

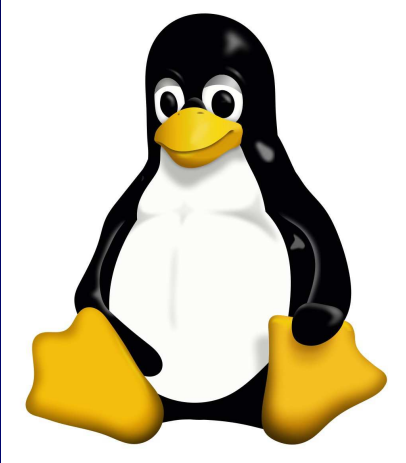
# GHID DE UTILIZARE LINUX (V)

## Lucrul la linia de comandă în UNIX, partea a IV-a: Interpretoare de comenzi UNIX, partea a II-a – *scripting* BASH

Cristian Vidrașcu

`cristian.vidrascu@info.uaic.ro`

Martie, 2025



# Sumar

- Introducere
- Facilitățile limbajului de *scripting* bash
- Variabile de *shell*
- Expresii aritmetice și logice
- Structuri de control alternative și repetitive
- Funcții *shell*
- Referințe bibliografice

- Introducere
  - Facilitățile limbajului de *scripting* bash**
    - Proceduri *shell* – *script*-uri
    - Execuția procedurilor *shell*
    - Demo*: un *script* “Helloworld”
    - Alte facilități
  - Variabile de *shell***
    - Definire (creare) și utilizare (substituție)
    - Instrucțiunea de atribuire
    - Alte forme de substituție
    - Variabile poziționale și alte variabile speciale
    - Comenzi interne utile
    - Demo*: un *script* “PrintArgs”
  - Expresii aritmetice și logice**
    - Comenzi pentru calcule aritmetice
    - Comenzi pentru expresii condiționale
  - Structuri de control alternative și repetitive**
    - Structura alternativă IF
    - Structura alternativă CASE
    - Structura repetitivă WHILE
    - Structura repetitivă UNTIL
    - Structura repetitivă FOR
    - Structura repetitivă SELECT
    - Alte comenzi interne utile
  - Funcții *shell***
    - Definire (declarare) și apelare (execuție)
    - Exemple de funcții
  - Referințe bibliografice**



# Introducere

## Introducere

Facilitățile limbajului de  
*scripting* `bash`

Variabile de *shell*

Expresii aritmetice și logice

Structuri de control alternative  
și repetitive

Funcții *shell*

Referințe bibliografice

În prima parte a acestei prezentări am discutat despre:

### ■ Comenzi simple

- Modul de execuție a comenzilor simple
- Execuția comenzilor simple în *background*
- Redirecțări I/O
- Valoarea de exit (codul de terminare)

### ■ Comenzi compuse

- Lanțuri de comenzi simple
- Execuția secvențială a mai multor comenzi
- Execuția paralelă, neînălțuită a mai multor comenzi
- Execuția condițională a mai multor comenzi
- Sintaxa extinsă: liste de comenzi și comenzi compuse

### ■ Specificarea numelor de fișiere

- Specificarea fișierelor individuale
- Șabloane pentru specificarea multiplă de fișiere

### ■ Fișierele de configurare a interpretorului de comenzi

- Inițializarea sesiunilor interactive ale *shell*-ului
- Istoricul comenzilor executate

Acum vom continua prezentarea cu subiectele sumarizate pe *slide*-ul anterior.



## Introducere (cont.)

### Introducere

Facilitățile limbajului de  
*scripting* `bash`

Variabile de *shell*

Expresii aritmetice și logice

Structuri de control alternative  
și repetitive

Funcții *shell*

Referințe bibliografice

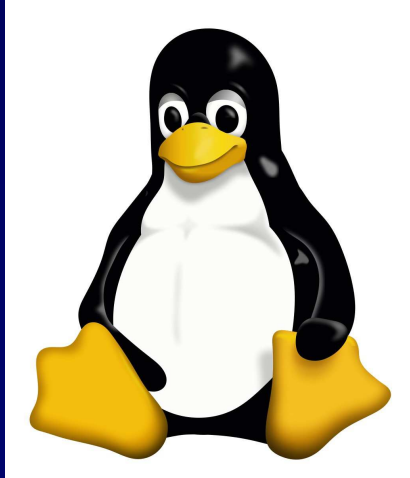
Într-un sistem UNIX, **interpretorul de comenzi** este un program executabil ce oferă două funcționalități de bază:

- preia comenzile introduse de utilizator, le interpretează și le execută, realizând astfel interfața dintre utilizator și sistemul de operare;
- **oferă facilități de programare într-un limbaj de comandă specific**, cu ajutorul căruia se pot scrie *script*-uri, *i.e.* fișiere text ce conțin secvențe de comenzi UNIX.  
(*Important: limbajul de scripting permite automatizarea lucrului la linia de comandă în acel shell.*)

Reamintesc faptul că în sistemele de operare din familia UNIX avem la dispoziție mai multe *shell*-uri: `sh` (*Bourne SHell*), `bash` (*Bourne Again SHell*), `csh` (*C SHell*), `ksh` (*Korn SHell*), `ash`, `zsh`, ș.a.

Referitor la cea de-a doua funcționalitate de mai sus (*i.e.*, aceea de **limbaj de scripting**), *shell*-urile disponibile în UNIX posedă toate facilitățile specifice oricărui limbaj de programare de nivel înalt: variabile, instrucțiuni de atribuire, instrucțiuni de control structurate – alternative și repetitive (de genul `if`, `case`, `while`, `for`, ș.a.), proceduri și funcții, parametri de apel, etc. Sintaxa acestor facilități este specifică fiecărui *shell*, fiind astfel o caracteristică prin care se diferențiază între ele *shell*-urile UNIX.

Vom prezenta în continuare toate aceste facilități, cu referiri explicite la sintaxa utilizată de limbajul de *scripting* oferit de interpretorul de comenzi `bash`.



# Agenda

Introducere

Facilitățile limbajului de *scripting* bash

Proceduri *shell* – *script*-uri

Execuția procedurilor *shell*

*Demo*: un *script* “Helloworld”

Alte facilități

Variabile de *shell*

Expresii aritmetice și logice

Structuri de control alternative și repetitive

Funcții *shell*

Referințe bibliografice

Introducere

**Facilitățile limbajului de *scripting* bash**

Proceduri *shell* – *script*-uri

Execuția procedurilor *shell*

*Demo*: un *script* “Helloworld”

Alte facilități

**Variabile de *shell***

Definire (creare) și utilizare (substituție)

Instrucțiunea de atribuire

Alte forme de substituție

Variabile poziționale și alte variabile speciale

Comenzi interne utile

*Demo*: un *script* “PrintArgs”

**Expresii aritmetice și logice**

Comenzi pentru calcule aritmetice

Comenzi pentru expresii condiționale

**Structuri de control alternative și repetitive**

Structura alternativă IF

Structura alternativă CASE

Structura repetitivă WHILE

Structura repetitivă UNTIL

Structura repetitivă FOR

Structura repetitivă SELECT

Alte comenzi interne utile

**Funcții *shell***

Definire (declarare) și apelare (execuție)

Exemple de funcții

**Referințe bibliografice**





## Proceduri *shell* – *script*-uri

Introducere

Facilitățile limbajului de  
*scripting* *bash*

Proceduri *shell* – *script*-uri

Execuția procedurilor *shell*

Demo: un *script* “Helloworld”

Alte facilități

Variabile de *shell*

Expresii aritmetice și logice

Structuri de control alternative  
și repetitive

Funcții *shell*

Referințe bibliografice

O **procedură *shell***, denumită și ***script*** (de la termenul în engleză), este practic un fișier text ce conține secvențe de comenzi UNIX, simple și/sau compuse.

