

Capitolul 3

INTERPRETOARE DE COMENZI

3.1. Introducere

Interpretorul de comenzi este un program care realizează o interfață între utilizator și sistemul de operare, interfață care preia comenzile utilizatorului și le transmite sistemului de operare spre execuție. Pentru Unix și clonele acestuia (*Linux*) interpretoarele de comenzi sunt independente față de sistemul de operare ceea ce a dus la dezvoltarea unui număr mare de astfel de interpretoare.

Primul interpretor de comenzi important, numit *Bourne shell*, a fost dezvoltat de Steven Bourne și a fost inclus într-o distribuție de Unix lansată în 1979. Acest interpretor de comenzi este cunoscut în cadrul sistemului de operare sub numele de *sh*. Un alt interpretor de comenzi utilizat pe scară largă este *C shell* cunoscut și sub numele de *csh*. *Csh* a fost scris de Billy Joy ca parte integrantă a unei distribuții de Unix numită BSD (Berkley System Distribution), și a apărut la câțiva ani după *sh*. Numele de *Csh* l-a primit datorită asemănării sintaxei comenzilor sale cu cea a instrucțiunilor limbajului *C*, fapt ce îl face mult mai ușor în utilizare celor familiarizați cu limbajul *C*. Toate aceste interpretoare au fost inițial dezvoltate pentru Unix dar s-au dezvoltat versiuni ale acestora și pentru *Linux*.

Unul dintre cele mai cunoscute interpretoare de comenzi pentru *Linux* este *bash* (bourne again shell), acesta va fi și interpretorul de comenzi ce va fi utilizat în continuare.

Atenție!

La intrarea în sistem fiecare utilizator primește o instanță a interpretorului de comenzi. Acesta pastrează un mediu distinct pentru fiecare utilizator din sistem.

În multe sisteme *Linux* comanda */bin/sh* este de multe ori un *link* către interpretorul de comenzi curent din sistemul de operare.

Interpretorul de comenzi din *Linux* deși are unele similitudini cu procesorul de comenzi *DOS* este un instrument mult mai puternic oferind utilizatorului o multitudine de facilități.

Un exemplu de fișier de comenzi este:

```
#!/bin/sh
echo "hello world"
```

Rularea lui determină afișarea mesajului *hello world* la consola

3.2. Facilități ale interpretoarelor de comenzi

Expandarea numelor de cale – În exemplele de până acum s-au folosit numai nume de cale complet precizate. Există mai multe metode prin care numele de cale să fie abreviate, iar interpretorul de comenzi va expanda aceste nume înainte de ale transmite comenzilor. Un astfel de caracter de abreviere este ***. De exemplu

```
$ cp text/* backup
```

Aici prin *** se înțelege – toate fișierele (cu excepția celor ce încep cu *.*) din catalogul *text*. Sensul comenzii este – toate fișierele din catalogul *text* vor fi copiate în catalogul *backup*.

Un alt caracter special utilizat în numele fișierelor pe care interpretorul de comenzi le va expanda este semnul întrebării *?*. Rolul acestuia este de a substitui un singur caracter :

```
$ ls /dev/tty?
/dev/tty0 /dev/tty2 /dev/tty4 /dev/tty6 /dev/tty8
/dev/tty1 /dev/tty3 /dev/tty5 /dev/tty7 /dev/tty9
```

De asemenea este posibilă precizarea unei liste de caractere, și numai caracterele conținute în această listă să poată fi expandate. Această listă este delimitată de paranteze drepte (*[]*). Ex:

```
$ ls /dev/tty[pq][235]
/dev/tty2 /dev/tty3 /dev/tty5 /dev/ttyq2 /dev/ttyq3 /dev/ttyq5
```

O ultimă formă de expandare pentru numele de cale este prin precizarea unei liste de cuvinte, cuvintele vor fi separate prin virgulă și cuprinse între paranteze *{}*. Ex:

```
$ mkdir /usr/tmp/{bin,doc}
```

Atenție! Toate numele de cale sunt expandate de interpretorul de comenzi înainte a fi transmise către comenzi.

