Shell Unix

Autor : prof. Georges ROSENTHAL – IUT Robert Schuman Strasbourg
Traducere şi adaptare : conf. Emilia PECHEANU – Univ. Dunarea de Jos Galati

1

Introducere (1)

Limbajele de programare pot fi clasificate în **două categorii**:

- limbaje **compilate**
- limbaje **interpretate**

Shell -ul Unix este :
- un limbaj de comenzi (limbaj **interpretat**) şi
- un limbaj de **programare**

Un program scris în limbaj Shell se numeşte **script**

- Sub alte SO, terminologia este : **limbaj de comenzi** sau **limbaj de control** (ex. **JCL**)

2

Introducere (2)

- Principalele variante de shell Unix:
  - shell (Bourne, **sh**)
  - C-shell (B. Joy, **csh**)
  - K-shell (D. Korn, **ksh**)
  - Bourne again (**bash**)
  - **pdksh** (public domain ksh), **tcsh**, **zsh**, .. shell-ri restrânse (**rsh**, **rksh**, ...) sau prin meniu-uri (**vsh**, **help**, ...

3

Introducere (3)

- **Avantajele utilizării shell-ului**
  - Interfaţa cu **comenzile**
  - **Stil concis** de programare şi **modularitate**
  - **Administrare** Unix

Nerecomandat pentru prelucrări care cer performanţe de timp

- **Funcţiunile shell-ului**
  - interpretarea **numelor generice** de fişiere
  - **redirectare** şi **pipe-lining**
  - controlul execuţiei programelor
  - **programare** (variabile, parametri, funcţii)
  - **substituţia** comenzilor

4

Caractere şi expresii generice

?	**un caracter oarecare**
*****	**un şir oarecare de caractere (eventual vid)**
[*lista*]	**un caracter din lista**
[*interval ...*]	**un caracter din interval**
[!*lista*]	**un caracter care nu este în lista**
[!*interval ...*]	**un caracter care nu este în interval**

⇨ **în bash se pot utiliza clasele de caractere POSIX:**

```
[:alpha:] [:upper:] [:lower:] [:digit:]
[:xdigit:][:alnum:] [:space:] [:print:]
[:punct:] [:graph:] [:cntrl:]
```

5

Caractere şi expresii generice (exemple)

```
ls –d /bin/???
ls –d *p*
ls –d /bin/*[!A-Za-z]
ls –d .[!.]*
ls /usr/bin/[nm]*[[:digit:]]
```

Observaţie : Dacă numele de fişier conţine caractere generice, căutarea se efectuează implicit în toate subdirectoarele directorului specificat. Opţiunea –d limitează căutarea la directorul specificat.

6

Definiţii Shell (1)

spaţiu alb
- caracterul spaţiu sau caracterul tab

cuvânt
- secvenţă de caractere considerată de Shell ca entitate elementară (atomică) – denumit şi **token**

nume
- cuvânt format din caractere alfanumerice şi underscore care începe cu literă sau underscore – denumit şi **identificator**

cuvânt rezervat
- un cuvânt care are semnificaţie specială pentru Shell şi este interpretat de Shell atunci când nu este protejat (plasat între apostroafe, ghilimele)

```
! case do done elif  else  fi for function if  in
  select  then  until  while  {}  time  [[  ]]
```

Obsevaţie : numele comenzilor sunt cuvinte rezervate

7

Definiţii Shell (2)

metacaracter
- un caracter special care separă cuvinte

```
;  & (  ) | <   >   space    tab
```

operator
- un cuvânt care are rol activ, de control

```
|| &  &&  ;  ;;  (  )  |  NL (newline)
```

comentariu
- simbolul # marchează începutul unui comentariu iar sfârşitul comentariului este **NL (new line)**

8

Redirectarea fişierelor (1)

Trei fişiere <u>implicit</u> ataşate oricărei sesiuni de lucru:

Nume fişier	Descriptor	Semnificaţie
STDIN (standard input)	**0**	intrarea standard de date (tastatura)
STDOUT (standard output)	**1**	ieşirea standard de date (ecranul)
STDERR (standard error)	**2**	ieşirea standard de eroare (ecranul)