*Notă:* și în MS-DOS sau Windows avem un concept similar – fișierele batch \*.bat.

Pentru a indica un comentariu într-un *script*, se utilizează caracterul '#' urmat de un text (până la finalul acelei linii de text).

Accesarea în corpul unei proceduri (*i.e.*, în acel fișier *script*) a argumentelor de apel a acesteia, se realizează cu ajutorul unor variabile speciale (ce pot fi doar citite, nu și modificate), având numele: \$1, \$2, . . . , \$9, \${10}, \${11}, . . .

\* \* \*

**Apelul unui *script*** înseamnă invocarea numelui său, folosind oricare dintre formele sintactice permise pentru comenzi externe, prezentate deja în prima parte: formele de apel al unei comenzi simple în *foreground* sau în *background*, forma de apel al unui *pipeline* (*i.e.*, lanț de comenzi) și respectiv celelalte forme de specificare a comenzilor compuse.



## Execuția procedurilor *shell*

Introducere

Facilitățile limbajului de  
*scripting* *bash*

Proceduri *shell* – *script*-uri

Execuția procedurilor *shell*

Demo: un *script* “Helloworld”

Alte facilități

Variabile de *shell*

Expresii aritmetice și logice

Structuri de control alternative  
și repetitive

Funcții *shell*

Referințe bibliografice

Astfel, pentru lansarea unui *script* în execuție în *foreground*, se poate utiliza oricare dintre cele trei forme de apel amintite deja, cu următoarele particularități de execuție:

```
UNIX> nume_script [opțiuni] [argumente] [redirectări I/O]
```

*Efect*: se creează un nou proces ce rulează neinteractiv *shell*-ul specificat pe prima linie a *script*-ului, prin construcția: ***#!nume\_shell*** (sau *shell*-ul curent, dacă nu este specificat vreun *shell* pe prima linie), iar acesta va executa linie cu linie comenzile din acel fișier (ca și cum ar fi fost introduse la prompterul său).

```
UNIX> nume_shell nume_script [parametri și redirectări I/O]
```

*Efect*: la fel ca mai sus, procesul *shell* nou creat fiind cel specificat la prompter.

```
UNIX> source nume_script [parametri și redirectări I/O]
```

```
sau UNIX> . nume_script [parametri și redirectări I/O]
```

*Efect*: nu se mai creează un nou proces *shell*, ci însuși procesul *shell* curent va executa linie cu linie comenzile din acel fișier.



## Execuția procedurilor *shell* (cont.)

Introducere

Facilitățile limbajului de  
*scripting* *bash*

Proceduri *shell* – *script-uri*

Execuția procedurilor *shell*

Demo: un *script* “Helloworld”

Alte facilități

Variabile de *shell*

Expresii aritmetice și logice

Structuri de control alternative  
și repetitive

Funcții *shell*

Referințe bibliografice

Iar pentru lansarea unui *script* în execuție în *background*, se adaugă caracterul ‘&’ la finalul oricăreia dintre cele trei forme de lansare în execuție specificate anterior:

```
UNIX> nume_script [parametri și redirectări I/O] &
```

```
UNIX> nume_shell nume_script [parametri și redirectări I/O] &
```

```
UNIX> source nume_script [parametri și redirectări I/O] &
```

```
sau UNIX> . nume_script [parametri și redirectări I/O] &
```

*Efect*: *script*-ul se execută în maniera descrisă la cele trei forme de apel pentru execuția în *foreground*, doar că de data aceasta interpretorul curent nu mai așteaptă terminarea execuției aceluia *script*, ci reafișează imediat prompterul, oferind astfel utilizatorului posibilitatea să introducă o nouă comandă pentru a fi executată **înainte** de a se termina execuția *script*-ului respectiv (*i.e.*, noua comandă se va executa în paralel cu acel *script*).





# Demo: un script “Helloworld”

- Introducere
- Facilitățile limbajului de *scripting* bash
- Proceduri *shell* – *script*-uri
- Execuția procedurilor *shell*
- Demo: un *script* “Helloworld”
- Alte facilități
- Variabile de *shell*
- Expresii aritmetice și logice
- Structuri de control alternative și repetitive
- Funcții *shell*
- Referințe bibliografice

Iată un exemplu, numit `Hello.sh`, cu care vom ilustra modul de execuție prin cele trei forme de apel:

```
#!/bin/bash
echo -e Hello, $1! \\n "Lista proceselor active in sesiunea de lucru curenta:"
ps -f
```

UNIX> `./Hello.sh world`

```
Hello, world!
  Lista proceselor active in sesiunea de lucru curenta:
UID          PID    PPID  C STIME TTY          TIME CMD
vidrascu  24249   24248  0  13:08 pts/4        00:00:00 -bash
vidrascu  24286   24249  0  13:08 pts/4        00:00:00 /bin/bash ./Hello.sh world
vidrascu  24287   24286  0  13:08 pts/4        00:00:00 ps -f
```

*Notă:* la prima formă de apel, trebuie setat dreptul de execuție (*i.e.*, `chmod u+x Hello.sh`) și specificat numele prin cale absolută sau relativă.

UNIX> `dash Hello.sh Cristian`

```
-e Hello, Cristian!
  Lista proceselor active in sesiunea de lucru curenta:
UID          PID    PPID  C STIME TTY          TIME CMD
vidrascu  24249   24248  0  13:08 pts/4        00:00:00 -bash
vidrascu  25002   24249  0  13:20 pts/4        00:00:00 dash Hello.sh Cristian
vidrascu  25003   25002  0  13:20 pts/4        00:00:00 ps -f
```

UNIX> `source Hello.sh`

```
Hello, !
  Lista proceselor active in sesiunea de lucru curenta:
UID          PID    PPID  C STIME TTY          TIME CMD
vidrascu  24249   24248  0  13:08 pts/4        00:00:00 -bash
vidrascu  25082   24249  0  13:21 pts/4        00:00:00 ps -f
```



## Alte facilități

[Introducere](#)

[Facilitățile limbajului de \*scripting\* `bash`](#)

[Proceduri \*shell\* – \*script\*-uri](#)

[Execuția procedurilor \*shell\*](#)

[Demo: un \*script\* “Helloworld”](#)

[Alte facilități](#)

[Variabile de \*shell\*](#)

[Expresii aritmetice și logice](#)

[Structuri de control alternative și repetitive](#)

[Funcții \*shell\*](#)

[Referințe bibliografice](#)

În secțiunile următoare vom prezenta o serie de facilități ale limbajului de *scripting* oferit de interpretorul de comenzi `bash`, precum ar fi:

- Variabile de *shell*
- Expresii aritmetice și logice
- Structuri de control alternative și repetitive
- Funcții *shell*

*Avertisment:* prezentarea acestor facilități va fi la un nivel simplificat. Pentru o descriere completă a tuturor acestor facilități vă recomand studierea documentației interpretorului de comenzi `bash` ([3]).