În cazul în care se dorește utilizarea caracterelor speciale cu sensul lor inițial se poate folosi unul din următoarele trei mecanisme de marcare:

- caractere escape (**) – caracterul *backslash* îi indică interpretorului de comenzi să ignore sensul special al caracterelor de după. Ex: ***.
- ghilimele ‘ ‘ – orice șir de caractere cuprins între aceste marcaje își pierde sensul special al tuturor caracterelor cuprinse în el. Ex: *'a*?'*
- ghilimele duble “ “ – un șir de caractere pierde sensul special al caracterelor cuprinse în el (este valabil numai pentru caracterele prezentate până acum) , cu excepția caracterelor *\$* și **.

Ex:

`cmd`. Executa comanda **cmd**. De exemplu: \$ cale='pwd'
atribuie variabilei cale rezultatul comenzii **pwd**.

Redirectarea ieșirii și a intrării – Filosofia de bază Linux este aceea de ține lucrurile cât mai simple posibil, prin punerea la dispoziția utilizatorului a unui număr mare de funcții simple. Iar prin gruparea acestor funcții împreună se pot obține comenzi mai complexe. Implicit, majoritatea comenzilor Linux sunt configurate astfel încât întreaga acestora să o reprezinte tastatura iar datele de ieșire sunt trimise către monitor. În unele situații ar fi util dacă intrarea, respectiv ieșirea unei comenzi ar fi altele decât cele implicite. Unele comenzi au această facilități, dar dacă pentru fiecare în parte s-ar prevedea acest lucru dimesiunea executabilelor ar crește, de aceea această sarcină și-a asumat-o sistemul de operare prin intermediul acestui mecanism de redirectare.

Un exemplu clasic este cel de creare a unui nou fișier prin care ieșirea standard a comenzii *cat* este redirectată către fișierul care urmează a fi creat:

```
$ cat > text
text de exemplu
Ctrl-d
$
```

Se observă că în această situație se folosește caracterul '>'. Acesta are rolul de a indica redirectarea ieșirii standard către o altă comandă. Dacă este urmat de un nume de fișier atunci acest fișier este creat dacă nu există, iar dacă există atunci va rescris. Un alt simbol utilizat de acest mecanism de redirectare este '>>'. În cazul în care ieșirea este reprezentată de un fișier datele redirectate vor fi concatenate la sfârșitul fișierului. Ex:

```
$ ls -l /usr/bin >>text
```

Rezultat: numele fișierelor din */usr/bin*, împreună cu informațiile despre acestea sunt adăugate la fișierul text.

Identic cu redirectarea ieșirii se poate redirecta și ieșirea de eroare:

```
$ ls /usr/bin 2>text.err
```

Și intrarea standard:

```
$ wc -l < /etc/passwd
12
```

Rezultat: numărul de intrări din fișierul de parole.

Pipes sau conectarea proceselor – prin redirectionarea ieșirii standard a unei comenzi într-un fișier, iar apoi redirectionarea aceluiași fișier ca și intrare standard pentru altă comandă se pun practic în legătură două procese prin intermediul unui fișier temporar. Acest lucru se poate face implicit de către sistemul de operare prin intermediul unui *pipe*. Pentru aceasta se folosește operatorul |.

