

Interpretorul de comenzi

Sisteme de Operare

Ciprian Oprea și Adrian Colea

Universitatea Tehnică din Cluj-Napoca
Departamentul Calculatoare

Cursul 2

Cuprins

- 1 Descriere generală
- 2 Comenzi Linux folosite frecvent
- 3 Utilizarea terminatorilor
 - Modul asincron
 - Redirectarea intrării și ieșirii
 - Înlănțuirea comenzilor
- 4 Script-uri (fișiere de comenzi)



Cuprins

- 1 Descriere generală
- 2 Comenzi Linux folosite frecvent
- 3 Utilizarea terminatorilor
 - Modul asincron
 - Redirectarea intrării și ieșirii
 - Înlănțuirea comenzilor
- 4 Script-uri (fișiere de comenzi)



Shell-ul

Definiție

Aplicație care permite utilizatorului să interacționeze cu SO.

- permite utilizatorului să introducă comenzi, pe care apoi le execută (direct, sau pornind alte programe)
- rulează în *user mode*, deci nu face parte din SO
- majoritatea SO-urilor sunt distribuite împreună cu un interpretor de comenzi
 - Linux: **sh**, *bash*
 - Windows: *Command Prompt*, *PowerShell*
 - OSX: *zsh*
- în general asociem *shell-ul* cu CLI (*Command-Line Interface*) dar definiția lui acoperă și GUI (*Graphical User Interface*)



Funcționalitate

- ❶ pornește atunci când utilizatorul se loghează
- ❷ afișează un *prompt*
- ❸ citește comanda tastată
- ❹ execută comanda tastată
 - comenzi interne → executate direct de interpretor
 - ex: `cd`, `echo`, `alias`
 - comenzi externe → se lansează alte procese pentru execuția lor
 - ex: `ls`, `cp`, `/usr/bin/zip`
- ❺ revine la *prompt* (pasul 2)



Algoritmul *shell*-ului (simplificat)

SHELL()

```
1  while TRUE
2      PRINT(PROMPT)
3      command_string = READ()
4      name, params, extra = PARSE(command_string)
5      if IS-INTERNAL(name)
6          EXECUTE-INTERNAL(name, params, extra)
7      else
8          prog_path = SEARCH-PATH(name)
9          proc = CREATE-PROCESS(prog_path, params, extra)
10         if not extra.asynchronous
11             WAIT-TERMINATION(proc)
```



Sintaxa unei comenzi

`user@machine:~$ ls -l --color=never dir1 >list.txt`

Diagram illustrating the components of a command line:

- `user@machine:~$` is labeled **prompt**.
- `ls` is labeled **nume comandă**.
- `-l --color=never` is labeled **opțiuni**.
- `dir1` is labeled **parametri**.
- `>list.txt` is labeled **terminatori**.

- prompt-ul e afișat de *shell*
- primul cuvânt tastat reprezintă numele comenzii
- opțiunile sunt tot parametri (*shell*-ul nu le tratează diferit)
- spațiul e pe post de separator
 - "parametru compus" (cu ghilimele sau apostroafe)
 - parametru\ compus



Variabile

- declararea variabilelor
 - se scrie nume=valoare fără spații
 - exemple
 - NAME=Ion
 - TEXT1="Ana are mere"
 - MY_DIR='pwd' – dacă folosim apostroafe inverse, se va rula comanda scrisă între ele, iar variabila va conține rezultatul



Variabile

- declararea variabilelor
 - se scrie nume=valoare fără spații
 - exemple
 - `NAME=Ion`
 - `TEXT1="Ana are mere"`
 - `MY_DIR='pwd'` – dacă folosim apostroafe inverse, se va rula comanda scrisă între ele, iar variabila va conține rezultatul
- utilizarea variabilelor
 - punem `$` în fața numelui pentru a extrage valoarea
 - exemple
 - `echo $TEXT1`
 - `echo "My name is $NAME"` – va afișa "My name is Ion"
 - `echo 'My name is $NAME'` – va afișa "My name is \$NAME"



Variabile

- declararea variabilelor
 - se scrie nume=valoare fără spații
 - exemple
 - `NAME=Ion`
 - `TEXT1="Ana are mere"`
 - `MY_DIR='pwd'` – dacă folosim apostroafe inverse, se va rula comanda scrisă între ele, iar variabila va conține rezultatul
- utilizarea variabilelor
 - punem `$` în fața numelui pentru a extrage valoarea
 - exemple
 - `echo $TEXT1`
 - `echo "My name is $NAME"` – va afișa "My name is Ion"
 - `echo 'My name is $NAME'` – va afișa "My name is \$NAME"
- calcule aritmetice `RESULT=$(((VAR1 + 5) / VAR2))`