# Agenda

- Introducere
- Facilitățile limbajului de *scripting* bash
- Variabile de shell*
- Definire (creare) și utilizare (substituție)
- Instrucțiunea de atribuire
- Alte forme de substituție
- Variabile poziționale și alte variabile speciale
- Comenzi interne utile
- Demo: un script “PrintArgs”*
- Expresii aritmetice și logice
- Structuri de control alternative și repetitive
- Funcții *shell*
- Referințe bibliografice

- Introducere
- Facilitățile limbajului de *scripting* bash**
  - Proceduri *shell* – *script*-uri
  - Execuția procedurilor *shell*
  - Demo: un script “Helloworld”*
  - Alte facilități
- Variabile de shell***
  - Definire (creare) și utilizare (substituție)
  - Instrucțiunea de atribuire
  - Alte forme de substituție
  - Variabile poziționale și alte variabile speciale
  - Comenzi interne utile
  - Demo: un script “PrintArgs”*
- Expresii aritmetice și logice**
  - Comenzi pentru calcule aritmetice
  - Comenzi pentru expresii condiționale
- Structuri de control alternative și repetitive**
  - Structura alternativă IF
  - Structura alternativă CASE
  - Structura repetitivă WHILE
  - Structura repetitivă UNTIL
  - Structura repetitivă FOR
  - Structura repetitivă SELECT
  - Alte comenzi interne utile
- Funcții *shell***
  - Definire (declarare) și apelare (execuție)
  - Exemple de funcții



## Definire (creare) și utilizare (substituție)

Introducere

Facilitățile limbajului de  
*scripting* `bash`

Variabile de *shell*

Definire (creare) și  
utilizare (substituție)

Instrucțiunea de atribuire

Alte forme de substituție

Variabile poziționale și alte  
variabile speciale

Comenzi interne utile

Demo: un *script* "PrintArgs"

Expresii aritmetice și logice

Structuri de control alternative  
și repetitive

Funcții *shell*

Referințe bibliografice

O facilitate comună tuturor interpretoarelor de comenzi disponibile în sistemele UNIX este utilizarea de *variabile*, cu mențiunea că variabilele au, implicit, valori de tipul *șir de caractere*. Ele mai sunt numite și *parametrii shell*-ului respectiv.

Variabilele sunt păstrate într-o zonă de memorie a procesului *shell* în care sunt definite (*i.e.*, create), sub formă de perechi *nume=valoare*.

O variabilă este definită (*i.e.*, creată) în momentul execuției primei *instrucțiuni de atribuire* ce o implică (*i.e.*, în care apare în partea stângă a semnului de atribuire) sau a primei comenzi `read` ce o specifică drept argument.

*Remarcă:* așadar, variabilele *shell* nu trebuie declarate în avans (*i.e.*, la începutul procedurii sau funcției în care sunt utilizate), precum în alte limbaje de programare de nivel înalt (*e.g.*, C/C++).

*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 în care apare.

*Remarcă:* și interpretoarele de comenzi din MS-DOS sau Windows au un concept similar de variabile.





# Instrucțiunea de atribuire

Introducere

Facilitățile limbajului de  
*scripting* `bash`

Variabile de *shell*

Definire (creare) și  
utilizare (substituție)

Instrucțiunea de atribuire

Alte forme de substituție

Variabile poziționale și alte  
variabile speciale

Comenzi interne utile

*Demo*: un *script* “PrintArgs”

Expresii aritmetice și logice

Structuri de control alternative  
și repetitive

Funcții *shell*

Referințe bibliografice

*Instrucțiunea de atribuire* este comanda internă cu 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 (poate fi inclusiv șirul vid).

**Atenție:** caracterul ‘=’ nu trebuie să fie precedat ori urmat de spații albe!

Câteva exemple de atribuiri și de expresii în care se utilizează valorile unor variabile:

```
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= # v este “distrusă” (i.e., primește ca valoare “șirul vid”).
```

**Observație:** se folosește sintaxa *\${var}suffix* pentru a indica substituția dorită atunci când numele variabilei este urmat imediat, în acea expresie, de alte caractere ce pot face parte dintr-un identificator.





## Alte forme de substituție

Introducere

Facilitățile limbajului de  
*scripting* `bash`

Variabile de *shell*

Definire (creare) și  
utilizare (substituție)

Instrucțiunea de atribuire

Alte forme de substituție

Variabile poziționale și alte  
variabile speciale

Comenzi interne utile

Demo: un *script* "PrintArgs"

Expresii aritmetice și logice

Structuri de control alternative  
și repetitive

Funcții *shell*

Referințe bibliografice

Alte forme de substituții / expresii ce implică variabile :

- *Use default values:*  $\${var:-sir}$

*Efect:* rezultatul expresiei este valoarea variabilei  $var$ , dacă aceasta este definită, altfel este valoarea  $sir$  (iar dacă  $sir$  lipsește, atunci se afișează un mesaj standard de eroare ce spune că variabila este nedefinită).

- *Assign default values:*  $\${var:=sir}$

*Efect:* rezultatul expresiei este valoarea variabilei  $var$ , după ce eventual acesteia i se asignează valoarea  $sir$  (asignarea având loc doar dacă  $var$  era nedefinită).

- *Display error if null or unset:*  $\${var:?sir}$

*Efect:* rezultatul expresiei este valoarea variabilei  $var$ , dacă aceasta e definită, altfel se afișează  $sir$  (sau un mesaj standard de eroare, dacă  $sir$  lipsește).

- *Use alternate value:*  $\${var:+sir}$

*Efect:* dacă  $var$  e deja definită, atunci i se asignează valoarea  $sir$ , altfel rămâne fără valoare. Așadar, asignarea are loc doar în cazul în care  $var$  era deja definită.



## Alte forme de substituție (cont.)

Introducere

Facilitățile limbajului de  
*scripting* `bash`

Variabile de *shell*

Definire (creare) și  
utilizare (substituție)

Instrucțiunea de atribuire

Alte forme de substituție

Variabile poziționale și alte  
variabile speciale

Comenzi interne utile

Demo: un *script* "PrintArgs"

Expresii aritmetice și logice

Structuri de control alternative  
și repetitive

Funcții *shell*

Referințe bibliografice

Alte forme de substituții / expresii ce implică variabile (cont.) :

- *Parameter length*: o expresie de forma `${#var}`  
Efectul este de a fi substituită cu lungimea cuvântului / valorii variabilei *var*.
- *Substring expansion*: `${var:start:length}` sau `${var:start}`  
Efectul este de a fi substituită cu subcuvântul, din valoarea variabilei *var*, ce începe de la poziția *start* și de lungime *length* (respectiv, până la finalul cuvântului, în cazul celei de-a doua forme).

Iată câteva exemple cu astfel de expresii / substituții:

```
UNIX> word=12345 # Variabila word primește valoarea "12345".
UNIX> echo $word # Se afișează pe ecran textul "12345".
UNIX> echo ${#word} # Se afișează pe ecran textul "5".
UNIX> echo ${word:0:2} # Se afișează pe ecran textul "12".
UNIX> echo ${word:1:2} # Se afișează pe ecran textul "23".
UNIX> echo ${word:2} # Se afișează pe ecran textul "345".
```



## Alte forme de substituție (cont.)

Introducere

Facilitățile limbajului de  
*scripting* `bash`

Variabile de *shell*

Definire (creare) și  
utilizare (substituție)

Instrucțiunea de atribuire

Alte forme de substituție

Variabile poziționale și alte  
variabile speciale

Comenzi interne utile

Demo: un *script* "PrintArgs"

Expresii aritmetice și logice

Structuri de control alternative  
și repetitive

Funcții *shell*

Referințe bibliografice

- *Remove matching prefix pattern:* `${var#word}` sau `${var##word}`  
*Efect:* va fi substituită cu valoarea variabilei *var*, din care se elimină cel mai scurt, respectiv lung, prefix al acesteia, egal cu valoarea expresiei *word*.
- *Remove matching suffix pattern:* `${var%word}` sau `${var%%word}`  
*Efect:* va fi substituită cu valoarea variabilei *var*, din care se elimină cel mai scurt, respectiv lung, sufix al acesteia, egal cu valoarea expresiei *word*.

