

# Bash ca limbaj de programare

– Ștefan Tanasă și Sabin Corneliu Buraga

## Caracterizare

Un *shell* reprezintă un macro-procesor capabil de a executa comenzi. Un *shell* Linux (UNIX) este atât un interpretor de comenzi, interfață între utilizator și un bogat set de comenzi și utilitare, cât și materializarea unui limbaj de programare care oferă mecanisme complexe de operare cu sistemul. *Bash* este un *shell* (interpretor de comenzi) specific sistemului de operare Linux, conceput sub auspiciile GNU (*GNU: is Not Unix*). Numele este un acronim de la *Bourne-Again Shell*, după Steve Bourne, autorul *shell*-ului *sh* pentru UNIX, predecesorul *bash*-ului. Pentru sistemele de operare Linux, *shell*-ul implicit este *bash*. Există mai multe versiuni de *bash* disponibile, în prezent cea mai utilizată pe un sistem Linux fiind *bash* 2.0. Ca și alte pachete de programe GNU, *bash*-ul este portabil și se găsește în aproape toate versiunile de UNIX, iar independent îl putem rula în OS/2, DOS și Windows NT. *Shell*-ul *bash* oferă o multitudine de posibilități administratorilor și programatorilor de sistem, posedând toate caracteristicile unui limbaj de programare de nivel înalt. În cele ce urmează vom descrie o parte dintre cele mai interesante și utile aspecte ale acestui *shell*.

## Script-uri bash

Comenzile *bash* pe care dorim să le execute *shell*-ul pot fi stocate în fișiere. Acestor fișiere li dau drepturi de execuție (cu comanda `chmod +x fișier`), după care pot fi executate ca orice altă comandă. Fișierele conținând comenzi ale unui limbaj de tip *script*, cum este cazul *bash*-ului, se mai numesc și *script*-uri.

De obicei, la începutul fiecărui fișier *script* se stabilește *shell*-ul care va fi invocat de către sistemul de operare pentru a se executa comenzile și construcțiile *bash*. Pentru *bash* vom avea: `#!/bin/bash`. Pentru a executa *script*-urile putem utiliza următoarele modalități de apelare:

```
(infoiasi)$ . script [ parametri ]
(infoiasi)$ ./script [ parametri ]
(infoiasi)$ bash script [ parametri ]
```

Parametrii pot să lipsească, dacă *script*-ul dorit a fi rulat se poate apela și fără aceștia. Comentariile se introduc prin simbolul „#” și sunt valabile până la sfârșitul liniei (din acest punct de vedere sunt similare comentariilor // din C++ sau Java).

În cele ce urmează vom vedea că *shell*-ul *bash* oferă toate construcțiile unui limbaj de programare de nivel înalt, punând la dispoziția administratorilor de sistem un bogat set de facilități.

