6. FACILITATI ALE SHELL-URILOR SI SCRIPTING

6.1. Interpretorul de comenzi bash si facilitati ale sale

6.1.1. Utilitate si moduri de lucru

Interpretorul de comenzi Linux/Unix (shell-ul) reprezinta interfata intre utilizator si kernel-ul sistemului de operare. El are ca rol procesarea liniilor de comanda primite; pentru fiecare linie, shell-ul:

- efectueaza eventuale transformari ale liniei de comanda (vezi paragrafele despre variabile, alias-uri etc.)
- identifica in cadrul liniei primite comanda, optiunile si argumentele
- executa comanda pasandu-i optiunile si argumentele extrase anterior

In plus fata de simpla executie de comenzi ce constituie scopul sau principal, shell-ul ofera o multitudine de facilitati - unele care tin de lucrul in linia de comanda, altele gandite sa transforme shell-ul intr-un adevarat mediu de programare. Ele se indreapta in doua directii:

- usurarea lucrului in linia de comanda. Presupune facilitati precum:
  - **autocomplete** – completarea automata a numelor partiale de comenzi sau argumente la apasarea tastei TAB
  - **redirectionari** – posibilitatea ca output-ul sau erorile unei comenzi, care in mod normal sunt afisate pe ecran, sa fie directionate catre un fisier sau catre o alta comanda (ca input al acesteia). Este posibila de asemenea redirectionarea input-ului unei comenzi, astfel incat aceasta sa citeasca dintr-un fisier in loc sa-si preia datele de la tastatura
  - **history** – shell-ul mentine o lista a comenzilor executate anterior, utilizatorul putand-o consulta si executa comenzi componente
  - **alias-uri** – posibilitatea definirii de noi comenzi pe baza celor deja existente
  - **job control** – o facilitate ce permite utilizatorului ca, din cadrul unui singur shell, sa ruleze mai multe comenzi in paralel
  - **variabile** – zone de memorie ce contin informatie si referite printr-un nume. Variabilele pot fi folosite fie pentru a stabili optiuni din oficiu pentru comenzile rulate, fie pentru a memora date prelucrate in cazul programelor de shell (vezi sectiunile ulterioare din cadrul acestui material)
- crearea de programe formate din comenzi de shell (“script-uri”). Astfel de programe sunt alcatuite folosind comenzi obisnuite, compuse prin intermediul unor facilitati ale shell-ului ce tin de programare:
  - **structuri de control al executiei** (instructiuni decizionale, instructiuni repetitive etc)
  - **operatori** - simboluri sau cuvinte cheie utilizate pentru a combina si procesa informatie
  - **functii** - secvente de instructiuni ce primesc un nume si pot fi astfel apelate de oricate ori este nevoie fara a le re-scrie de fiecare data

Nota: *un program scris in limbaj de shell poarta denumirea de* ***shell script*** *(vezi capitolul dedicat scripting-ului).*

Shell-ul are doua moduri posibile de lucru:

- modul interactiv, in care shell-ul este conectat la un terminal, iar comenzile sunt introduse manual, de la tastatura, de catre utilizator
- modul non-interactiv ("batch"), in care shell-ul citeste dintr-un fisier comenzile si le executa una cate una (cazul rularii de script-uri)

In functie de modul de lucru, la pornire shell-ul cauta si executa diferite fisiere de configurare, dupa cum se va explica intr-un paragraf ulterior.

Vor fi prezentate in continuare cateva dintre facilitatile descrise mai sus (o parte dintre ele au fost deja acoperite in capitole anterioare ale cursului).

6.1.2. History

Aceasta facilitate a shell-ului presupune memorarea automata a comenzilor utilizatorului pe masura ce acesta le introduce. Lista de comenzi este pastrata in memorie; ea se salveaza intr-un fisier la parasirea shell-ului si este incarcata inapoi in memorie la pornirea unui shell nou de catre acelasi utilizator.

Fisierul folosit pentru salvarea/incarcarea listei de comenzi este cel desemnat de valoarea variabilei HISTFILE, a carei valoare din oficiu este **~/.bash_history**. Dimensiunea acestui fisier poate fi limitata memorand maximul dorit (exprimat in octeti) in variabila HISTFILESIZE, iar numarul maxim de linii din fisier poate fi limitat la randul sau setand valoarea sa maxima in variabila HISTSIZE.

Nota: *daca variabila HISTFILE nu este setata sau fisierul referit nu are permisiuni de scriere, lista de comenzi nu este salvata la parasirea shell-ului.*

Memorarea istoricului comenzilor ofera utilizatorului facilitati precum:

- accesarea facila a ultimelor comenzi din lista, folosind sagetile sus-jos
- afisarea listei complete de comenzi memorate, folosind comanda **history**
- cautarea unei comenzi anterioare pe baza unui fragment al liniei de comanda, apasand **CTRL-r** si apoi scriind o portiune a liniei cautate

6.1.3. Alias-uri

6.1.3.1. Conceptul de alias

Un alias reprezinta o noua comanda creata prin atribuirea unui nou nume altei comenzi sau unei succesiuni de comenzi existente. Alias-urile sunt create/vizualizate cu ajutorul comenzii **alias** si pot fi sterse cu comanda **unalias**. Alias-urile sunt create in memorie si dispar odata cu inchiderea shell-ului in care au fost definite; daca se doreste ca ele sa fie disponibile intotdeauna, este necesara crearea lor intr-unul dintre scripturile de initializare ale shell-ului.

6.1.3.2. Crearea unui alias

Sintaxa generala de creare a unui alias este:

```
alias comanda_noua='sir de caractere'
```

Atentie! *Nu lasati spatii in stanga sau in dreapta semnului = (egal)!*

Ori de cate ori utilizatorul introduce o linie de comanda, shell-ul verifica daca numele comenzii se regaseste in lista de alias-uri. In caz afirmativ, alias-ul este inlocuit in linia de comanda cu valoarea sa si abia apoi se incearca impartirea in campuri si executarea liniei de comanda.

Exemplu: crearea unui alias pentru pornirea din linia de comanda a browserului Firefox:

```
alias f='mozilla-firefox'
```

De acum inainte, ori de cate ori utilizatorul va folosi comanda *f* in cadrul unei linii de comanda, ea va fi substituita cu sirul de caractere *mozilla-firefox*. Aceasta inseamna ca *f* accepta si optiuni/argumente:

```
f –width 800 –height 600 yahoo.com
```

Comanda *f* va fi gasita in lista de alias-uri si inlocuita cu valoarea sa, linia de comanda finala devenind *mozilla-firefox –width 800 –height 600 yahoo.com*.