Iată câteva exemple cu astfel de expresii / substituții:

```
UNIX> v=AbcAbcAbcDEF                # Se inițializează variabila v, cu valoarea "AbcAbcAbcDEF".
UNIX> echo ${v#A*c}                   # Se afișează pe ecran textul "AbcAbcDEF".
UNIX> echo ${v##A*c}                  # Se afișează pe ecran textul "DEF".
UNIX> fisier=/thor/profs/vidrascu/subdir/sursa.c  # Se inițializează variabila fisier.
UNIX> echo ${fisier#/thor/profs/vidrascu/}        # Se afișează textul "subdir/sursa.c".
UNIX> echo ${fisier##*/}                  # Se afișează pe ecran textul "sursa.c".
UNIX> echo ${fisier%.c}.txt              # Se afișează textul "/thor/profs/vidrascu/subdir/sursa.txt".
```

Mai putem utiliza și comenzile externe `dirname` și `basename` ([8]), astfel :

```
UNIX> dirname $fisier                # Se afișează pe ecran textul "/thor/profs/vidrascu/subdir".
UNIX> basename $fisier .c            # Se afișează pe ecran textul "sursa".
```





## Alte forme de substituție (cont.)

Introducere

Facilitățile limbajului de  
*scripting* `bash`

Variabile de *shell*

Definire (creare) și  
utilizare (substituție)

Instrucțiunea de atribuire

Alte forme de substituție

Variabile poziționale și alte  
variabile speciale

Comenzi interne utile

Demo: un *script* "PrintArgs"

Expresii aritmetice și logice

Structuri de control alternative  
și repetitive

Funcții *shell*

Referințe bibliografice

- *Command substitution*: o substituție specială este expresia `$(comanda)` sau, echivalent, ``comanda``.  
Efectul este de a fi substituită, în linia de comandă sau în contextul în care apare, cu textul afișat pe ieșirea normală standard prin execuția comenzii specificate; aceasta este executată într-un *subshell* (*i.e.*, se creează un proces fiu al instanței curente de *shell*, ce va rula o altă instanță a *shell*-ului respectiv).
- *Arithmetic expansion*: o altă substituție specială este `$((expression))`.  
Efectul este de a fi substituită cu valoarea calculată a acelei expresii aritmetice.

*Word splitting*: în urma interpretării liniei de comandă (*i.e.*, după ce se evaluează substituțiile de variabile, de comenzi și cele aritmetice descrise anterior), instanța curentă de *shell* "parsează" rezultatul acestor substituții, împărțindu-l în cuvinte folosind ca și separatori caracterele `<space>`, `<tab>` și `<newline>`.

*Pathname expansion*: după împărțirea în cuvinte, urmează substituțiile șabloanelor pentru specificarea multiplă de fișiere, descrise în prima parte a acestei prezentări.



# Variabile poziționale și alte variabile speciale

- Introducere
- Facilitățile limbajului de *scripting* `bash`
- Variabile de *shell*
  - Definire (creare) și utilizare (substituție)
  - Instrucțiunea de atribuire
  - Alte forme de substituție
  - Variabile poziționale și alte variabile speciale
  - Comenzi interne utile
  - Demo: un *script* "PrintArgs"
- Expresii aritmetice și logice
- Structuri de control alternative și repetitive
- Funcții *shell*
- Referințe bibliografice

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

- `$1, $2, . . . , $9, ${10}, ${11}, . . .`      Semnificația: parametrii poziționali cu care a fost apelat procesul curent (*i.e.*, parametrii din linia de apel în cazul unui *script*).
  - `$0`      Semnificația: numele procesului curent (*i.e.*, al *script*-ului în care este referită).
  - `$#`      Semnificația: numărul parametrilor poziționali din linia de apel (fără argumentul `$0`).
  - `$*`      Semnificația: lista parametrilor poziționali din linia de comandă (fără argumentul `$0`).
  - `@`      Semnificația: lista parametrilor poziționali 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.
- `$$`      Semnificația: PID-ul procesului *shell* curent (*i.e.*, instanța *shell*-ului ce execută acel *script*).
  - `$?`      Semnificația: codul de terminare returnat de ultimul *pipeline* executat în *foreground*.
  - `!`      Semnificația: PID-ul ultimului proces executat în *background*.
  - `-`      Semnificația: atributele cu care a fost lansat procesul *shell* respectiv.
- Observație:* aceste atribute (*i.e.*, opțiuni de execuție) pot fi manevrate cu comanda internă `set`.

*Shell*-ul `bash` posedă și o serie de variabile de mediu *predefinite* (prin fișierele de inițializare) :  
`$HOME` , `$USER` , `$LOGNAME` , `$SHELL` , `$MAIL` , `$PS1` , `$PS2` , `$TERM` , `$PATH` , `$CDPATH` , `$IFS` , s.a.  
18 / 38





# Comenzi interne utile

Introducere

Facilitățile limbajului de  
*scripting* `bash`

Variabile de *shell*

Definire (creare) și  
utilizare (substituție)

Instrucțiunea de atribuire

Alte forme de substituție

Variabile poziționale și alte  
variabile speciale

Comenzi interne utile

Demo: un *script* "PrintArgs"

Expresii aritmetice și logice

Structuri de control alternative  
și repetitive

Funcții *shell*

Referințe bibliografice

Iată câteva comenzi interne ale interpretorului `bash`, utile pentru lucrul cu variabile :

## ■ Comanda de *citire* :

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

Are ca efect citirea, de la intrarea standard `stdin`, de valori și atribuirea lor variabilelor specificate.

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

## ■ Comanda de *declarare read-only* :

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

Are ca efect declararea variabilelor specificate ca fiind *read-only* (*i.e.*, ele nu mai pot fi modificate după execuția comenzii, ci rămân cu valorile pe care le aveau când s-a executat această comandă).

## ■ Comanda de *exportare* :

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

Are ca efect "exportul" variabilelor specificate în toate procesele fii ale procesului *shell* respectiv (în mod obișnuit, variabilele nu sunt "vizibile" în procesele fii, ele fiind locale procesului *shell* respectiv, fiind păstrate în memoria acestuia). Mai putem utiliza și combinația:

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

Are ca efect atribuirea și exportul variabilei printr-o singură comandă.

