

*METODE AVANSATE DE GESTIUNE A
DOCUMENTELOR ȘI A SISTEMELOR
DE CALCUL
- CURS 5 -*

Asist. Diana – Florina Șotropa

www.cs.ubbcluj.ro/~diana.sotropa

LINUX

Redirecționarea I/O

- Intrările sunt generate de la tastatură (dispozitiv standard de intrare = **stdin**)
- Ieșirile sunt afișate pe ecran (dispozitiv standard de ieșire = **stdout**)
- Redirecționările sunt făcute prin **>** și **|**

```
nancy:~> cat test1
some words

nancy:~> cat test2
some other words

nancy:~> cat test1 test2 > test3

nancy:~> cat test3
some words
```



LINUX

Redirecționarea I/O

- Redirecționarea intrărilor se face folosind ca operand simbolul <
- Combinarea redirecționărilor:

```
andy:~> mail mike@somewhere.org < to_do
```

- Operandul >> - nu suprascrie conținutul ci adaugă la finalul fișierului

```
mike:~> less --help | grep -i examine
:e [file]      Examine a new file.
:n            * Examine the (N-th) next file from the command line.
:p            * Examine the (N-th) previous file from the command line.
:x            * Examine the first (or N-th) file from the command line.
```

```
mike:~> cat wishlist
more money
less work
```

```
mike:~> date >> wishlist
```

```
mike:~> cat wishlist
more money
less work
Thu Feb 28 20:23:07 CET 2002
```



LINUX

Redirecționarea I/O

- Sunt trei tipuri de I/O. Fiecare având propriul identificator, denumit descriptor de fișiere:
 - Intrarea standard: 0
 - ieșirea standard: 1
 - Eroarea standard: 2

```
[nancy@asus /var/tmp]$ ls 2> tmp

[nancy@asus /var/tmp]$ ls -l tmp
-rw-rw-r-- 1 nancy nancy 0 Sept  7 12:58 tmp

[nancy@asus /var/tmp]$ ls 2 > tmp
ls: 2: No such file or directory
```



LINUX

Editoare de text

- Emacs
 - CTRL + H – prezintă opțiunile pe care le avem la dispoziție
 - Se pot personaliza anumite definiții ale comenzilor Emacs
- Vi sau Vim
 - :n – mutarea pe linia n a fișierului
 - :w – va salva fișierul
 - :q – ieșire din editor
 - :q! – forțează ieșirea fără salvare
 - :wq – salvează modificările și iese din editor



LINUX

- Afișarea conținutului unui fișier
 - **CAT** – trimite fișierele la ieșirea standard
 - **MORE** – afișează textul la ieșirea standard pagină cu pagină
 - **LESS** – similar cu MORE, dispune de mai multe facilități care permit evidențierea criteriilor de căutare, derulare
 - **HEAD, TAIL** – afișează primele/ultimele n linii dintr-un fișier
 - Opțiunea **-t** a comenzii TAIL se arată continuu ultimele n linii ale unui fișier care are un conținut în permanentă schimbare

```
tony:~> tail -10 .bash_history
locate configure | grep bin
man bash
cd
xawtv &
grep usable /usr/share/dict/words
grep advisable /usr/share/dict/words
info quota
man quota
echo $PATH
frm
```



LINUX

SORT

- Comanda SORT aranjează implicit liniile în ordine alfabetică

```
thomas:~> cat people-I-like | sort  
Auntie Emmy  
Boyfriend  
Dad  
Grandma  
Mum  
My boss
```



LINUX

GREP

- GREP scanează ieșirile, linie cu linie, în căutarea tiparelor;
- Toate liniile care conțin tiparul vor fi afișate la ieșirea standard;
- Comportamentul poate fi inversat prin opțiunea **-v**



LINUX GREP

```
cathy ~> grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
```

```
cathy ~> grep -n root /etc/passwd
1:root:x:0:0:root:/root:/bin/bash
12:operator:x:11:0:operator:/root:/sbin/nologin
```

- Exemplul 1: ce fișiere au fost modificate în februarie?
`jenny:~> ls -la | grep Feb`
- Comanda GREP face deosebire între literele mari și cele mici
- Opțiunea `-i` se folosește pentru ca această comandă să nu facă diferența între majuscule și litere mici
- Opțiunea `--color` ajută la evidențierea tiparelor de căutare în liniile prea lungi
- Opțiunea `--after-context` afișează numărul liniilor după ultima linie care se potrivește
- Opțiunea `-r` caută recursiv în toate subdirectoarele unui director
- Opțiunea `-n` prefixează fiecare linie cu numărul ei



LINUX

Expresii regulare

- Regex = prezintă o modalitate de identificare a unui șir de caractere dintr-un text dat conform anumitor reguli;
- Reguli
 1. Blocul fundamental reprezintă un singur caracter și se selectează pe sine
 2. O expresie paranteză pătrată reprezintă o listă de caractere cuprinse între [și] și descrie un singur caracter din acea listă
 3. Dacă primul caracter dintre [] este ^ descrie orice caracter care nu se regăsește în listă
 4. În interiorul unei expresii paranteză pătrată o paletă de valori este reprezentată de două caractere separate prin - și identifică orice caracter care se găsește între cele două
 5. ^ în afara parantezelor drepte identifică începutul unei linii (dacă este primul caracter din regex)
 6. \$ identifică sfârșitul unei linii (dacă este ultimul caracter din regex)