De exemplu seria de comenzi:

```
$ ls /usr/bin > /tmp/temp.txt
$ wc -w </tmp/temp.txt
500
$ rm /tmp/temp.txt
```

poate fi înlocuită prin:

```
$ ls /usr/bin | wc -w
500
```

Controlul proceselor – Permite întreruperea unui proces și de asemenea reluarea execuției aceluși proces la un moment de timp ulterior. Întreruperea execuției unui proces se poate face direct de la tastatură prin secvența *Ctrl-z*, după care procesul va fi întrerupt iar utilizatorul reprimește controlul putând executa alte comenzi.

```
$ cat > file.txt
Ctrl-z
[1]+ Stopped
$
```

Pentru gestionarea proceselor interpretorul de comenzi are implementate o serie de comenzi:

- *fg* – este comanda prin care este reluată execuția unui proces, aceasta poate primi ca și parametru numărul de ordine al procesului întrerupt. Pentru a relua execuția ultimului proces întrerupt avem:

```
$fg %1
cat >file.txt
```

- *bg* – se folosește pentru suspendarea unui proces, echivalent cu *Ctrl-z*
- *jobs* – afișează procesele suspendate.

Alte facilități ale interpretorului de comenzi Linux:

- *history* posibilitatea de rechemare rapidă a ultimilor comenzi executate.
- *command completion* afișarea posibilelor comenzi pornind de la primele caractere ale numelor acestora.

3.3. Sintaxa interpretorului de comenzi.

Programarea în cazul interpretorului de comenzi se face în același mod ca și în cazul altor limbaje de programare. Și în acest caz vom avea:

Variabile – în mod normal nu sunt declarate decât atunci când este nevoie de ele. Implicit variabilele snt considerate și memorate ca și șiruri de caractere, chiar și atunci când primesc valori numerice.

Variabilele pot fi de mai multe tipuri: variabile predefinite sau speciale, parametrii poziționali sau variabile utilizator.

Variabile predefinite sau speciale – la pornirea sistemului o serie de variabile sunt inițializate cu valorile implicite mediului de operare. Cele mai importante sunt următoarele:

\$HOME – catalogul gazdă a utilizatorului curent

\$PATH – o listă de cataloage în care interpretorul caută pentru fișierul executabil asociat unei comenzi.

\$SHELL – numele interpretorului de comenzi curent.

\$PS1 – definește prompterul interetorului, implicit acesta este \$.

\$IFS – Internal Field Separator utilizat pentru separarea cuvintelor și a liniilor .

\$0 – numele fișierului de comenzi.

\$# – numărul de parametri transmiși fișierului de comenzi.

\$\$ – numărul de identificare prin care sistemul identifică *scriptul* în momentul execuției.

Parametrii poziționali – Dacă la apelarea unui fișier de comenzi (*script*) sunt folosiți parametri atunci create câteva variabile suplimentare. Chiar dacă nu se folosesc parametri variabila tot este creată variabila \$# dar va avea valoarea 0.

Variabile de acest tip sunt:

\$1, \$2, ... – parametrii trasmiși scriptului.

\$* - lista cu parametrii transmiși scriptului.

Acesarea conținutului variabileie se face prin prefixarea acesteia cu caracterul \$.

Ex:

\$ salut=Hello

\$ echo \$salut

Hello

Structuri de control – și în acest caz se întâlnesc structuri de control la fel ca și în cazul altor limbaje de programare.

If – testează rezultatul unui comenzi și în funcție de rezultatul acesteia se execută una dintre ramurile de declarații. Format:

```
if condiție
then declarații
[ elif condiție
then declarații]
[ else declarații]
fi
```

Se execută *condiție*; dacă codul returnat este zero se executa declarațiile ce urmeaza primului **then**. In caz contrar, se executa *condiția* de după **elif** si daca codul returnat este zero se executa declarațiile de dupa al doilea **then**.

Ex: - testarea tipului unui fișier ce este transmis ca și argument:

```
#!/bin/sh
if test -f $1
then echo $1 este un fisier obisnuit
elif test -d $1
then echo $1 este un catalog
else echo $1 este altceva
fi
```

for – folosit pentru trecerea printr-un anumit număr de valori(pot fi și șiruri de caractere). Sintaxa:

```
for nume [in valori]
do declarații
done
```

Variabila *nume* ia pe rand valorile din lista ce urmeaza lui **in**. Pentru fiecare valoare se executa ciclul **for**. Daca **in valori** este omis, ciclul se executa pentru fiecare parametru pozitional. Conditia poate fi si **in ***, caz in care variabila *nume* ia pe rand ca valoare numele intrarilor din directorul curent.