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



## Comenzi interne utile (cont.)

Introducere

Facilitățile limbajului de  
*scripting* `bash`

Variabile de *shell*

Definire (creare) și  
utilizare (substituție)

Instrucțiunea de atribuire

Alte forme de substituție

Variabile poziționale și alte  
variabile speciale

Comenzi interne utile

Demo: un *script* "PrintArgs"

Expresii aritmetice și logice

Structuri de control alternative  
și repetitive

Funcții *shell*

Referințe bibliografice

- Comanda de *shift*-are, cu sintaxa: `shift [n]`, unde  $n$  este 1, în caz că lipsește. Are ca efect "deplasarea" spre stânga cu  $n$  poziții a tuturor parametrilor poziționali (e.g., pentru  $n = 1$ , în `$1` se copie valoarea lui `$2`, în `$2` se copie valoarea lui `$3`, ș.a.m.d.), iar variabilele `$#`, `$@` și `$*` se actualizează în mod corespunzător. Este utilă, de pildă, atunci când dorim să procesăm secvențial argumentele de apel a procedurii respective (a se vedea exemplul [\[FirstScript\], iii\)](#), disponibil [aici](#)).
- Comanda de *eval*-uare, cu sintaxa: `eval parametri`. Efect: se evaluează parametrii specificați și se execută rezultatul evaluării (ca și cum ar fi fost introdus de utilizator la prompter).  
Un exemplu: `eval newvar=\$$varname`. Efect: valoarea variabilei `varname` este considerată drept identificator de variabilă, iar valoarea acestuia din urmă este atribuită variabilei `newvar` (i.e., obținem practic o referință indirectă). A se vedea și exemplul [\[FirstScript\], v\)](#), disponibil [aici](#)).
- Comanda `set` se utilizează pentru a seta (ori a reseta) attributele procesului *shell* curent.  
Un exemplu: `set -o noexec` sau `set -n`. Efect: citește comenzile, dar fără să le execute. Acest atribut este util pentru *analiza sintactică* – se poate verifica un script pentru erori de sintaxă.  
Alt exemplu: `set -o xtrace` sau `set -x`. Efect: pentru fiecare linie de comandă introdusă de utilizator, se afișează rezultatul interpretării acelei linii, înainte de a o executa efectiv. Acest atribut este util pentru depanare, ca să vedem exact ce se va executa, de către *shell*, în urma citirii și interpretării liniei de comandă respective.  
*Sfat*: adăugați comanda `set -x` în *script*-uri, la început, pentru a vă ajuta în etapa de scriere și depanare; iar apoi, când ați ajuns la forma dorită a *script*-ului la care lucrați, o puteți elimina.  
Pentru mai multe detalii, consultați [\[Exemplul #1 de script cu erori\]](#), disponibil [aici](#).



# Demo: un script “PrintArgs”

Un exemplu, numit PrintArgs.sh , ce ilustrează valorile parametrilor poziționali prin execuția sa:

```
#!/bin/bash
echo Positional parameters are: '$1'=$1, '$2'=$2, '$3'=$3, '$4'=$4, ...
echo The parameter '$0'=$0 , the parameter '$@'=$@ , and the parameter '$#'=$#
ps -o user,pid,ppid,cmd
```

UNIX> ./PrintArgs.sh hello world

```
Positional parameters are: $1=hello, $2=world, $3=, $4=, ...
The parameter $0=./PrintArgs.sh , the parameter $@=hello world , and the parameter $#=2
USER      PID    PPID  CMD
vidrascu  29472  29471  -bash
vidrascu  30212  29472  /bin/bash ./PrintArgs.sh hello world
vidrascu  30214  30212  ps -o user,pid,ppid,cmd
```

UNIX> dash PrintArgs.sh val1 val2 val3

```
Positional parameters are: $1=val1, $2=val2, $3=val3, $4=, ...
The parameter $0=PrintArgs.sh , the parameter $@=val1 val2 val3 , and the parameter $#=3
USER      PID    PPID  CMD
vidrascu  29472  29471  -bash
vidrascu  30351  29472  dash PrintArgs.sh val1 val2 val3
vidrascu  30352  30351  ps -o user,pid,ppid,cmd
```

UNIX> source PrintArgs.sh p1 p2 p3 p4 p5 p6

```
Positional parameters are: $1=p1, $2=p2, $3=p3, $4=p4, ...
The parameter $0=-bash , the parameter $@=p1 p2 p3 p4 p5 p6 , and the parameter $#=6
USER      PID    PPID  CMD
vidrascu  29472  29471  -bash
vidrascu  30409  29472  ps -o user,pid,ppid,cmd
```

- Introducere
- Facilitățile limbajului de scripting bash
- Variabile de shell
- Definire (creare) și utilizare (substituție)
- Instrucțiunea de atribuire
- Alte forme de substituție
- Variabile poziționale și alte variabile speciale
- Comenzi interne utile
- Demo: un script “PrintArgs”
- Expresii aritmetice și logice
- Structuri de control alternative și repetitive
- Funcții shell
- Referințe bibliografice



# Agenda

- Introducere
- Facilitățile limbajului de *scripting* bash
- Variabile de *shell*
- Expresii aritmetice și logice
- Comenzi pentru calcule aritmetice
- Comenzi pentru expresii condiționale
- Structuri de control alternative și repetitive
- Funcții *shell*
- Referințe bibliografice

- Introducere
- Facilitățile limbajului de *scripting* bash**
  - Proceduri *shell* – *script*-uri
  - Execuția procedurilor *shell*
  - Demo*: un *script* “Helloworld”
  - Alte facilități
- Variabile de *shell***
  - Definire (creare) și utilizare (substituție)
  - Instrucțiunea de atribuire
  - Alte forme de substituție
  - Variabile poziționale și alte variabile speciale
  - Comenzi interne utile
  - Demo*: un *script* “PrintArgs”
- Expresii aritmetice și logice**
  - Comenzi pentru calcule aritmetice
  - Comenzi pentru expresii condiționale
- Structuri de control alternative și repetitive**
  - Structura alternativă IF
  - Structura alternativă CASE
  - Structura repetitivă WHILE
  - Structura repetitivă UNTIL
  - Structura repetitivă FOR
  - Structura repetitivă SELECT
  - Alte comenzi interne utile
- Funcții *shell***
  - Definire (declarare) și apelare (execuție)
  - Exemple de funcții





# Comenzi pentru calcule aritmetice

Introducere

Facilitățile limbajului de  
*scripting* `bash`

Variabile de *shell*

Expresii aritmetice și logice

Comenzi pentru calcule  
aritmetice

Comenzi pentru expresii  
condiționale

Structuri de control alternative  
și repetitive

Funcții *shell*

Referințe bibliografice

*Expresiile aritmetice* se pot calcula cu comanda internă `let`, cu comenzile externe `expr` sau `bc` ([6]), ori cu substituțiile *arithmetic expansion*. Câteva exemple de calcule:

```
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 (i.e., 2 ridicat la puterea 3).
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.
```

A se vedea și exemplul [\[FirstScript\]](#), i)-v) din suportul de laborator, disponibil [aici](#).

O altă posibilitate este lucrul cu *variabile de tip întreg*: comanda `declare -i n` setează atributul “cu valori întregi” pentru variabila `n`.