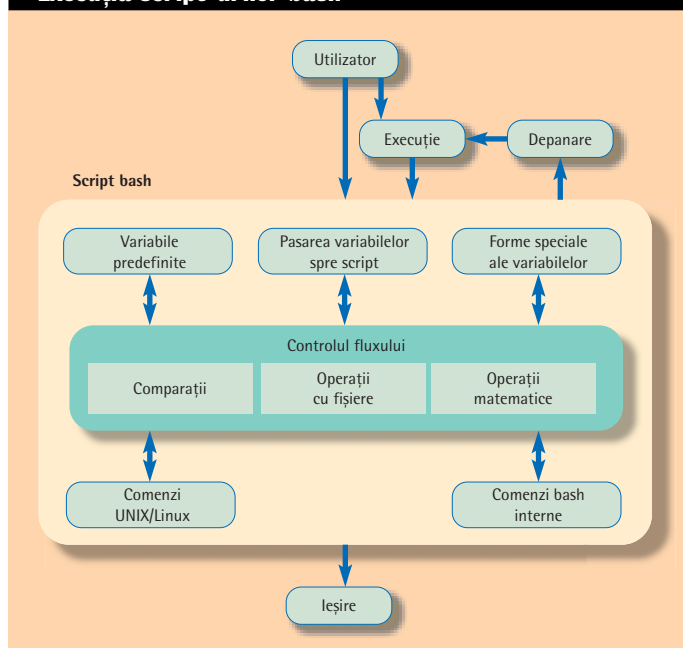
## Variabile

Deseori am dori ca anumite rezultate să le stocăm temporar în memorie pentru prelucrări ulterioare. Acest lucru poate fi realizat fie cu ajutorul unor fișiere temporare (soluție ineficientă, consumatoare de resurse ale sistemului), fie prin intermediul *variabilelor* puse la dispoziție de *shell*. Pentru *shell*-ul *bash*, toate variabilele sunt de tip șir de caractere, ele fiind create „din zbor”. Pentru a vizualiza toate variabilele definite și valorile corespunzătoare acestora trebuie utilizată comanda `set`. Inițializarea unei variabile se realizează cu operatorul „=” (acest operator nu trebuie să fie precedat sau succedat de spații): `variabila=valoarea`. Numele variabilei trebuie precedat de simbolul „\$” atunci când referim valoarea respectivei variabile. Pentru *bash*, avem la dispoziție și comanda internă `let` pentru a realiza atribuire de valori unei variabile. Sunt acceptați și operatorii `+=`, `-=` etc. prezenți în C ori Perl. De asemenea, *shell*-ul *bash* pune la dispoziție un bogat set de facilități pentru evaluări matematice utilizând numere întregi (în alte *shell*-uri posibile doar cu ajutorul comenzii `expr`). Astfel, pentru a evalua o expresie vom scrie acea expresie între paranteze rotunde duble precedate de caracterul „\$”. Pentru efectuarea de calcule fracționare se poate utiliza comanda `bc`. Afișarea conținutului unei variabile se poate realiza cu ajutorul comenzii `echo`.

```
(infoiasi)$ a = 10
bash: a: command not found
(infoiasi)$ a=10
(infoiasi)$ echo $a
10
(infoiasi)$ v="-o cgi-bin"
(infoiasi)$ ls $v
total 4
-rwxr-xr-x  1 user    99 Nov 27 18:42 form.cgi
(infoiasi)$ echo $((12 + 21/3))
19
(infoiasi)$ v=15 ; z=4
(infoiasi)$ echo $((5 - $v % $z))
2
(infoiasi)$ echo `expr 3 - 1`
2
```

Pentru ca `echo` să nu treacă automat la rând nou după afișarea valorilor se va utiliza opțiunea „-n”. Opțiunea „-e” permite utilizarea codurilor *escape* (cele introduse de *backslash*). Aceste coduri *escape* (similare celor prezente în limbajele C sau Perl) sunt:

## Execuția script-urilor bash



- \a emite un sunet (*alarm*).
- \b deplasează cursorul cu o poziție spre stânga (*backspace*).
- \f trece cursorul pe rândul următor, rămânând pe aceeași coloană.
- \n trece cursorul pe prima poziție de pe linia următoare.
- \r mută cursorul la începutul liniei curente.
- \t inserează un caracter *tab*.
- \nnn inserează caracterul care are codul ASCII nnn (poate avea una, două sau trei cifre), în octal.
- \xnnn inserează caracterul care are codul ASCII nnn (poate avea una, două sau trei cifre), cod dat în hexa.

O comandă înrudită este `printf` (foarte asemănătoare ca funcționalitate cu funcția `printf()` din C). Atunci când dorim după o variabilă să afișăm imediat un alt șir de caractere, numele variabilei trebuie încadrat de acolade.

```
(infoiasi)$ nume=Maria
(infoiasi)$ echo Ana$nume
AnaMaria
(infoiasi)$ echo ${nume}na
Mariana
(infoiasi)$ echo $numena
(infoiasi)$
```

Ultima comandă afișează conținutul variabilei `vna` (în cazul nostru nefiind definită în prealabil va fi considerată vidă).