Variabile de mediu

- o variabilă obișnuită este “vizibilă” doar în terminalul curent
 - nu este accesibilă din alte terminale
 - nu este accesibilă proceselor pornite din terminalul curent
- dacă dorim ca o variabilă să fie accesibilă și altor procese lansate din terminalul curent, trebuie să o exportăm
 - o variabilă exportată se numește **variabilă de mediu** (eng. *environment variable*)
 - exemple
 - `export VAR1`
 - `export VAR2="some value"`
- comanda `env` afișează toate variabilele de mediu din terminal



Rularea comenzilor externe

- comenzile externe sunt programe
- se pot specifica prin
 - cale completă (ex. `/usr/bin/ls`)
 - cale relativă (ex. `./myprog` – rulăm un program din folderul curent; nu va fi găsit fără `./`)
 - nume (ex. `ls`)



Rularea comenzilor externe

- comenzile externe sunt programe
- se pot specifica prin
 - cale completă (ex. `/usr/bin/ls`)
 - cale relativă (ex. `./myprog` – rulăm un program din folderul curent; nu va fi găsit fără `./`)
 - nume (ex. `ls`)
- de unde știe interpretorul de comenzi unde se găsește programul specificat doar prin nume?



Rularea comenzilor externe

- comenzile externe sunt programe
- se pot specifica prin
 - cale completă (ex. `/usr/bin/ls`)
 - cale relativă (ex. `./myprog` – rulăm un program din folderul curent; nu va fi găsit fără `./`)
 - nume (ex. `ls`)
- de unde știe interpretorul de comenzi unde se găsește programul specificat doar prin nume?
- pe sistem există niște căi (eng. *path-uri*) comune unde se găsesc programele
 - `/bin`, `/usr/bin`, `/usr/local/bin`, ...
- var. de mediu `$PATH` conține toate aceste căi separate prin `:`
- interpretorul caută programul pe rând, în fiecare



Cuprins

- 1 Descriere generală
- 2 Comenzi Linux folosite frecvent**
- 3 Utilizarea terminatorilor
 - Modul asincron
 - Redirectarea intrării și ieșirii
 - Înlănțuirea comenzilor
- 4 Script-uri (fișiere de comenzi)



Comenzi pentru lucrul cu fișiere

- crearea unui fișier

```
touch file_name
```

- ștergerea unui fișier

```
rm file_name
```

- copierea unui fișier

```
cp src_name dst_name
```

- mutarea / redenumirea unui fișier

```
mv src_name dst_name
```

- concatenarea / afișarea fișierelor

```
cat file_name
```

```
cat file1 file2
```




Comenzi pentru lucrul cu directoare

- crearea unui director

```
mkdir dir_name
```

- ștergerea recursivă a unui director

```
rm -R dir_name
```

- afișarea conținutului unui director

```
ls
```

```
ls -l /path/to/dir
```

- căutarea într-un director

```
find . -name "*.txt"
```



Comenzi pentru procesarea conținutului

- numărarea cuvintelor / liniilor / caracterelor (*word count*)

```
wc file_name
```

```
wc -l file_name
```

- filtrarea conținutului

- afișarea liniilor care conțin un *pattern* dat

```
grep pattern file_name
```

- afișarea liniilor care **nu** conțin un *pattern* dat

```
grep -v pattern file_name
```



Comenzi pentru lucrul cu procese

- afișarea tuturor proceselor

```
ps -A
```

- închiderea forțată a unui proces, după PID

```
kill -9 1234
```

- închiderea forțată a proceselor cu un anumit nume

```
killall -9 proc_name
```

- afișarea interactivă a proceselor

```
top  
htop
```



Cuprins

- 1 Descriere generală
- 2 Comenzi Linux folosite frecvent
- 3 Utilizarea terminatorilor**
 - Modul asincron
 - Redirectarea intrării și ieșirii
 - Înlănțuirea comenzilor
- 4 Script-uri (fișiere de comenzi)



Modurile de execuție sincron și asincron

- modul **sincron**
 - implicit
 - interpretorul așteaptă terminarea comenzii curente
 - doar ulterior afișează prompt-ul și așteaptă comenzi noi
- modul **asincron**
 - se activează adăugând '&' la finalul comenzii
 - interpretorul afișează prompt-ul imediat, nu așteaptă terminarea comenzii curente
 - comanda din *background* împarte terminalul cu alte comenzi
 - mesajele afișate de mai multe programe se pot “amesteca”
 - are sens să rulăm pe fundal programe care nu (prea) scriu în terminal
- exemplu

```
gnome-text-editor file.txt &
```