Alte exemple:

- definirea unui alias scurt pentru o comanda cu nume lung folosita des:

```
alias ncc='named-checkconf'
```

- definirea unui nume alternativ pentru o comanda, deoarece suntem obisnuiti ca acea comanda sa aiba alt nume in alt sistem de operare

```
alias dir='ls'
```

- crearea unui alias pentru o comanda folosita frecvent

```
alias ll='ls –l'
```

- definirea unui alias ce corespunde unei succesiuni de comenzi pe care le utilizam frecvent

```
alias testnet='ping –c 3 yahoo.com|tail -2'
```

- crearea unui alias cu acelasi nume ca o comanda deja existenta, pentru a executa intotdeauna acea comanda cu un anumit set de optiuni implicite. Ex: pentru ca stergerea de fisiere sa fie intotdeauna interactiva, putem scrie:

```
alias rm='rm –i'
```

Un alias care are nume identic cu o comanda deja existenta are prioritate fata de aceasta (in caz contrar, nu ar mai avea rost crearea unui alias, caci ar fi executata intotdeauna comanda originala). Daca, atunci cand exista o comanda si un alias cu acelasi nume, dorim sa executam comanda originala, putem folosi *\nume_comanda*:

```
\rm –rf /test
```

6.1.3.3. Vizualizarea si stergerea alias-urilor

Vizualizarea alias-urilor deja definite se realizeaza cu comanda **alias**, insa fara argumente:

```
bash$ alias
alias md='mkdir'
alias rm='rm -i'
```

Stergerea unui alias se realizeaza cu comanda **unalias**:

```
bash$ unalias md
bash$ alias
alias rm='rm -i'
# a disparut alias-ul md prezent anterior
```

6.1.4. Variabile

6.1.4.1. Generalitati. Utilitate

O variabila reprezinta o zona de memorie in care se stocheaza date, accesibila prin intermediul unei etichete text ce reprezinta numele variabilei. Datele stocate in variabila sunt denumite “valoarea variabilei”. Termenul de “variabila” provine din faptul ca valoarea ei poate fi citita si modificata de oricate ori avem nevoie de-a lungul existentei sale.

Variabilele au cel putin doua utilizari in linia de comanda Linux/Unix:

- multe comenzi Linux verifica existenta anumitor variabile si tin cont de valoarea acestora. In acest fel, variabilele actioneaza ca niste optiuni din oficiu pentru diferite comenzi sau utilitare. Spre exemplu, interpretorul bash va verifica existenta variabilei PS1 si va afisa in stanga cursorului din fereastra de terminal un sir de caractere ce corespunde valorii acestei variabile; comanda ls verifica existenta unei variabile LS_COLORS si, daca ea exista, va colora diferitele tipuri de fisiere conform setarilor specificate in aceasta variabila
- in programele scrise in limbaj de shell (asa-numitele “script-uri de shell”) variabilele pot fi definite si folosite ca in orice alt limbaj de programare, ele putand lua valori, participa in expresii sau decizii etc.

6.1.4.2. Definirea unei variabile si modificarea valorii acesteia

Spre deosebire de alte limbaje de programare, in care era necesara declararea variabilelor inaintea folosirii, in interpretorul de comenzi Linux/Unix definirea unei variabile se realizeaza atribuind acesteia o (prima) valoare. Atribuirea se efectueaza folosind numele variabilei, operatorul de atribuire = si valoarea dorita:

```
USER=student
```

Atentie! Nu trebuie introduse spatii in stanga sau in dreapta operatorului =! *Spatiile sunt interpretate ca separatori intre campurile liniei de comanda, ceea ce ar cauza o interpretare diferita de intentiile noastre a liniei de comanda ce creeaza variabila.*

Numele variabilei poate fi format din litere mici, mari sau o combinatie de acestea. Se practica numele de variabile formate exclusiv din majuscule deoarece astfel devin mai vizibile in scripturi, insa acest lucru nu este obligatoriu. Atentie insa - numele de variabile sunt case-sensitive! Variabilele *VAR* si *Var* sunt diferite.

Atunci cand valoarea unei variabile este formata din unul sau mai multe cuvinte, trebuie folosita una dintre modalitatile de escaping (vezi sectiunea 6.2.7) pentru a anula semnificatia speciala a caracterului spatiu:

```
DIR=”Program Files”
DIR='Program Files'
DIR=Program\ Files
```

Modificarea ulterioara a valorii unei variabile se realizeaza cu aceeasi sintaxa; spre exemplu, comanda de mai jos va inlocui valoarea variabilei DIR cu *Poze concediu*:

```
DIR=Poze\ Concediu
```

6.1.4.3. Citirea valorii unei variabile

Citirea valorii memorate anterior intr-o variabila se realizeaza prefixand numele variabilei cu metacaracterul $. Atunci cand o astfel de referire la o variabila apare in cadrul liniei de comanda, interpretorul va inlocui numele variabilei cu valoarea acesteia si abia apoi va interpreta si executa linia de comanda:

```
bash$ DIR=/tmp
bash$ ls -ld $DIR
drwxrwxrwt 15 root root 784 2015-05-22 14:44 /tmp
```

Atunci cand incercam sa citim valoarea unei variabile ce nu a fost inca definita, valoarea rezultanta va fi sirul vid, fara a se genera vreo eroare. Spre exemplu, daca incercam sa afisam caracteristicile unui director al carui nume este dat de valoarea variabilei NEWDIR, iar aceasta variabila nu a fost inca definita, comanda ls va afisa detaliile directorului curent deoarece este practic apelata fara argumente:

```
bash$ ls -ld $NEWDIR
drwxr-xr-x 59 student users 4096 oct 15 10:11 .
```

Atentie! Numele de variabile Linux/Unix sunt case sensitive! *Aceasta inseamna ca, daca definim variabila DIR si apoi citim $dir, cea din urma este o variabila diferita de prima si, in cazul in care nu a fost inca definita, va fi substituita cu sirul vid.*

6.1.4.4. Afisarea variabilelor definite si a valorilor acestora

Avem la dispozitie doua instructiuni pentru vizualizarea valorilor variabilelor:

- comanda **set** – data fara argumente, afiseaza lista tuturor variabilelor definite in shell-ul curent, impreuna cu valorile acestora
- comanda **echo** – urmata de numele unei variabile (prefixat cu $) ea afiseaza pe ecran valoarea variabilei cerute

```
bash$ set
BASH=/bin/bash
BASH_ARGC=()
BASH_ARGV=()
BASH_COMPLETION=/etc/bash_completion
[...]
bash$ echo $BASH_COMPLETION
/etc/bash_completion
```

6.1.4.5. Stergerea unei variabile

Aceasta operatie se realizeaza cu ajutorul comenzii **unset** – fie urmata de optiunea -v, fie fara optiuni:

```
student@Desktop $ OPTION=2
student@Desktop $ echo "Ati ales optiunea $OPTION"
Ati ales optiunea 2
student@Desktop $ unset OPTION
student@Desktop $ echo "Ati ales optiunea $OPTION"
Ati ales optiunea
```

Nota: *unset este folosit si pentru anularea definitiilor de functii. Atunci cand exista variabile si functii cu acelasi nume, optiunea -v este utilizata pentru a ne referi explicit la vairabile, iar optiunea -f pentru functii.*

6.1.4.6. Exemple de variabile utile

Amintim doua dintre variabilele de interes pentru utilizatorul de shell:

- **PS1** – valoarea sa determina prompt string-ul (sirul de caractere afisat in stanga cursorului in ferestrele de terminal sau console). In cadrul valorii lui PS1 pot fi folosite secventele: \u pentru username, \h pentru hostname, \w pentru directorul curent etc. (lista completa se gaseste in manpage-ul lui bash)

```
[student@Desktop:/tmp]$ echo $PS1
[\u@\h:\w]\$        → in traducere: [ username@nume_statie: director_curent] $
[student@Desktop:/tmp]$ PS1='\u -->\w'
[student --> /tmp]
```