Dacă în loc de ghilimele vom folosi apostrofuri, atunci *shell*-ul nu va mai expanda valoarea variabilei:

```
(infoiasi)$ curs=Web
(infoiasi)$ echo "Cursul meu preferat este $curs"
Cursul meu preferat este Web
(infoiasi)$ echo 'Cursul meu preferat este $curs'
Cursul meu preferat este $curs
```

O variabilă poate primi ca valoare rezultatul execuției unei comenzi. Pentru acest lucru comanda trebuie încadrată de apostrofuri inverse:

```
(infoiasi)$ v=`wc -l void.html`
(infoiasi)$ echo $v
8 void.html
```

O variantă identică semantic este următoarea (preferată în versiunea 2.0 a *shell*-ului `bash`):

```
(infoiasi)$ v=$(wc -l void.html)
```

Pentru a șterge o variabilă se poate utiliza una dintre variantele:

```
(infoiasi)$ unset variabila
(infoiasi)$ variabila=
```

Citirea de la tastatură a valorii unei variabile se realizează cu comanda `read`. Comanda `readonly` stabilește că valorile variabilelor specificate nu pot fi modificate (așadar, variabilele devin constante).

```
(infoiasi)$ read nume
Sabin
(infoiasi)$ echo $nume
Sabin
(infoiasi)$ nume="Sabin Corneliu"
(infoiasi)$ echo $nume
Sabin Corneliu
(infoiasi)$ readonly nume
(infoiasi)$ nume=Victor
```

```
bash: nume: readonly variable
(infoiasi)$ unset nume
bash: unset: nume: cannot unset: readonly variable
```

Pentru ca o variabilă să aibă valoarea disponibilă proceselor copil (*e.g.* sub-*shell*-uri) ale *shell*-ului curent, va trebui exportată cu ajutorul comenzii `export`:

```
export nume TERM PS1
```

În mod normal, variabilele nu sunt vizibile în procesele copil ale *shell*-ului, ele fiind considerate locale procesului *shell* respectiv.

Atribuirea de valori unei variabile poate fi o atribuire condiționată.

- Construcția `${var:-sir}`, unde `var` este numele unei variabile, iar `sir` este un șir de caractere, se evaluează la valoarea variabilei `var` dacă aceasta este definită, iar în caz contrar la șirul specificat.

```
(infoiasi)$ echo ${anotimp:-Iarna}
Iarna
(infoiasi)$ echo $anotimp
(infoiasi)$ anotimp="E iarna iar"
(infoiasi)$ echo ${anotimp:-Iarna}
E iarna iar
(infoiasi)$ echo $anotimp
E iarna iar
```

- Expresia `${var:=sir}` se evaluează asemănător expresiei precedente, iar în plus în cazul în care variabila `var` nu este setată se inițializează cu șirul de caractere indicat (util pentru cazul în care dorim să asignăm o valoare implicită unei variabile, dacă aceasta nu este definită).

```
(infoiasi)$ unset cale
(infoiasi)$ echo ${cale:=/tmp}
/tmp
(infoiasi)$ cale=/home/user/tmp
(infoiasi)$ echo ${cale:=/tmp}
/home/user/tmp
(infoiasi)$ unset cale
(infoiasi)$ echo ${cale:=/tmp}
/tmp
(infoiasi)$ echo $cale
/tmp
```

- Dacă variabila `var` este setată, atunci valoarea expresiei `${var:+sir}` este dată de șirul specificat și valoarea variabilei nu se modifică, altfel valoarea respectivei expresii este șirul vid.

```
(infoiasi)$ unset comanda
(infoiasi)$ echo ${comanda:+ls}
(infoiasi)$ echo $comanda
(infoiasi)$ comanda=pwd
(infoiasi)$ echo ${comanda:+ls}
ls
(infoiasi)$ echo $comanda
pwd
```

- Construcția `${var:?sir}` generează un mesaj de eroare `sir` dacă variabila `var` nu este setată, iar în caz contrar se evaluează la valoarea variabilei specificate.

```
(infoiasi)$ cale=
(infoiasi)$ echo ${cale:? "Calea de directoare este vida."}
bash: cale: Calea de directoare este vida.
(infoiasi)$ cale=/home/busaco
```

```
(infoiasi)$ echo ${cale:?"Calea de directoare este vida."}
/home/busaco
```

Numărul de caractere memorate într-o variabilă `var` se obține în urma evaluării expresiei `${#var}`.

```
(infoiasi)$ un_autor="Sabin Corneliu Buraga"
(infoiasi)$ echo ${#un_autor}
21
```

### Variabile predefinite

În cadrul *shell*-ului avem acces la un număr de variabile predefinite, cele mai semnificative dintre acestea regăsindu-se în tabelul „Variabile“ (prin convenție, variabilele predefinite ale sistemului au numele dat cu majuscule).