Intrările și ieșirile standard

- fiecare aplicație care are un terminal asociat poate
 - citi intrări de la tastatură
 - afișa pe ecran
- fiecare aplicație are în mod implicit trei descriptori de fișiere (asociați terminalului în care rulează)
 - **0 (STDIN)** – implicit tastatura
 - **1 (STDOUT)** – implicit ecranul
 - **2 (STDERR)** – implicit ecranul



Redirectarea intrării standard

- se face adăugând la finalul comenzii simbolul '<' urmat de numele fișierului
- în loc să se citească de la tastatură (STDIN), se va citi din fișierul dat
- ex. un apel de scanf va ajunge să apeleze read(0, ...)
 - descriptorul 0 nu va mai indica tastatura, ci fișierul dat
- exemple

```
read var1 var2 < file1
```

```
cat < file2.txt
```

```
sort 0< list.txt
```



Redirectarea ieșirii standard

- se adaugă '>' urmat de numele fișierului la finalul comenzii
- dacă nu dorim să suprascriem ci să adăugăm date, folosim '>>'
- scrierile care în mod normal ajung la ecran (STDOUT) vor ajunge în fișier
- ex. un apel de `printf` va ajunge să apeleze `write(1, ...)`
 - prin redirectare, descriptorul 1 va indica fișierul dat
- exemple

```
ls > file_names.txt
```

```
cat <file1 >file2
```

```
find / -name "*.c" -print 1>c_files.txt
```




Redirectarea erorilor

- se adaugă '2>' urmat de numele fișierului la finalul comenzii
- are sens pentru comenzi care folosesc STDERR
 - ex. un program C care apelează perror sau
fprintf(stderr, ...)
 - ambele funcții ajung să apeleze write(2, ...)
- exemplu

```
ls -R / >reachable_files.txt 2>/dev/null
```



Mai multe comenzi pe aceeași linie

- se pot scrie mai multe comenzi pe aceeași linie, cu ajutorul separatorilor
- `cmd1 SEP cmd2 SEP ... SEP cmdn`
- separatorii pot fi `' ; '`, `' && '`, `' || '` sau `' | '`
 - `cmd1 ; cmd2` – se execută întâi `cmd1` apoi `cmd2`
 - `cmd1 && cmd2` – se execută `cmd2` doar dacă `cmd1` s-a încheiat cu succes
 - `cmd1 || cmd2` – se execută `cmd2` doar dacă `cmd1` a eșuat
- exemple

```
test -f file.txt && rm file.txt
```

```
cd /home/user1 || cd /home/user2
```



Înlănțuirea comenzilor prin *pipe*

- $\text{cmd}_1 \mid \text{cmd}_2$
- STDOUT de la cmd_1 devine STDIN pentru cmd_2
- pentru a face legătura, se folosește un fișier special, de tip *pipe*
- tehnică foarte utilă pentru *one liners* (obținerea de funcționalități complexe cu o singură linie)
- exemple

```
cat file1.txt | sort | less
```

```
grep "filter" file2.txt | wc -l
```



Cuprins

- 1 Descriere generală
- 2 Comenzi Linux folosite frecvent
- 3 Utilizarea terminatorilor
 - Modul asincron
 - Redirectarea intrării și ieșirii
 - Înlănțuirea comenzilor
- 4 Script-uri (fișiere de comenzi)



Script-uri *shell*

Un script este un fișier text ce conține mai multe comenzi.

- se folosesc pentru automatizarea unor sarcini
- limbajul de scripting este un limbaj de programare
 - structuri de control (`if`, `while`, `for`)
 - variabile, funcții



Script-uri *shell*

Un script este un fișier text ce conține mai multe comenzi.

- se folosesc pentru automatizarea unor sarcini
- limbajul de scripting este un limbaj de programare
 - structuri de control (if, while, for)
 - variabile, funcții

```
#!/bin/bash
for FNAME in *.tmp
do
    if test -f $FNAME
    then
        rm $FNAME
    fi
done
```



Accesul la parametrii din linia de comandă

- \$0: numele script-ului (comenzii)
- \$1, \$2, ..., \${10}, ...: parametrii
- \$#: numărul parametrilor
- \$@: string-ul cu toți parametrii



Accesul la parametrii din linia de comandă

- \$0: numele script-ului (comenzii)
- \$1, \$2, ..., \${10}, ...: parametrii
- \$#: numărul parametrilor
- \$@: string-ul cu toți parametrii

```
#!/bin/bash
echo "we have $# parameters. the first one is $1"
for PARAM in $@
do
    echo $PARAM
done
```




