2. Fisiere de comenzi

1. Înlănţuirea comenzilor

În Unix, dar şi, parţial, în DOS, majoritatea comenzilor folosesc aşa-numitele *fişiere standard de intrare* şi *fişiere standard de ieşire*. Acestea sunt concepte abstracte care reprezintă sursa din care comenzile îşi iau datele de intrare, respectiv destinaţia în care ele scriu rezultatele. Deci comenzile citesc din intrarea standard şi scriu în ieşirea standard. În mod normal, intrarea standard este reprezentată de tastatura calculatorului, iar ieşirea standard de către dispozitivul de afişare (monitorul).

Exemplu: comanda **sort** (existentă atât în DOS cât şi în UNIX) funcţionează după principiul enunţat. Dacă este apelată fără nici un parametru, ea va aştepta introducerea liniilor de text de la tastatură (*intrarea standard*), până la introducerea caracterului **^Z** urmat de **Enter** în MS-DOS, sau a caracterului **^D**, în Unix, după care va sorta liniile şi le va afişa în ordine pe ecran (*ieşirea standard*).

Intrarea şi ieşirea standard pot fi schimbate folosind operatorii de ***redirectare***. Redirectarea "conectează" intrarea sau ieşirea comenzilor la un fişier dat. Pentru redirectarea intrării se foloseşte operatorul '<', iar pentru redirectarea ieşirii operatorul '>'.

Exemplu: comanda următoare preia liniile care trebuie sortate din fişierul **date.txt**, iar rezultatele vor fi afişate pe ecran. Se redirectează, deci, numai intrarea standard:

```
sort < date.txt
```

Pentru a redirecta numai ieşirea, adică liniile de text să fie citite de la tastatură, dar rezultatul să fie scris într-un fişier, se foloseşte următoarea formă:

```
sort > ordonat.txt
```

Redirectările se pot combina, astfel încât liniile să fie citite dintr-un fişier, iar rezultatul să fie scris în altul:

sort < date.txt > ordonat.txt

Încercaţi aceste exemple.

Comenzile se pot şi *înlănţui*, în sensul că ieşirea generată de una devine intrare pentru alta. Pentru aceasta, se foloseşte operatorul '|', numit uneori operatorul *pipe* (conductă).

Exemplu: Comanda **more** realizează afişarea pagină cu pagină a datelor citite din intrarea standard. O construcţie de forma:

```
ls | more
```

face ca ieşirea lui **dir** să fie legată la intrarea lui **more**, astfel încât, efectul va fi afişarea pagină cu pagină a fişierelor din directorul curent.
Se pot înlănţui oricâte comenzi şi, prin urmare, pentru afişarea pagină cu pagină, ordonate alfabetic, a numelor tuturor fişierelor din directorul curent, se foloseşte comanda:

```
ls | sort | more
```

Încercaţi acest exemplu.

Observaţie: *Toate aceste considerente sunt valabile atât în DOS, cât şi în UNIX.*

2. Fişiere de comenzi

2.1 *Fişiere de comenzi UNIX*

2.1.1 Interpretoare de comenzi

In sistemul de operare Unix exista mai multe interpretoare de comenzi, selectabile de catre utilizator. Fiecare interpretor accepta un limbaj specific, astfel ca fisierele de comenzi care pot fi

scrise difera in functie de acest limbaj. Interpretorul de comenzi "standard" este **sh**, in Linux el fiind inlocuit cu interpretorul **bash**. In continuare, ne vom referi la comenzile si directivele specifice acestui interpretor, pentru detalii referitoare la celelalte variante putand fi consultate paginile de manual corespunzatoare.

Ca terminologie, in limba engleza interpretorul de comenzi mai este numit ***shell***, iar un program (fisier de comenzi) scris in limbajul recunoscut de acesta se numeste ***shell script***.

Lansarea in executie a unui fisier de comenzi se face fie tastand direct numele acestuia (el trebuie sa aiba dreptul de executie setat) sau apeland interpretorul de comenzi cu un parametru reprezantand numele fisierului de comenzi (exemplu: **sh *fisier***). In UNIX nu exista o "extensie" dedicata care sa identifice fisierele de comenzi, asa ca numele lor pot fi alese liber.

Comenzile UNIX pot fi grupate in liste de comenzi trimise spre executie intepretorului. Ele vor fi executate pe rand, o comanda fiind lansata in executie numai dupa ce comanda anterioara s-a terminat. Listele se formeaza scriind un sir de comenzi separate prin caracterul '**;**'. Exemplu:

```
cd exemplu ; ls -al
```

Daca intr-o lista, in loc de separatorul '**;**' se foloseste separatorul '**&&**', atunci o comanda nu va fi executata decat in cazul in care precedenta s-a terminat cu cod de succes (codul 0). Daca se foloseste '||', atunci conditia este ca precedenta sa se fi terminat cu cod de eroare (cod diferit de 0).