Câteva exemple:

```
(infoiasi)$ echo $BASH_VERSION
2.04.11(1)-release
(infoiasi)$ echo $MACHTYPE
i386-redhat-linux-gnu
(infoiasi)$ echo $MAIL
/var/spool/mail/stanasa
(infoiasi)$ echo $HOME
/home/stanasa
(infoiasi)$ echo $TERM
linux
(infoiasi)$ echo $RANDOM
21144
```

Valorile variabilelor de sistem predefinite pot fi consultate prin intermediul comenzii `set`. De asemenea, asupra lor se pot utiliza comenzile `export`, `readonly` sau `unset`.

### Variabile speciale

Există câteva variabile speciale foarte utile în *script*-uri:

Variabile	
Variabilă	Descriere
HOME	Conține calea completă a directorului corespunzător utilizatorului curent. În cadrul specificatorilor de fișier, <code>\$HOME</code> poate fi substituită de caracterul <b>tilda</b> „~”.
USER	Furnizează numele de cont al utilizatorului curent.
LOGNAME	Conține numele de cont al utilizatorului curent.
HOSTNAME	Desemnează numele serverului.
HOSTTYPE	Furnizează tipul mașinii (procesorului).
OSTYPE	Desemnează tipul sistemului de operare.
MACHTYPE	Describe tipul sistemului în formatul <b>procesor - firma producătoare - tipul sistemului de operare</b> .
shell	Indică <b>shell</b> -ul implicit.
BASH	Indică fișierul care a generat această instanță a <b>shell</b> -ului.
BASH_VERSION	Furnizează versiunea bash.
MAIL	Conține numele fișierului unde sunt depozitate mesajele de <b>e-mail</b> primite.
PS1	Desemnează structura <b>prompt</b> -ului principal al <b>shell</b> -ului.
PS2	Reprezintă <b>prompt</b> -ul secundar al <b>shell</b> -ului (apare atunci când o comandă este scrisă pe mai multe rânduri).
TERM	Furnizează tipul de terminal.
PATH	Conține lista de directoare utilizată de <b>shell</b> pentru căutarea comenzilor (fișierelor executabile).
PWD	Furnizează directorul curent de lucru (cel care a fost stabilit de comanda <code>cd</code> ).
RANDOM	Conține un număr generat aleator cuprins între 0 și 32767. După utilizare, valoarea variabilei se modifică automat.

- `$0` conține numele *script*-ului;
- `$1`, `$2`, ..., `$9` reprezintă parametrii din linia de comandă (`$1` conține primul parametru, `$2` al doilea etc.);
- `*$` furnizează lista tuturor parametrilor din linia de comandă;
- `@` similar cu `*$`, dar parametrii sunt considerați elemente separate;
- `#` desemnează numărul parametrilor din linia de comandă;
- `$$` furnizează PID-ul procesului curent (această variabilă se poate folosi pentru a crea fișiere temporare cu nume unic, de exemplu având numele `/tmp/nume$$`);
- `$?` conține codul întors de ultima comandă executată (zero, semnificând `true`, sau un număr pozitiv desemnând valoarea logică `false`);
- `!` furnizează PID-ul ultimului proces executat în fundal;
- `-` desemnează opțiunile cu care a fost lansat *shell*-ul respectiv.

În cazul în care avem mai mult de nouă parametri în linia de comandă, pentru a putea avea acces la valorile tuturor parametrilor, vom utiliza comanda `shift`. Aceasta realizează o deplasare a elementelor listei de parametri în sensul următor: valoarea lui `$1` se pierde și primește valoarea lui `$2`, `$2` ia valoarea lui `$3` și așa mai departe, iar `$9` va lua valoarea parametrului următor celui referit de `$9` înainte.

Pentru a observa cum lucrează câteva din variabilele speciale de mai sus vom considera fișierul `cmd`:

```
#!/bin/bash
echo "Numele scriptului: $0"
echo "Parametri: $*"
echo "Numarul de parametri: $#"
```

```
echo "PID: $$"
```

Apoi considerăm următoarele execuții:

```
(infoiasi)$ . cmd Perl C Java bash
Numele scriptului: bash
Parametri: Perl C Java bash
Numarul de parametri: 4
PID: 926
(infoiasi)$ ./cmd Victor Stefan Sabin
Numele scriptului: ./cmd
Parametri: Victor Stefan Sabin
Numarul de parametri: 3
PID: 1465
```

Se observă faptul că pentru primul apel fișierul care se execută este `bash` (*shell*-ul), care execută comenzile din fișierul `cmd` (numele său este parametru). A doua variantă de apel aduce rezultatul scontat: `cmd` este fișierul care se execută.