9

Redirectarea fişierelor (2)

Cele trei fişiere standard pot fi **redirectate** către **fişiere pe disc**

< fişier	redirectare **STDIN**	**intrare** date **din *fişier***
> fişier	redirectare **STDOUT**	**ieşire** date **către *fişier***
>> fişier	redirectare **STDOUT**	**adăugare** date **în *fişier***
2 > fişier	redirectare **STDERR**	**stocare** mesaje de eroare **în *fişier***

10

Redirectarea fişierelor (3)

- Expresii speciale de redirectare :

2>&1	**fuzionare STDOUT şi STDERR**
<&*n*	**asociere STDIN la descriptorul de fişier *n***
>&-	**închidere STDOUT**
<&-	**închidere STDIN**

- Redirectarea intrării standard a unei comenzi către shell (document integrat)

  comanda **<<[-]*cuvânt***

Pentru comanda ***comanda*** Shell-ul redirecteză STDIN dintr-un fişier (intern) temporar. Fişierul stochează toate liniile introduse de utilizator cu excepţia celei conţinând ***cuvânt***

- Fişierul special « coş de gunoi » **: /dev/null**

11

Redirectarea fişierelor (Exemple 1)

```
# Se creează fişierul vid f1  prin redirectare STDOUT
>f1

# Se listează conţinutul directorului / în fişierul f2  prin redirectare STDOUT
ls / >f2

# Se adaugă data calendaristică în fişierul f2 prin redirectare STDOUT
date >>f2

# Prin redirectare STDOUT se creează fişierul (vid) f3
# Directorul /nimic  nu există  iar pe ecran se afişează un mesaj de eroare
ls /nimic >f3

# Prin redirectare STDOUT se creează fişierul vid f3
# Mesajul de eroare se stochează în fişierul fis_err prin redirectare STDERR
ls /nimic >f3 2>fis_err
```

12

Redirectarea fişierelor (Exemple 2)

```
# Care din comenzile următoare afişează mesajul de eroare pe ecran ?
ls /nimic >f3 2>&1
ls /nimic 2>&1 >f3
```

Indicaţie : Evaluarea redirectărilor dintr-o comandă se face de la stânga la dreapta.

```
# Redirectare STDIN de la fişierul f3
# Conţinutul fişierului f3 va fi trimis prin un e-mail utilizatorului curent
mail $LOGNAME <f3
```

13

Redirectarea fişierelor (Exemple 3)

```
# Afişarea unui text pe ecran ca document integrat
# În liniile până la linia cu cuvântul sfarsit se face substiuţia de variabile şi comenzi
cat <<sfarsit
Numele meu este $LOGNAME
Directorul curent este `pwd`
sfarsit

# Afişarea unui text pe ecran ca document integrat
cat <<-gata
      Acest text va fi aliniat la stanga
gata

# Trimiterea unui mesaj – document integrat către toţi utilizatorii conectaţi
wall <<sf_text
Serverul LINUX va fi oprit peste 5 min
Administrator sistem
sf_text
```

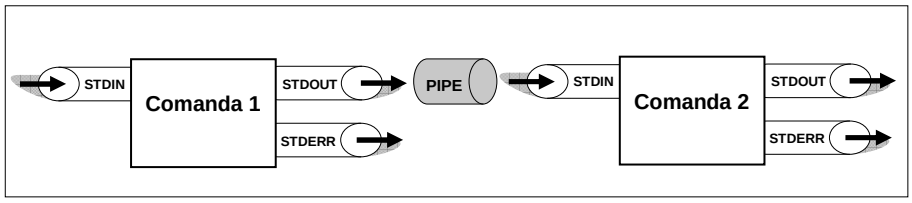
14

Pipe-lining

O « **conductă »** (***pipe***, **|**) constitue o formă dublă de redirectare :

