Ghid de utilizare Linux (IV) :

Interpretoare de comenzi UNIX*, partea a II-a: Programare* BASH

Cristian Vidrașcu

vidrascu@info.uaic.ro

Sumar

- Introducere
- Proceduri *shell* (*script*-uri)
- Variabile de *shell*
- Structuri de control pentru *script*-uri
- Alte comenzi utile pentru *script*-uri
- Funcții *shell*

Introducere

Interpretorul de comenzi oferă facilități de programare într-un limbaj propriu, cu ajutorul căruia se pot scrie *script*-uri, adică fișiere text ce conțin secvențe de comenzi `UNIX`.

Shell-ul posedă toate noțiunile de bază specifice oricărui limbaj de programare, precum ar fi: variabile, instrucțiuni de atribuire, instrucțiuni de control structurate (`if`, `while`, `for`, `case` ș.a.), proceduri și funcții, parametri, etc.

Proceduri *shell* (*script*-uri)

Procedurile *shell* (numite și *script*-uri) sunt fișiere text ce conțin secvențe de comenzi `UNIX`.

(*Remarcă*: în `MS-DOS` sau `Windows` avem similar fișierele batch `*.bat`)

Recuperarea în procedura *script* a argumentelor de apel se face cu ajutorul unor variabile speciale, cu numele $1, $2, . . . , $9.

Caracterul '#' urmat de un text este folosit pentru a indica o linie de comentariu într-un *script*.

Apelul unui *script* se poate face prin oricare dintre cele 3 forme de apel disponibile:

Execuția procedurilor *shell*

Formele de apel (*i.e.*, lansare în execuție) a unui script:

`UNIX>` *`nume_script`* [*`parametri`*]

Se creează un nou proces ce rulează neinteractiv *shell*-ul specificat pe prima linie a *script*-ului, prin construcția: #!*nume_interpretor_comenzi* (sau *shell*-ul curent, dacă nu este specificat), și acesta va executa linie cu linie comenzile din acel fișier.

`UNIX>` *`nume_shell nume_script`* [*`parametri`*]

Ca mai sus, procesul *shell* nou creat fiind cel specificat la prompter.

`UNIX>` `source` *`nume_script`* [*`parametri`*]

sau `UNIX>` `.` *`nume_script`* [*`parametri`*]

Nu se mai creează un nou proces *shell*, ci *shell*-ul curent va executa linie cu linie comenzile din acel fișier.

Variabile de *shell*

O facilitate comună tuturor interpretoarelor de comenzi UNIX este utilizarea de *variabile*, cu mențiunea că variabilele sunt, implicit, de tip șir de caractere.

Variabilele sunt păstrate într-o zonă de memorie a procesului *shell* respectiv, sub formă de perechi *nume=valoare*.

(*Remarcă*: interpretoarele din MS-DOS sau Windows au conceptul similar de variabile.)

Instrucțiunea de atribuire are sintaxa următoare:

UNIX> *var=expr*

unde *var* este un identificator (*i.e.*, un nume) de variabilă, iar *expr* este o expresie care trebuie să se evalueze la un șir de caractere.

Atenție: semnul '=' nu trebuie să fie precedat ori urmat de spații albe!

Variabile de *shell* (cont.)

Referirea la valoarea unei variabile (*i.e.*, atunci când avem nevoie de valoarea variabilei într-o expresie) se face prin numele ei precedat de simbolul '$', efectul fiind substituția numelui variabilei prin valoarea ei în expresia respectivă.

Exemple:

```
UNIX> v=123                          # variabila v primește valoarea "123"
UNIX> echo $v                        # se afișează pe ecran textul "123"
UNIX> cat xyv                        # se afișează conținutul fișierului "xyv"
UNIX> cat xy$v                       # se afișează conținutul fișierului "xy123"
UNIX> v=abc${v}xyz                   # variabila v primește valoarea "abc123xyz"
UNIX> set                            # se afișează lista variabilelor definite
UNIX> unset v   sau   UNIX> v=       # variabila v este distrusă
```

Variabile de *shell* (cont.)