### Instrucțiuni

*Shell*-ul `bash` pune la dispoziția programatorilor o serie de structuri de test: `if` și `case` și repetitive: `for`, `while`, `until`.

Structura condițională `if` are următoarea sintaxă:

```
if
then
    lista_de_comenzi_1
then
    lista_de_comenzi_2
[ elif
    lista_de_comenzi_3
then
    lista_de_comenzi_4 ]
...
[ else
    lista_de_comenzi_N ]
fi
```

Secvența `elif` poate apărea de câte ori este nevoie. Dacă ultima comandă din prima listă de comenzi se termină cu succes (returnează valoarea zero), se execută instrucțiunile care urmează lui `then`, altfel se

continuă cu următorul `elif` sau `else`. Când se ajunge pe o ramură `elif` se execută ca și cum ar fi un alt `if`. În programul de mai jos (denumit `rmtemp`) apare structura `if`:

```
#!/bin/bash
if
    echo "Stergerea fisierului \"temp\""
    rm temp 2>/dev/null
then
    echo "Fisierul \"temp\" a fost sters."
else
    echo "Fisierul \"temp\" nu a putut fi sters."
fi
```

Comanda `rm` șterge fișierul `temp` dacă există și poate fi șters (e.g. utilizatorul care apelează *script*-ul are drepturile necesare), altfel afișează un mesaj de eroare (mesajul nu apare pe ecran întrucât ieșirea standard de eroare este redirectată la `/dev/null`).

Sintaxa structurii condiționale `case` este cea de mai jos:

```
case expresie in
    [ sir_de_valori_1 ") lista_de_comenzi_1 ";; ]
    ...
    [ sir_de_valori_N ") lista_de_comenzi_N ";; ]
esac
```

Mai întâi se evaluează expresia după care se încercă o potrivire cu una din valorile din șirurile specificate. Dacă s-a găsit o potrivire (acesta va fi prima în ordinea definiției valorilor) se va executa lista de comenzi corespunzătoare, după care structura `case` se va termina. Cuvântul `esac` este oglinditul lui `case` și termină o construcție `case`.

Drept exemplu, considerăm că fișierul `opt` are următorul conținut:

```
#!/bin/bash
case $1 in
    -a|[fx-z] ) echo "S-a detectat o optiune valida";;
    -* ) echo "S-a detectat o optiune" ;;
    stop|start ) echo "S-a detectat un parametru valid" ;;
    ?* ) echo "S-a detectat un parametru" ;;
    * ) echo "Mod de utilizare: $0 param" ;;
esac
```

Pentru delimitarea mai multor valori se utilizează caracterul „|”. Pot apare *wildcard*-uri („\*“ va ține loc de zero sau multe caractere, iar „?” va ține locul unui singur caracter). Primul șir este echivalent cu `-a|-f|-x|-y|-z`. Al doilea va selecta toate cuvintele care încep cu caracterul „-“. Construcția `?*` va accepta toate cuvintele nevide (semnul de întrebare cere existența unui caracter).

Pentru a ieși forțat dintr-un *script* sau dintr-o funcție definită de utilizator (vezi mai jos) vom utiliza comanda `return` urmată de un cod de stare. Comanda `exit` va determina părăsirea *shell*-ului curent.

O altă comandă utilă este `select` care permite realizarea de interacțiuni în mod text. Următorul exemplu dă posibilitatea utilizatorului să aleagă între două opțiuni:

```
OPTIUNI="Salutari Iesire"
select opt in $OPTIUNI; do
    if [ $opt = "Iesire" ]; then
        echo "Am terminat..."
        exit
    elif [ $opt = "Salutari" ]; then
        echo "Salut! Ce mai faceți?"
    else # nici una din optiuni
        clear # stergem ecranul
```

```
        echo "Opțiune necunoscută..."
    fi
done
```

Comenzile pot fi executate condiționat folosind operatorii „&&” și „||”:

- Construcția: comanda1 && comanda2 funcționează astfel: se execută prima comandă, iar dacă aceasta se încheie cu succes (returnează codul 0) se execută și cea de-a doua comandă.
- Pentru comanda1 || comanda2 lucrurile decurg similar, doar că a doua comandă se execută atunci când prima întoarce un cod de eroare (nenul).