- **PATH** – valoarea sa consta dintr-o succesiune de cai catre directoare ce contin executabile, separate prin ; . Cand shell-ul are de executat o comanda care nu este built-in, alias sau functie, el trebuie sa determine locatia executabilului corespunzator in sistemul de fisiere; shell-ul nu va cauta acel executabil decat in caile prezente in PATH. De aceea, daca dorim sa executam comenzi folosind numele lor scurt, este necesar ca directorul in care se gaseste executabilul sa fie prezent in PATH; altminteri va fi necesara specificarea caii complete catre executabil

```
# sa presupunem ca avem executabilul skype in directorul /opt/skype/bin
[student@Desktop:/tmp]$ skype
bash: skype: command not found...
# cu cale absoluta, comanda ruleaza corect:
[student@Desktop:/tmp]$ /opt/skype/bin/skype
# reconfigurare PATH
[student@Desktop:/tmp]$ PATH=$PATH:/opt/skype/bin
[student@Desktop:/tmp]$ skype
# comanda ruleaza acum si cu numele scurt
```

6.1.4.7. Domeniul de vizibilitate al unei variabile

O variabila definita intr-un shell este vizibila numai in cadrul acelui shell, nu si in afara acestuia. Valorile variabilelor stau in memorie; atunci cand shell-ul in care au fost definite se inchide, ele dispar. Daca se doreste ca o variabila sa "persiste" dincolo de inchiderea shell-ului, iluzia persistentei poate fi data prin includerea unei instructiuni in fisierele de initializare ale shell-ului care sa re-creeze acea variabila la pornirea interpretorului de comenzi (vezi sectiunea 6.1.5).

Implicit, variabilele definite intr-un shell nu vor fi mostenite de catre procesele pornite din acel shell. Exista insa un mecanism prin care variabilele dorite pot fi marcate ca mostenite: comanda **export**. Ea este urmata de unul sau mai multe nume de variabile, care vor fi astfel pasate automat tuturor proceselor ulterioare pornite din shell-ul curent. Variabilele exportate sunt denumite *variabile de mediu* ("environment variables") deoarece ansamblul lor formeaza mediul (contextul informational de pornire) in care ruleaza un proces.

Observatie: *variabilele de mediu reprezinta de fapt o modalitate de comunicare inter-proces, deoarece procesul parinte depoziteaza informatie intr-o astfel de variabila iar subprocesele o preiau*

S-a explicat anterior ca multe comenzi Linux tin cont de valorile unor variabile in masura in care acestea exista. Avand in vedere ca acele comenzi reprezinta de obicei noi procese pornite din cadrul unui shell, este necesara exportarea in prealabil a variabilelor de interes pentru ca ele sa fie disponibile comenzii care ruleaza ca subproces al shell-ului. Spre exemplu, comanda man verifica la lansare existenta unei variabile PAGER; daca aceasta exista, valoarea ei va fi folsita ca utilitar de paginare a informatiei afisate. In Linux, pager-ul din oficiu este less; daca dorim sa facem experimentul de a schimba utilitarul de paginare cu wc -l, nu este suficienta definirea variabilei, ci este necesara si exportarea sa:

```
student@Desktop $ man ls
LS(1)                          User Commands                          LS(1)
NAME
       ls - list directory contents
[...restul output-ului a fost omis...]
student@Desktop $ PAGER='wc -l'
student@Desktop $ man ls
LS(1)                          User Commands                          LS(1)
NAME
       ls - list directory contents
[...restul output-ului a fost omis...]
student@Desktop $ export PAGER
student@Desktop $ man ls
231
```

Shell-ul bash permite crearea din start a unei variabile exportate, cu sintaxa **export variabila=valoare:**

```
export PAGER='wc -l'
```

Afisarea variabilelor exportate se realizeaza cu comanda **env**:

```
student@Desktop $ env|grep PAGER
PAGER=wc -l
```

6.1.5. Fisiere de initializare ale shell-ului

6.1.5.1. Descriere si utilitate

Fisierele de initializare ale shell-ului sunt fisiere cu nume si locatii predefinite, pe care shell-ul le cauta si le executa automat la pornirea sa intr-o anumita ordine. Aceste fisiere contin comenzi de shell si sunt utilizate in general pentru definirea de variabile, alias-uri, functii etc ce vor fi folosite mai apoi de catre utilizator sau de catre programele rulate de acesta.

Succesiunea fisierelor de initializare executate la rularea shell-ului depinde de tipul de shell. Un shell poate fi:

- **interactiv sau non-interactiv** – un shell interactiv este unul conectat la un terminal si care interpreteaza comenzi provenite de la tastatura utilizatorului. Un shell non-interactiv citeste comenzile de executat dintr-un fisier si le executa una cate una (cum este cazul la executarea unui script de shell)
- **login shell sau non-login shell**. Un *login shell* este cel pornit intr-o consola virtuala, dupa ce utilizatorul s-a autentificat cu succes (sau - mai rar - unul pornit cu optiunea --login sau -l). Un *non-login shell* este,

spre exemplu, cel pornit ca parte a unui emulator de terminal in mediul grafic (ex: konsole, gnome-terminal, xterm), unde utilizatorului nu i se mai solicita informatiile de autentificare

Nota: *o verificare indirecta daca un shell este sau nu interactiv se poate face analizand variabila PS1, care este definita numai in cazul shell-urilor interactive.*

Exista doua categorii de fisiere de initializare folosite de shell-uri:

- **fisiere globale** ("system-wide") – sunt memorate in /etc si contin setari care se aplica tuturor utilizatorilor sistemului. Exemplele tipice sunt ***/etc/profile***, ***/etc/bashrc*** si fisierele din directorul ***/etc/profile.d/***
- **fisiere per-user** – sunt memorate in directorul personal al fiecarui utilizator, continand setari ce se aplica numai utilizatorului in cauza si care le pot suprascrie pe cele efectuate in fisierele de configurare system-wide. Exemple tipice: ***~/.profile, ~/.bash_login, ~/.bashrc, ~/.bash_logout, ~/.bash_profile***

6.1.5.2. Fisiere de initializare executate in functie de tipul de shell

In functie de tipul shell-ului, distingem 3 cazuri:

- shell de login interactiv (ex: consola virtuala). In acest caz, shell-ul cauta si executa urmatoarele fisiere:
  - la pornire:
    - **/etc/profile**. In unele distributii, pentru o mai buna structurare, este creat un director suplimentar /etc/profile.d/ in care stau fisiere de initializare cu setari per-aplicatie, fisierele din acest director fiind executate explicit din /etc/profile in cadrul unei instructiuni repetitive
    - primul dintre fisierele **~/.bash_profile, ~/.bash_login, ~/.profile** care exista si are permisiune de read (cautate in aceasta ordine)
  - la inchiderea shell-ului
    - **~/.bash_logout** – daca fisierul exista si are permisiune de read
- shell non-login interactiv (ex: fereastra de terminal). Fisiere cautate si executate:
  - **~/.bashrc** . De remarcat ca acest fisier este de obicei executat explicit si din cadrul lui **~/.bash_profile**, astfel incat si shell-urile de login sa beneficieze de setarile incluse aici
- shell non-interactiv (ex: rulare script):
  - daca exista variabila de mediu **BASH_ENV**, valoarea sa este considerata ca fiind calea catre fisierul de initializare ce trebuie executat

Interpretorul bash executa doua categorii de scripturi de initializare – unele proprii, si altele in virtutea compatibilitatii cu shell-uri mai vechi (sh, ksh etc). Fisierele */etc/profil*e si *~/.profile* fac parte din ultima categorie; *~/.bash_** sunt scripturi proprii bash. Avand in vedere ca bash are multe facilitati suplimentare fata de shell-urile mai vechi, distributiile evita in general sa plaseze comenzi specifice bash in fisiere precum */etc/profile* sau *~/.profile*, deoarece acestea sunt executate automat la pornire si de catre alte shell-uri disponibile in sistem. De aceea, atunci cand se doresc setari system-wide dar specifice bash, ele nu se includ in */etc/profile*, ci intr-un fisier */etc/bashrc* sau */etc/bash/bashrc* care este apelat din *~/.bash_profile*, din *~/.bashrc* sau chiar din */etc/profile*. Exemplu: in Fedora exista *~/.bash_profile* care executa *~/.bashrc*, iar *~/.bashrc* la randul sau executa */etc/bashrc*.