LINUX

Expresii regulare

- Regex = prezintă o modalitate de identificare a unui șir de caractere dintr-un text dat conform anumitor reguli;
- Reguli
 7. `\` urmat de un caracter special selectează caracterul special respectiv (`.`, `*`, `[`, `\`)
 8. `?` Selectează o sau 1 caractere anterioare
 9. `*` selectează o sau mai multe caractere anterioare
 10. `+` selectează caracterul anterior o dată sau de mai multe ori
 11. `{n}` elementul precedent este selectat de exact n ori
 12. `{n,}` elemenul precedent este selectat de cel puțin n ori
 13. `{n,m}` elementul precedent este selectat de cel puțin n ori, dar nu mai mult de m ori



LINUX

Expresii regulare

- Exemple:
 - [0123456789] – reprezintă o singură cifră
 - [^0123456789] – reprezintă orice caracter care nu este cifră
 - ^ab – linie care începe cu ab
 - ab\$ - linie care se termină cu ab
 - ^\$ - linie goală
 - ^a[a-z0-9] – linie care începe cu "a" urmat de orice caracter între "a" și 'z' sau "0" și "9"

GREP `[01][0-9]\{12\}`

- găsirea unui șir de caractere de tip CNP



LINUX GREP

```
cathy ~> grep ^root /etc/passwd  
root:x:0:0:root:/root:/bin/bash
```

- Găsirea liniilor care încep cu **root** în fișierul /etc/passwd

```
cathy ~> grep '\<c...h\>' /usr/share/dict/words  
catch  
clash  
cloth  
coach  
couch  
cough  
crash  
crush
```

- Găsirea liniilor care conțin cuvinte care încep cu **c** și se termină cu **h** din fișierul dat ca parametru și care au exact 5 litere

```
cathy ~> grep '\<c.*h\>' /usr/share/dict/words  
caliph  
cash  
catch
```

- Găsirea liniilor care conțin cuvinte care încep cu **c** și se termină cu **h** din fișierul dat ca parametru



LINUX SED Optiuni

Citește datele de intrare, editează liniile și scrie rezultatul

Comportamentul implicit al lui SED este sa afiseze fiecare linie pe care o parseaza, dupa care afiseaza rezultatul in urma executiei comenzii.

Optiuni:

- **a** = adaugă textul sub linia curentă
- **d** = șterge textul
- **i** = inserează text deasupra liniei curente
- **p** = afișează text-ul modificat după linia care respectă expresia regulară
- **s** = caută și înlocuiește un text
- **-e SCRIPT** = adaugă comenzile din SCRIPT setului de comenzi pe care le execută odată cu procesarea datelor de intrare
- **-n** = mod silențios
- **-V** = afișează versiunea
- **-i** = editeaza fisierul sursa
- Simbolul & denota intregul sir de caractere care respecta o expresia regulara



LINUX SED

- **SED -e "s/old/new"** - înlocuiește prima apariție a șirului de caractere **old** cu **new**
- **SED -e "s/old/new/g"** - înlocuiește toate aparițiile șirului de caractere **old** cu **new**
- **old** – poate fi o expresie regulată

```
sed -e '/^$/d' $filename – se sterg liniile goale
```

```
sed -n '/xzy/p' $filename – se afiseaza doar liniile care contin xzt
```

```
sed -e 's./*/& ADD DUPA PATTERN' $filename – se inlocuieste linia
```



LINUX SED

```
sed -n '/diana/p' pas.txt | sed '1d'
```

- Se afiseaza toate liniile care contin diana, in afara de prima

Notatie	Efect
8d	Sterge a 8a linie din input
1,/^\\$/d	Sterge toate liniile pana la intalnirea unei linii goale
s/*\$/	Sterge spatiile de la sfarsitul fiecarei linii
s/oo*/o/g	Restrange toate caracterele o consecutive intr-un singur caracter
echo "Working on it." sed -e '1i How far are you along?'	Afiseaza "How far are you along?" ca prima linie si apoi insereaza linia "Working on it".
echo "Working on it." sed -e '1a How far are you along?'	Afiseaza "Working on it" ca prima linie si apoi insereaza linia "How far are you along?".
5i 'Linux is great.' file.txt	Insereaza 'Linux is great.' pe a 5-a linie din fisier
/SO/i 'Linux is great.' file.txt	Insereaza 'Linux is great.' inainte de liniile care contin SO
/GUI/d	Sterge toate liniile care contin GUI
s/GUI//g	Sterge toate aparitiile cuvantului GUI, nemodificand in rest continutul liniei



LINUX SED

```
sandy ~> cat -n example
 1 This is the first line of an example text.
 2 It is a text with errors.
 3 Lots of errors.
 4 So much errors, all these errors are making me sick.
 5 This is a line not containing any errors.
 6 This is the last line.
```

```
sandy ~> sed '/errors/p' example
This is the first line of an example text.
It is a text with errors.
It is a text with errors.
Lots of errors.
Lots of errors.
So much errors, all these errors are making me sick.
So much errors, all these errors are making me sick.
This is a line not containing any errors.
This is the last line.
```