De remarcat faptul ca, atunci cand comenzile se inlantuie prin caracterul '|' (pipe) ele vor fi executate in paralel.

O comanda poate fi lansata si in fundal (in *background*), adica executia ei se va desfasura in paralel cu cea a interpretorului de comenzi, acesta afisand promptul imediat ce a lansat-o, fara sa-i mai astepte terminarea. Acest lucru se realizeaza adaugand caracterul '**&**' la sfarsitul liniei care contine comanda respectiva. Exemplu:

```
du > du.log &
```

2.1.2 Variabile de mediu

Variabilele de mediu pot sa contina ca valoare un sir de caractere. Atribuirea de valori se face astfel:

variabila=valoare

De exemplu

v=ABCD

va asigna variabilei cu numele *v* sirul "ABCD". Daca sirul asignat contine si spatii, el trebuie incadrat intre ghilimele.
ATENTIE: Nu se pune spatiu intre numele variabilei si semnul egal (si nici dupa semnul egal)! Altfel, interpretorul de comenzi va considera ca este vorba de o comanda cu parametri, si nu de o atribuire.

Referirea unei variabile se face prin numele ei, precedat de simbolul $. De exemplu,

echo $v

va determina afisarea textului ABCD.

In UNIX exista cateva variabile predefinite.

- variabile *read-only*, actualizate de interpretor:
  - **$?** - codul returnat de ultima comanda executata
  - $**$** - identificatorul de proces al interpretorului de comenzi
  - **$!** - identificatorul ultimului proces lansat in paralel
  - **$#** - numarul de argumente cu care a fost apelat fisierul de comenzi curent
  - **$0** - contine numele comenzii executate de interpretor
  - **$1, $2** ... - argumentele cu care a fost apelat fisierul de comenzi care se afla in executie
- variabile initializate la intrarea in sesiune:
  - **$HOME** - numele directorului "home" afectat utilizatorului;

- **$PATH** - caile de cautare a programelor;
- **$PS1** - prompter-ul pe care il afiseaza interpretorul atunci cand asteapta o comanda;
- **$PS2** - al doilea prompter;
- **$TERM** - tipul terminalului pe care se lucreaza.

Apelaţi comanda `set` fără parametri şi analizaţi variabilele de mediu definite în sistem.

2.1.3 Directive de control

Directivele de control ale interpretorului **sh** sunt structurile de limbaj care pot fi utilizate in scrierea de programe. In continuare vor fi prezentate cateva din cele mai folosite structuri de control.

Consultati pagina de manual a interpretorului de comenzi bash sau sh si analizati directivele si facilitatile pe care acesta le pune la dispozitie.

2.1.3.1. Instructiuni de decizie

- *Instructiunea **if***

if *lista1*
then *lista2*
else *lista3*
fi

if *lista1*
then *lista2*
elif *lista3*
then *lista4*
else *lista5*
fi

O comanda returneaza o valoare la terminarea ei. In general, daca o comanda s-a terminat cu succes ea va returna 0, altfel va returna un cod de eroare nenul.

In prima forma a comenzii **if**, se executa *lista1*, iar daca *si* ultima instructiune din lista returneaza codul 0 (succes) se executa *lista2*, altfel se executa *lista3*.

In a doua forma se pot testa mai multe conditii: daca *lista1* se termina cu succes, se va executa *lista2*, altfel se executa *lista3*. Daca aceasta se termina cu succes se executa *lista4*, altfel se executa *lista5*.

- *Instructiunea case*

case *cuvant* **in**
tipar1) *lista1*;;
tipar2) *lista2*;;
...
esac

Aceasta instructiune implementeaza decizia multipla. Sablonul *tipar* este o constructie care poate contine simbolurile ? si *, similara celor folosite la specificarea generica a numelor de fisiere. Comanda expandeaza (evalueaza) sirul *cuvant* si incearca sa il potriveasca pe unul din tipare. Va fi executata lista de comenzi pentru care aceasta potrivire poate fi facuta.

2.1.3.2 Instructiuni de ciclare

- *Instructiunea **while***

while *lista1*
do *lista2*
done

Se executa comenzile din *lista2* in mod repetat, cat timp lista de comenzi *lista1* se incheie cu cod

de succes.

- *Instructiunea **until***

until *lista1*
do *lista2*
done

Se executa comenzile din *lista2* in mod repetat, pana cand lista de comenzi *lista1* se incheie cu cod de succes.