Alte forme de substituție:

- `${`*var*`:-`*sir*`}`
  Efect: rezultatul expresiei este valoarea variabilei *var*, dacă aceasta este definită, altfel este valoarea *sir*.
- `${`*var*`:-}`
  Efect: rezultatul expresiei este valoarea variabilei *var*, dacă este definită, altfel este afișat un mesaj standard de eroare care spune că acea variabilă este nedefinită.
- `${`*var*`:=`*sir*`}`
  Efect: rezultatul expresiei este valoarea variabilei *var*, după ce eventual acesteia i se asignează valoarea *sir* (asignarea având loc doar dacă *var* era nedefinită).
- `${`*var*`:?`*sir*`}`
  Efect: rezultatul expresiei este valoarea variabilei *var*, dacă aceasta este definită, altfel este afișat mesajul *sir* (sau un mesaj standard de eroare, dacă *sir* lipsește).
- `${`*var*`:+`*sir*`}`
  Efect: dacă *var* este deja definită, atunci i se asignează valoarea *sir*, altfel rămâne fără valoare (deci asignarea are loc doar în cazul în care *var* era deja definită).

Variabile de *shell* (cont.)

Alte forme de substituție (cont.):

- Substituții de forma: `${#var}`
  Efectul este de a fi substituită cu lungimea cuvântului `$var`.

- Substituții de forma: `${var:start:length}`
  Efectul este de a fi substituită cu subcuvântul ce începe de la poziția *start* și de lungime *length*, din valoarea variabilei *var*.
  Exemple:

```
UNIX> index=1234          # variabila index primește valoarea "1234"
UNIX> echo $index         # se afișează pe ecran textul "1234"
UNIX> echo ${#index}      # se afișează pe ecran valoarea "4"
UNIX> echo ${index:0:1}   # se afișează pe ecran textul "1"
UNIX> echo ${index:0:2}   # se afișează pe ecran textul "12"
```

- O substituție specială este expresia:
  `$(comanda)`, sau echivalent, `` `comanda` `` .
  Efectul este de a fi substituită, în linia de comandă sau contextul în care e folosită, cu textul afișat pe ieșirea normală standard prin execuția comenzii specificate.

Variabile de *shell* (cont.)

Câteva comenzi interne, utile pentru lucrul cu variabile:

- comanda de *citire*:

  `UNIX>` `read` *`var`* [*`var2 var3 ...`*]

  Exemplu: `UNIX>` `read -p "Dați numărul n:" n`

- comanda de *declarare read-only*:

  `UNIX>` `readonly` *`var`* [*`var2 var3 ...`*]

- comanda de *exportare*:

  `UNIX>` `export` *`var`* [*`var2 var3 ...`*]

  `UNIX>` `export` *`var`*=*valoare* [*`var2`*=*valoare2* `...`]

În terminologia `UNIX`, pentru o variabilă exportată se folosește termenul de *variabilă de mediu*.

Variabile de *shell* (cont.)

Există o serie de variabile ce sunt modificate dinamic de către procesul *shell* pe parcursul execuției de comenzi, cu scopul de a le păstra semnificația pe care o au:

- **`$0`** Semnificația: numele procesului curent (*i.e.*, al *script*-ului în care este referită).
- **`$1,$2,...,$9`** Semnificația: parametrii cu care a fost apelat procesul curent (i.e. parametrii din linia de apel în cazul unui *script*).
- **`$#`** Semnificația: numărul parametrilor din linia de apel (fără argumentul `$0`).
- **`$*`** Semnificația: lista parametrilor din linia de comandă (fără argumentul `$0`).
- **`$@`** Semnificația: lista parametrilor din linia de comandă (fără argumentul `$0`).

Observație: diferența dintre $@ și `$*` apare atunci când sunt folosite între ghilimele: la substituție `"$*"` produce un singur cuvânt ce conține toți parametrii din linia de comandă, pe când `"$@"` produce câte un cuvânt pentru fiecare parametru din linia de comandă.

Variabile de *shell* (cont.)

Există o serie de variabile ce sunt modificate dinamic de către procesul *shell* pe parcursul execuției de comenzi, cu scopul de a le păstra semnificația pe care o au:

- **$$** Semnificația: `PID`-ul procesului curent (*i.e.*, *shell*-ul ce execută acel *script*).
- **$?** Semnificația: codul returnat de ultima comandă executată.
- **$!** Semnificația: `PID`-ul ultimului proces executat în *background*.
- **$-** Semnificația: opțiunile cu care a fost lansat procesul *shell* respectiv.