Apoi se pot scrie expresii aritmetice în mod direct, fără a mai utiliza explicit comanda `let`. 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).
```

Putem lucra și cu *variabile de tip vector*: comanda `declare -a v` setează atributul “*array*” pentru variabila `v`. Referirea la un element al vectorului se face prin `v[i]`.

Pentru ilustrare, a se vedea exemplul [\[Iterative math #4\]](#) din suportul de laborator, disponibil [aici](#).





## Comenzi pentru calcule aritmetice (cont.)

Introducere

Facilitățile limbajului de  
*scripting* `bash`

Variabile de *shell*

Expresii aritmetice și logice

Comenzi pentru calcule  
aritmetice

Comenzi pentru expresii  
condiționale

Structuri de control alternative  
și repetitive

Funcții *shell*

Referințe bibliografice

Comanda `bc` ([6]) – un limbaj de programare pentru calcule în virgulă mobilă. Iată câteva exemple de calcule:

```
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  $\pi$ ,
    cu 10 zecimale (și nu cu 20 de zecimale!) ; a(1) este apelul funcției de bibliotecă  $\arctan(x)$  .
UNIX> echo "scale=5; sqrt(2)" | bc # Se afișează 1.41421, i.e.  $\sqrt{2}$  cu 5 zecimale .
UNIX> echo "scale=2; v = 1; v += 3/2; v+10" | bc # Se afișează valoarea 12.50 .
```

A se vedea și exemplul `[MyExpr]` din suportul de laborator, disponibil [aici](#) .

*Arithmetic expansion* folosind construcțiile `((...))` și `${(...)}` . Câteva exemple:

```
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?11:22 )) # Variabilei b i se atribuie valoarea expresiei condiționale, i.e. 11 .
UNIX> echo ${0xFFFF} # Se afișează 65535, i.e. valoarea acelui număr hexazecimal .
UNIX> echo ${4#1203} # Se afișează 99, i.e. valoarea numărului 1203 scris în baza 4 .
```



# Comenzi pentru expresii condiționale

Introducere

Facilitățile limbajului de  
*scripting* `bash`

Variabile de *shell*

Expresii aritmetice și logice

Comenzi pentru calcule  
aritmetice

Comenzi pentru expresii  
condiționale

Structuri de control alternative  
și repetitive

Funcții *shell*

Referințe bibliografice

Comanda internă de *evaluare a unei expresii condiționale* ([7]) are sintaxa următoare:

`test condiție` sau `[ condiție ]` ,

unde expresia condițională *condiție* poate fi de una dintre următoarele forme:

## ■ Operatori relaționali pe *șiruri de caractere*:

- `test -z string` # Adevărat dacă *string* are lungimea 0 .
- `test -n string` sau `test string` # Adevărat dacă *string* nu este șirul vid .
- `test string_1 = string_2` # Adevărat dacă cele două șiruri sunt egale .
- `test string_1 != string_2` # Adevărat dacă cele două șiruri nu sunt egale .
- `test string_1 < string_2` # Comparatie folosind ordinea lexicografică .
- `test string_1 > string_2` # Comparatie folosind ordinea lexicografică .

## ■ Operatori relaționali între două valori *numere întregi*:

- `test val_1 -eq val_2` # Adevărat dacă cele două valori întregi sunt egale .
- `test val_1 -ne val_2` # Adevărat dacă cele două valori întregi nu sunt egale .
- `test val_1 -gt val_2` # Testează inegalitatea “mai mare strict” .
- `test val_1 -ge val_2` # Testează inegalitatea “mai mare sau egal” .
- `test val_1 -lt val_2` # Testează inegalitatea “mai mic strict” .
- `test val_1 -le val_2` # Testează inegalitatea “mai mic sau egal” .



## Comenzi pentru expresii condiționale (cont.)

Introducere

Facilitățile limbajului de  
*scripting* `bash`

Variabile de *shell*

Expresii aritmetice și logice

Comenzi pentru calcule  
aritmetice

Comenzi pentru expresii  
condiționale

Structuri de control alternative  
și repetitive

Funcții *shell*

Referințe bibliografice

(cont.) Alte expresii condiționale utilizabile ca argumente ale comenzii interne `test`:

■ Teste referitoare la fișiere: `test -opt fișier`, opțiunea de testare `-opt` putând fi:

- `-e` : testează existența aceluși fișier (de orice tip) ;
- `-d` / `-f` / `-p` : testează dacă fișierul există și este de tip director / fișier obișnuit / *fifo* ;
- `-b` / `-c` : testează dacă fișierul există și este de tip dispozitiv în mod bloc / caracter ;
- `-r` / `-w` / `-x` : testează dacă utilizatorul curent poate citi / modifica / executa fișierul ;
- `-s` : testează dacă fișierul are conținut nevid ;
- ș.a. (a se vedea `help test` și `man 1 test`).

■ O expresie logică (negație, conjuncție, sau disjuncție de condiții):

- `test ! condiție_1` # Negația condiției `condiție_1` ;
- `test condiție_1 -a condiție_2` # Conjuncția celor două condiții ;
- `test condiție_1 -o condiție_2` # Disjuncția celor două condiții ,

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

Valoarea de exit a comenzii `test` este 0 dacă condiția testată este adevărată, sau 1 în caz contrar.

*Observație:* mai există și comanda compusă `[[ condiție ]]` disponibilă în interpretorul `bash`, cu aceleași condiții ca la `test`, dar cu anumite diferențe la execuție (e.g., `string_2` poate fi o expresie regulată și operatorii relaționali pe șiruri de caractere vor face *pattern matching* în această situație).



# Agenda

- Introducere
- Facilitățile limbajului de *scripting* bash
- Variabile de *shell*
- Expresii aritmetice și logice
- Structuri de control alternative și repetitive
- Structura alternativă IF
- Structura alternativă CASE
- Structura repetitivă WHILE
- Structura repetitivă UNTIL
- Structura repetitivă FOR
- Structura repetitivă SELECT
- Alte comenzi interne utile
- Funcții *shell*
- Referințe bibliografice

- Introducere
  - Facilitățile limbajului de *scripting* bash**
    - Proceduri *shell* – *script*-uri
    - Execuția procedurilor *shell*
    - Demo*: un *script* “Helloworld”
    - Alte facilități
  - Variabile de *shell***
    - Definire (creare) și utilizare (substituție)
    - Instrucțiunea de atribuire
    - Alte forme de substituție
    - Variabile poziționale și alte variabile speciale
    - Comenzi interne utile
    - Demo*: un *script* “PrintArgs”
  - Expresii aritmetice și logice**
    - Comenzi pentru calcule aritmetice
    - Comenzi pentru expresii condiționale
  - Structuri de control alternative și repetitive**
    - Structura alternativă IF
    - Structura alternativă CASE
    - Structura repetitivă WHILE
    - Structura repetitivă UNTIL
    - Structura repetitivă FOR
    - Structura repetitivă SELECT
    - Alte comenzi interne utile
  - Funcții *shell***
    - Definire (declarare) și apelare (execuție)
    - Exemple de funcții
- Referințe bibliografice**





# Structura alternativă IF

Introducere

Facilitățile limbajului de  
*scripting* bash

Variabile de *shell*

Expresii aritmetice și logice

Structuri de control alternative  
și repetitive

Structura alternativă IF

Structura alternativă CASE

Structura repetitivă WHILE

Structura repetitivă UNTIL

Structura repetitivă FOR

Structura repetitivă SELECT

Alte comenzi interne utile

Funcții *shell*

Referințe bibliografice

1) *Structura alternativă IF* este realizată de comanda internă **if**, având *sintaxa* următoare:

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

Sau, oricare dintre cele trei apariții ale caracterului ';' poate fi înlocuită cu <newline>. Spre exemplu:

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

*Semantica*: se execută mai întâi comenzile din *lista\_comenzi\_1* și, dacă valoarea de exit a acestei liste este 0 (*i.e.*, terminare cu succes), atunci se execută comenzile din *lista\_comenzi\_2*. Iar în caz contrar (dar numai dacă există și ramura **else**), se execută comenzile din *lista\_comenzi\_3*.

*Structura alternativă “Multiple IFs”* – putem avea mai multe **if**-uri imbricate (cu prescurtarea **elif**):