- Găsește toate liniile care conțin cuvântul **errors** și le afișează
- Dacă s-ar folosi și opțiunea **-n** atunci s-ar afișa doar liniile care corespund căutării.

```
sandy ~> sed -n '/errors/p' example
It is a text with errors.
Lots of errors.
So much errors, all these errors are making me sick.
```



LINUX SED

```
sandy ~> cat -n example
 1 This is the first line of an example text.
 2 It is a text with errors.
 3 Lots of errors.
 4 So much errors, all these errors are making me sick.
 5 This is a line not containing any errors.
 6 This is the last line.
```

```
sandy ~> sed -n '/errors/p' example
It is a text with errors.
Lots of errors.
So much errors, all these errors are making me sick.
```

- Găsește toate liniile care conțin cuvântul **errors**
- Pentru a afișa doar liniile care se potrivesc căutării



LINUX SED

```
sandy ~> cat -n example
 1 This is the first line of an example text.
 2 It is a text with errors.
 3 Lots of errors.
 4 So much errors, all these errors are making me sick.
 5 This is a line not containing any errors.
 6 This is the last line.
```

```
sandy ~> sed '/errors/d' example
This is the first line of an example text.
This is a line not containing any errors.
This is the last line.
```

- Ștergerea liniilor care nu conțin cuvântul **errors**



LINUX SED

```
sandy ~> cat -n example
 1 This is the first line of an example text.
 2 It is a text with errors.
 3 Lots of errors.
 4 So much errors, all these errors are making me sick.
 5 This is a line not containing any errors.
 6 This is the last line.
```

```
sandy ~> sed -n '/^This.*errors.$/p' example
This is a line not containing any errors.
```

- Afișarea liniilor care încep cu o anumită expresie și se termină cu alta



LINUX SED

```
sandy ~> cat -n example
 1 This is the first line of an example text.
 2 It is a text with errors.
 3 Lots of errors.
 4 So much errors, all these errors are making me sick.
 5 This is a line not containing any errors.
 6 This is the last line.
```

```
sandy ~> sed '2,4d' example
This is the first line of an example text.
This is a line not containing any errors.
This is the last line.
```

```
sandy ~> sed '3,$d' example
This is the first line of an example text.
It is a text with errors.
```

- Eliminarea unor linii



LINUX SED

- Find & Replace
 - Prima apariție
 - Toate aparițiile

```
sandy ~> cat -n example
 1 This is the first line of an example text.
 2 It is a text with errors.
 3 Lots of errors.
 4 So much errors, all these errors are making me sick.
 5 This is a line not containing any errors.
 6 This is the last line.
```

```
sandy ~> sed 's/errors/errors/' example
This is the first line of an example text.
It is a text with errors.
Lots of errors.
So much errors, all these errors are making me sick.
This is a line not containing any errors.
This is the last line.
```

```
sandy ~> sed 's/^/> /' example
> This is the first line of an example text.
> It is a text with errors.
> Lots of errors.
> So much errors, all these errors are making me sick.
> This is a line not containing any errors.
> This is the last line.
```



LINUX SED

```
sandy ~> cat -n example
 1 This is the first line of an example text.
 2 It is a text with errors.
 3 Lots of errors.
 4 So much errors, all these errors are making me sick.
 5 This is a line not containing any errors.
 6 This is the last line.
```

```
sandy ~> sed 's/$/EOL/' example
This is the first line of an example text.EOL
It is a text with errors.EOL
Lots of errors.EOL
So much errors, all these errors are making me sick.EOL
This is a line not containing any errors.EOL
This is the last line.EOL
```

- Inserarea unui șir de caractere la sfârșitul fiecărei linii



LINUX SED

```
sandy ~> cat -n example
 1 This is the first line of an example text.
 2 It is a text with errors.
 3 Lots of errors.
 4 So much errors, all these errors are making me sick.
 5 This is a line not containing any errors.
 6 This is the last line.
```

```
sandy ~> sed -e 's/errors/errors/g' -e 's/last/final/g' example
This is the first line of an example text.
It is a text with errors.
Lots of errors.
So much errors, all these errors are making me sick.
This is a line not containing any errors.
This is the final line.
```

- Find & Replace multiplu, considerând opțiunea `-e`
- Comanda este echivalenta cu:

```
sed 's/errors/errors/g; s/last/final/g' example
```



Exemple

```
sed "s/\(.\)/\1\n/g" /etc/passwd
```

⇒ Inlocuieste fiecare caracter cu el urmat de new line

```
sort
```

⇒ Sorteaza alfabetic

```
uniq -c
```

⇒ Omite liniile consecutive care se repeat

⇒ prin optiunea -c de prefixeaza liniile cu numarul de repetari

```
sort -n -r
```

⇒ Sorteaza numeric si descrescator

```
head
```

⇒ Afiseaza primele 10 linii

```
sed "s/\(.\)/\1\n/g" /etc/passwd | sort | uniq -c | sort -n -r | head
```



LINUX

- Atunci când folosiți în alte scopuri caracterele care au un înțeles aparte pentru consolă, ele trebuie să fie separate de acest înțeles special (escaped)
- Caracterul din Bash, precum și în alte console, care face acest lucru este linia oblică inversă (\);



LINUX

- Drepturile de acces
 - Pentru ca utilizarea comenzilor împreună cu fișierele să fie cât mai facilă, atât permisiunile sau modurile, drepturile de acces, cât și grupurile de utilizatori au un cod