Ex:

```
#!/bin/sh
for i in $( ls );
do
echo $i
done
```

while – în cazul în care se dorește trecerea printr-un număr mai mare de valori este utilă folosirea acestei structuri de control. Sintaxă:

```
while condiție  
do declarații  
done
```

Ex:

```
#!/bin/sh  
i=0  
while [ $i -lt 5 ];  
do  
    echo $i  
    ((i=i+1))  
done
```

until – ciclul identic cu *for* dar în acest caz testarea condițiilor se face la sfârșit. Sintaxă:

```
until lista_1  
do lista  
done
```

case – această structură are sintaxa:

```
case condiție in  
sablon_1) declarații;;  
sablon_2) declarații;;  
...  
esac
```

De studiat!

```
#!/bin/sh  
echo "$#=$#"   
echo "$0='$0'"   
i=1  
while [ $# -gt 0 ]; do  
    echo "$$i='$1'"   
    i='expr $i + 1'   
    shift  
done
```

Se compară *condiție* cu fiecare din sabloanele prezente și se execută lista de comenzi unde se constată potrivirea.

Proceduri shell – reprezintă o modalitate de a grupa mai multe comenzi shell la un singur loc în vederea execuției lor ulterioare. Forma generală a unei astfel de proceduri este:

```
nume() { lista; }
```

Unde nume este numele procedurii (sau funcției), iar lista este o listă de comenzi, și *pipe*-uri care urmează să fie executate.

Ex:

```
#!/bin/sh  
function hello {  
    echo Hello World  
}  
function quit {  
    sleep 5  
    exit  
}  
hello  
quit
```

Câteva comenzi specifice interpretorilor de comenzi:

Test – este folosită pentru a testa o expresie dacă este adevărată atunci *test* va returna 0, în caz contrar va returna 1. Uzual *test* este folosit pentru a verifica tipul unor fișiere:

```
test -e fișier – returnează adevărat dacă fișierul există  
test -f fișier – returnează adevărat dacă fișierul este unul obișnuit  
test -d fișier – returnează adevărat dacă fișierul este catalog
```

Atenție! Comenzile *test condiție* și *[condiție]* sunt echivalente.

Export – face disponibile variabilele primite ca și argument în *subshell*-urile viitoare. Implicit variabilele create într-un shell nu sunt disponibile într-un shell creat ulterior.

Echo – afișează argumentele la ieșirea standard

Eval – realizează evaluarea și apoi execuția argumentelor

Exit – forțează întreruperea procesului curent care va returna ca și cod de ieșire valoarea primită ca și argument. Ex.: *Exit n*

Read – citește o linie din fișierul standard de intrare.

Break – Comanda de parasire a celei mai interioare bucle *for*, *while* sau *until* ce contine **break**. Daca *n* este specificat se iese din *n* bucle.

Continue – comanda permite trecerea la o nouă interacție a buclei *for*, *while* sau *until*.

Return – revenirea dintr-o funcție cu valoarea *n*, unde *n* este valoarea transmisă ca și argument. Dacă această valoare nu este precizată atunci codul returnat este cel al ultimei comenzi executate.

Set – folosit pentru activarea sau dezactivarea unor opțiuni sau pentru poziționarea unor parametrii poziționali.

Sleep – suspendă execuția pentru un număr *n* de secunde primit ca și argument.

Ex:

```
#!/bin/sh
```

```
function print {
```

```
    echo $1
```

```
}
```

```
print NUME:
```

```
read nume
```

```
print "Hello: $nume"
```

De studiat!

- *man bash*
- să se realizeze un script care să afișeze toate fișierele speciale dintr-un catalog dat.