Universitatea Politehnica București
Facultatea de Electronică, Telecomunicații și Tehnologia Informațiilor

Interpretatorul de comenzi in Linux

Lăcătușu Raluca – Cristina
431 A

1. Structura sistemului Unix

Codul care executa apelurile de sistem este sistemul de operare. Chiar daca sunt foarte folositoare si nu putem lucre fara ele, editoarele, compilatoarele, asambloarele si interpretatoarele de comenzi nu fac parte din sistemul de operare.

Sistemul UNIX este un sistem de operare cu destinatie generala, multiutilizator, multiproces, lucrind in regim time-sharring. Dintre caracteristicile care-l fac atractiv enumeram urmatoarele:

- este un sistem de operare cu o interfata relativ simpla, avand numeroase programe utilitare, care acopera o gama larga de aplicatii (retele, grafica, baze da date , inteligenta artificiala). Sunt disponibile compilatoare si medii de programare pentru limbaje de programare de nivel inalt (Basic, Fortran 77, Pascal, C, Lisp, Prolog)
- Unix are un sistem de fisiere arborescent, cu fisiere organizate simplu (sir de octeti) si grupate in directoare.
- Unix dispune de un sistem de intrare/iesire generalizat, care trateaza identic fisiere, periferice, memoria si comunicatiile interproces
- Unix are un mecanism simplu de gestiune a memoriei, folosind conceptul de memorie virtuala, asigurind un nivel inalt de multiprogramare.
- Unix dispune de un sistem eficient de protectie a informatiei, folosind parole si drepturi de acces
- Unix are o interfata utilizator simpla, numita SHELL, care se poate extinde la nivelul unui limbaj de programare de nivel inalt, de tip C, cu un interpretor de comenzi diferit de nucleu (utilizatorul poate incarca interpretorul de comenzi pe care-l doreste).
- Procesele Unix sunt reentrante (procesele care au partea de cod identica folosesc o singura copie a codului, rezidenta in memorie).

Sistemul de operare Unix, deci și Linux, conține un nucleu (partea rezidentă) și un număr mare de programe utilitare accesibile prin intermediul interpretatorului de comenzi. Utilizatorul are 3 niveluri de acces la system:

- Prin intermediul utilitarelor (nivel extranucleu);
- Prin funcții de bibliotecă standard a limbajului C (nivel intermediar);
- Prin directiva sistem (nivel scăzut).

Nucleul sau ”kernel-ul” este intermediarul între interfața furnizată de apelurile sistem și echipamentele fizice. Nucleul realizeză:

- Gestiunea fișierelor;
- Gestionarea memoriei;
- Planificarea proceselor pentru execuție.

1.1. Fisiere executabile

În sistemul de operare Linux există două tipuri de fișiere executabile:

- fișiere executabile de tip ELF (Executable Linking Format)
- fișiere executabile de tip script (fișiere de comenzi sau script shell

1.1.1. Fișierele executabile de tip ELF

Fișierele executabile de tip ELF sunt fișiere compilate binary, iar fișierele de comenzi sunt fișiere text care conțin seturi de comenzi specific interpretatorului de comenzi utilizat.

Primele caractere ale unui fișier executabil de tip ELF sunt chiar caracterele ELF, restul caracterelor reprezentând date criptate de compilator. În cazul fișierelor de comenzi, prima linie este întotdeauna numele interpretatorului de comenzi, precedat de caracterele ”#!”.

Fișierele executabile care se află în directoarele /bin, /sbin, /usr/bin și /usr/sbin nu trebuie să fie precedate de “./” în momentul apelării. Interpretorul de comenzi va căuta numele executabilului apelat de la tastatură în aceste directoare. Dacă executabilul nu este găsit, interpretorul va afișa mesajul de eroare: -bash: comanda: command not found.

1.1.2. Variabile

Variabilele sunt seturi de date care conțin diferite informații utilizabile de către aplicații sau de către alte variabile. În interpretoarele de comenzi din Linux există trei tipuri de variabile: **variabile sistem, variabile ale scripturilor shell și variabile definite de utilizatori.** Orice variabilă în Linux se notează cu “$nume variabilă”.