- **Redirectarea ieşirii standard (STDOUT) a unei comenzi către conductă**
- **Redirectarea intrării standard (STDIN) a comenzii următoare către aceeaşi conductă care va « conecta » astfel cele 2 comenzi**

Shell-ul asigură sincronizarea celor două comenzi (procese)



15

Pipe-lining (Exemple)

 # Fişierele script din directorul ***/usr/bin*** *conţin o linie cu textul « shell script »*
 # Care sunt fişierele script din directorul ***/usr/bin*** *?*
```
file /usr/bin/*|grep "shell script"
```

 # Care sunt fişierele script din directorul ***/etc*** *?*
```
find /etc –exec file{} \; 2>/dev/null|grep "shell script"
```

 # Care este numele celui mai mare fişier din directorul **/etc** *?*
```
ls –al /etc|tr –s " "|sort –nr +4 -5|head -1|cut –d" " –f9
```

 # Câte directoare proprii ale utilizatorilor nu se află în ***/home*** *?*
```
getent passwd|cut –d: -f6|grep -cv /home/
```

 # Care este primul nume de utilizator (în ordine alfabetică), dar ultimul ?
```
getent passwd|cut –d: -f1|sort|tee lista|head -1; tail -1 lista
```

 # Care este utilizatorul care s-a conectat primul ?
```
who|sort –k4|line|cut -f1 -d" "
```

16

Citarea (Protejare – Quoting) (1)

- **forţează Shell-ul să ignore** semnificaţia specială a unor metacaractere sau cuvinte rezervate
- **se împiedică interpretarea caracterelor speciale**, se evită recunoaşterea cuvintelor rezervate, sau se **intezice expandarea parametrilor**
- **trei caractere** utilizate pentru protejare :

****	**anulează semnificaţia shell pentru caracterul următor se mai numeşte caracter de escape**
' .... '	**apostoafele (quotes) anulează semnificaţia shell pentru toate caracterele dintre cele 2 apostroafe**
"...."	**ghilimelele (double quotes) anulează semnificaţia shell pentru toate caracterele înafară de $ ` **

- toate **metacaracterele** **; & ( ) | < > space tab** au semnificaţie specială pentru Shell, deci trebuie protejate pentru a-şi păstra propria valoare.

17

Citarea (Protejare – Quoting) (2)

În cazul special al **citării** unui **caracter special sau a unui metacaracter** (**plasării sale între apostroafe inverse, apostroafe sau ghilimele**) :

caractere

delimitatori	\	$	*	`	"	'
`	da	da	da	sfârşit	nu	nu
"	da	da	nu	da	sfârşit	nu
'	nu	nu	nu	nu	nu	sfârşit

nu = caracterul **nu este** interpretat conform semnificaţiei sale ca şi caracter Shell
da = caracterul **este** interpretat conform semnificaţiei sale ca şi caracter Shell
sfârşit = sfârşit delimitare cu caracter de citare

18

Citarea (Protejare – Quoting) (3)

În cadrul unei expresii **$'sir'** sau cu comanda **echo –e "..."** secvenţele **\caracter(e)** sunt interpretate cf. limbajului C :

\b backspace	***\e escape***	***\f formfeed***
\n newline	***\r return***	***\t tab***
\v vertical tab	***\\ back slash***	

nnn caracterul al cărui cod ASCII în octal este ***nnn*** (1 la 3 cifre)
\x*nnn* caracterul al cărui cod ASCII în hexazecimal este ***nnn*** (1 la 3 cifre)

19

Citarea (Exemple 1)

```
>\*a    # creează un fişier având numele *a
echo *
*a Desktop f1.sh f2.sh p1.c p2.c Perles tmp
echo \*
*
echo \**
*a
echo "*"
*
echo pwd
pwd
echo `pwd`
/home/emipec
echo "`pwd`"
/home/emipec
echo '`pwd`‘
`pwd`
# ce afişează comanda ?
echo `echo "*"`
```