6.2. Scripting de shell

6.2.1. Generalitati

Un script (intalnit in Windows sub denumirea de *batch file*) este un fisier text ce contine comenzi executabile fie de catre un shell, fie de catre un alt interpretor, specificat de obicei pe prima linie din fisier (php,perl etc). Odata scriptul creat si salvat, functionalitatea oferita de el poate fi refolosita prin apelarea lui repetata.

Etapele pe care le presupun elaborarea si folosirea unui script sunt:

- scrierea si salvarea scriptului
- adaugarea permisiunii de executie
- rularea scriptului

6.2.2. Scrierea si salvarea scriptului

Un script este un fisier text neformatat, asadar orice editor de text este potrivit pentru aceasta operatie. La salvare nu exista impuneri asupra numelui fisierului – deseori scripturile nu au nicio extensie. Se practica insa folosirea extensiei *.sh* pentru a indica explicit continutul fisierului.

6.2.3. Adaugarea permisiunii de executie

Precum se stie, un fisier Linux/Unix poate fi executat numai in masura in care contul cu care logat utilizatorul ce-l executa are permisiuni de read si execute pe fisierul in cauza.

bash$ chmod u+rx *numescript*

6.2.4. Executarea scriptului

Exista diferite modalitati de executie a unui script, fiecare cu caracteristicile sale:

a) **./numescript** (daca scriptul se gaseste in directorul curent; sau orice alta cale relativa ce-l vizeaza)
/cale/absoluta/catre/numescript

Executarea scriptului folosind o cale absoluta sau relativa catre fisierul respectiv. Acest mod de executie porneste un nou shell in care se executa comenzile cuprinse in script, una cate una, in ordinea in care apar in fisier. Este necesara specificarea caii atunci cand fisierul de executat se afla intr-un director care nu se regaseste in PATH; in caz contrar, este suficient simplul nume al scriptului.

Din cauza faptului ca instructiunile din script se executa intr-un shell nou, niciuna dintre variabilele definite in script nu va fi disponibila in shell-ul din care se ruleaza scriptul; toate variabilele dispar la incheierea executiei shell-ului in care au fost definite:

```
student@Desktop $ cat s1.sh
COLOR=blue
student@Desktop $ ./s1.sh
student@Desktop $ echo "Culoarea este $COLOR"
Culoarea este
# $COLOR nu e definit si deci se inlocuieste cu sirul vid
```

b) bash numescript
sh numescript

Executa fisierul curent intr-un shell nou (cu aceleasi implicatii ca la punctul anterior), diferenta fiind insa ca fisierul executat nu mai are nevoie de permisiuni de executie.

c) source ./numescript
. ./numescript
source /cale/absoluta/catre/script
. /cale/absoluta/catre/script

Executa toate comenzile din script in shell-ul curent. In acest fel, orice variabila definita in script va deveni disponibila in shell-ul din care a fost rulat scriptul:

```
student@Desktop $ cat s1.sh
COLOR=red
student@Desktop $ source ./s1.sh
student@Desktop $ echo "Your color is now $COLOR"
Your color is now red
```

d) exec ./numescript
exec /cale/absoluta/catre/script

Executia scriptului inlocuieste shell-ul curent, fara a crea un nou proces; cand scriptul se incheie, shell-ul curent se inchide.

6.2.5. Elemente de baza ale unui script

6.2.5.1. Comentarii

In scripturile bash, simbolul # indica inceputul unui comentariu. Tot restul liniei este ignorat de catre interpretor. Comentariul se poate afla la inceput de linie sau ulterior:

```
# comentariu de o linie
alias # afiseaza alias-urile curente
```

6.2.5.2. Specificarea interpretorului

Exista si cazuri in care comentariile nu sunt ignorate. Unul dintre ele este acela in care, pe prima linie a unui script, se gaseste combinatia #! (numita si "shebang"), urmata de calea catre un executabil. Rolul unei prime astfel de linii este specificarea interpretorului ce va procesa instructiunile cuprinse in fisier. In Linux se folosesc multiple limbaje de scripting, de aceea prima linie poate avea valori precum:

- **#!/bin/bash** – pentru scripturile de shell Linux
- **#!/usr/bin/perl** – pentru programe scrise in Perl
- **#!/usr/bin/php** – pentru programe PHP gandite a fi executate din linia de comanda etc.

Specificarea in acest fel a interpretorului are si un alt efect: editoarele de text din Linux vor folosi aceasta informatie pentru a face formatarea si colorarea codului in functie de limbajul in care este scris scriptul. Spre exemplu, daca scriem un script de shell pentru care initial nu specificam interpretorul, la deschiderea sa cu un

editor intregul cod va fi afisat cu aceeasi culoare; daca adaugam insa linia shebang si apoi re-deschidem fisierul, de data aceasta diversele elemente de sintaxa vor fi colorate in functie de rolul lor.

6.2.5.3. Comenzi de uz general in scripturi de shell

Vom mentiona aici doua dintre comenzile necesare in scripting-ul de shell:

- **echo** – este folosit pentru a afisa pe ecran un sir de caractere. Dupa sirul cerut, echo introduce din oficiu NEWLINE (trecere pe linie noua). Sirul in cauza poate fi scris ca atare sau poate reprezenta rezultatul unei expresii in care pot interveni si valori de variabile:

```
student@Desktop $ COLOR=red
student@Desktop $ echo “A color has been defined” # afisare sir de caractere specificat explicit
A color has been defined
student@Desktop $echo “The color is: $COLOR”  # afisare sir ce contine si valori de variabile
The color is: red
student@Desktop $echo “3 x 2 =” $((3*2)) # afisare sir ce contine si expresii aritmetice
3 x 2 = 6
```

  Comanda echo accepta si cateva optiuni utile:
  - **-n** – dupa afisarea sirului cerut nu se mai trece pe linia urmatoare, lasand cursorul pe linia curenta. Utila atunci cand se doreste afisarea unei singure linii folosind mai multe comenzi echo
  - **-e** – activeaza interpretarea de catre echo a secventelor escape (\n – newline, \t – tab etc)

- **exit** – determina incheierea fortata a executiei scriptului (inainte de parcurgerea tuturor instructiunilor din fisier). Primeste un argument ce reprezinta valoarea de iesire – valoarea 0 indicand succesul si valorile nenule esecul. Apelat fara argumente va folosi implicit codul de iesire 0.