Un exemplu:

```
(infoiasi)$ gcc gaen.c -o gaend -O2 && strip gaend
```

Structura `for` are următoarea sintaxă:

```
for var [ in text ]
do
    lista_de_comenzi
done
```

Variabila `var` va lua succesiv valori din textul specificat (câte o linie). Dacă textul lipsește variabila va lua ca valori parametri transmiși în linia de comandă.

Programul de mai jos va genera un fișier conținând informații despre fiecare utilizator conectat:

```
for U in `who | cut -c1-8`
do
    finger $U >> lista_utilizatori
done
```

După cum am văzut, în `bash` în locul construcției ``who | cut -c1-8`` putem scrie `$( who | cut -c1-8 )`.

Sintaxa structurii repetitive `while` este furnizată în continuare:

```
while
    lista_de_comenzi_1
do
    lista_de_comenzi_2
done
```

Se execută prima listă de comenzi. Dacă ultima comandă din prima listă se încheie cu succes (returnează un cod de eroare nul), atunci se execută și a doua listă după care se reia bucla, altfel se iese din structura repetitivă.

Următorul *script* va simula execuția comenzii `cat`:

```
while read -r linie
do
    echo "$linie"
done
```

Pentru a ieși forțat dintr-un ciclu repetitiv se poate folosi `break`, iar pentru a trece direct la următoarea iterație se va utiliza `continue`, ca în limbajele C sau Java. Un exemplu:

```
for dir in $PATH; do
    test -z "$dir" && dir=.
    if test -f $dir/tail; then
        tail="$dir/tail"
        break
    fi
done
echo "Am gasit comanda tail: $tail"
```

## Comanda test

Bash-ul permite utilizarea expresiilor condiționale prin intermediul comenzii `test`. Aceasta are o formă scurtă în care lipsește numele comenzii, iar condiția este încadrată de parantezele drepte „`[]`”. Cu ajutorul lui `test` se pot efectua comparații aritmetice și asupra unor șirurilor de caractere, precum și teste asupra fișierelor. Pentru a testa diverse condiții, vom utiliza următoarele:

- *teste privitoare la fișiere*

- a fișier - true dacă fișierul există.
- d fișier - true dacă este director.
- e fișier - true dacă fișierul există.
- f fișier - true dacă fișierul există și este un fișier obișnuit.
- g fișier - true dacă fișierul există și aparține grupului.
- h fișier - true dacă fișierul există și este o legătură simbolică.
- p fișier - true dacă fișierul există și este de tip *pipe*.
- r fișier - true dacă fișierul există și poate fi citit.
- s fișier - true dacă fișierul există și are dimensiunea nenulă.
- w fișier - true dacă fișierul există și poate fi modificat.
- x fișier - true dacă fișierul există și este executabil.

- *teste referitoare la variabile*

- z variabilă - true dacă variabila este nesetată (sau conține un șir vid de caractere).

[ -n ] variabilă - true dacă variabila este setată.

- *teste privitoare la șiruri de caractere și numere*

- șir1 == șir2 - true dacă șirul de caractere 1 coincide cu șirul 2; se poate utiliza un singur egal.
- șir1 != șir2 - true dacă cele două șiruri sunt diferite.
- șir1 < șir2 - true dacă șirul 1 este înaintea șirului 2 în ordinea lexicografică.
- șir1 > șir2 - true dacă șirul 1 este după șirul 2 în ordinea lexicografică.
- arg1 OP arg2, unde OP poate fi: -eq, -ne, -lt, -le, -gt sau -ge - true dacă operatorii aritmetici binari returnează true; aceștia au semnificațiile de egal, diferit, mai mic, mai mic egal, mai mare și respectiv mai mare egal, iar arg1 și arg2 pot fi numere pozitive sau negative.

De menționat faptul că în bash valoarea de adevăr true este echivalentă cu 0, iar valoarea logică false este dată de un număr nenul.