Cod	Înțeles
0 sau -	Drepturile de acces asociate fișierului nu sunt acordate
4 sau r	Categoria de utilizatori definită are drepturi de citire
2 sau w	Categoria de utilizatori definită are drepturi de Scriere
1 sau x	Categoria de utilizatori definită poate rula fișierul

Cod	Înțeles
u	Permiuni acordate utilizatorilor
g	Permiuni acordate grupurilor
o	Permiuni acordate celorlalți



LINUX

- Numele de utilizator (variabila **\$USER**) și celelalte grupuri ale căror membru suntem: **ID**

```
tilly:~> id  
uid=504(tilly) gid=504(tilly) groups=504(tilly),100(users),2051(org)
```

```
tilly:~> echo $USER  
tilly
```



LINUX

- Comanda **CHMOD**
 - Schimbarea drepturilor de acces
- Operanzii + și – sunt folosiți pentru a acorda sau interzice drepturile de acces

```
asim:~> ./hello
bash: ./hello: bad interpreter: Permission denied

asim:~> cat hello
#!/bin/bash
echo "Hello, World"

asim:~> ls -l hello
-rw-rw-r--  1 asim  asim  32 Jan 15 16:29 hello

asim:~> chmod u+x hello

asim:~> ./hello
Hello, World

asim:~> ls -l hello
-rwxrw-r--  1 asim  asim  32 Jan 15 16:29 hello*
```

```
asim:~> chmod u+rwx,go-rwx hello

asim:~> ls -l hello
-rwx-----  1 asim  asim  32 Jan 15 16:29 hello*
```



LINUX

Comanda	Înțelesul comenzii
chmod 400 fișier	Pentru protejarea unui fișier de o suprascriere accidentală
chmod 500 director	Pentru a vă opri pe dumneavoastră să ștergeți, redenumiți sau să mutați accidental fișiere care aparțin acestui director.
chmod 600 fișier	Un fișier privat, care poate fi schimbat doar de utilizatorul care a introdus această comandă.
chmod 644 fișier	Un fișier care poate fi accesat public dar care poate fi schimbat doar de utilizatorul care a introdus această comandă.
chmod 660 fișier	Utilizatorii care aparțin grupului tău pot schimba acest fișier pe când ceilalți nu au nici un fel de drepturi asupra lui.
chmod 700 fișier	Numai utilizatorul are drepturi depline, ceilalți, indiferent de grupul aparținător, nu au nici un fel de drepturi.
chmod 755 director	Pentru fișierele care trebuie să fie citite sau rulate și de către alți utilizatori, dar scrise (schimbate) doar de utilizatorul care a introdus comanda descrisă.
chmod 775 fișier	Modul standard de acordare a permisiunilor pentru un grup.
chmod 777 fișier	Oricine poate face orice cu acest fișier.



Linux

Setarea drepturilor

- Drepturi de acces
 - Read (r) = 4
 - Write (w) = 2
 - Execute (x) = 1
 - Nici un drept = 0
- Moduri de acces pentru
 - Utilizator (u)
 - Grup (g)
 - Altii (o)
- Chmod 643 fisier reprezinta:
 - Drepturile acordate utilizatorului 6 (4+2+0), adica r w -
 - Drepturile acordate grupului 4 (4+0+0), adica r - -
 - Drepturile acordate altora 3 (0+2+1), adica - w x
- In final fisierul va avea drepturile: r w - r - - - w x



LINUX

- Modificarea apartenenței la un utilizator sau la un grup:
CHOWN sau **CHGRP**

```
jacky:~> id
uid=1304(jacky) gid=(1304) groups=1304(jacky),2034(pproject)

jacky:~> ls -l my_report
-rw-rw-r-- 1 jacky  project      29387 Jan 15 09:34 my_report

jacky:~> chown jacky: my_report

jacky:~> chmod o-r my_report

jacky:~> ls -l my_report
-rw-rw---- 1 jacky  jacky      29387 Jan 15 09:34 my_report
```



LINUX

- Modificarea apartenenței la un utilizator sau la un grup:
CHOWN sau **CHGRP**

```
jacky:~> ls -l report-20020115.xls
-rw-rw---- 1 jacky  jacky  45635 Jan 15 09:35 report-20020115.xls

jacky:~> chgrp project report-20020115.xls

jacky:~> chmod o= report-20020115.xls

jacky:~> ls -l report-20020115.xls
-rw-rw---- 1 jacky  project 45635 Jan 15 09:35 report-20020115.xls
```



LINUX

Variabile

- Setarea unei variabile
VARIABILA = VALOARE
- Utilizarea variabile
echo \$VARIABILA
- Clasificare
 - Variabile modificate dinamic de către interpretor
 - Variabile atribuite la intrarea în sesiune



LINUX

Variabile

```
#!/bin/sh
echo What is your name?
read MY_NAME
echo "Hello $MY_NAME!"
```

- Posibilitate de declarare:
 - Declare OPTION(s) VARIABLE = value
- Opțiuni
 - -a = variabila este un array (șir)
 - -i = variabile este un întreg

Exemple:

```
declare -a var=(1 2 3)
echo ${var[1]} => 2
```

```
declare -i vari=10
echo $vari
```