6.2.5.4. Elemente de sintaxa generala in scripturile de shell

Spre deosebire de multe alte limbaje, in scripting-ul de shell linia de comanda nu este obligatoriu sa se termine cu ; (punct si virgula). Interpretorul considera trecerea pe linia urmatoare ca fiind indicatia incheierii comenzii de executat. Daca insa se doreste includerea mai multor comenzi pe aceeasi linie, este obligatorie separarea acestora prin ; ca in exemplul de mai jos:

```
student@Desktop $ echo Unu
Unu
student@Desktop $ echo Doi
Doi
student@Desktop $ echo -n Unu; echo Doi
UnuDoi
```

Exista si cazul in care se doreste distribuirea unei singure linii de comanda pe mai multe randuri (spre exemplu, cand comanda este foarte lunga). In acest scop, fiecare trecere pe linia urmatoare (in cadrul aceleiasi comenzi) va fi precedata de caracterul \ (backslash):

```
cat /etc/services | egrep ".*sh.*" \
      | grep 22 | \
      wc -l
```

6.2.6. Variabile – notiuni avansate

6.2.6.1. Modul de procesare a liniei de comanda

Inainte de a executa linia de comanda primita de la utilizator, interpretorul efectueaza cateva operatii asupra sa:

- inlocuieste variabilele cu valorile corespunzatoare acolo unde este cazul
- inlocuieste alias-urile cu valorile corespunzatoare
- inlocuieste constructiile `....` cu output-ul executiei comenzii continute (vezi 6.2.8)
- imparte linia in campuri, folosind ca separator de campuri caracterul spatiu si ca separator de comenzi caracterul ;
- identifica comanda care trebuie executata

Se observa ca executarea unei comenzi are loc abia dupa incheierea acestui set de operatii. In functie de valorile variabilelor, alias-urilor etc este posibil ca o linie de comanda aparent corecta sa rezulte in final intr-o eroare de executie, cauzata de negasirea comenzii sau de erori de sintaxa aparute dupa efectuarea substitutiilor. Asadar, spre deosebire de alte limbaje, este posibil sa obtinem sau nu eroare de sintaxa in functie de valorile variabilelor prezente in linia de comanda.

Iata un scurt exemplu: fie o instructiune conditionala *if* care compara valoarea curenta a unei variabile cu sirul de caractere “exit”:

```
if [ $OPTION == exit ]; then [...diverse alte instructiuni...]
```

Daca variabila OPTION a fost initializata anterior cu valoarea “ignore”, shell-ul va efectua inlocuirea numelui de variabila cu valoarea acesteia in linia de comanda, linia rezultata fiind:

```
if [ ignore == exit ]; then [...diverse alte instructiuni...]
```

Sintactic, linia de mai sus este corecta; ea va fi acum executata de catre shell, iar rezultatul comparatiei celor doua siruri de caractere va fi desigur valoarea de adevar “fals”. Daca insa variabila OPTION nu a fost definita, numele său va fi substituit cu sirul vid, iar linia de comanda devine:

```
if [ == exit ]; then [...diverse alte instructiuni...]
```

Operatorului de comparare == ii lipseste in acest caz primul operand, ceea ce determina afisarea unei erori de sintaxa (!) la executarea comenzii.

Aceasta particularitate face scripting-ul de shell sa fie diferit fata de limbajele de programare obisnuite; in mod normal, erorile nu sintaxa nu puteau fi cauzate de valorile variabilelor, ci doar de felul in care a fost scris codul sursa.

6.2.6.2. Tipuri de date si alte atribute ale variabilelor

Implicit, valorile variabilelor sunt siruri de caractere, ceea ce le lasa libertatea de a memora orice fel de informatie (sub forma de siruri de caractere pot fi stocate numere, cai in sistemul de fisiere etc). In bash, variabilele dispun de atribute suplimentare, ce ne permit fie sa impunem tipul de date al variabilei, fie sa o declaram ca read-only (constanta). Comanda folosita in acest scop este **declare**, iar optiunile de interes sunt:

- **-i** – impune ca variabila sa fie de tip numeric intreg. Orice operatie in care va participa variabila va trata valoarea variabilei ca numar
- **-a** – impune ca variabila sa fie de tip array (tablou)
- **-r** – creeaza o constanta. Odata valoarea variabilei stabilita, ea nu va mai putea fi modificata

```
# crearea unei variabile de tip int
student@Desktop $ declare -i LEVEL=5
# crearea unei constante MAX
student@Desktop $ declare -r MAX=10
# la incercarea de modificare a valorii constantei primim eroare:
student@Desktop $ MAX=5
bash: MAX: readonly variable
```

6.2.6.3. Variabile speciale

Variabilele speciale sunt variabile cu nume deosebit, definite automat de catre shell si a caror valoare este tinuta "la zi" de catre acesta. Ele contin diferite informatii utile in scripturi. Iata cateva exemple:

- **$?** - memoreaza automat valoarea de iesire a ultimei comenzi (0-succes, nenul - esec). Este utila pentru a determina in mod algoritmic daca o comanda s-a incheiat sau nu cu succes
- **$#** - numarul de argumente pasate scriptului la apelare
- **$@** - lista de argumente pasate scriptului la apelare
- **$$** - process ID-ul shell-ului din care este accesata variabila

```
student@Desktop $ ls -l /etc/hosts
-rw-r--r-- 1 root root 253 2009-05-03 18:16 /etc/hosts
student@Desktop $ echo $?
0
student@Desktop $ ls -l /etc/inexistent
ls: cannot access /etc/inexistent: No such file or directory
student@Desktop $ echo $?
2
```

6.2.6.4. Accesarea argumentelor pasate scriptului la apelare

Deseori este necesar ca executia unui script sa fie parametrizata pasandu-i-se acestuia argumente la apelare. Pentru ca scriptul sa poata tine cont de valorile acestor argumente, trebuie ca ele sa fie disponibile din interiorul scriptului. In acest scop, in interiorul oricarui script (si, de fapt, al oricarui shell) sunt definite automat urmatoarele variabile, denumite si *parametri pozitionali*:

- **$0** – reprezinta numele exact cu care a fost apelat scriptul. Un script poate fi executat in mai multe moduri – cu o cale absoluta, una relativa sau chiar prin intermediul unui symlink catre script; de fiecare data, $0 va reflecta calea exacta folosita pentru apelarea lui
- **$1, $2,** .... – reprezinta primul, al doilea,...etc. argument pasat scriptului. Incepand cu zecelea argument este necesara includerea numarului intre acolade: ${10}, ${11} etc.

Lista de parametri pozitionali impreuna cu variabilele speciale prezentate mai sus, $# si $@, fac usoara atat accesarea punctuala a valorilor anumitor argumente pasate in linia de comanda, cat si parcurgerea automatizata a listei de argumente si prelucrarea acestora.

Exemplu: fie scriptul *culoare.sh* de mai jos, care foloseste argumentul pasat in linia de comanda pentru a-si genera output-ul:

```
#!/bin/bash
echo “Culoarea mea preferata este $1“
```

Scriptul poate fi acum rulat cu diverse argumente:

```
student@Desktop $ ./culoare.sh rosu
Culoarea mea preferata este rosu
student@Desktop $ ./culoare.sh verde
Culoarea mea preferata este verde
```

6.2.7. Escaping

Termenul de *escaping* se refera la procedee prin care se elimina momentan semnificatia speciala a unora dintre metacaracterele shell-ului. Spre exemplu, in contexte in care nu vrem ca * sa fie inteles ca “0 sau mai multe caractere”, ci ca simplul caracter *, va trebui sa luam masuri speciale in acest sens.