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

În interpretoarele de comenzi din Linux există mai multe tipuri de variabile: variabile predefinite sau speciale, parametrii poziţionali sau variabile utilizator.

Variabile predefinite sau speciale – la pornirea sistemului o serie de variable sunt iniţializate 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 fişierul executabil asociat unei comenzi.
- $SHELL – numele interpretorului de comenzi curent.
- $PS1 – defineşte prompterul interetorului, implicit acesta este $.
- $IFS – Internal Field Separator utilizat pentru separarea cuvintelor şi a liniilor .
- $0 – numele fişierului de comenzi.

- $# – numărul de parametri transmişi fişierului de comenzi.
- $$ – numărul de identificare prin care sistemul identifică scriptul în momentul execuţiei.

Parametrii poziţionali – Dacă la apelarea unui fişier de comenzi (script) sunt folosoţi parametrii atunci create câteva variabile suplimentare. Chiar dacă nu se folosesc parametrii variabila tot este creată variabila $# dar va avea valoarea 0.

Variabile de acest tip sunt:

- $1, $2, .. – parametrii trasmişi scriptului.
- $* - lista cu parametrii transmişi scriptului.

Accesarea conţinutului variabileie se face prin prefixarea acesteia cu caracterul $.

Ex:

```
$ salut=Hello
$ echo $salut
Hello
```

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

If – testează rezultatul unui comenzi şi în funcţie de rezultatul acesteia se execută una dintre ramurile de declaraţii.
Format:

```
if condiţie
then declaraţii
[ elif condiţie
then declaraţii]
[ else declaraţii]
fi
```

Se execută condiţie; dacă codul returnat este zero se executa declaraţiile ce urmeaza primului *then*. In caz contrar, se executa condiţia de după *elif* și dacă codul returnat este zero se execută declaraţiile de după al doilea *then*.

Ex: - testarea tipului unui fişier ce este transmis ca şi 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 şi şiruri de caractere).
Sintaxa:

for nume [în valori]
do declaraţii
done

Variabila nume ia pe rând valorile din lista ce urmeaza lui *in*. Pentru fiecare valoare se execută ciclul *for*. Dacă în valori este omis, ciclul se execută pentru fiecare parametru poziţional. Condiția poate fi și **in *** , caz in care variabila nume ia pe rand ca valoare numele intrărilor din directorul curent.

1.1.3. Scripturi SHELL

Un script shell este un fișier executabil care conține seturi de comenzi de bază și/sau comenzi specifice shell -urilor. Orice script shell este inițial un fișier text în care se scriu comenzile necesare. După editare, scripturile trebuie transformate în fișiere executabile utilizând comanda chmod. Lansarea în execuție a unui script shell se poate face în trei moduri:

- ./nume fișier argumente - dacă fișierul executabil se află în directorul current
- /locație/./nume fișier argumente - dacă fișierul nu se află în directorul current
- nume fișier argumente - dacă fișierul executabil se află într-unul din directoarele salvate în variabila $PATH.

Shell-ul este un program care conferă utilizatorului o interfață prin care se poate comunica cu sistemul de operare (la nivelul nucleului). Are rolul de a citi și interpreta comenzile primite de la utilizator și efectuează operații cu fișiere și procese conform acestor comenzi. În același timp, controlează modul și momentul în care sunt executate instrucțiunile.

Rolul interpretatorului de comenzi

Interpretorul de comenzi este un program care realizează o interfaţă între utilizator şi sistemul de operare, interfaţă care preia comezile utilizatorului şi le transmite sistemului de operare spre execuţie. Pentru Unix şi clonele acestuia (Linux) interpretoarele de comenzi sunt independente faţă de sistemul de operare ceea ce a dus la dezvoltarea unui număr mare de astfel de interpretoare. Interpretatorul de comenzi utilizeaza multe din caracteristicile sistemului de operare.

Atunci cand un utilizator se conecteaza in sistem, se porneste si un interpretor. Acesta vede terminalul ca intrare si iesire standard.

