



Sisteme de Operare 1 - Curs 2

Curs tinut in 2012-2013 de catre lector dr. Sanda-Maria Dragos

<h2>Shell si programare in Shell</h2> <p>Shell-ul este un program special care furnizeaza o interfata intre nucleu sistemului de operare Unix (kernel) si utilizator. Din aceasta perspectiva, un shell poate fi privit ca:</p> <ul style="list-style-type: none">» <i>limbaj de comanda</i> - In momentul in care un utilizator isi deschide o sesiune de lucru, un shell se instaleaza ca interpretor de comenzi.» <i>limbaj de programare</i>, ce are ca element de baza <i>comanda</i> Unix. Shell-urile dispun de conceptele de variabila, constanta, expresie, structuri de control si subprogram.	<div>slide 2</div> <div></div>
<h2>Interpretoare de comenzi shell</h2> <p>Primele shelluri (primul in 1978): simple si restrictive ca si restul interpretoarelor de comenzi disponibile pe celelalte sisteme de operare.</p> <ul style="list-style-type: none">» <u><i>Bourne Shell sh</i></u>: (1977). Primul shell care a revolutionat produsele din aceasta categorie a fost proiectat de Steve R. Bourne de la Bell Laboratories. Este shell-ul implicit pe majoritatea versiunilor de Unix.» <u><i>C Shell csh</i></u>: (1978) proiectat de catre Bill Joy de la Universitatea Berkeley. Extinde <i>sh</i> prin mecanismul de retinere a istoricului ultimelor comenzi introduse de utilizator.» <u><i>Korn Shell ksh</i></u>: (early 1980) dezvoltat de David Korn de la Bell Laboratories, este o imbunatatire a <i>sh</i>, dar are si caracteristici din <i>csh</i> (de ex. istoricul ultimelor comenzi).» <u><i>Z Shell zsh</i></u>: (1990) scris de Paul Falstad de la Universitatea Princeton. E o extensie a <i>sh</i>.» <u><i>Restricted Shell rsh</i></u>: Versiune a <i>sh</i> care contine urmatoarele restrictii. Nu permite utilizatorului:	<div>slide 3</div> <div></div>

- » sa schimbe directorul (cd)
- » sa modifice valoarea variabilei de mediu PATH
- » specificarea absoluta sau relativa a unei comenzi (numele comenzii a nu continua /)
- » redirectarea iesirii (> sau >>)

» **TENEX C Shell *tcs***: (1970-1990) versiune imbunatatita a *csh*. TENEX e un sistem de operare care l-a inspirat pe Ken Greer, autorul *tch*. Altii ca Paul Placeway de la Ohio State University si Wilfredo Sanchez de la MIT au mai lucrat in anii 1980, respectiv 1990 la *tcs*.

» **Bourne Again Shell *bash***: (create 1986; 1990) Dezvoltat sub auspiciile GNU, avandul ca autor principal pe Brian Fox de la Free Software Foundation, este shell-ul implicit pe majoritatea distributiilor Linux. Mosteneste in intregime *sh* (orice comanda care merge in *sh* merge si in *bash*; invers nu e valabil) si se conformeaza cerintelor POSIX si specificatiilor IEEE privind shell-urile. Contine elemente din *ksh* si *csh*.

/etc/shells	- contine shelurile de pe un sistem Linux
/etc/passwd	- locul unde e setat shellul implicit

Documentatie suplimentara: **Bash Guide for Beginners** (<http://www.tldp.org/LDP/Bash-Beginners-Guide/html/>).

Functionarea unui interpretor de comenzi shell

slide 4

```

CatTimp (nu s-a inchis sesiunea)
Afiseaza prompter;
Citeste linia de comanda;
Daca ( linia se termina cu '&' ) atunci
    Creeaza un proces; ii da spre executie comanda
    Nu asteapta ca executia sa se termine
Altfel
    Creeaza un proces; ii da spre executie comanda
    Asteapta ca executia comenzii sa se termine
SfDaca
SfCatTimp
  
```



Recomandare: **Debugging Bash scripts** (http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_02_03.html).

Programarea in shell

slide 5



slide 6



COMANDA ELEMENTARA

```
ls
ls > a
nume=Ion
echo $nume
```

&cifra, &- - specifica o intrare sau o iesire standard

&0, &1, &2 - referire la fisierele standard de in, out, err. Se folosesc pentru redirectarea fisierelor si eventual suprapunerea lor.

```
ls so1 so2 # so1 exista ca director; so2 nu exista
ls so1 so2 > outfile # redirecteaza continutul dir so1 in outfile
ls so1 so2 1> outfile # ca mai sus
ls so1 so2 2> errfile # redirecteaza err (so2 nu exista) in errfile
5. ls so1 so2 > file 2>&1 # scrie atat iesirea std cat si err in file

ls so1 so2 &> file # echivalenta cu ^^
ls so1 so2 2> file >&2

10. ls so1 so2 | sort
ls so1 so2 2>&1 |sort # redirecteaza iesirea standard SI ERR catre
# intrarea standard a comenzii sort

grep curs < file
grep curs 0< file
```

Exact la fel ca si in exemplele de mai sus se poate folosi >>, operatie care realizeaza adaugare la fsarsitul fisierului, pe cand > realizeaza suprascriere.