LINUX SHELL PROGRAMMING

- Array
 - Creare:
 - ARRAY[INDEXNR] = value
 - Declare -a ARRAYNAME
 - ARRAY=(value1 value2 ... valueN)
 - Pentru a face referire la o valoare din array se utilizează acolade
 - Ștergerea unei variabile

```
[bob in ~] ARRAY=(one two three)

[bob in ~] echo ${ARRAY[*]}
one two three

[bob in ~] echo $ARRAY[*]
one[*]

[bob in ~] echo ${ARRAY[2]}
three

[bob in ~] ARRAY[3]=four

[bob in ~] echo ${ARRAY[*]}
one two three four
```

```
[bob in ~] unset ARRAY[1]

[bob in ~] echo ${ARRAY[*]}
one three four

[bob in ~] unset ARRAY

[bob in ~] echo ${ARRAY[*]}
<--no output-->
```



LINUX SHELL PROGRAMMING

- Lungimea unei variabile: `${#VAR}`

```
[bob in ~] echo $SHELL
/bin/bash

[bob in ~] echo ${#SHELL}
9

[bob in ~] ARRAY=(one two three)

[bob in ~] echo ${#ARRAY}
3
```

- Eliminarea subșirurilor de caractere:
`${VAR:OFFSET:LENGTH}`

```
[bob in ~] export STRING="thisisaverylongname"

[bob in ~] echo ${STRING:4}
isaverylongname

[bob in ~] echo ${STRING:6:5}
avery
```



LINUX

Variabile

```
myvar2.sh
#!/bin/sh
echo "MYVAR is: $MYVAR"
MYVAR="hi there"
echo "MYVAR is: $MYVAR"
```

```
$ ./myvar2.sh
MYVAR is:
MYVAR is: hi there
```

```
$ MYVAR=hello
$ ./myvar2.sh
MYVAR is:
MYVAR is: hi there
```

```
$ export MYVAR
$ ./myvar2.sh
MYVAR is: hello
MYVAR is: hi there
```



LINUX

Variabile

Care este diferenta?

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

```
echo "What is your name?"
```

```
read USER_NAME
```

```
echo "Hello $USER_NAME"
```

```
echo "I will create you a file called $USER_NAME_file"
```

```
touch $USER_NAME_file
```

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

```
echo "What is your name?"
```

```
read USER_NAME
```

```
echo "Hello $USER_NAME"
```

```
echo "I will create you a file called ${USER_NAME}_file"
```

```
touch "${USER_NAME}_file"
```

LINUX

Variabile

- Variabile modificate dinamic de către interpretor
 - \$# = numărul de parametri ai unei proceduri shell
 - \$? = codul de revenire al ultimei comenzi executate
 - \$\$ = identificatorul de proces asociat shell-ului
 - \$! = identificatorul ultimului proces lansat în background
 - \$- = opțiunile cu care a fost lansat shell-ul
 - \$n = parametrii transmiși procedurilor shell pe linia de comandă (n=1..9)



LINUX

Variabile de mediu

- Sunt gestionate din consolă și sunt moștenite de oricare program pe care îl porniți
- \$PATH
- \$HOME
- Afișarea conținutului unei variabile:

```
debby:~> echo $PATH
/usr/bin:/usr/sbin:/bin:/sbin:/usr/X11R6/bin:/usr/local/bin

debby:~> echo $MANPATH
/usr/man:/usr/share/man/:/usr/local/man:/usr/X11R6/man
```

- Schimbarea conținutului unei variabile:

```
debby:~> PATH=$PATH:/opt/FlightGear/bin
```

```
debby:~> export PATH
```



LINUX

Variabile de mediu

- \$HOSTNAME
- \$LANG
- \$LOGNAME
- \$MAIL
- \$OSTYPE
- \$PWD
- \$SHELL



LINUX

Constante

- Readonly OPTION VARIABLE(s)

⇒ Se foloseste pentru declararea constantelor



LINUX SHELL PROGRAM MING

- Fișier care conține comenzi pentru configurarea consolei
- Un script shell poate fi făcut executabil folosind comanda **chmod** care determină luarea în considerare a bit-ului care marchează scriptul ca executabil

chmod +x nume_script

- Executarea **./nume_fisier ARGUMENTE** este echivalentă cu **bash nume_fisier ARGUMENTE**
- Script-urile Bash încep de regulă cu

```
#!/bin/bash
```



LINUX SHELL PROGRAM MING

- Substituirea comenzilor cu output-ul lor
 - `$(command)` sau ``command``

```
franky ~-> echo `date`  
Thu Feb 6 10:06:20 CET 2003
```

- Evaluarea expresiilor aritmetice
 - `$(expresie)` sau `$(expresie)` în Bash

```
franky ~-> echo $[365*24]  
8760
```



LINUX SHELL PROGRAMMING

- `EXPR` – evaluateza expresii
- `expr EXPRESSION`
- `EXPRESSION`:
 - `ARG1 | ARG2`
 - `ARG1 & ARG2`
 - `ARG1 < ARG2`
 - `ARG1 <= ARG2`
 - `ARG1 = ARG2`
 - `ARG1 != ARG2`
 - `ARG1 >= ARG2`
 - `ARG1 > ARG2`
 - `ARG1+ARG2`
 - `ARG1 – ARG2`
 - `ARG1 * ARG2`
 - `ARG1 / ARG2`
 - `ARG1 % ARG2`
 - `STRING : REGEXP`
 - `match STRING REGEXP`
 - `substr STRING POS LENGTH`
 - `index STRING CHARS`
 - `length STRING`