Primul interpretor de comenzi important, numit Bourne shell, a fost dezvoltat de Steven Bourne şi a fost inclus într-o distribuţie de Unix lansată în 1979. Acest interpretor de comenzi este cunoscut în cadrul sistemului de operare sub numele de sh. Un alt interpetor de comenzi utilizat pe scară largă este C shell cunoscut şi sub numele de csh. Csh a fost scris de Billy Joy ca parte integrantă a unei distribuţii de Unix numită BSD (Berkley System Distribution), şi a apărut la câţiva ani după sh. Numele de Csh l-a primit datorită asemănării sintaxei comenzilor sale cu cea a instrucţiunilor limbajului C, fapt ce îl

face mult mai uşor în utilizare celor familiarizaţi cu limbajul C. Toate aceste interpretoare au fost iniţial dezvoltate pentru Unix dar s-au dezvoltat versiuni ale acestora şi pentru Linux.

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

Un avantaj al interpretatorului bach este faptul ca îmbină o serie de opțiuni îmbunătățite specific limbajelor de programare (csh) cu soluții eficiente pentru comunicarea interactive utilizator – nucleu. În acelși timp, include editoare de text ușor de utilizat, evaluări matematice cu numere întregi în orice baza 2 – 64, poate rula majoritatea script-urilor fără modificări de sintaxă.

1.2. Interfața cu utilizatorul

Shell-ul este apelat de cele mai multe ori interactic, în sensul dialogului cu utilizatorul, interpretând și executând comenzile introduse de acesta. Utilizarea interactive a interpretatorului de comenzi constituie o sesiune de lucru. Pornirea interpretatorului de comenzi se poate face în mai multe feluri:

- în mod automat, la conectarea în sistem;
- de la consola calculatorului;
- de la distanță, prin ssh sau telnet, după autentificarea user-ului. Pentru fiecare utilizator este specificat un asemenea interpretator de comenzi implicit, administratorul sistemului putând stabili totuși interdicțiile privind folosirea vreunuia dintre ele;
- în cadrul altui program care necesită rularea interpretatorului de comenzi pentru a executa diferite comenzi la cererea utilizatorului, cum este cazul ferestrelor terminal. În termenii utilizați de Unix, fiecare program în execuție este un proces.

Dialogul cu un sistem Unix are loc prin intermediul unui terminal. După inițializarea sesiunii de lucru a unui utilizator (introducerea numelui de login și a parolei) se lansează interpretatorului de comenzi (shell). În cazul Unix-ului, interpretatorul de comenzi este un program obișnuit. Utilizatorul are posibiliatatea să aleagă ce program să fie lansat la începutul sesiunii.

După ce procesul de login a aflat numele utilizatorului și a verificat corectitudinea parolei, el schimbă catalogul curent cu directorul respectivului utilizator (indicat în fișierul /etc/passwd) și lansează în execuție prgramul al cărui nume figurează în ultimul camp al aceluiași intrări în etc/passwd. Acest program este un interpretor de comenzi.

Lansarea în execuție are loc fără crearea unui process nou, procesul login fiind practice înlocuit cu noul program. Interpretorul de comenzi lucrează asemănător interpretoarelor de comenzi din sistemele de operare cunoscute:

1. afișează un prompt
2. preia comanda și argumentele
3. dacă acea comandă este comandă internă, o execută

4. în caz contrar, numele comenzii este folosit pentru indentificarea unui fișier executabil care este încărcat și executat.
5. După terminarea execuției comenzii reapare prompt-ul care așteaptă o nouă comandă.

Spre deosebire de alte sisteme de operare, unde interpretorul de comenzi este un program cu privilegii, în Unix, shell-ul este considerat ca fiind un program obișnuit, neavând prioritate mai mare decât alte procese.

In cadrul interpretorului de comenzile , instructiunile contin de obicei trei segmente :

- numele comenzii ;
- optiunile comenzii ;
- fisierele de intrare sau iesire.