Exista urmatoarele solutii de escaping:

- folosirea lui \ (asa-numitul **escape character**). Rolul lui escape character (care este caracter special la randul sau) este de a inlatura semnificatia speciala a caracterului ce-i urmeaza (ex: \$, *)
- folosind **ghilimele**. Atunci cand un string este incadrat intre ghilimele, metacaracterele din interior isi pierd semnificatia speciala, cu exceptia lui \ si $ (\ ramane special doar inainte de $,” si NEWLINE). In interior se poate obtine caracterul ” prin precedarea cu \ (ex: ”Caracterul \” apare ca atare”). Faptul ca $ ramane metacaracter inseamna ca variabilele sunt in continuare recunoscute si inlocuite cu valoarea lor
- folosind **apostroafe**. Incadrarea intre apostroafe a unui string face ca toate metacaracterele continute sa isi piarda semnificatia speciala. Astfel, nu se mai efectueaza niciun fel de substitutii de variabile, iar in interiorul apostroafelor nu mai putem folosi apostrof (nici macar precedat de \ )

```
# crearea unui director numit *
mkdir \*
# crearea unui director al carui nume contine un spatiu; fara quoting s-ar fi creat 2 directoare!
mkdir “Poze Tabara”
# intrarea in directorul creat anterior se poate face acum in trei moduri
cd “Poze Tabara”
cd 'Poze Tabara'
cd Poze\ Tabara
# variabilele dintre ghilimele se inlocuiesc automat cu valoarea lor
CULOARE=bleu
echo “Culoare aleasa: $CULOARE” # afiseaza Culoare aleasa: bleu
# variabilele dintre apostroafe nu mai sunt substituite, din cauza ca $ nu mai este metacaracter
echo 'Culoare aleasa: $CULOARE' # afiseaza Culoare aleasa: $CULOARE
```

6.2.8. Capturarea output-ului unei comenzi

Atunci cand avem nevoie de a include rezultatul unei comenzi intr-o alta linie de comanda (ex: a-l atribui unei variabile) putem incadra comanda ce face subiectul capturarii intre caractere ` (back quote – aflate in general pe tasta ~). Atunci cand in cadrul unei linii de comanda apare o astfel de constructie, se va executa comanda continuta intre back quotes si intreaga constructie va fi inlocuita cu ceea ce ea ar fi afisat pe ecran daca rula de sine statator. Iata exemple:

```
# memorarea numarului de linii ale unui fisier intr-o variabila
NRLINII=`cat /etc/termcap|wc -l`
# folosirea output-ului unei comenzi ca partea a alteia
FISIER=/etc/termcap
echo "Fisierul $FISIER are `cat $FISIER|wc -l` linii"
```

Nota: *o sintaxa alternativa pentru `comanda` este $(comanda).*

6.2.9. Functii

O functie reprezinta o succesiune de comenzi careia i se atribuie un nume, putand astfel fi apelata ulterior in mod repetat prin folosirea numelui. A crea o functie in shell-ul Linux inseamna de fapt a "inventa" o noua comanda – numele comenzii fiind cel al functiei, iar efectul fiind dat de instructiunile continute in functie. Ca si o comanda, functia poate primi argumente (dar cu aceeasi sintaxa ca o comanda, nu cu sintaxa din matematica sau programarea uzuala!). Sintaxa pentru definirea unei functii este:

```
function numefunctie() {
      comanda 1
      comanda 2
      ...
}
```

Atentie! *Dupa acolada deschisa trebuie sa existe fie un spatiu sau TAB, fie NEWLINE inainte de lista de comenzi.*

Apelarea unei functii este identica cu cea a unei comenzi. Atunci cand dorim sa pasam parametri functiei, nu vom folosi sintaxa functie(p1,p2), ci cea pentru comenzi: **functie p1 p2**. Argumentele pasate functiei sunt accesibile in interiorul acesteia sub forma de parametri pozitionali ($1, $2 etc).

La fel ca o comanda, o functie produce un cod de iesire ce indica succesul executiei. Instructiunea folosita in acest scop este **return** urmata de codul de iesire. Daca nu exista *return*, codul de iesire al functiei va fi implicit cel al ultimei comenzi executate in cadrul ei.

Nota: *diferenta intre return si exit este ca primul determina iesirea din functie, pe cand ultimul incheie executia intregului script din care face parte functia.*

Stergerea unei functii se realizeaza cu comanda **unset**, folosind optiunea -f.

```
student@Desktop $ function hello() { echo "Salut $1, bine ai venit"; return 100; }
student@Desktop $ hello Mihai
Salut Mihai, bine ai venit
student@Desktop $ echo $?
100
student@Desktop $ unset -f hello
```

Nota: *cu informatiile de pana acum, deducem ca o comanda executata in shell poate fi de unul dintre urmatoarele tipuri:*

- *built-in (comanda cunoscuta chiar de catre executabilul shell-ului) – exemplu: exit, unset*
- *alias*
- *functie*
- *fisier executabil (binar sau script)*

In caz de suprapuneri (ex: alias, functie si executabil cu acelasi nume), ordinea in care shell-ul ia on considerare comenzile este chiar cea de mai sus: o functie "ia fata" executabilului cu acelasi nume, un alias este preferat unei functii etc.
Pentru a determina tipul unei comenzi se poate folosi comanda **type**.

6.2.10. Instructiuni decizionale

6.2.10.1. Generalitati

Instructiunile decizionale permit executia conditionata a diverse portiuni de cod in functie de conditii puse de catre programator. Spre exemplu, dorim ca, daca argumentul pasat unui script este vid, sa afisam utilizatorului o eroare, iar daca este corect sa efectuam o anumita operatie ce il implica.

Instructiunile decizionale sunt de doua tipuri:

- instructiunea decizionala simpla, **if** – primeste o conditie care se evalueaza adevarat sau fals, iar in functie de valoarea ei se va executa una din doua portiuni de cod (sau se va executa conditionat o singura portiune de cod - vezi mai jos)
- instructiunea decizionala multipla, **case** – primeste o expresie ce poate lua diferite valori; pentru fiecare dintre valorile posibile (sau pentru cateva dintre ele) se executa cate o portiune de cod specifica

O particularitate de sintaxa a instructiunilor decizionale este ca delimitatorul de final al comenzii este delimitatorul de inceput scris invers: **if** se incheie cu **fi**, iar **case** se incheie cu **esac** (vezi mai jos sintaxele generale).

6.2.10.2. Instructiunea decizionala simpla (if)

Sintaxa generala a comenzii **if** este:

```
# pentru executia conditionata a unei portiuni de cod
if comanda_conditie; then
	lista_comenzi
fi

# pentru executia uneia din doua portiuni de cod
# lista_comenzi_1 va fi executata numai daca comanda_conditie se executa cu succes;
# in caz contrar se executa lista_comenzi_2
if comanda_conditie; then
	lista_comenzi_1
else
	lista_comenzi_2
fi
```

Conditia este o comanda in functie de codul de iesire al careia if va decide ce ramura de cod sa execute. Pentru a folosi conditii uzuale din programare (comparari ale valorilor variabilelor cu diferite praguri, comparatii numerice etc) se pot folosi doua moduri de a pune conditia:

- **test** expresie – comanda *test* primeste ca argument o expresie in care pot participa valori ale variabilelor definite combinate cu ajutorul a diferiti operatori (vezi mai jos). Codul de iesire al comenzii *test* indica succesul in cazul in care expresia se evalueaza ca true
- **[expresie]** - acelasi efect ca in cazul lui *test* si acelasi format al expresiei. ***Atentie insa! Expresia trebuie separata de parantezele drepte prin spatii!***