```
if lista_1; then lista_2; [elif lista_3; then lista_4; [elif ... [else lista_N;] ...]] fi
```

*Observație*: Valoarea de exit a comenzii **if** este valoarea de exit a ultimei comenzi executate, sau zero dacă nicio condiție nu a fost testată adevărată.





# Structura alternativă CASE

Introducere

Facilitățile limbajului de  
*scripting* bash

Variabile de *shell*

Expresii aritmetice și logice

Structuri de control alternative  
și repetitive

Structura alternativă IF

Structura alternativă CASE

Structura repetitivă WHILE

Structura repetitivă UNTIL

Structura repetitivă FOR

Structura repetitivă SELECT

Alte comenzi interne utile

Funcții *shell*

Referințe bibliografice

2) *Structura alternativă CASE* este realizată de comanda internă **case**, cu *sintaxa* următoare:

```
case expresie in
    lista_valori_1    ) lista_comenzi_1    ;;
    lista_valori_2    ) lista_comenzi_2    ;;
    . . . . .
    lista_valori_N-1 ) lista_comenzi_N-1 ;;
    lista_valori_N   ) lista_comenzi_N
esac
```

*Semantica*: se evaluează **expresie** și se “scanează” fiecare linie în căutarea primei linii ce o conține: dacă valoarea expresiei se regăsește în lista de valori **lista\_valori\_1**, atunci se execută **lista\_comenzi\_1** și apoi execuția comenzii **case** se încheie. Altfel, dacă valoarea expresiei se regăsește în lista de valori **lista\_valori\_2**, atunci se execută **lista\_comenzi\_2** și apoi execuția comenzii case se încheie. Altfel, ... ș.a.m.d.

*Observații*: i) Valoarea de exit a comenzii **case** este zero dacă nu se găsește nicio valoare pentru expresia căutată. În caz contrar, este valoarea de exit a ultimei comenzi executate din lista selectată.

ii) dacă se folosește “;**&**” în loc de “;”, atunci se vor executa listele de comenzi asociate tuturor șirurilor de valori ce conțin valoarea expresiei respective (*i.e.*, execuția comenzii **case** nu se mai încheie după găsirea primei linii ce conține valoarea expresiei).



# Structura repetitivă WHILE

Introducere

Facilitățile limbajului de  
*scripting* bash

Variabile de *shell*

Expresii aritmetice și logice

Structuri de control alternative  
și repetitive

Structura alternativă IF

Structura alternativă CASE

Structura repetitivă WHILE

Structura repetitivă UNTIL

Structura repetitivă FOR

Structura repetitivă SELECT

Alte comenzi interne utile

Funcții *shell*

Referințe bibliografice

3) *Buclo repetitivă WHILE* este realizată de comanda internă **while**, având *sintaxa* următoare:

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

Oricare dintre cele două apariții ale caracterului ';' poate fi înlocuită cu <newline>. Spre exemplu:

```
while lista_comenzi_1
do
    lista_comenzi_2
done
```

*Semantica*: la fiecare iterație a buclei, se execută comenzile din *lista\_comenzi\_1* și, dacă valoarea de exit a acestei liste (*i.e.*, codul de terminare al ultimei comenzi din această listă) este zero (*i.e.*, terminare cu succes), atunci se execută comenzile din *lista\_comenzi\_2* și se reia bucla. Altfel, se termină execuția buclei **while**.



# Structura repetitivă UNTIL

Introducere

Facilitățile limbajului de  
*scripting* bash

Variabile de *shell*

Expresii aritmetice și logice

Structuri de control alternative  
și repetitive

Structura alternativă IF

Structura alternativă CASE

Structura repetitivă WHILE

Structura repetitivă UNTIL

Structura repetitivă FOR

Structura repetitivă SELECT

Alte comenzi interne utile

Funcții *shell*

Referințe bibliografice

4) *Bucula repetitivă UNTIL* este realizată de comanda internă **until**, având *sintaxa* următoare:

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

Oricare dintre cele două apariții ale caracterului ';' poate fi înlocuită cu <newline>. Spre exemplu:

```
until lista_comenzi_1  
do  
    lista_comenzi_2  
done
```

*Semantica*: la fiecare iterație a buclei, se execută comenzile din *lista\_comenzi\_1* și, dacă valoarea de exit a acestei liste (*i.e.*, codul de terminare 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**.

*Observație* (valabilă pentru ambele comenzi **while** și **until**):

Valoarea de exit returnată este valoarea de exit a ultimei comenzi din *lista\_comenzi\_2* care este executată, sau zero dacă nicio comandă nu a fost executată.



# Structura repetitivă FOR

Introducere

Facilitățile limbajului de  
*scripting* bash

Variabile de *shell*

Expresii aritmetice și logice

Structuri de control alternative  
și repetitive

Structura alternativă IF

Structura alternativă CASE

Structura repetitivă WHILE

Structura repetitivă UNTIL

Structura repetitivă FOR

Structura repetitivă SELECT

Alte comenzi interne utile

Funcții *shell*

Referințe bibliografice

5) *Bucula iterativă FOR* este realizată de comanda internă **for**, având *sintaxa* următoare:

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

Oricare dintre cele două apariții ale caracterului ' ; ' poate fi înlocuită cu <newline>. Spre exemplu:

```
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*.

*Observație:* această formă a buclei **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. În acest scop se poate folosi comanda **seq**, astfel:

```
UNIX> for v in $(seq first increment last) ; do lista_comenzi ; done
```

Alternativ, se mai poate folosi și comanda **for** (( , adică a doua formă sintactică a comenzii **for** :

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

unde *exp1*, *exp2* și *exp3* sunt expresii aritmetice, cu aceleași semnificații ca în limbajul C/C++ .





# Structura repetitivă SELECT

6) *Buclo iterativă SELECT* este realizată de comanda internă `select`, având *sintaxa* următoare:

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

Oricare dintre cele două apariții ale caracterului `'` poate fi înlocuită cu `<newline>`. Spre exemplu:

```
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 *lista\_cuvinte* ce este selectat prin interacțiune cu utilizatorul. Execuția buclei `select` se încheie tot prin interacțiune cu utilizatorul (apăsând tastele `CTRL+D`, i.e. EOF).

*Observații* (valabile pentru ambele comenzi `select` și `for`):

- i) Valoarea de exit returnată este valoarea de exit a ultimei comenzi din *lista\_comenzi* care este executată, sau zero dacă nicio comandă nu a fost executată (e.g., dacă evaluarea elementelor din *lista\_cuvinte* are ca rezultat o listă goală, nu se execută nicio comandă).
- ii) Dacă lipsește partea opțională *in lista\_cuvinte* , atunci drept listă de cuvinte se prelucrează valoarea variabilei speciale `$@` .

- Introducere
- Facilitățile limbajului de *scripting* `bash`
- Variabile de *shell*
- Expresii aritmetice și logice
- Structuri de control alternative și repetitive
- Structura alternativă IF
- Structura alternativă CASE
- Structura repetitivă WHILE
- Structura repetitivă UNTIL
- Structura repetitivă FOR
- Structura repetitivă **SELECT**
- Alte comenzi interne utile
- Funcții *shell*
- Referințe bibliografice



# Alte comenzi interne utile

Iată alte câteva comenzi interne, utile atât în *script*-uri, cât și la linia de comandă a interpretorului `bash` :

- 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: în cazul  $n = 1$  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`.
- Comanda `exit`, cu sintaxa: `exit [cod]` , unde `cod` este valoarea variabilei speciale `$?`, în caz că lipsește. Efect: se încheie execuția *script*-ului (sau instanței de *shell*) în care este apelată, iar codul său de terminare (*i.e.*, valoarea de `exit`) va fi valoarea specificată.
- Comanda `wait`, cu sintaxa: `wait [pid]` . Efect: se suspendă execuția *script*-ului (sau instanței de *shell*) în care este apelată, așteptându-se terminarea procesului având PID-ul specificat.
- Comanda `exec`, cu sintaxa: `exec comandă` . Efect: se execută comanda specificată, fără a se crea un nou proces în acest scop. Astfel, *shell*-ul ce execută această comandă se va “reacoperi” cu procesul asociat comenzii (deci nu este *reentrant*).
- Comanda `trap`, cu sintaxa: `trap comandă eveniment` . Efect: se va executa comanda specificată atunci când se va produce acel eveniment (*i.e.*, când se va primi semnalul respectiv).  
Exemplu: `UNIX> trap 'rm /tmp/ps$$ ; exit' 2`  
Notă: 2 = *interrupt signal* (semnalul generat prin apăsarea tastelor `CTRL+C`), 3 = *quit signal* (semnalul generat prin apăsarea tastelor `CTRL+\`), ș.a.

- Introducere
- Facilitățile limbajului de *scripting* `bash`
- Variabile de *shell*
- Expresii aritmetice și logice
- Structuri de control alternative și repetitive
- Structura alternativă IF
- Structura alternativă CASE
- Structura repetitivă WHILE
- Structura repetitivă UNTIL
- Structura repetitivă FOR
- Structura repetitivă SELECT
- Alte comenzi interne utile
- Funcții *shell*
- Referințe bibliografice



# Agenda

- Introducere
- Facilitățile limbajului de *scripting* bash
- Variable de *shell*
- Expresii aritmetice și logice
- Structuri de control alternative și repetitive
- Funcții *shell*
- Definire (declarare) și apelare (execuție)
- Exemple de funcții
- Referințe bibliografice

- Introducere
- Facilitățile limbajului de *scripting* bash**
  - Proceduri *shell* – *script*-uri
  - Execuția procedurilor *shell*
  - Demo*: un *script* “Helloworld”
  - Alte facilități
- Variable de *shell***
  - Definire (creare) și utilizare (substituție)
  - Instrucțiunea de atribuire
  - Alte forme de substituție
  - Variable poziționale și alte variabile speciale
  - Comenzi interne utile
  - Demo*: un *script* “PrintArgs”
- Expresii aritmetice și logice**
  - Comenzi pentru calcule aritmetice
  - Comenzi pentru expresii condiționale
- Structuri de control alternative și repetitive**
  - Structura alternativă IF
  - Structura alternativă CASE
  - Structura repetitivă WHILE
  - Structura repetitivă UNTIL
  - Structura repetitivă FOR
  - Structura repetitivă SELECT
  - Alte comenzi interne utile
- Funcții *shell***
  - Definire (declarare) și apelare (execuție)
  - Exemple de funcții
- Referințe bibliografice**





# Definire (declarare) și apelare (execuție)

Introducere

Facilitățile limbajului de *scripting* `bash`

Variabile de *shell*

Expresii aritmetice și logice

Structuri de control alternative și repetitive

Funcții *shell*

Definire (declarare) și apelare (execuție)

Exemple de funcții

Referințe bibliografice

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*) fie într-o procedură *shell*, fie direct la promptul unei instanțe de *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`.

**Apelul unei funcții** înseamnă *execuția acelei secvențe de comenzi* și se face similar ca pentru orice comandă simplă, adică prin numele său, plus parametri și eventuale redirectări I/O.

La fel ca la proceduri, în corpul unei funcții (*i.e.*, în `lista_comenzi`) folosim variabilele poziționale `$1`, `$2`, ..., `$9`, `${10}`, `${11}`, ... pentru a ne referi la parametrii de apel ai funcției, iar prin variabilele speciale `$#`, respectiv `$*` și `$@`, ne referim la numărul, respectiv lista, tuturor parametrilor de apel ai acelei funcții.

*Observații*: i) în declarația unei funcții, fie `function`, fie `()` pot fi omise, dar nu simultan amândouă!  
ii) între perechea de paranteze `()` nu se scrie niciodată nimic, chiar dacă dorim să declarăm funcția ca având unul sau mai multe argumente!  
iii) în concluzie, conceptul de *funcție* de la *shell*-uri NU este similar, d.p.d.v. sintactic și semantic, cu cel de funcție din limbaje de programare precum C/C++.





# Exemple de funcții

- Introducere
- Facilitățile limbajului de *scripting* `bash`
- Variabile de *shell*
- Expresii aritmetice și logice
- Structuri de control alternative și repetitive
- Funcții *shell*
- Definire (declarare) și apelare (execuție)
- Exemple de funcții
- Referințe bibliografice

1) Iată un exemplu de funcție, declarată și apelată direct la linia de comandă a *shell*-ului:

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

Efect: se va afișa conținutul directorului specificat ca argument al funcției.

2) Iată un alt exemplu de funcție, declarată și apelată în interiorul unui *script*:

```
#!/bin/bash  
function count_params ()  
{  
    echo "Apel cu $# argument(e) : $*"  
}  
count_params "$*"      # Primul apel al functiei.  
count_params "$@"      # Al doilea apel al functiei.
```

Dacă apelăm acest script cu următoarea linie de comandă, vom obține ca output:

```
UNIX> ./script.sh a b c
```

```
Apel cu 1 argument(e) : a b c  
Apel cu 3 argument(e) : a b c
```

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

3) Alte exemple: a se vedea [\[MyGccOrCat\]](#) și [\[Recursive math #1 & #2\]](#) în suportul de laborator disponibil [aici](#).



# Bibliografie obligatorie

Introducere

Facilitățile limbajului de  
*scripting* `bash`

Variabile de *shell*

Expresii aritmetice și logice

Structuri de control alternative  
și repetitive

Funcții *shell*

Referințe bibliografice

[1] Cap. 2, §2.4 din cartea “*Sisteme de operare – manual pentru ID*”, autor C. Vidrașcu, editura UAIC, 2006. *Notă*: este accesibilă, în format PDF, din pagina disciplinei “Sisteme de operare”:

- <https://edu.info.uaic.ro/sisteme-de-operare/SO/books/ManualID-SO.pdf>

[2] Suportul de laborator online asociat acestei prezentări:

- [https://edu.info.uaic.ro/sisteme-de-operare/SO/support-lessons/bash/suport\\_lab4.html](https://edu.info.uaic.ro/sisteme-de-operare/SO/support-lessons/bash/suport_lab4.html)

- [https://edu.info.uaic.ro/sisteme-de-operare/SO/support-lessons/bash/suport\\_lab5.html](https://edu.info.uaic.ro/sisteme-de-operare/SO/support-lessons/bash/suport_lab5.html)

## Bibliografie suplimentară:

[3] Documentația interpretorului de comenzi `bash` : `man 1 bash` și “GNU Bash manual”

[4] *Linux Documentation Project Guides* → “Advanced Bash-Scripting Guide”

[5] Cartea “Bash Pocket Reference” (1st edition), by A.Robbins, O’Reilly Media Inc., 2010.

[6] `help let` , `man 1 expr` și `man 1 bc` / `man 1p bc`

[7] `help test` și `man 1 test`

[8] `man 1 basename` și `man 1 dirname`