- *Instructiunea **for***

for *variabila* [**in** *val1 val2 ...*]
do *lista*
done

Se executa lista de comenzi in mod repetat, variabila luand pe rand valorile *val1, val2, ...* Daca lipseste
cuvantul cheie **in**, valorile pe care le va lua pe rand *variabila* vor fi parametrii din linia de comanda pe care i-a primit fisierul de comenzi atunci cand a fost lansat in executie.

2.1.3.3. Alte comenzi

- **break** - permite iesirea din ciclu inainte de indeplinirea conditiei;
- **continue** - permite reluarea ciclului cu urmatoarea iteratie, inainte de terminarea iteratiei curente;
- **exec** *cmd* - comenzile specificate ca argumente sunt executate de interpretorul de comenzi in loc sa se creeze procese separate de executie; daca se doreste rularea comenzilor in procese separate ele se scriu direct, asa cum se scriu si in linia de comanda
- **shift** - realizeaza deplasarea argumentelor cu o pozitie la stanga ($2$1, $3$2 etc);
- **wait** [*pid*] - permite sincronizarea unui proces cu sfarsitul procesului cu pid-ul indicat sau cu

sfarsitul tuturor proceselor "fii";

- **expr** *expresie* - permite evaluarea unei expresii.

2.1.4 Substitutia comenzilor

Atunci cand intr-un *shell script* o comanda este incadrata de caractere ` (accent grav), interpretorul de comenzi va executa comanda, dupa care rezultatul acesteia (textul) va substitui locul comenzii in program. De exemplu, comanda

```
director=`pwd`
```

va atribui variabilei *director* rezultatul executiei comenzii **pwd**, adica sirul de caractere ce contine numele directorului curent.

Un exemplu de utilizare a substitutiei este construirea de expresii aritmetice:

```
contor=1
contor=`expr $contor + 1`
```

Aceasta secventa initializeaza o variabila *contor* la valoarea 1 (sir de caractere !) si apoi o "incrementeaza", in sensul ca la sfarsit, ea va contine sirul de caractere "2".
ATENTIE: Operatorii si operanzii sunt argumente diferite ale comenzii `expr`! Prin urmare, comanda

```
expr 1+2
```

este gresita. Corect este:

```
expr 1 + 2
```

2.1.5 Exemple

1.

```
while test -r fisier
do sleep 5
done
```

Programul testeaza daca "fisier" este accesibil la citire; in caz afirmativ programul se suspenda 5 secunde.

2.

```
until test -r fisier
do sleep 5
done
```

Programul se suspenda cate 5 secunde cat timp "fisier" nu este accesibil la citire.

3.

```
contor=$#
cmd=echo
while test $contor -gt 0
do
cmd="$cmd \$$contor"
contor=`expr $contor - 1`
done
eval $cmd
```

Programul realizeaza tiparirea argumentelor cu care a fost apelat, in ordine inversa. Se executa cate un ciclu pentru fiecare argument, incepand cu ultimul. Argumentul prelucrat este indicat de variabila contor, care pleaca de la valoarea $# (numarul de argumente) si se decrementeaza la fiecare parcurgere a ciclului. Programul construieste o comanda echo, la care adauga argumentele in ordine inversa (linia 5); sirul \$$contor are ca scop sa creeze un text format din $ si valoarea curenta a lui contor (\ semnifica faptul ca $ care urmeaza trebuie luat ca atare si nu ca si

caracterul de inceput al numelui unei variabile).

4.

```
contor=$#
cmd=echo
while true
do
cmd="$cmd \$$contor"
contor=`expr $contor - 1`
if test $contor -eq 0
then break
fi
done
eval $cmd
```

5.

```
contor=$#
cmd=echo
while true
do
cmd="$cmd \$$contor"
contor=`expr $contor - 1`
if test $contor -gt 0
then continue
fi
eval $cmd
exit
done
```

6.

```
if test $# -eq 0
then ls -l | grep '^d'
```

```
    else for i
    do
    for j in $i/*
    do
    if test -d $j
    then echo $j
    fi
    done
    done
    fi
```

Explicati functionarea programelor 4., 5. si 6. .

**Modificati programul din exemplul 6. pentru a-i extinde efectul asupra tuturor directoarelor din
subarborele directorului argument (sau al celui curent, in lipsa argumentelor).**

Nota: exemplele au fost preluate din lucrarea *Sisteme de Operare - indrumator de laborator*, de Roxana Chilintan si Sorin Babii, aparut in 1996 la Centrul de Multiplicare al Universitatii "Politehnica" Timisoara

Autor: ***Dan Cosma***