Expresia folosita pe post de conditie reprezinta o alaturare de conditii elementare, agregate prin SAU/SI logic. Fiecare conditie elementara presupune folosirea unui operator si a uneia sau mai multor valori – fie scrise ca atare in codul sursa, fie valori de variabile. Iata principalii operatori:

- operatori care isi trateaza operanzii ca fiind căi in sistemul de fisiere
  - **-f** *cale* – verifica daca fisierul referit exista si este de tip fisier obisnuit
  - **-d** *cale* – verifica daca fisierul referit exista si este de tip director
  - **-r, -w** sau **-x** *cale* – verifica daca fisierul exista si scriptul are drept de read, write sau execute asupra sa
- operatori care isi trateaza operanzii ca siruri de caractere
  - *sir1* == *sir2* - verifica egalitatea a doua siruri de caractere
  - *sir1* **!=** *sir2* - verifica daca cele doua siruri de caractere sunt diferite
  - **-n** *sir* – verifica daca sirul de caractere este nevid
  - **-z** *sir* – verifica daca sirul de caractere este vid
- operatori care isi trateaza operanzii ca numere (corespunzator diferitilor operatori de comparare din matematica)
  - *val1* **-eq** *val2* – verifica daca cele doua valori sunt numeric egale
  - *val1* **-ne** *val2* – verifica daca cele doua valori sunt numeric diferite
  - *val1* **-gt** *val2* – verifica daca prima valoare este strict mai mare decat a doua
  - *val1* **-lt** *val2* – verifica daca prima valoare este strict mai mica decat a doua
  - *val1* **-ge** *val2* – verifica daca prima valoare este mai mare sau egala cu a doua
  - *val1* **-le** *val2* – verifica daca prima valoare este mai mica sau egala cu a doua
- operatori logici
  - **-o** – OR (SAU logic)
  - **-a** – AND (SI logic). Are prioritate superioara lui -o
  - **!** – NOT (negare logica). Este folosit pentru inversarea valorii de adevar a conditiilor (ex: *if [! -f FILE];* va verifica daca fisierul NU este de tip fisier obisnuit)

Nota: *lista completa de operatori poate fi gasita in* man test.

Exemplu: un script care citeste de la tastatura un nume de fisier si afiseaza continutul acestuia daca fisierul contine cel putin 10 linii:

```
#!/bin/bash

read FISIER

# a fost introdus ceva?
if [ -z $FISIER ]; then echo “Lipseste numele fisierului”; exit 1; fi

# fisierul introdus este unul obisnuit asupra caruia scriptul are drept de citire?
if [ ! -f $FISIER -o ! -r $FISIER ]; then
      echo “Fisierul trebuie sa fie unul obisnuit si sa poata fi citit”; exit 2;
fi

# fisierul este gol? (verificam daca sirul reprezentat de continutul lui este vid)
if [ -z “`cat $FISIER`” ];then echo “Eroare – fisierul este gol”; exit 3; fi

# fisierul are cel putin 10 linii? (capturam output-ul lui wc si il comparam cu 10)
if [ “`cat $FISIER|wc -l`” -lt 10 ];then echo “Fisierul nu are suficiente linii”; exit 4; fi

# daca am trecut de toate filtrele si am ajuns aici, putem afisa continutul sau
less $FISIER
```

6.2.10.3. Instructiunea decizionala multipla (case)

Sintaxa generala a comenzii **case** este:

```
case expresie in
      valoare1) lista_instructiuni_1;
                ;;
      valoare2) lista_instructiuni_2;
                ;;
      valoare3 |valoare4) lista_instructiuni_3
                ;;
      *) lista_instructiuni_default;
            ;;
esac
```

Daca expresia ia valoarea *valoare1*, va fi executata *lista_instructiuni_1*; daca ia valoarea *valoare2*, se va executa *lista_instructiuni_2*. Daca ia una dintre valorile *valoare3* sau *valoare4*, va fi executata cea de-a treia lista de instructiuni. Cazul * “prinde” toate celelalte valori care nu au fost listate explicit si executa pentru ele *lista_instructiuni_default*.

6.2.11. Instructiuni pentru executie repetitiva

6.2.11.1. Aspecte generale

Folosim astfel de instructiuni atunci cand avem nevoie sa executam in mod repetat, controlat, o secventa de comenzi. Exista trei comenzi de shell gandite in acest scop, asemanatoare cu structurile din limbajele de programare obisnuite: **while**, **until** si **for**.

Instructiunile repetitive debuteaza cu cuvantul cheie corespunzator si au ca delimitator de final cuvantul cheie *done*.

6.2.11.2. Instructiunile *while* si *until*

Instructiunea **while** realizeaza executia repetata a unui bloc de instructiuni atata timp cat o conditie este adevarata. Spre exemplu, putem solicita utilizatorului sa introduca de la tastatura un username atata timp cat username-ul introdus este invalid. Sintaxa comenzii este:

```
while comanda_test; do lista_comenzi; done
```

Conditia, pusa sub forma lui *comanda_test*, functioneaza pe acelasi principii ca in cazul lui if (putem folosi *test expresie* sau *[expresie]*). Iata cum am proceda pentru a solicita de la tastatura un username cat timp acesta este vid sau nu este format exclusiv din litere:

```
while [ -z $USR -o -z “`cat $USR|egrep “^[A-Za-z]+$”`” ];do
      echo “Introduceti un username format numai din litere”
      read USR
done
```

Nota: *a se observa procedeul folosit pentru verificarea formatului: s-a folosit comanda egrep pentru a verifica daca valoarea variabilei USR este formata numai din litere. Rezultatul comenzii egrep este fie valoarea variabilei USR (in cazul in care formatul acesteia este valid), fie sirul vid pentru format incorect. Asadar*

output-ul lui grep este vid sau nevid in functie de corectitudinea formatului, ceea ce ne permite sa folosim testul -z aplicat sirului de caractere ce constituie output-ul comenzii.

Comanda **until** functioneaza pe aceleasi principii si cu aceeasi sintaxa ca si while, insa cu singura diferenta ca executa secventa de instructiuni continuta cat timp conditia este falsa.

6.2.11.3. Instructiunea for

Instructiunea **for** este folosita atunci cand dorim sa parcurgem o succesiune de elemente, executand o secventa de cod pentru fiecare dintre ele. Sintaxa generala a comenzii este:

```
for VAR in lista_cuvinte_sau_linii; do lista_comenzi; done
```

Lista de cuvinte poate fi specificata ca atare in instructiunea for, sau, in cazul cel mai util, poate proveni din capturarea output-ului altei comenzi. Iata un exemplu cu primul caz:

```
for SPORT in tennis hiking soccer; do
      echo "I love $SPORT"
done
```

Variabila SPORT ia pe rand ca valoare fiecare cuvant din lista (*tennis*, *hiking* si *soccer*); pentru fiecare valoare se executa codul din interiorul constructiei for, rezultand trei linii afisate pe ecran – cate una pentru fiecare sport.