In continuare pezentam un exemplu de lucru cu descriptori de fisiere. (*descriptor de fisier = modalitate de a asign o valoare numerica unui fisier*)

```
echo 1234567890 > File  # scrie sirul de cifre in fisierul File
exec 3<> File           # deschide File si ii asigneaza descript. 3
                        # se poate verifica cu ls -l /proc/self/fd (sau /dev/fd)

read -n 4 <&3            # citeste 4 caractere.

5. echo -n . >&3         # scrie "." in acel loc. (-n - no new line)
exec 3>&-                # inchide fd 3;
                        # sterge legatura simbolica /proc/self/fd/3

cat File               # ==> 1234.67890
```

COMANDA COMPUSA	slide 7
Comenzi simple si succesiuni de comenzi	
Liste de comenzi	
sunt insiruiuri de comenzi legate in pipe si separate de catre unul dintre operatorii	
<pre>c1&&c2 # Lista de forma AND # (returns an exit status of zero) c1 c2 # Lista de forma OR # (returns a non-zero exit status) 5. c1;c2;c3 # Codul de retur al aceste secvente # este cel al ultimei comenzi c1&c2 # c1 se executa intr-un subshell</pre>	

Comenzi compuse

Compunere	Explicatie
	lista_de_comenzi este executat intr-un subshel, adica variabilele

(lista_de_comenzi)	folosite si comenzile din lista care in mod normal ar afecta mediul shellului nu vor mai avea efect dupa terminarea aceste secvente
{ lista_de_comenzi; }	O astfel de secvente se numeste comanda grup.
((expresie))	expresie este evaluata aritmetic. Codul de retur al aceste secvente este 0 in cazul in care expresia evaluata este diferita de zero si 1 altfel.
[[expresie]]	expresie este o conditie care va fi evaluata.
(expresie)	returneaza valoarea expresiei
! expresie	neaga valuarea de adevar a expresiei

```

pwd; (cd ../; pwd;); pwd
pwd; { cd ../;pwd; }; pwd;           # contextul dintre {} se mosteneste

```

Exemple de utilitate pentru {}.

```

# 1. redirectarea unei comenzi grupate
{ echo da; echo nu; } > fis

# 2. comentarea unei secvente de cod
5. # in loc de          putem folosi

                                false &&{

# comm1                      comm1
# comm2                      comm2
10. # comm3                  comm3
# comm4                      comm4
# comm5                      comm5
# comm6                      comm6
# comm7                      comm7

15.                             }

# 3. legarea in pipe a unui grup de comenzi
ps -u raoul | cut -c1-5; echo begin; ps -u root | cut -c1-3
ps -u raoul |{ cut -c1-5; echo begin; ps -u root; }| cut -c1-3



```

Evitari

\ evitarea caracterului urmator
'...' evitarea caracterelor din ... cu exceptia '
"..." evitarea caracterelor din ... cu exceptiile \$ ` \ "

Comentariu

- comenteaza toate caracterele dupa el
#! - asezat la inceputul liniei reprezinta o exceptie, caz in care shell-ul interpreteaza restul liniei ca si o conamda shell pe care o executa. De ex. *#!/bin/sh*

Ex de fisier de comenzi	slide 8
<pre>for fis in *.c do vi \$i gcc \$i done</pre>	
<pre>\$ chmod 755 compilari \$./compilari \$ sh compilari</pre>	
	slide 9
<h2>Variabile shell si mecanisme de substitutie</h2>	
<h2>Substitutie cu iesire standard</h2>	
<p><code>`comanda`</code> - iesirea comenzii comanda</p>	
<pre>echo Calea curenta este `pwd` echo Calea curenta este pwd</pre>	

Variabile shell si substitutia cu valorile lor

```

nume=cuvant      - definirea unei variabile
                 nume="val msms"      - evitare pentru spatii
                 export nume          - va fi mostenita de procesul parinte
$nume            - valoarea variabilei sau sirul vid daca variabila nu e definita
${nume}          - ----- || -----
${nume-cuvant}   - valoarea variabilei sau cuvant daca variabila nu e definita
${nume=cuvant}   - val. var. sau cuvant daca variabila nu e definita; nume=cuvant
${nume+cuvant}   - schimba val. var. in cuvant; daca var. nu e def. va ramane asa
${nume?cuvant}   - pt. validari; val. var. sau cuvant daca variabila nu e definita;
                 daca variabila nu e definita, cuvant se afiseaza pe stderr

