, deci b=3
a=$[$a+$b]      # 5
```

7. Operatii pe siruri

Cele mai importante prelucrari de siruri:

- ${#sir} – reprezinta lungimea sirului
- ${sir:pos}, ${sir:pos1:pos2} – extrag din **sir**, ***subsirul*** de la pozitia *pos* +1 pana la sfarsit sau de la pozitia *pos1*+1, un numar de caractere egal cu *pos2*.
- ${sir#subsir}, ${sir##subsir}, ${sir%subsir}, ${sir%%subsir} – elimina din **sir** (de la inceput daca se foloseste simbolul # sau de la sfarsit daca se foloseste simbolul %) cea mai scurta portrivire (daca e un singur simbol) sau cea mai lunga potrivire (daca sunt doua simboluri consecutive).
- ${sir/vechi/nou}, ${sir//vechi,nou}, ${sir/#vechi,nou}, ${sir/%vechi,nou} – sunt folosite pentru inlocuirea de ***subsiruri*** in **sirul** dat. ***Subsirul,*** $nou inlocuieste ***subsirul*** $vechi astfel: numai pentru prima aparitie, pentru toate aparitiile, numai daca $sir incepe cu $vechi, numai daca $sir se termina cu $vechi.

Oriunde in constructiile de mai sus apare un subsir, el poate fi constituit dintr-o expresie regulata, nu neaparat dintr-un sir fixat. Explicatii foarte clare despre expresiile regulate folosite in Unix se gasesc in sectiunea din **`man grep`**.

```
#!/bin/bash

sir=abcABC123ABCabc
echo ${sir}           # abcABC123ABCabc
echo ${#sir}          # 15
echo ${sir:10}        # BCabc
echo ${sir:3:6}       # ABC123
echo ${*:2:2}         # parametrii 2 si 3 ai scriptului

echo ${sir#a*C}       # sterge cea mai mica potrivire a sirului "a*C"
                      # pornind de la inceput, deci subsirul "abcABC"
                      # rezultatul este "123ABCabc"

echo ${sir##a*C}      # sterge cea mai mare potrivire a sirului "a*C"
                      # rezultatul este "abc"

echo ${sir%b*c}       # sterge potrivirea minima de la coada
                      # rezultatul este "abcABC123ABCa"
echo ${sir%%b*c}      # sterge potrivirea maxima de la coada
                      # rezultatul este "a"

FILE=arhiva.tar.gz
echo "fisierul '${FILE%%.*}' are extensia '${FILE#*.}'"
                      # fisierul 'arhiva' are extensia 'tar.gz'
```

8. Comenzi de decizie

Sintaxa pentru if este urmatoarea

```
if COMANDA-TEST; then
COMENZI;
[elif ALTE-COMENZI-TEST; then
ALTE-COMENZI;]
[else COMENZI-ALTERNATIVE;]
fi
```

Daca COMANDA-TEST are un exit-code egal cu 0, atunci se executa COMENZI; altfel, se testeaza in continuare eventualele comenzi din “elif”-uri (ALTE-COMENZI-TEST), pana cand se gaseste o

conditie care are exit-code-ul 0, caz in care se executa ALTE-COMENZI. Daca exista o clauza “else” si s-a ajuns la ea fara a se executa ceva, atunci se executa COMENZI-ALTERNATIVE. Observati ca blocul if se termina cu “fi”.

Pentru comenzile de test se foloseste cel mai des forma [operatii de test] . Intre operatori si operanzi trebuie lasate spatii; la fel, trebuie sa existe spatii dupa ’[’ si inainte de ’]’. Explicatia este urmatoarea: bash emuleaza intern comportarea programului test (a se vedea man test); de altfel si ’[’ este doar un symlink catre test. Nu trebuie sa uitati ca operatorii si operanzii sunt de fapt parametri ai unui program si de aceea trebuie separati prin spatii, la fel ca si caracterul ’]’ care trebuie sa fie ultimul ’parametru’. Sunt permise operatii asupra fisierelor, numerelor intregi si string-urilor, intr-o maniera uniforma; cele mai importante sunt sintetizate in tabelul urmator:

Operator	Descriere	Exemplu => exit-code
-e fisier	fisierul exista	-e /etc/passwd => 0
-d fisier	fisierul este un director	-d /root => 0
-f fisier	e un fisier obisnuit	-f ~/.bashrc => 0
-L fisier	fisierul e un symlink	-L /etc/rc2.d/klogd => 0
-s fisier	are dimensiunea >0	-s /proc sau -s /dev/null =>1, -s / =>0
-r fisier	fisierul e 'readable'	-r /var/log/messages => 0
-w fisier	fisierul e 'writeble'	-w /etc/passwd => 1 -w $HOME => 0
-x fisier	fisierul este executabil	-x /etc/fstab => 1, -x /etc =>0
fisier1 -nt fisier2	e mai actual decat	-
fisier1 -ot fisier2	e mai vechi decat	-
num1 -eq num2	egalitate numerica	3 -eq 3 =>0
num1 -ne num2	inegalitate numerica	3 -ne 3 =>1
num1 -lt num2	mai mic	4 -lt 4 =>1
num1 -le num2	mai mic sau egal	4 -lt 4 =>0
num1 -gt num2	mai mare sau egal	4 -gt 4 =>1
num1 -ge num2	mai mare sau egal	4 -gt 4 =>0
-z string	string de lungime 0	-z „” =>0
-n string	string nevid	-n „” =>1, -z „abc” =>0
string1=string2	egalitate de stringuri	„borring”=”borring” =>0
string1!=string2	inegalitate de stringuri	„Borring”!=”borring” =>0
string1\>string2	mai mare lexicografic	„borring” \> „Borring” =>0
string1\<string2	mai mic lexicografic	„borring”\<”borrinG” =>1

Testele simple se pot conecta logic cu altele folosind operatori de AND, OR si NOT, adica ’-a’, ’-o’ respectiv ’!’

O alta contructie pentru decizie este **case**. Sintaxa ei este urmatoarea:

case CUVANT in
[[(] SABLON [| SABLON]...) LISTA-COMENZI ;;]
...
esac

Aceasta contructie sintactica este compusa din mai multe clauze dintre care una singura se executa (prima care se potriveste); regula este potrivirea lui CUVANT cu expandarea de cale a lui SABLON (deci in sablon pot aparea expresii regulate) . Astfel, daca se doreste executia unor comenzi in cazul in care nici una din clauzele anterioare nu s-a potrivit, se poate scrie sablonul '*)' ca singurul sablon al ultimei clauze; acesta va realiza o potrivire oricare ar fi CUVANT. Fiecare clauza trebuie terminata prin ';;' , iar sabloanele multiple(alternative) in aceeasi clauza trebuie separate prin '|' . La sfarsitul constructiei trebuie sa se scrie 'esac'