Accesul la parametrii din linia de comandă în C

- `argc`: numărul argumentelor
- `argv[0]`: numele comenzii
- `argv[1]`, `argv[2]`, ... `argv[argc-1]`: parametrii

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("The program name: %s\n", argv[0]);

    for(i=1; i<argc; i++){
        printf("The i-th param: %s\n", argv[i]);
    }
    return 0;
}
```



Accesul la variabile și la variabile de mediu

- la fel ca în comenzi, prefixăm variabila cu '\$'
- exemplu: afișarea directorilor din \$PATH pe linii separate

```
#!/bin/bash
NR_DIRS='echo $PATH | grep -o ":" | wc -l'
NR_DIRS=$(( NR_DIRS + 1 ))
for i in $(seq 1 $NR_DIRS)
do
    echo $PATH | cut -d : -f $i
done
```



Accesul la variabile de mediu în C

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv, char **envp)
{
    int i=0;
    char *path;
    printf("The env. vars of the process %s are:\n", argv[0]);
    while(envp[i]) {
        printf("envp[%d]: %s\n", i, envp[i]);
        ++i;
    }
    path = getenv("PATH");
    if(path != NULL) printf("The current path is: %s\n", path);
    return 0;
}
```



Considerente de securitate

- poate fi păcălit utilizatorul să ruleze codul atacatorului?
- de ce trebuie să folosim "."/" ca să rulăm un program din directorul curent?



Considerente de securitate

- poate fi păcălit utilizatorul să ruleze codul atacatorului?
- de ce trebuie să folosim "."/" ca să rulăm un program din directorul curent?

Scenariu

- atacatorul crează un executabil malițios numit 1s
- îl pune într-un folder pe care utilizatorul e probabil să îl viziteze
- ???



Considerente de securitate

- poate fi păcălit utilizatorul să ruleze codul atacatorului?
- de ce trebuie să folosim "."/" ca să rulăm un program din directorul curent?

Scenariu

- atacatorul crează un executabil malițios numit `ls`
- îl pune într-un folder pe care utilizatorul e probabil să îl viziteze
- nu va fi rulat, se comenzile se caută doar în `$PATH`



Considerente de securitate

- poate fi păcălit utilizatorul să ruleze codul atacatorului?
- de ce trebuie să folosim "."/" ca să rulăm un program din directorul curent?

Scenariu

- atacatorul crează un executabil malițios numit `ls`
- îl pune într-un folder pe care utilizatorul e probabil să îl viziteze
- nu va fi rulat, se comenzile se caută doar în `$PATH`
- dar dacă un executabil ar fi căutat în directorul curent doar dacă nu există în `$PATH`?



Considerente de securitate

- poate fi păcălit utilizatorul să ruleze codul atacatorului?
- de ce trebuie să folosim "."/" ca să rulăm un program din directorul curent?

Scenariu

- atacatorul crează un executabil malițios numit `ls`
- îl pune într-un folder pe care utilizatorul e probabil să îl viziteze
- nu va fi rulat, se comenzile se caută doar în `$PATH`
- dar dacă un executabil ar fi căutat în directorul curent doar dacă nu există în `$PATH`?
 - atacatorul ar putea folosi nume de comenzi scrise greșit
 - vezi `sl`



Exit status

- orice program sau script are un status de ieşire (la terminare)
 - 0: succes
 - \neq 0: eșec
- ieşirea cu un *exit status*
`exit n`
- verificarea statusului
 - \$? conține statusul ultimei comenzi rulate
 - if consideră true o comandă cu status 0 (invers față de C)

```
if cd $1
then
    echo "$1 is accessible"
else
    echo "$1 is not accessible or does not exist"
fi
```



Navigarea prin sistemul de fișiere în Linux

- ierarhie de fișiere și directoare
- folderul rădăcină se numește "/"
- folderul de lucru al fiecărui utilizator este de obicei de forma `/home/username/`
- în orice folder există două subfoldere speciale
 - "." - referință la folderul curent
 - ".." - referință la folderul părinte
- numele fișierelor ascunse începe cu '.'
 - comanda `ls` nu le afișează decât dacă se folosește opțiunea `-a`

Bibliografie

- [LAB] C. Oprea și A. Colea, *Sisteme de operare - îndrumător de laborator*, UTPress, 2021, Capitolul 2
- [MOS] A. Tanenbaum și H. Bos, *Modern Operating Systems*, 4th Edition, Pearson Education, 2015, Secțiunea 1.5.6