5. Interpretorul de comenzi (Shell)

Cuprins

- Introducere
- Facilităţi Shell fundamentale
- Facilităţi de programare în Shell
- Controlul joburilor
- Alte facilităţi ale interpretoarelor de comenzi

5.1. Introducere

- Interpretorul de comenzi poate fi văzut ca orice program care rulează în spaţiul utilizator şi este analog celorlalte comenzi
- Orice program rulează într-o *ambianţă*; stabilirea acesteia este una din sarcinile Shell; ambianţa pentru Shell-ul de login este stabilită implicit de SO
- Shell-ul este un program ciclic care execută următoarele acţiuni:
  - Citeşte o linie din fişierul de intrare
  - Stabileşte ambianţa de execuţie a liniei
  - Lansează în execuţie fiecare comandă din linie, ca proces distinct (fiu al procesului shell)
  - Aşteaptă terminarea execuţiei comenzilor lansate
  - Reia ciclul de la prima acţiune
- Acest ciclu de bază este extins cu facilităţi de înlănţuire a comenzilor
- Se consideră în principal facilităţile interpretorului de comenzi "clasic" (Bourne shell, sh)

5.2. Facilităţi Shell fundamentale

- Pentru stabilirea ambianţei de execuţie Shell execută, în ordine, următoarele:
  - Împarte linia de comandă în cuvinte (atomi)
  - Interpretează caracterele de citare (ghilimelele simple şi duble)
  - Redirectează intrările şi ieşirile
  - Substituie variabilele de mediu
  - Substituie comenzile
  - Expandează numele de fişiere
- După stabilirea ambianţei se lansează în execuţie comanda
- Exemplu:

  `$ ls –l $HOME/*.c | grep "Nov 18"`
  - Atomi: `ls, -l, $HOME/*.c, | , grep şi "Nov 18"`
  - Caractere de citare: ghilimelele de la ultimul atom, sunt necesare pentru ca tot şirul să fie văzut ca un singur atom
  - Caracter de redirectare: `|`, shell-ul face aranjamentele ca ieşirea de la `ls` să fie intrare pentru `grep`
  - Variabilă de mediu: `$HOME`, se înlocuieşte cu valoarea ei (ex. `/home/staff/ionel`)
  - Expandarea numelor de fişiere: se caută toate fişierele cu nume având sufixul .c şi de adaugă în linia de comadă
  - Shell creează două procese fii, unul pentru `ls`, altul pentru `grep` şi le lansează în execuţie simultan

Mecanisme de citare în Shell

- Pentru Bourne shell următoarele caractere au semnificaţie specială:
  `# & * ? [ ] ( ) = | ^ ; < > ` $ ´ ˝ \`
- Spaţiile albe (spaţiu, tab, newline) servesc ca separatori de cuvinte
- Prin mecanismele de citare se dezautorizează semnificaţia caracterelor speciale (nu sunt afectate spaţiile albe)
- Trei mecanisme de citare:

´*text*´ dezautorizează toate caracterele speciale din *text*

˝*text*˝ nu dezautorizează `$`, `` ` `` şi `\`

`\`*c* dezautorizează caracterul *c*

- Exemple:

`$ echo 45 \* 3` duce la rezultatul `45 * 3`

`$ echo 45 * 3` înlocuieşte * cu toate fişierele din directorul curent

```
$ echo Hey!     What's next? Mike's #1 friend has $$.
```

va produce:

```
   Hey!  Whats next? Mikes
$ echo "Hey!     What's next? Mike's #1 friend has $$."
```

va produce:

```
Hey!     What's next? Mike's #1 friend has 23812.
```

Redirectarea intrărilor şi ieşirilor

- Fişiere standard – disponibile fiecărui proces: intrare (descriptor de fişier 0), ieşire (1), raportare de erori (2)
- Implicit: intrarea standard este tastatura, ieşirea şi raportarea de erori: ecranul
- În Shell se pot specifica în linia de comandă alte asocieri pentru fişierele standard de intrare (cu <) şi de ieşire (cu >). Exemplu:

```
$ cat main.c > program.c
```

are efectul că trimite conţinutul din `main.c` în `program.c` în loc de a-l afişa pe ecran. Shell-ul *închide* fişierul standard momentan folosit şi *deschide* imediat fişierul program.c, care este şi creat dacă nu există.