Observație: aceste opțiuni pot fi manevrate cu comanda internă `set`.

Există o serie de variabile de mediu *predefinite* (prin fișierele de inițializare): $`HOME` , $`USER` , $`LOGNAME` , $`SHELL` , $`MAIL` , $`PS1` , $`PS2` , $`TERM` , $`PATH` , $`CDPATH` , $`IFS` , ș.a.

Expresii aritmetice

Expresiile aritmetice se pot calcula cu comanda internă `let`, ori cu comenzile externe `expr` sau `bc`. Exemple:

```
UNIX> let v=v-1                    # variabila v este decrementată
UNIX> let v+=10                    # valoarea lui v este mărită cu 10
UNIX> let "v += 2 ** 3"            # valoarea lui v este mărită cu 8

UNIX> expr 1 + 2 \* 3              # se afișează valoarea expresiei, i.e.7
UNIX> v=`expr $v - 1`              # variabila v este decrementată
UNIX> v=$(expr $v + 10)            # valoarea lui v este mărită cu 10
```

O altă posibilitate este lucrul cu *variabile de tip întreg*:

```
UNIX> declare -i n
```

Efect: setează atributul "cu valori întregi" pentru variabila n.

Apoi se pot scrie expresii aritmetice în mod direct, fără a mai utiliza explicit comenzile de mai sus. Exemple:

```
UNIX> n=5*4                        # variabila n primește valoarea 20 (i.e., 5 înmulțit cu 4)
UNIX> n=2**3                       # variabila n primește valoarea 8 (i.e., 2 ridicat la puterea 3)
```

Expresii aritmetice (cont.)

`bc` – un limbaj de programare pentru calcule în virgulă mobilă. Exemple:

```
UNIX> echo 3 ^ 2 | bc                          # se afișează valoarea 9
UNIX> echo 3 / 2 | bc                          # se afișează valoarea întreagă 1
UNIX> echo 3 / 2 | bc -l                       # se afișează 1.500...000 (i.e., cu 20 de zecimale)
UNIX> echo "scale=4; 3/2" | bc                 # se afișează 1.5000 (i.e., cu 4 zecimale)
UNIX> echo "scale=10; 4*a(1)" | bc -l          # se afișează 3.1415926532, i.e.
                numărul π, cu 10 zecimale; a(1) este apelul funcției de bibliotecă arctan (x)
UNIX> echo "scale=5; sqrt(2)"|bc               # se afișează 1.41421, i.e. √2 cu 5 zecimale
UNIX> echo "scale=2; v = 1; v += 3/2; v+10"|bc -l   # se afișează 12.50
```

Arithmetic expansion folosind construcțiile `((...))` și `$((...))`:

```
UNIX> a=$(( 4 + 5 ))                           # variabilei a i se atribuie valoarea 9
UNIX> (( a += 10 ))                            # valoarea lui a este mărită cu 10
UNIX> (( b = a<45?7:11 ))                      # lui b i se atribuie 7, i.e. valoarea expresiei condiționale
UNIX> echo $((0xFFFF))                         # afișează 65535, i.e. valoarea acelui număr hexazecimal
UNIX> echo $((4#1203))                         # afișează 99, i.e. valoarea numărului 1203 scris în baza 4
```

Structuri de control pentru *script*-uri

1) **Bucla iterativă FOR** – *sintaxa*:

```
for variabila [ in lista_cuvinte ]
do
    lista_comenzi
done
```

sau

```
for variabila [in lista_cuvinte ]; do lista_comenzi ; done
```

Semantica: *lista_cuvinte* descrie o listă de valori pe care le ia *variabila* în mod succesiv și, pentru fiecare asemenea valoare, se execută comenzile din *lista_comenzi*.

Structuri de control pentru *script*-uri

1) **Bucla iterativă FOR** (cont.)

Observație: Forma precedentă a structurii `for` se folosește pentru mulțimi neordonate de valori, date prin enumerare. Însă dacă avem o mulțime ordonată de valori, am putea să o specificăm prin valoarea minimă, cea maximă, și pasul de incrementare.
Pentru aceasta se poate folosi comanda `seq` , e.g.