```

```

fruct=mar
activitate=joc
echo Dupa $activitate copiii mananca $fruct.      # ==> Dupa joc copiii mananca
                                                    # mar.
5. echo Salba de $fructgaritare.                  # ==> Salba de .
echo Salba de ${fruct}garitare.                   # ==> Salba de margaritare.

nume=Maria

10. echo ${nume-Ana}                             # ==> Maria | Ana daca nume nu E
echo $nume                                           # ==> Maria |

echo ${nume=Ana}                                   # ==> Maria | Ana
echo $nume                                           # ==> Maria | Ana

15. echo ${nume+Ana}                             # ==> Ana |
echo $nume                                           # ==> Maria |

echo ${nume?Ana}                                   # ==> Maria | -bash: nume: Ana

20. echo $nume                                       # ==> Maria |

```

Variabile shell predefinite

```

printenv      # listeaza variabilele predefinite

```

- » **HOSTNAME** - numele masinii pe care suntem logati (linux.scs.ubbcluj.ro)
- » **TERM** - tipul terminalului folosit (xterm)
- » **SHELL** - tipul interpretorului de comenzi folosit ([/bin/bash](http://bin/bash))

- » **USER** - numele sub care utilizatorului curent ([snmro123](#))
- » **MAIL** - fisierul care contine posta electronica a utilizatorului ([/var/spool/mail/snmro123](#))
- » **PATH** - caile de cautare ale fisierelor executabile

slide 10

?

- ([/usr/java/jdk1.6.0/bin:/usr/local/bin:/bin:/usr/bin](#))
- » **HOME** - directorul gazda al utilizatorului ([/home/scs/an2/gr321/snmro123](#))
- » **LOGNAME** - numele sub care utilizatoru si-a deschis sesiunea de lucru ([snmro123](#))
- » **IFS** - separatorii shell pentru cuvinte
- » **PS1** - prompterul principal Unix ([\[\u@\h \W\]\\\$](#))
- » **PS2** - prompterul secundar Unix (>)

```
PS1=ceva
PS1=\>
PS1=`pwd` \>
PS1=\[\u@\h \W]\$ \ # ultimul caracter e spatiu
```

Referirea argumentelor din linia de comanda

slide 11

?

comanda	arg1	arg2	...	arg9	arg10	...argn
^	^	^		^		
\$0	\$1	\$2		\$9		

- » **\$#** - intoarce numarul argumentelor comenzii
- » **\$*** - indica toate argumentele liniei de comanda, privite ca un singur sir
- » **\$@** - indica toate argumentele liniei de comanda, ca succesiune de siruri
- » **\$-** - indica argumentul ce contine optiunile liniei de comanda
- » **\$?** - intoarce codul de terminare al comenzii precedente

» **\$!** - indica PID-ul ultimului proces lansat in background

» **\$\$** - indica PID-ul procesului parinte

```
$ echo $0          # numele comenzii
```

```
-bash
```

```
$ echo $?          # codul de retur al comenzii precedente
```

5. 0

```
$ ls nuExista
```

```
ls: cannot access nuExista: No such file or directory
```

10.

```
$ echo $?          # codul de retur al comenzii precedente
```

```
2
```

```
$ echo $$          # pidul procesului parinte
```

```
1361
```

15.

```
$ ls -ls & echo $!  # pidul ultimului proces lansat in background
```

```
[1] 2084
```

```
2084
```

slide 12

COMENZI utile in context shell

```
$ read ana ion
```

```
george maria
```



```
$ echo $ion
```

```
maria
```

5.

```
$ echo $ana
```

```
george
```

```
$ sleep 5; echo da #va afisa dupa 5 sec
```

```
da
```

10.


```
$ cut -d: -f5 /etc/passwd
```

```
$ echo dada
```

```
dada
```

Comanda *test*

slide 13

	<pre>\$ [3 -lt 4 -a 3 -gt 2]</pre>	
	<pre>\$ echo \$?</pre>	
	0	
	<pre>\$ test -f infoCurs</pre>	
		
5.	<pre>\$ echo \$?</pre>	
	0	
	<pre>\$ test -f info</pre>	
	<pre>\$ echo \$?</pre>	
	1	
10.	<pre>\$ [-d \$HOME]</pre>	
	<pre>\$ echo \$?</pre>	
	0	
	<pre>\$ test -z \$PATH</pre>	
	<pre>\$ echo \$?</pre>	
15.	1	

slide 15



slide 14