- Pot fi redirectate şi fişiere cu alţi descriptori decât 0 şi 1, folosind notaţia `n>` (`2>` redirectează raportarea de erori)
- Pentru nu suprascrie un fişier folosit la redirectare se foloseşte notaţia >>, cu semnificaţia de *append (adaugă)*
- Caz special: utilizarea notaţiei << *c* pentru intrare: următoarele informaţii necesare unei comenzi se preiau de la tastatură sau din textul unei proceduri Shell, până la întâlnirea lui *c* (caracter sau şir)
- Mecanimul de *pipe*, redat prin |

```
$ command1 | command2
```

are semnificaţia:

- `command1` şi `command2` vor fi lansate concurent în execuţie, ieşirea lui `command1` fiind utilizată ca intrare pentru `command2`

Se foloseşte pentru combinarea prelucrărilor realizate de mai multe comenzi. Ex.:

```
$ ps –aux | grep ionel
```

Variabile de mediu. Substituţia comenzilor

- Variabilele de mediu păstrează informaţii depre ambianţa de execuţie. Au ca valori şiruri de caractere şi se pot iniţializa prin *comenzi* de forma:

```
name=value
```

- Referirile la valoarea unei variabile de mediu se fac cu $ în faţa numelui variabilei. Ex.:

```
$ VAR=text
$ echo $VAR
text
```

- Pentru ca o variabilă să fie validă pe toată durata sesiunii (nu numai pentru activarea curentă a Shell), ea trebuie *exportată*. Sintaxa:

```
name=value; export name
```

- Listarea tuturor variabilelor de mediu definite la un moment dat se poate face cu comanda `env`
- Fiecare program poate avea variabile de mediu proprii, dar există câteva predefinite:

`PATH` lista directoarelor în care se caută comenzi. Pentru a adăuga noi directoare:

```
PATH=$PATH:/usr/local/bin:$HOME/bin
```

`HOME` directorul gazdă al utilizatorului

`TERM` tipul terminalului folosit de utilizator

`MAIL` numele de cale pentru cutia poştală

`PS1` şirul folosit ca prompt. La `bash` se poate scrie, de ex., `PS1=\u@\h:\w\$`

`PS2` promptul suplimentar; când o comandă se întinde pe mai multe linii

`$` numărul procesului (PID) pentru shell-ul curent

`#` numărul de argumente din linia de comandă

`0` numele comenzii executate

`n` pentru valori `n` între 1 şi 9 reprezintă al n-lea argument din linia de comandă

- Substituţia comezilor: ieşirea produsă de o comandă se substituie acelei comenzi scrise între accente grave. Ex.:

```
$ mail `who | cut –c1-8 | sort –u`
```

5.3. Facilităţi de programare în Shell (1)

- Facilităţile de programare se folosesc în special în proceduri Shell, dar pot fi utilizate şi în lucrul interactiv
- Fişierele cu proceduri Shell pot fi lansate în execuţie în mai multe moduri:

```
$ sh numefis argumente
```

(se execută cu un nou proces shell)

sau:

```
$ . numefis argumente
```

(se execută cu shell-ul curent)

sau:

```
$ chmod a+x numefis
$ numefis argumente
```

(fişierul cu procedura shell trebuie să fie executabil)

- Structuri pentru controlul secvenţial:

a) `;` permite specificarea mai multor comenzi pe o linie. Comenzile se execută strict secvenţial. Ex.:

```
cmd1; cmd2; cmd3
```

Execuţia lui `cmd2` nu poate începe decât dupa ce se termină `cmd1`

b) ( ) se folosesc pentru gruparea comenzilor, ceea ce înseamnă combinarea fişierelor standard de ieşire. Exemplu:

```
(cmd1; cmd2 | grep) |  wc
```

are ca efect faptul că intrarea pentru `wc` este formată din ieşirea lui `cmd1` urmată de ieşirea lui `grep`

c) `&&` a doua comandă (ex. `cmd1 && cmd2`) se execută numai dacă prima are cod de retur 0 (adică s-a executat corect)

d) `||` a doua comandă se execută numai dacă prima are cod de retur diferit de zero