`UNIX>` `for v in `seq 2 $n` ; do` *lista_comenzi* `; done`

sau se poate folosi `for ((` , *i.e.* a doua formă sintactică a lui `for` :

```
for (( exp1; exp2; exp3 )); do lista_comenzi; done
```

unde `exp1`, `exp2` și `exp3` sunt expresii aritmetice.

Structuri de control pentru *script*-uri

2) **Bucla repetitivă WHILE** – *sintaxa*:

```
while  lista_comenzi_1
do
    lista_comenzi_2
done
```

sau

```
while lista_comenzi_1 ; do lista_comenzi_2 ; done
```

Semantica: se execută comenzile din *lista_comenzi_1* și, dacă codul de retur al ultimei comenzi din această listă este 0 (*i.e.*, terminare cu succes), atunci se execută comenzile din *lista_comenzi_2* și se reia bucla. Altfel, se termină execuția buclei `while`.

Structuri de control pentru *script*-uri

3) **Bucla repetitivă UNTIL** – *sintaxa*:

```
until lista_comenzi_1
do
    lista_comenzi_2
done
```

sau

```
until lista_comenzi_1 ; do lista_comenzi_2 ; done
```

Semantica: se execută comenzile din *lista_comenzi_1* și, dacă codul de retur al ultimei comenzi din această listă este diferit de 0 (*i.e.*, terminare cu eșec), atunci se execută comenzile din *lista_comenzi_2* și se reia bucla. Altfel, se termină execuția buclei `until`.

Structuri de control pentru *script*-uri

4) **Structura alternativă IF** – *sintaxa*:

```
if  lista_comenzi_1
then
    lista_comenzi_2
[ else
    lista_comenzi_3 ]
fi
```

sau

```
if lista_comenzi_1; then lista_comenzi_2; [else lista_comenzi_3;] fi
```

Semantica: se execută comenzile din *lista_comenzi_1* și, dacă codul de retur al ultimei comenzi din această listă este 0 (*i.e.*, terminare cu succes), atunci se execută comenzile din *lista_comenzi_2*. Iar altfel, numai dacă există și ramura `else`, se execută comenzile din *lista_comenzi_3*.

Structuri de control pentru *script*-uri

5) **Structura alternativă CASE** – *sintaxa*:

```
case expresie in
  șir_valori_1   ) lista_comenzi_1   ;;
  șir_valori_2   ) lista_comenzi_2   ;;
  ...
  șir_valori_N-1 ) lista_comenzi_N-1 ;;
  șir_valori_N   ) lista_comenzi_N
esac
```

Semantica: dacă valoarea expresiei *expresie* se găsește în lista de valori *șir_valori_1*, atunci se execută *lista_comenzi_1* și apoi execuția comenzii `case` se încheie. Altfel, dacă valoarea expresiei *expresie* se găsește în lista *șir_valori_2*, atunci se execută *lista_comenzi_2* și apoi execuția comenzii `case` se încheie. Altfel, . . . ș.a.m.d.

Structuri de control pentru *script*-uri

6) **Bucla iterativă SELECT** – *sintaxa*:

```
select variabila [ in lista_cuvinte ]
do
    lista_comenzi
done
```

sau

```
select variabila [in lista_cuvinte]; do lista_comenzi; done
```

Semantica: este o combinație între `for` și `case`: la fiecare iterație variabila primește ca valoare acel cuvânt din lista de cuvinte ce este selectat prin interacțiune cu utilizatorul. Execuția buclei `select` se încheie tot prin interacțiune cu utilizatorul.

Alte comenzi interne, utile pentru *script*-uri

- Comanda internă de testare a unei condiții:

  `test` *`condiție`* sau `[` *`condiție`* `]`

  unde expresia *condiție* poate fi:

  - operatori pe șiruri de caractere
    - `test -z` *`string`*
    - `test -n` *`string`* sau `test` *`string`*
    - `test` *`string_1`* `=` *`string_2`*
    - `test` *`string_1`* `!=` *`string_2`*
    - `test` *`string_1`* `<` *`string_2`*
    - `test` *`string_1`* `>` *`string_2`*

Alte comenzi interne, utile pentru *script*-uri

- Comanda internă de testare a unei condiții:

  `test` *`condiție`* sau `[` *`condiție`* `]`

  unde expresia *condiție* poate fi:

  - condiții relaționale între două valori numerice:
    - `test` *`val_1`* `-eq` *`val_2`*
    - `test` *`val_1`* `-ne` *`val_2`*
    - `test` *`val_1`* `-gt` *`val_2`*
    - `test` *`val_1`* `-ge` *`val_2`*
    - `test` *`val_1`* `-lt` *`val_2`*
    - `test` *`val_1`* `-le` *`val_2`*

Alte comenzi interne, utile pentru *script*-uri

- Comanda internă de testare a unei condiții:

  `test` *`condiție`* sau `[` *`condiție`* `]`

  unde expresia *condiție* poate fi:

  - condiții referitoare la fișiere: `test -`*`opt fișier`*

    unde opțiunea de testare `-`*`opt`* poate fi:

    - `-e` : testează existența acelui fișier
    - `-d` : testează dacă fișierul este un director
    - `-f` : testează dacă fișierul este un fișier obișnuit
    - `-p` : testează dacă fișierul este un fișier de tip *fifo*
    - `-b` : testează dacă fișierul este un fișier de tip dispozitiv în mod bloc
    - `-c` : testează dacă fișierul este un fișier de tip dispozitiv în mod char
    - `-s` : testează dacă fișierul are conținut nevid
    - `-r` : testează dacă fișierul poate fi citit de către utilizatorul curent
    - `-w` : testează dacă fișierul poate fi modificat de către utilizatorul curent
    - `-x` : testează dacă fișierul poate fi executat de către utilizatorul curent
    - ș.a.

Alte comenzi interne, utile pentru *script*-uri

- Comanda internă de testare a unei condiții:

  `test` *`condiție`* sau `[` *`condiție`* `]`

  unde expresia *condiție* poate fi:

  - o expresie logică (negație, conjuncție, sau disjuncție de condiții):
    - `test !` *`condiție_1`*
    - `test` *`condiție_1`* `-a` *`condiție_2`*
    - `test` *`condiție_1`* `-o` *`condiție_2`*

    unde *condiție_1* și *condiție_2* sunt condiții de oricare dintre formele specificate anterior.

Pentru restul opțiunilor a se vedea `help test` și `man 1 test`.

Alte comenzi interne, utile pentru *script*-uri

Comenzi interne utile în *script*-uri (sau la linia de comandă):

- comanda `break`, cu sintaxa:

  `break [`*n*`]`

  unde *n* este 1 în caz că lipsește.

  Efect: se iese afară din *n* bucle `do-done` imbricate, execuția continuând cu următoarea instrucțiune de după `done`.

- comanda `continue`, cu sintaxa:

  `continue [`*n*`]`

  unde *n* este 1 în caz că lipsește.

  Efect: pentru $n = 1$ se reîncepe bucla curentă `do-done` (de la pasul de reinițializare), respectiv pentru $n > 1$ efectul este ca și cum s-ar executa de *n* ori comanda `continue 1`.

Alte comenzi interne, utile pentru *script*-uri

Comenzi interne utile în *script*-uri (sau la linia de comandă):

- comanda `exit`, cu sintaxa:

  `exit [`*cod*`]`

  unde *cod* este valoarea variabilei $?, în caz că lipsește.

  Efect: se încheie execuția *script*-ului în care apare și se întoarce drept cod de retur valoarea specificată.

- comanda `exec`, cu sintaxa:

  `exec` *comandă*

  Efect: se execută comanda specificată fără a se crea o nouă instanță de *shell* (astfel *shell*-ul ce execută această comandă se va “reacoperi” cu procesul asociat comenzii, deci nu este *reentrant*).

Alte comenzi interne, utile pentru *script*-uri

Comenzi interne utile în *script*-uri (sau la linia de comandă):

- comanda `wait`, cu sintaxa:

  `wait` *pid*

  Efect: se suspendă execuția *script*-ului curent, așteptându-se terminarea procesului având `PID`-ul specificat.

- comanda `eval`, cu sintaxa:

  `eval` *parametri*

  Efect: se evaluează parametrii specificați și se execută rezultatul.
  Exemplu:
  `UNIX>` `eval newvar=\$$varname`
  Obținem practic o referință indirectă (precum pointerii din limbajul C).