$ comanda {argument1} {argument2}...{argumentn} nume fișier

Desigur, nu toate cele trei segmente sunt obligatorii, ci în funcție de scopul comenzii opțiunile sau numele fișierelor pot să lipsească, numele comenzii fiind minimul necesar pentru a putea fi interpretată.

Comanda reprezintă o comanda internă (executată direct de shell) sau numele unui program executabil, cautat in urmatoarea secventa:

- în directorul cure
- în directorul /b
- în directorul /usr/b
- mesaj de eroare dacă nu este găsit în locurile specificate mai sus

Celelalte câmpuri ale comenzii definesc parametrii comenzii și se numesc *argumente*. Ele sunt separate prin blank-uri. Se pot introduce mai multe comenzi pe aceeași linie dacă le saparăm prin ";" . Multe dintre argumentele unei comenzi sunt nume de fisiere, de aceea au fost introduse niste *metacaractere* (caracterele *jocker*) pentru a ne putea referi la o multime de fisiere.

Metacaracterele sunt urmatoarele:

- * -orice sir de caracter (inclusiv sirul vi
- ? -orice caracte
- [...]-o multime de caracter
- - _-secventa lexicografică de caractere

Există 2 categorii de comenzi care pot fi apelate prin intermeniul interpretorului:

- comenzi interne care se găsesc implementate în cadrul shell-ului; exemple: *cd, kill*
- comenzi externe care se găsesc separate, fiecare fiind în fișierele executabile; ex: *passwd, ls, mail.*

Pentru a evidenția modul în care shell-ul asigură interfața cu sistemul de operare și prelucrarea comenzii primate, vom alege o comandă și vom comenta modul de lucru al

interpretorului de comenzi.

Părăsirea shell-ului interactiv (sesiunii curente de lucru) se realizează prin intermediul comenzii *exit* sau acționând combinația CTRL + a muta cursorul la începutul liniei, iar CTRL + e, la sfârșitul ei.

Shell-ul oferă facilitate numită *tab completion*, care dă posibilitatea introducerii parțiale a numelui unui fișier, la apăsarea tastei *Tab,* interpretorul completând cea ma bună potrivire cu textul.

Shell-ul nu scrie niciodată un rezultat pe ecran, scrie decât mesajele de eroare. Shell-ul citește un nume de comandă pe care o execută ca pe un process separat. Acest process tipărește rezultatele pe ecran.

2. Redirectarea ieşirii şi a intrării

Filozofia de bază Linux este aceea de tine lucrurile cât mai simple posibil, prin punerea la dispoziţia utilizatorului a unui număr mare de funcţii simple. Iar prin gruparea acestor funcţii înpreună se pot obţine comenzi mai complexe. Implicit, majoritatea comenzilor Linux sunt configurate astfel încât intrea acestora să o reprezinte tastatura iar datele de ieşire sunt trimise către monitor. În unele situaţii ar fi util dacă intrarea , respectiv ieşirea unei comenzi ar fi altele decât cele implicite. Unele comenzi au această facilitate, dar dacă pentru fiecare în parte s-ar prevedea acest lucru dimesiunea executabilelor ar creşte, de aceea acestă sarcină şi-a asumat-o sistemul de operare prin intermediul acestui mecanism de redirectare.

- Cu ajutorul semnului "<" se redirecteză fișierul standard de intrare. Acest lucru înseamnă că programul lansat de shell nu va mai lua datele de la tastatură, ci dintr-un fișier indicat după operatorul "<".
- Cu ajutorul operatorului ">" se redirectează ieșirea standard. Comanda *ls -l > fis* va lista conținutul directorului curent în fișierul "fis". Dacă fișierul există deja, prin folosirea operatorului ">>" în loc de ">" se va adauga informația la sfârșitul fișierului "fis"
- Un program poate fi lansat cu redirectarea simultană a intrării și ieșirii standard: *crypt '<'fis1> fis2***,** comandă care cifreaza fișierul fis1 si depune rezultatul în fis2.