20

Citarea (Exemple 2)

```
echo "\\"
\

echo "\""
"

echo "\`"
`

echo "\$HOME"
$HOME

echo "\'"
\'


echo 'aaa\tbbbb'
aaa\tbbbb
echo $'aaa\tbbbb'
aaa    bbbb
```

21

Substituţia comenzilor

`*comanda*`

sau

$(*comanda*) în bash

- caracterele NL din intrarea standard sunt suprimate (o singură linie)
- Perechile de apostroafe inverse **` ..... `** pot fi imbricate cu condiţia de a folosi **citarea cu backslash ** pentru aposroafele inverse ` interne
  Citarea nu este necesară daca se foloseste **$(...)**

22

Substituţia comenzilor (Exemple)

```
# comanda id afişează identificatorii uid real şi gid real pentru un utilizator
id
uid=15510(popescu) gid=1500(automatica)
echo $LOGNAME are identificatorul `id|cut –f1 –d"("|cut –f2 –d=`
popescu are identificatorul 15510

# Ce fac comenzile urmatoare?
cd /tmp
cp `getent passwd|grep popescu|cut –d: -f6`/fis1 fis2
cd `dirname \`find / -name lista –print 2>/dev/null \``
```

23

Execuţia comenzilor

; sau **NL**	execuţie sincronă (**foreground**)
&	execuţie asincronă (**background**)
&&	şi
||	sau
|	comunicaţie inter-procese
{ }	group de comenzi executate în shell-ul curent
( )	group de comenzi executate într-un sub-shell

- Orice comandă poate să citească sau să scrie într-o conductă (pipe-line) în urma redirectăriii intrării sale standard (STDIN) sau a ieşirii sale standard (STDOUT)
- Intrarea standard a unei comenzi care se execută în background (**&**) este **/dev/null**

24

Execuţia comenzilor (Exemple 1)

```
# Execuţia comenzii find se face în background
find / -name *aa* 2>/dev/null &

# conectorii && şi || simulează funcţionarea unei structuri if
# codul de retur al primei comenzi condiţionează execuţia comenzilor următoare
date "+%A %d %B" && echo "Corect!" || echo "Comanda gresita!"
Joi 8 noiembrie
Corect!

date "%A %d %B" && echo "Corect!" || echo "Comanda gresita!"
Date: invalid date %A %d %B
Comanda gresita!


# Cum funcţionează comenzile următoare ?
ls –l |grep f1 && echo "fişier  gasit!" || echo "fişier
negasit"
```

25

Execuţia comenzilor (Exemple 2)

```
# Gruparea comenzilor cu ajutorul parantezelor

(echo Salut ; exit)
{echo salut; exit;}


pwd; (cd ..; pwd; echo salut) ; pwd
pwd; {cd ..; pwd; echo salut} ; pwd
```

26

Instrucţiuni Shell de programare

`while...do...done`	structura repetitivă condiţională
`for..in..do...done`	structura iterativă
`if...then...else...fi`	structura alternativă
`case..in..esac`	lista de cazuri

- aceste instrucţiuni functionează ca şi **comenzi interne**
- cuvintele cheie **`while, for ...`** nu sunt recunoscute decât după separatorii de cuvânt sau după operatorii următori :

  `;  NL  &  &&  |  ||`
- o **condiţie** se exprimă prin **codul de retour** reîntors de **o comandă** sau **o listă de comenzi** (**0** pentru **ADEVARAT**, **diferit de 0** pentru **FALS**)

27

Structura alternativă

```
if  lista-1
      then  lista-2
     [else  lista-3]
fi
```

sau :

```
if lista-1
     then lista-2
     [elif lista-3
          then lista-4
          else  lista-5]
fi
```

unde : ***`lista-1`*** , ***`lista-2 ...`*** reprezintă o comandă sau un grup de comenzi

28

Structura alternativă (Exemple)

```
# Afişare mesaje de eroare în limba română
if cat fifi 2>/dev/null
       then
              echo OK
       else
              echo "fişier  negasit sau fara drept de citire"

# Condiţia se exprimă prin două comenzi în pipe-line
if who|grep root 2>/dev/null
       then echo "Utilizatorul root este conectat! "
       else
            echo "Utilizatorul root nu este conectat!"
fi

# Condiţia se poate exprima şi prin operatorii && şi ||
who|grep root 2>/dev/null && echo "Root conectat!" || echo OK
```