Alte comenzi interne, utile pentru *script*-uri

Comenzi interne utile în *script*-uri (sau la linia de comandă):

- comanda `shift`, cu sintaxa:

  `shift[`*n*`]`

  unde *n* este 1 în caz că lipsește.

  Efect: variabilele poziționale $n+1, $n+2, ... se redenumesc în $1, $2, ...
  Utilă atunci când dorim să apelăm un script cu mai mult de 9 argumente.

- comanda `set`

  Se utilizează pentru a seta valorile opțiunilor de execuție a procesului *shell* curent, sau a variabilelor poziționale.

  Exemplu: `set -o xtrace` sau `set -x`
  Efect: afișează fiecare linie de comandă interpretată, înainte de a o executa.
  Utilă pentru depanare, ca să vedem exact ce se va executa de către *shell* în urma citirii și interpretării liniei de comandă.

Alte comenzi interne, utile pentru *script*-uri

Comenzi interne utile în *script*-uri (sau la linia de comandă):

- comanda `trap`, cu sintaxa:

  `trap` *`comandă eveniment`*

  Efect: când se va produce evenimentul specificat (*i.e.*, când se va primi semnalul respectiv), se va executa comanda specificată.
  Evenimente (semnale) posibile:

  - semnalul 1 = *hang-up signal*
  - semnalul 2 = *interrupt signal* (generat prin apăsarea tastelor `CTRL + C`)
  - semnalul 3 = *quit signal* (generat prin apăsarea tastelor `CTRL + \`)
  - semnalul 9 = *kill signal* (semnal ce “omoară” procesul)
  - semnalul 15 = semnal de terminare normală a unui proces
  - ș.a.

  Exemplu:

```
UNIX> trap 'rm /tmp/ps$$ ; exit'  2
```

Funcții *shell*

O funcție *shell* este un nume pentru o secvență de comenzi UNIX, analog cu procedurile *shell*, cu deosebirea că o funcție nu se scrie într-un fișier text separat, ca în cazul acestora, ci se scrie (*i.e.*, se declară) în interiorul unei proceduri *shell*, folosind sintaxa:

```
function nume_funcție () { lista_comenzi ; }
```

Semantica: comanda internă `function` declară *nume_funcție* ca fiind o variabilă de tip funcție, adică un "alias" pentru secvența de comenzi *lista_comenzi*.

Observație: fie `function`, fie `()` pot fi omise, dar nu simultan amândouă. Iar între parantezele `()` nu se scrie niciodată nimic, chiar dacă dorim să apelăm funcția cu unul sau mai multe argumente!

Funcții *shell* (cont.)

În corpul funcției (*i.e.*, în *lista_comenzi*) putem folosi variabilele poziționale $1, $2, . . . , $9 pentru a ne referi la parametrii de apel ai funcției, iar prin $* și $@ ne referim la lista tuturor parametrilor de apel.

Apelul unei funcții (*i.e.*, lansarea sa în execuție) se face similar ca pentru orice comandă, prin numele său, plus parametri.

Un exemplu de funcție, definită și apelată direct la linia de comandă:

```
UNIX> function listing () { \
echo "Listingul directorului: $1" ; \
if test -d $1 ; then ls -lA $1 ; else echo "Eroare" ; fi }
UNIX> listing ~vidrascu/so/
```

Observație: conceptul de funcție *shell* NU este deloc similar cu cel de funcție din limbaje de programare precum C ori C++.

Funcții *shell* (cont.)

Alt exemplu de funcție, scrisă în interiorul unui *script*:

```
#!/bin/bash
function cntparm ()
{
  echo "$# params: $*"
}
cntparm "$*"
cntparm "$@"
```

Dacă apelăm acest script cu următoarea linie de comandă:

```
UNIX> ./script a b c
1 params:  a b c
3 params:  a b c
```

mesajele afișate pe ecran ne demonstrează diferența dintre modul de evaluare al variabilelor `$*` și $@ atunci când sunt cuprinse între ghilimele.

Bibliografie obligatorie

Cap.2, §2.4 din manualul, în format PDF, accesibil din pagina disciplinei “Sisteme de operare”:

- `http://profs.info.uaic.ro/~vidrascu/SO/books/ManualID-SO.pdf`