LINUX SHELL PROGRAM MING

```
expr test : \.*'
```

=> Rezultatul este **4** (numarul de caractere care respecta expresia regulara)

```
expr text : tex
```

=> Rezultatul este **3**

```
expr text : '\(.*\)'
```

=> Rezultatul este **text** (caracterele care respecta expresia regulara)

```
expr 5 = 5 => Rezultatul este 1 (true)
```

```
expr '5' = '5' => Rezultatul este 1 (true)
```

```
expr 5 \> 10 => Rezultatul este 0 (false) – evitarea caracterelor speciale!!!
```

```
expr 5 \!= 5 => Rezultatul este 0 (false)
```



LINUX SHELL PROGRAM MING

```
#!/bin/bash
count=0
echo $count
count=`expr $count + 1`
echo $count
count=`expr $count + 1`
echo $count
```

=> Rezultatul este **1 2**



LINUX SHELL PROGRAM MING

```
if [ ... ]; then  
    # do something  
fi
```

```
if [ ... ]  
then  
    # if-code  
else  
    # else-code  
fi
```

```
if [ something ]; then  
    echo "Something"  
elif [ something_else ]; then  
    echo "Something else"  
else  
    echo "None of the above"  
fi
```



LINUX SHELL PROGRAM MING

- **if TEST-COMMANDS; then CONSEQUENT-COMMANDS; fi**
- **TEST-COMMANDS - TRUE** dacă
 - [-a FILE] =fișierul există
 - [-d FILE] =fișierul există și este un director
 - [-z STRING] = lungimea lui STRING e zero
 - [STRING₁ == STRING₂] = șiruri egale
 - [STRING₁ != STRING₂] = șiruri diferite
 - [STRING₁ < STRING₂] = șiruri ordonate alfabetic
 - [STRING₁ > STRING₂] = șiruri ordonate descrescător alfabetic
 - [ARG₁ OP ARG₂] = OP (-eq, -ne, -lt, -le, -gt, -ge) – operatori aritmetici asupra întregilor ARG₁ și ARG₂



LINUX SHELL PROGRAM MING

- **if TEST-COMMANDS; then CONSEQUENT-COMMANDS; fi**
- TEST-COMMANDS (combinarea expresiilor) - TRUE dacă
 - [! Expr] = expresia Expr este falsă
 - [(Expr)] = returnează valoarea lui Expr
 - [Expr1 -a Expr2] = ambele expresii Expr1 și Expr2 sunt adevărate
 - [Expr1 -o Expr2] = cel puțin una din expresiile Expr1 și Expr 2 este adevărată



LINUX SHELL PROGRAMMING

```
anny ~> if [ $? -eq 0 ]
More input> then echo 'That was a good job!'
More input> fi
That was a good job!
```

```
if [ "$(whoami)" != 'root' ]; then
    echo "You have no permission to run $0 as non-root user."
    exit 1;
fi
```

```
anny ~> if ! grep $USER /etc/passwd
More input> then echo "your user account is not managed locally"; fi
your user account is not managed locally
```

```
anny > echo $?
0
```

```
anny > num=`wc -l work.txt`
```

```
anny > echo $num
201
```

```
anny > if [ "$num" -gt "150" ]
More input> then echo ; echo "you've worked hard enough for today."
More input> echo ; fi
```

```
you've worked hard enough for today.
```



LINUX SHELL PROGRAM MING

```
anny > gender="female"
```

```
anny > if [[ "$gender" == f* ]]
More input> then echo "Pleasure to meet you, Madame."; fi
Pleasure to meet you, Madame.
```

```
freddy scripts> gender="male"
```

```
freddy scripts> if [[ "$gender" == "f*" ]]
More input> then echo "Pleasure to meet you, Madame."
More input> else echo "How come the lady hasn't got a drink yet?"
More input> fi
How come the lady hasn't got a drink yet?
```



LINUX SHELL PROGRAMMING

```
anny ~> cat weight.sh
#!/bin/bash
# This script prints a message about
your weight if you give it your
# weight in kilos and height in
centimeters.
weight="$1"
height="$2"
idealweight=$((height - 110))
if [ $weight -le $idealweight ] ; then
    echo "You should eat a bit more fat."
else
    echo "You should eat a bit more
fruit."
fi
```

```
anny ~> bash -x weight.sh 55 169

+ weight=55
+ height=169
+ idealweight=59
+ '[' 55 -le 59 ']'
+ echo 'You should eat a bit more fat.'

You should eat a bit more fat.
```



LINUX SHELL PROGRAMMING

```
anny ~> cat weight.sh
#!/bin/bash
if [ ! $# == 2 ]; then
    echo "Usage: $0 weight_in_kilos
length_in_centimeters"
    exit
fi
weight="$1"
height="$2"
idealweight=$((height - 110))
if [ $weight -le $idealweight ] ; then
    echo "You should eat a bit more fat."
else
    echo "You should eat a bit more fruit."
fi
```

```
anny ~> weight.sh 70 150
You should eat a bit more fruit.
```

```
anny ~> weight.sh 70 150 33
Usage: ./weight.sh weight_in_kilos
length_in_centimeters
```