29

Lista de cazuri

case *variabilă* **in**
model-a [**|** *model-b*]...**)** *lista-1* **;;**
model-m [**|** *model-n*]...**)** *lista-2* **;;**
…
model-x [**|** *model-y*]**...)** *lista-99*
esac

unde : ***lista-1*** , ***lista-2 …*** reprezintă o comandă sau un grup de comenzi

30

Lista de cazuri (Exemple)

```
# Afişare mesaje în funcţie de o valoare numerică tastată de utilizator
read numar
case $numar in
   [0-9]) echo "Ati tastat un numar de o cifra!" ;;
   [0-9][0-9]) echo "Ati tastat un numar de 2 cifre!" ;;
   *) echo "Ati tastat altceva decat un numar" ;;
esac

# Afişare mesaje în funcţie de o valoare calculată
nr_util=`who|wc -l`
case $nr_util in
       1) echo "Un singur utilizator este conectat! " ;;
       [2-5]) echo "Sub 5 utilizatori conectati!" ;;
       ?|?? ) echo "Mai mult de 5 utilizatori conectati!"
esac
```

31

Structuri repetitive

Structura iterativă

for *variabila* [**in** *cuvânt-1* [*cuvânt-2*] ...]
    do lista-comenzi
done

Se poate utiliza comanda shell **seq** pentru a genera lista de valori (cuvinte)

Structura repetitivă condiţională

while *lista-1*
    do *lista-2*
done

Este disponibilă şi structura **until** ... **do ...done**

32

Comenzi interne pentru controlul fluxului de execuţie

`break` **[*n*]**	iesire din bucla	(1)
`continue` **[*n*]**	salt la iteratia urmatoare	(1)
`exit` **[*r*]**	iesire din shell-ul (scriptul) curent , care corespunde ultimei comenzi executate	
`.` ***fişier*** catre *fişier*	redirijarea intrarii standard (STDIN)	
`wait` **[*n*]** background)	asteptare in vederea executiei (proces în	

33

Variabile Shell (1)

- **numele unei variabile** este o suită de caractere de tip **literă**, **cifră** sau _ , în care primul caracter este obligatoriu literă
- declararea **implicită** a unei variabile se face atunci când se efectuează prima atribuire a unei valori

nume-de-variabila=sir_de_caractere

ATENŢIE : Fără spatiu înainte şi după semnul **=**

- o variabilă este **implicit locală** shell-ului în care a fost creată, **dar poate fi "exportată"** comenzilor apelate cu comanda **`export`**

34

Variabile Shell

- **la execuţia unei comenzi, orice variabilă care face parte din acea comandă va fi înlocuită (substituită) prin valoarea sa curentă, care poate fi şi rezultatul unei comenzi**
- **referinţa la o variabilă (înlocuirea ei cu valoarea sa curentă) se face cu ${*nume*} sau cu $*nume* , dacă nu există ambiguitate**
- **un şir de caractere vid sau conţinând caractere speciale şi/sau spatii va fi delimitat cu ghilimele " sau cu apostroafe '**

35

Variabile speciale ale shell-ului

- **Variabile speciale**

$? codul de retur al ultimei comenzi
$# numarul de parametri pozitionali (din linia de comanda)
$$ pid-ul shell-ului curent
$! pid-ul ultimului proces executat in background
$0 numele script-ului curent (sau a shell-ului curent)

$1..$9 parametrii poziţionali (introduşi de utilizator în linia de c-dă)

- **Variabile create de shell (în afară de ?)**

"$*" toate argumentele separate de spatiu formează un singur cuvânt (un singur şir)
"$@" lista cuprinzând $# parametri poziţionali