Adevarata putere a lui *for* iese la iveala atunci cand lista de elemente prin care se itereaza este produsa de catre o alta comanda. Spre exemplu, putem parcurge lista de numere de mobil aflate intr-un fisier:

```
for NR in `cat $fisier | egrep "^07[[:digit:]]{8}$"`; do
      echo '+4'$NR; # afisarea cu prefix de tara, in formatul +40723456789
done
```

Atunci cand se doreste iterarea printr-o serie de valori numerice, se poate folosi pentru generarea seriei comanda **seq**, care primeste ca argumente limitele seriei. Spre exemplu, pentru a afisa numerele intre 1 si 10 putem scrie:

```
student@Desktop$ for n in `seq 1 3`; do echo $n; done
1
2
3
```

6.2.11.4. Instructiunile *break* si *continue*

Aceste doua instructiuni se folosesc in cadrul secventei de instructiuni din buclele *while*, *until* sau *for*, dupa cum urmeaza:

- **break** determina iesirea din bucla; iterarile ramase nu se mai efectueaza
- **continue** determina ignorarea restului de comenzi din iterarea curenta si trecerea la cea urmatoare

```
for i in `seq 1 10`; do echo -n "$i "; if [ $i -eq 2 ]; then break; fi; done; echo GATA
# afiseaza: 1 2 GATA
```

```
for i in `seq 1 3`; do
      echo -n "begin $i | "; if [ $i -eq 2 ]; then continue; fi; echo -n "end $i | "
done
# begin 1 | end 1 | begin 2 | begin 3 | end 3 | (end 2 este omis din cauza lui continue)
```

6.2.12. Obtinere de input de la utilizator

Pentru a citi date de la tastatura se poate folosi comanda **read**. Cand interpretorul de comenzi o intalneste in cadrul unui script, el va astepta utilizatorul sa introduca un sir de caractere si sa apese ENTER. Sirul primit va fi memorat in variabila data ca argument lui read:

```
echo “Introduceti data nasterii”
read DATA
if [ -z $DATA ]; then echo “Eroare - data introdusa este vida”
```

Comanda *read* primeste cateva optiuni utile:

- **-p** – specifica prompt string-ul (sirul afisat utilizatorului inainte de a se astepta input de la el). Este important sa utilizam aceasta optiune (sau sa notificam utilizatorul folosind *echo*, ca in exemplul de mai sus), deoarece implicit interpretorul nu ii atrage atentia in niciun fel asupra faptului ca asteapta input de la el
- **-n** – specifica numarul maxim de caractere citit. Daca utilizatorul a introdus acel numar de caractere, comanda read se incheie automat, fara sa mai fie nevoie ca utilizatorul sa apese ENTER, sirul introdus va fi memorat in variabila pasata ca parametru lui read, iar executia scriptului va continua cu comanda urmatoare
- **-t** – specifica intervalul de timp cat *read* va astepta ca utilizatorul sa furnizeze input. Dupa scurgerea acestui timp comanda *read* se va incheia indicand prin codul sau de iesire esecul, iar executia scriptului va continua cu comanda urmatoare

6.2.13. Alte comenzi utile in scripting

Dintre multiplele comenzi care fac programatorului de shell viata mai usoara (sau mai palpitanta) amintim:

- **cut** – este folosit pentru a imparti un sir de caractere in campuri, prin specificarea delimitatorului intre campuri si a campurilor dorite. Spre exemplu, daca dorim sa afisam username-ul si shell-ul utilizatorului student din /etc/passwd vom scrie:

```
cat /etc/passwd | grep student | cut -d ":" -f 1,6,7
# afiseaza: student:/home/student:/bin/bash
```

Optiunea -d specifica delimitatorul, iar -f indica campul (sau campurile) returnate. Atunci cand sunt solicitate mai multe campuri (ca in exemplul de mai sus) ele vor fi separate prin delimitatorul folosit pentru separare; in cazul unui singur camp, delimitatorul nu mai apare in output-ul lui cut.

6.3. BIBLIOGRAFIE

- **Bash Beginner's Guide**: http://tldp.org/LDP/Bash-Beginners-Guide/html/index.html
- **CompTIA Linux+ Certification Guide** – Cap 9: Using the Linux Shell
- **LPI Linux Certification in a Nutshell (2nd ed)** – Cap 17: Shells, scripting, programming and compiling
- **Bash Reference Manual:** http://www.gnu.org/software/bash/manual/bashref.html

Conventie

-referitoare la conditiile de desfasurare a examenului intermediar-

1. **Caracteristici generale ale examenului Intermediar teoretic :**

   a. are ca scop verificarea cunostintelor acumulate in prima parte a fiecarui modul
   b. este realizat din intrebari cuprinse in examenele partiale care trebuie sustinute pana la data examenului
   c. este accesibil pe http://www.infoacademy.net → Login → Nume_Clasa → TakeAssesment
   d. este de tip test grila si are in medie 20-30 de intrebari cu raspuns unic sau multiplu
   e. dureaza 30 de minute
   f. poate fi sustinut numai la sediul academiei „InfoAcademy” in prezenta unui delegat InfoAcademy
   g. se considera promovat daca nota weighted obtinuta este de cel putin 75%
   h. poate fi sustinut de maxim 2 ori la interval de 1 saptamana intre cele 2 incercari

2. **Data la care se sustine examenul intermediar**

   a. examenul intermediar (prima incercare) se sustine in cadrul celei de a 7-a sedinte de curs
   b. repetarea examenului intermediar (a doua incercare) se sustine in urmatoarea zi de vineri dupa prima sustinere a examenului, intre orele 8-12,00.

3. **Conditii de participare la examenul intermediar teoretic :**

   Poate participa la examenul intermediar teoretic orice student Infoacademy care:
   a. a promovat toate examenele partiale care trebuiau date pana in acel moment, cu cel putin 75%
   b. a achitat integral taxa de curs
   c. are asupra sa un act cu fotografie care ii atesta identitatea
   d. se obliga sa respecte conditiile de desfasurare a examenului intermediar incluse in aceasta Conventie

4. **In timpul desfasurarii examenului intermediar, studentul se obliga :**

   a. sa nu comunice direct sau prin dispozitive electronice cu alte persoane
   b. sa nu salveze pe nici un fel de suport intrebarile/imaginile/paginile incluse in examenul sustinut
   c. sa nu incerce sa transmita prin Intranet/Extranet/Internet continutul intrebarilor, imagini sau pagini web continute in examen sau aflate pe calculatorul de pe care sustine examenul
   d. sa nu utilizeze materiale scrise/inregistrate pentru a afla raspunsuri la intrebarile incluse in examen
   e. sa nu utilizeze dispozitive de calcul (de tip calculatorul inclus in telefon, ceas, sistemul de operare, etc..) pentru a efectua operatii de orice tip
   f. sa nu aiba asupra sa dispozitive de inregistrare audio/video (de tip telefon celular,etc..)
   g. sa nu afecteze prin tinuta sau comportament desfasurarea examenului

5. **Dupa incheierea examenului intermediar, studentul :**

   a. iese din cont (Logout), preda ciorna si paraseste sala de examen

6. **Este obligatoriu examenul intermediar?**

   a. conform contractului examenul intermediar este obligatoriu.

b. cei care nu sustin examenul intermediar si nu achita rata a 2-a (in cazul in care au platit partial cursul) nu mai sunt eligibili pentru participarea la cursuri/laboratoare si examen final.

7. Incalcarea Conventiei

a. incalcarea prevederilor articolului 4 a,b,c,d,e,f duce la eliminarea din examenul intermediar si din academie cu pierderea oricaror drepturi asupra modulului al carui examen intermediar se sustine.
b. incalcarea prevederilor articolului 4.g duce la eliminarea studentului din examenul intermediar cu pierderea uneia din cele doua sanse de sustinere a acestuia

Prezentarea la examenul intermediar echivaleaza cu acceptarea deplina a conditiilor acestei conventii.