LINUX SHELL PROGRAM MING

- **while CONTROL-COMMAND; do CONSEQUENT-COMMANDS; done**

```
#!/bin/bash

# This script opens 4 terminal windows.

i="0"

while [ $i -lt 4 ]
do
xterm &
i=$((i+1))
done
```

- **until TEST-COMMAND; do CONSEQUENT-COMMANDS; done**



LINUX SHELL PROGRAM MING

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

```
INPUT_STRING=hello
```

```
while [ "$INPUT_STRING" != "bye" ]
```

```
do
```

```
    echo "Please type something in (bye to quit)"
```

```
    read INPUT_STRING
```

```
    echo "You typed: $INPUT_STRING"
```

```
done
```

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

```
while :
```

```
do
```

```
    echo "Please type something in (^C to quit)"
```

```
    read INPUT_STRING
```

```
    echo "You typed: $INPUT_STRING"
```

```
done
```



LINUX SHELL PROGRAM MING

```
#!/bin/bash
for i in {0..10..2}
do
    echo "Welcome $i times"
done
```

- **FOR variable IN list**
DO
commands
DONE

```
#!/bin/bash
for i in 1 2 3 4 5
do
    echo "Welcome $i times"
done
```

```
#!/bin/bash
for i in {1..5}
do
    echo "Welcome $i times"
done
```



LINUX SHELL PROGRAM MING

```
for I in 1 2 3 4 5
do
    statements1
    statements2
    if (disaster-condition)
    then
        break
    fi
    statements3
done
```

- FOR ((EXP1; EXP2; EXP3))
DO
commands
DONE

```
#!/bin/bash
for (( c=1; c<=5; c++ ))
do
    echo "Welcome $c times"
done
```

```
#!/bin/bash
for (( ; ; ))
do
    echo "infinite loops [ hit CTRL+C to
stop]"
done
```



LINUX SHELL PROGRAM MING

```
for I in 1 2 3 4 5
do
    statements1
    statements2
    if (condition)
    then
        continue
    fi
    statements3
done
```



LINUX SHELL PROGRAM MING

```
case $variable-name in
    pattern1)
        command1
        ...
        commandN
        ;;
    pattern2)
        command1
        ...
        commandN
        ;;
    patternN)
        command1
        ...
        commandN
        ;;
    *)
esac
```



LINUX SHELL PROGRAM MING

```
case $variable-name in
    pattern1|pattern2|pattern3)
        command1
        ...
        commandN
    ;;
    pattern4|pattern5|pattern6)
        command1
        ...
        commandN
    ;;
    pattern7|pattern8|patternN)
        command1
        ...
        commandN
    ;;
*)
esac
```



LINUX SHELL PROGRAM MING

```
#!/bin/bash
if [ -z $1 ]
then
    rental="*** Unknown vehicle ***"
elif [ -n $1 ]
then
    # otherwise make first arg as a rental
    rental=$1
fi
# use case statement to make decision for rental
case $rental in
    "car") echo "For $rental rental is Rs.20 per k/m.>";;
    "van") echo "For $rental rental is Rs.10 per k/m.>";;
    "jeep") echo "For $rental rental is Rs.5 per k/m.>";;
    "bicycle") echo "For $rental rental 20 paisa per k/m.>";;
    "enfield") echo "For $rental rental Rs.3 per k/m.>";;
    "thunderbird") echo "For $rental rental Rs.5 per k/m.>";;
    *) echo "Sorry, I can not get a $rental rental for you!>";;
esac
```



LINUX SHELL PROGRAM MING

```
#!/bin/bash
NOW=$(date +"%a")
case $NOW in
    Mon)
        echo "Full backup";;
    Tue|Wed|Thu|Fri)
        echo "Partial backup";;
    Sat|Sun)
        echo "No backup";;
    *) ;;
esac
```



LINUX SHELL PROGRAM MING

myfile
hello
bonjour

```
#!/bin/sh
while read f
do
  case $f in
    hello)          echo English    ;;
    howdy)         echo American   ;;
    gday)          echo Australian  ;;
    bonjour)       echo French     ;;
    "guten tag")   echo German     ;;
    *)             echo Unknown Language: $f
                  ;;
  esac
done < myfile
```



LINUX SHELL PROGRAMMING

- Exemplul 1 – script care saluta utilizatorul

```
[jerry@nowhere ~] cat hello.sh
#!/bin/bash
echo "Hello $USER"
```

- Exemplul 2 – script care afișează utilizatorilor autentificați

```
#!/bin/bash
who | cut -d " " -f 1 | sort -u
```

- Exemplul 3 – script care face copii tuturor fișierelor dintr-un director

```
tille:~> cat bin/makebackupfiles.sh
#!/bin/bash
# make copies of all files in a directory
LIST=`ls`
for i in $LIST; do
    ORIG=$i
    DEST=$i.old
    cp $ORIG $DEST
    echo "copied $i"
done
```



LINUX SHELL PROGRAM MING

```
#!/bin/sh
# hi
echo "Hello, world!"
exit 0
```

```
1.sh -> sh 1.sh
Hello, world!
```

```
#!/bin/sh
# hi mike
name=Mike
echo "Hello, $name!"
exit 0
```

```
2.sh -> sh 2.sh
Hello, Mike!
```

```
#!/bin/sh
# rem
rm junk
echo "The return code
from rm was $?"
exit 0
```

```
3.sh -> sh 3.sh
rm: cannot remove
`junk': No such file
or directory
The return code from
rm was 1
```

```
-----
-----
The return code from
rm was 0
```