Exemplul următor listează directoarele din directorul curent:

```
#!/bin/bash
for director in * # variabila va lua ca valori numele
do # tuturor fișierelor din directorul curent
if [ -d $director ]
then # afiseaza numele directorului
echo $director
fi
done
```

## Resurse bibliografice

- ✗ M.Budiu, Articolele publicate în NET Report: <http://www.cs.cmu.edu/~mihaib/articles/articles.html>
- ✗ S. Buraga, G. Ciobanu, *Atelier de programare în rețele de calculatoare*, Editura Polirom, Iași, 2001: <http://www.infoiasi.ro/~lrc/>
- ✗ C. Ramey, B. Fox, *Bash Reference Manual*, Free Software Foundation, 1998
- ✗ R. Sage, *UNIX pentru profesioniști*, Editura de Vest, Timișoara, 1993
- ✗ \*\*\*, *The Linux Documentation Project*: <http://metalab.unc.edu/LDP/>

## Script-uri sistem

În continuare vom prezenta o serie de fișiere de comenzi bash speciale, utilizate îndeosebi la configurarea sesiunii de lucru. *Shell*-ul permite ca fiecare utilizator să poată scrie un *script* care să fie executat la fiecare început de sesiune de lucru în sistem. Acest fișier rezidă în directorul *home* (indicat după cum am văzut de variabila sistem HOME) al utilizatorului și poate fi regăsit sub numele de `.bash_profile` sau `.profile`. În cadrul acestui fișier se pot defini *alias*-urile comenzilor folosite frecvent, se pot stabili diferite valori ale variabilelor de mediu (e.g. TERM ori PS1) sau se pot executa diverse comenzi (de exemplu, să se afișeze cu echo un mesaj de bun venit sau data curentă). În plus, fiecare utilizator poate avea un *script* care va fi rulat la momentul părăsirii sesiunii, acest fișier purtând numele `.bash_logout` și fiind localizat tot în directorul *home* al aceluși utilizator.

Administratorul sistemului poate pregăti diferite fișiere de inițiere, valabile pentru toți utilizatorii. Aceste fișiere *script* sunt stocate în directorul `/etc`. De exemplu, `/etc/profile` care va fi executat la oricare deschidere a unei noi sesiuni de lucru a fiecărui utilizator. La crearea unui cont, în directorul *home* al utilizatorului proaspăt creat vor fi plasate copii ale fișierelor `.bash_profile` și `.bash_logout` regăsite în directorul `/etc/skel`, utilizatorul putându-le ulterior modifica după dorință.

Un exemplu de *script* `.bash_profile` standard poate fi:

```
# .bash_profile
# incarcam alias-urile si functiile
if [ -f ~/.bashrc ]; then
. ~/.bashrc
fi
# modificam mediul
PATH=$PATH:$HOME/bin
BASH_ENV=$HOME/.bashrc
USERNAME=""
export USERNAME BASH_ENV PATH
```

Ori de câte ori este lansat un proces *shell* interactiv, va fi lansat mai întâi fișierul `script` `/etc/bashrc`, apoi `$HOME/.bashrc`. De altfel, în orice sistem UNIX (Linux) prin convenție orice nume de fișier terminat în `rc` (de la *run commands*) desemnează un fișier script de configurare. Spre exemplu, *script*-urile de configurare a serviciilor sistem rulate la inițierea sistemului de operare se găsesc în directorul `/etc/rc.d/init.d`. Într-un fișier `.bashrc` se pot defini *alias*-uri și se pot stabili diverse valori ale unor variabile de sistem:

```
# .bashrc
alias h='history'
alias j="jobs -l"
alias l="ls -l"
alias f=finger
# modificam tipul de terminal
TERM=vt100
# schimbam prompt-ul
# \h - numele masinii
# \u - numele utilizatorului
# \w - directorul de lucru (curent)
PS1="\h (\u): \w>"
export TERM PS1
```

Pentru fiecare utilizator, mai există un fișier, denumit `.bash_history` care este stocat în directorul *home* și care păstrează ultimele comenzi executate de utilizator.

*Autorii sunt cadre didactice la Facultatea de Informatică, Universitatea „Al.I. Cuza” din Iași, putând fi contactați prin e-mail la [stanasa@infoiasi.ro](mailto:stanasa@infoiasi.ro) și [busaco@infoiasi.ro](mailto:busaco@infoiasi.ro). ■*