```
#!/bin/bash
# iesim daca nu avem parametru sau nu exista fisierul sau are dimensiune 0
if [ -z $1 -o ! -e $1 -o ! -s $1 ]; then exit 1; fi
if [ -f $1 ]; then              # e fisier obisnuit
      case "${1#*.}" in       # extensia
      tgz|tar.gz)             # selectie alternativa ?|?
            tar xzvf $1
            ;;              # trebuie ?;;?
      tar.bz2)
            tar xjvf $1
            ;;
      *) echo "Format necunoscut." ; exit 1;; # default
      esac

# altfel, daca e director & exista 2 param & al doilea e un caracter
elif [ -d $1 -a "$#" = "2" -a "${#2}" = "1" ]; then
      case $2 in
      g|G)
            TYPE=z; NAME=gz
            ;;
      b|B)
            TYPE=j; NAME=bz2
            ;;
      *)
            echo "Folositi parametrii 'g' pentu gzip, sau 'b' pentru bz2"
```

```
        esac
        tar ${TYPE}cvf ${1}.tar.${NAME} $1
    fi
    exit $? # exit-code-ul ultimei operatii
```

9. Instructiuni de ciclare

Instructiunea for are urmatoarea sintaxa

```
for CUVANT [in LISTA]
do
        COMENZI ...
done
```

La fiecare ciclu, CUVANT ia urmatoarea valoare precizata in LISTA. Daca 'do' este scris pe aceeasi linie cu 'for', atunci trebuie precedat de ';' (la fel cum se intampla cu then si else in cazul lui if). Constructia se termina obligatoriu cu done. Daca LISTA nu exista, atunci CUVANT ia pe rand valorile parametrilor de la rularea scriptului (adica LISTA este $@). Daca lista exista, atunci termenii ei sunt supusi expandarii de cale, ca mai jos; termenii distincti ai listei sunt considerati cei ce in urma expandarilor uzuale sunt separati prin spatii.

```
for x in /etc/r??? /var/lo* $HOME/.*; do
        echo $x; stat $x;
done
```

LISTA nu trebuie precizata neaparat de la inceput, ea poate rezulta in urma evaluarii unei variabile (ca mai sus) sau a executiei unei comenzi. Mai mult, output-ul sau input-ul poate fi redirectat pentru for in totalitate (ca de altfel pentru toate constructiile if, case, while, until).

Comanda **`while`**

```
while command
do
        list
done
```

command – este in mod normal o comanda de tip test
list – este o comanda sau un set de comenzi

Traducerea ar fi urmatoarea:

1. Executa *command*
2. Daca exit-status pentru *command* este diferit de 0 se iese din bucla
3. Daca exit-status pentru *command* este 0, executa *list*

4. Cand s-a terminat *list*, intoarcete la pasul 1

Exemplu:

```
#!/bin/bash
x=0                         # initializare variabila x
while [ $x -lt 10 ]# cat timp x<10
do
        echo $x  # afiseaza x
        x=$[$x+1]           # incrementeaza x
done
```