Pipes sau conectarea proceselor – prin redirecţionarea ieşirii standard a unei comenzi într-un fişier, iar apoi redirecţionarea aceluiaşi fişier ca şi intrare standard pentru altă comandă se pun practic în legătură două procese prin intermediul unui fişier temporar. Acest lucru se poate face implicit de către sistemul de operare prin intermediul unui pipe. Pentru aceatsa se foloseşte operatorul |.

Controlul proceselor – Permite întreruperea unui poces şi de asemenea reluarea execuţiei acelui proces la un moment de timp ulterior. Întreruperea execuţiei unui proces se poate face direct de la tastatură prin secvenţa Ctrl-z, după care procesul va fi întrerupt iar uitlizatorul reprimeşte controlul putând executa alte comenzi.

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

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

  $fg %1

  cat >file.txt
- bg – se foloseşte pentru suspendarea unui proces, echivalent cu Ctrl-z
- jobs – afişează procesele suspendate.

Alte facilittăţi ale interpretorului de comenzi Linux:

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

3. Comenzi specifice

Câteva comezi specifice interpretorelor de comenzi:

- *Test* – este folosită pentru a 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 fişiere:
- *test –e fişier* – returnează adevărat dacă fişierul există
- *test –f fişier* – returnează adevărat dacă fişierul este unul obişnuit
- *test –f fişier* – returnează adevărat dacă fişierul este catalog

Comenzile test condiţie şi [condiţie] sunt echivalente.

- *Export* – face disponibile variabilele primite ca şi argument în subshell-urile viitoare. Implicit variabilele create într-un shell nu sunt disponibile într-un shell creat ulterior.
- *Echo* – afişează argumentele la ieşirea standard
- *Eval* – realizeză evaluarea şi apoi execuţia argumentelor
- *Exit* – forţează întreruperea procesului curent care va returna ca şi cod de ieşire valoarea primită ca şi argument. Ex: *Exit n*
- *Read* – citeşte o linie din fişierul 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 interaţie a buclei *for*, *while* sau *until*.
- *Return* – revenirea dintr-o funcţie cu valoarea n, unde n este valoarea transmisă ca şi argument. Dacă această valoare nu este precizată atunci

codul returnat este cel al ultimei comenzi executate.

- *Set* – folosit pentru activarea sau dezactivarea unor opţiuni sau pentru poziţionarea unor parametrii poziţionali.
- *Sleep* – suspendă execuţia pentru un număr n de secunde primit ca şi argument.

Ex:

```
#!/bin/sh
function print {
        echo $1
}
print NUME:
read nume
print "Hello: $nume"
```

4. Script-uri Bash

Comenzile bash pe care dorim ca shell-ul sa le execute pot si stocate in fisiere. Acestor fisiere li se dau drepturi de executie(cu comanda ***chmod***), dupa care pot fi executate ca oricare alta comanda. Fisierele continand comenzi ale unui limbaj de tip script, cum este cazul bash-ului, se mai numes si ***script-uri bash***.

De obicei, la inceputul fiecarui fisier script se stabileste shell-ul care va fi invocat de catre sistemul de operare pentru a se executa comenzile si constructiile bash. Pentru bash de exemplu vom avea : **# !/bin/bash**

Pentru a putea executa script-urile putem utiliza urmatoarele comenzi:

- (demo)$ ***.script [parametri]***
- (demo)$ ***./script[parametri]***
- (demo)$ ***bash script [parametri]***

6. Bibliografie

“Bash Guide for Beginners”- _Machtelt Garrels
“Bash Prompt HOWTO”-Giles Orr
“Slackware-Linux Essentials Book”
“Introduction to Linux”- _Machtelt Garrels
“Operating Systems Design and Implementation (3rd Edition)**”** – Andrew S. Tannenbaum
http://linux.about.com/od/linux101/a/desktop11.htm
http://193.226.6.120/Miclea/MSOA2/so/shell.swf
http://snap.nlc.dcccd.edu/reference/shellguide/shells.html
http://en.wikipedia.org/wiki/Bash
http://www.arachnoid.com/linux/shell_programming.html
http://ss64.com/bash/
http://linux.about.com/library/cmd/blcmdl1_bash.htm
http://linuxconfig.org/Bash_scripting_Tutorial
http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html