Facilităţi de programare în Shell (2)

- **Structuri alternative**

a) if-then-else:

```
if [ test-conditions ]
then cmd1
else cmd2
fi
```

Exemple de condiţii de test:

`-f` *fişier* true dacă *fişier* există şi e obişnuit
`-d` *director* true dacă fişierul *director* există şi este director
`-w` *fişier* true dacă *fişier* există şi utilizatorul are drept de scriere
`-z` *şir* true dacă şir are lungime nulă
şir1 `!=` *şir2* true dacă cele două şiruri nu sunt egale
n1 `-gt` *n2* true dacă numărul *n1* este mai mare decât numărul *n2*
l1 `-a` *l2* true dacă *l1* şi *l2* sunt true

b) case:

```
case $variable in
   vallist1)   action1;;
   vallist2)   action2;;
   ...
  *) defaultaction;;
esac
```

Numai una dintre acţiuni ajunge să fie executată la o execuţie a unui `case`

Facilităţi de programare în Shell (3)

- **Structuri de ciclare**

a) for:

```
for variable [ in valuelist ]
do
     action
done
```

Dacă partea opţională lipseşte, ciclul se execută pentru *variable* luând pe rând ca valoare fiecare argument din linia de comandă

Exemple:

`for i do >$i; done` creează un număr oarecare de fişiere
`for i in * do`
`echo $i; done` listează numele fişierelor din directorul curent

b) while:

```
while command-list1
do command-list2
done
```

Valoarea testată de `while` este codul de retur al ultimei comenzi din *command-list1*

c) until:

```
until command-list1
do command-list2
done
```

Comenzi interne ale interpretorului de comenzi(1)

- **shift** elimină primul argument din linia de comandă şi deplasează spre stânga argumentele rămase. Exemplu:

```
while [ $# -gt 0 ]
do echo $1
    shift
done
```

- **read** citeşte câte o linie din fişierul standard de intrare şi atribuie cuvintele din linie unor variabile de mediu:

```
read [ -r ] [ name ... ]
```

Dacă sunt mai multe cuvinte, ultima variabilă primeşte toate cuvintele rămase

Dacă sunt mai puţine cuvinte, ultimele variabile primesc ca valori şirul vid

Dacă nu sunt specificate variabile, linia se dă ca valoare variabilei `REPLY`

Dacă este prezentă opţiunea –r perechea *backslash-newline* nu este ignorată şi se pot specifica valori pe mai multe linii

Codul de retur de la `read` este 0, până la întâlnirea unui sfârşit de fişier

Exemplu:

```
while read nume prenume telefon
  do
    echo  $nume ″ ″ $prenume ″ ″ $telefon >> agenda
done
```

Comenzi interne ale interpretorului de comenzi(2)

- **break** determină părăsirea unui ciclu `for`, `while` sau `until` şi returnează 0. Cu `break` *n* se părăsesc *n* cicluri încuibate
- **continue** determină trecerea la iteraţia următoare a unui ciclu. Se poate folosi şi `continue` *n*
- **eval** determină concatenarea argumentelor şi execuţia comenzii astfel rezultate:

```
eval [ args ]
```

Comanda returnează codul de retur al comenzii executate. Dacă nu există argumente returnează true.

- **enable** controlează autorizarea/dezautorizarea comenzilor interne ale Shell (unele comenzi au şi variantă externă: `cd`, `pwd`, `test`, `echo`, etc.). Are sintaxa:

```
enable [-n] [-a] [name ...]
```

Opţiunea `-n` determină dezautorizarea comenzilor interne specificate prin lista de nume. Fără listă se afişează comenzile interne momentan dezautorizate

Opţiunea `-a` afişează lista tuturor comenzilor interne, cu specificarea stării de autorizare a fiecăreia

- **set** permite specificarea unor condiţii de funcţionare a shell-ului. Sintaxa:

```
set [ options arg1 arg2 ... ]
```

Fără argumente listează valorile tuturor variabilelor cunoscute în shell-ul curent. Opţiunile se autorizează (`- option`) sau dezautorizează (`+ option`). Exemple de opţiuni:

`-a` marchează automat pentru export variabilele shell

`-b` afişează imediat mesajele de terminarea joburilor executate în fundal (background)