36

Comanda de testare obiect shell : `test(1)`

`test` *expresie* **sau** **`[` *expresie* `]`**

- **Evaluează *expresie* şi întoarce 0 dacă ea este adevarată , sau o valoare diferită de zero dacă este falsă**
- **se foloseşte cu instrucţiunile *if , while , until***
- **în bash, operatorul extins [ ] permite expresii simplificate**
- ***expresie* se construieşte pe baza unei sintaxe specifice comenzii `test`**

Pentru testare şiruri de caractere expresiile sunt :

-z *şir*	**adevărat dacă *şir* are lungimea egală cu 0 (şir vid)**
-n *şir*	**adevărat dacă *şir* are lungimea diferita de 0 (sir non-vid)**
şir1* = *şir2	**adevărat dacă *şir1* este identic cu *şir2***
şir1* != *şir2	**adevărat dacă *şir1* este diferit de *şir2***
şir	**adevărat dacă *şir1* nu este vid**

37

`test (2)`

Expresii pentru testare fişiere

`-d` *fişier*	**adevărat dacă *fişier* există şi este un director**
`-f` *fişier*	**adevărat dacă *fişier* există şi este un fişier ordinar (obişnuit)**
`-c` *fişier*	**adevărat dacă *fişier* există şi este de tip special-caracter**
`-b` *fişier*	**adevărat dacă *fişier* există şi este de tip special-bloc**
`-h` *fişier*	**adevărat dacă *fişier* există şi este de tip legatură simbolică**
`-p` *fişier*	**adevărat dacă *fişier* există şi este de tip conductă cu nume**
`-r` *fişier*	**adevărat dacă *fişier* există şi permite citire pentru utiliz. curent**
`-w` *fişier*	**adevărat dacă *fişier* există şi permite scriere pentru utiliz. curent**
`-x` *fişier*	**adevărat dacă *fişier* există şi permite executie pentru utiliz. curent**
`-k` *fişier*	**adevărat dacă sticky-bit pentru *fişier* este « on »**
`-u` *fişier*	**adevărat dacă bitul suid pentru *fişier* este « on »**
`-g` *fişier*	**adevărat dacă bitul sgid pentru *fişier* este « on »**
`-s` *fişier*	**adevărat dacă *fişier* are lungimea diferită de 0**
`-t` *descript*	**adevărat dacă fişierul având descriptorul *descript* (egal implicit cu 1) corespunde unui dispozitiv de tip terminal**

38

test (3)

Testare numere

n1 rel n2 — **adevărat dacă relatia rel dintre *intregii n1 şi n2* este adevarata**

- **relatia *rel* se exprimă prin `-eq` , `-ne` , `-gt` , `-ge` , `-lt` , `-le`**
- **ATENTIE : Nu confundaţi operatorii de comparare numere cu cei de comparare şiruri !**

Observaţii

- **conditiile compuse se exprima cu ajutorul operatorilor logici: `-a` (si) , `-o` (sau), `!` (non) plus paranteze rotunde ( ) pentru operanzi**
- **pentru a elimina ambiguitatile, delimitati variabilele cu ghilimele "**
- **operatorii `=` şi `!=` necesita intotdeuna specificarea operanzilor**
- **parantezele ( ) se citeaza (sunt precedate) in mod obligatoriu cu **
- **paranteza [(resp.]) trebuie sa fie urmata (resp. precedata) de spatiu**

39

test - extensii bash

`-e` ***fişier***	**adevărat dacă fişierul *fişier* există**
`-O` ***fişier***	**adevărat dacă utilizatorul curent este proprietarul fişierului**
`-G` ***fişier***	**adevărat dacă grupul din care face parte utilizatorul curent este grupul proprietarului**
`-N` ***fişier***	**adevărat dacă fişier ul *fişier* a fost modificat de la ultima citire**
f1 `-nt` ***f2***	**adevărat dacă f1 este mai recent decat f2**
f1 `-ot` ***f2***	**adevărat dacă f1 este mai vechi decat f2**
f1 `-ef` ***f2***	**adevărat dacă f1 şi f2 sunt legaturi (link-uri) fizice catre acelasi fişier**

40