`-f` ignoră metacaracterele în numele de fişiere

Controlul întreruperilor (semnalelor) în Shell

- O întrerupere de la terminal (de regulă prin Ctrl-C) determină terminarea unei proceduri Shell. Se poate controla comportarea la apariţia unui semnal prin comanda `trap`:

```
trap [-l] [arg] [sigspec ...]
```

Cu opţiunea `-l` se obţine o listă numelor semnalelelor şi a numerelor de identificare corespunzătoare

Dacă `arg` lipseşte sau este `-` semnalele specificate revin la tratarea implicită; pentru `arg` dat ca şir vid, semnalele specificate vor fi ignorate

De regulă `arg` apare ca o listă de comenzi între apostrofuri

`sigspec` se poate da ca listă de nume sau de numere. Valoarea 0 pentru sigspec permite specificarea comenzilor executate înainte de terminarea normală a procedurii

Un exemplu: când procedura e întreruptă de la tastatură să se şteargă fişierele termporare înainte de `exit`:

```
trap 'rm /tmp/ps$$; exit' SIGINT
```

Exemplu mai complex:

```
d=`pwd`
for i in *
do if [ -d $d/$i ]
   then cd $d/$i
             while echo "$i:"; trap exit 2;read x
             do trap : 2 ; eval $x ; done
   fi
done
```

5.4. Controlul joburilor

- În Unix există procese care se execută în *foreground* (prim-plan) - comunică direct cu terminalul şi procese *background* (fundal), deconectate de la terminal
- La un moment dat un singur proces poate fi în prim-plan, dar în fundal pot exista oricâte procese. Pentru controlul execuţiei, unele shell-uri prevăd comenzi speciale.
- Cea mai simplă facilitate: specificarea execuţiei în fundal prin `&`. Exemplu:

```
$ gv fis.ps &
[1] 12804
```

- Comanda `jobs` produce o listare a proceselor din fundal cu starea fiecăruia:

```
$ jobs
[1]+ Running gv fis.ps &
```

- Pentru aducerea în prim-plan a unui proces din fundal se foloseşte `fg`:

```
fg %1
```

În absenţa argumentului se aduce în prim-plan procesul listat cu + de `jobs`

- Procesul din prim-plan poate fi oprit (fără a fi terminat) prin *Ctrl-Z*, iar prin comanda `bg` poate fi trimis în fundal un proces suspendat:

```
$ gv Dev.ps.gz
CTRL-Z
[1]+ Stopped gv Dev.ps.gz
$ bg
[1]+ gv Dev.ps.gz &
```

- Terminarea unui proces din fundal se poate face utilizând comanda `kill` cu argument numărul procesului (jobului) precedat de % :

```
$ kill %1
```

5.5. Alte facilităţi ale interpretoarelor de comenzi

- **Istoria comenzilor**. Facilitate prezentă în majoritatea interpretoarelor de comenzi. Comanda `history` listează ultimele comenzi folosite în sesiune. Ex.:

```
$ history 5
  497 ls -l
  498 ftp b519
  499 ftp b519.timisoara.roedu.net
  500 exit
  501 history
```

Navigarea se face cu ajutorul tastelor cu săgeţi sau cu o notaţie de forma:

```
!n
```

unde n reprezintă numărul comenzii dorite din cele listate de `history`

- **Aliasuri**. Permit definirea de nume noi pentru o serie de comenzi. Exemple cu sintaxa din `bash`:

```
alias ri="rm -i"
alias clean="rm *~ .*~ core *.bak"
alias lr="ls -lR"
```

Aliasurile se introduc în fişierul de configurare al interpretorului (*.bash_profile* pentru `bash`)

- **Funcţii**. Au sintaxă asemănătoare funcţiilor din C. O funcţie echivalentă cu alias-ul `lr` de mai sus apare într-o procedură shell astfel (definiţie şi utilizare):

```
lr ( ) { ls -lR "$@"; }
lr
```

@ este o variabilă de mediu, cu semnificaţia "toţi parametrii din linia de comandă"

La utilizare se dă numele funcţiei urmat de lista de parametri (nume separate prin spaţii, fără parantezele de la definiţie). Aici linia a doua din procedură nu necesită parametri, conform definiţiei.