LINUX

SHELL PROGRAMMING

```
#!/bin/sh
# pars
echo "There are $# parameters."
echo "The parameters are $@"
echo "The script name is $0"
echo "The first parameter is $1"
echo "The second parameter is $2"
exit 0
```

```
4.sh -> sh 4.sh a 1 bc 2
```

```
There are 4 parameters.
The parameters are a 1 bc 2
The script name is 4.sh
The first parameter is a
The second parameter is 1
```

```
#!/bin/sh
# shifter
echo $1
shift
echo $1
shift
echo $1
shift
echo $1
exit 0
```

```
5.sh -> sh 5.sh ana are mere si
pere
ana
are
mere
si
```



LINUX

SHELL PROGRAMMING

```
#!/bin/sh
# sorter
rm -f /tmp/sorted
sort $1 > /tmp/sorted
cp /tmp/sorted $1
rm -f /tmp/sorted
exit 0
```

6.sh -> sh 6.sh names

cat names

```
#!/bin/sh
# hiyou
name=`whoami`
echo "Hello, $name!"
exit 0
```

7.sh -> sh 7.sh

Hello, diana.sotropa!

```
#!/bin/sh
# countem
echo "File \"$1\" contains \
exactly `wc -l $1` lines."
exit 0
```

8.sh -> sh 8.sh fisier

File names contains exactly 6
lines



LINUX SHELL PROGRAMMING

```
#!/bin/sh                                     9.sh -> sh 9.sh
# compares
echo "true yields 0, false yields          true yields 0, false yields 1
1"
x="005"
[ "$x" = "005" ]
echo "Are strings 005 and 005              Are strings 005 and 005 equal? 0
equal? $?"
[ "$x" = "5" ]
echo "Are strings 005 and 5                Are strings 005 and 5 equal? 1
equal? $?"
[ $x -eq 005 ]
echo "Are integers 005 and 005            Are integers 005 and 005 equal? 0
equal? $?"
[ $x -eq 5 ]
echo "Are integers 005 and 5              Are integers 005 and 5 equal? 0
equal? $?"
exit 0
```



LINUX

SHELL PROGRAMMING

```
#!/bin/sh
# adder
sum=0
for x in $@
do
sum=`expr $sum + $x`
done
echo "The sum is $sum."
exit 0
```

```
10.sh -> sh 10.sh 1 2 3 4 5 6
```

```
The sum is 21.
```



Example

```
sh a.sh 1 2 3 4 5 6 7 8 9
```

```
Ana are 5 mere
```

```
Ana are $A $B
```

```
a.sh
```

```
1 2 3 4 5 6 7 8 9
```

```
1 2 3 4 5 6 7 8 9
```

```
2 3 4 5 6 7 8 9
```

```
5 6 7 8 9
```

```
5 6 7 8 9
```

```
#!/bin/bash
```

```
A=5
```

```
B="mere"
```

```
echo "Ana are $A $B"
```

```
echo 'Ana are $A $B'
```

```
echo $0
```

```
echo $*
```

```
echo $1 $2 $3 $4 $5 $6 $7 $8 $9
```

```
shift
```

```
echo $1 $2 $3 $4 $5 $6 $7 $8 $9
```

```
shift 3
```

```
echo $1 $2 $3 $4 $5 $6 $7 $8 $9
```

```
echo $*
```



Exemple

```
#!/bin/bash
if test -e $1; then
    echo exista
else
    echo nu exista
fi
if test -e $1 && ! test -d $1; then
    echo exista dar nu e director
elif test -d $1; then
    echo director
else
    echo am ajuns la else
fi
if [ -e $1 ] && [ ! -d $1 ]; then
    echo exista dar nu e director
elif [ -d $1 ]; then
    echo director
else
    echo am ajuns la else
fi
```



Exemple

```
#!/bin/bash
for A in safd 3245 k rrr 098; do
    echo $A
done #(se parcurge lista care are 5 elemente)
for A in "safd 3245 k rrr 098"; do
    echo $A
done #(se parcurge lista care are 1 element)
for A in $*; do
    echo $A
done #(se parcurge lista argumentelor date in linia de comanda)
for A; do
    echo $A
done #(se parcurge lista argumentelor date in linia de comanda)
for F in *; do
    file $F
done #(se parcurge lista tuturor fisierelor din directorul curent)
for F in *.txt; do
    file $F
done #(se parcurge lista tuturor fisierelor txt din directorul curent)
```



Exemple!!!!

```
sh c.sh . 10
```

```
-----  
Afiseaza recursiv numele  
fisierelor text din  
directorul curent care au  
mai mult de 10 linii
```

```
#!/bin/bash  
D=$1  
N=$2  
for F in `find $D -type f`; do  
    if file $F | grep -q "ASCII text"; then  
        K=`wc -l $F | sed "s/ .*//"`  
        if [ $K -ge $N ]; then  
            echo $F  
        fi  
    fi  
done
```



Exemple

```
sh d.sh a.txt
```

```
-----
```

```
Afiseaza numarul de  
caractere, numarul de  
linii si numarul mediu de  
caractere / linie din  
fisierul a.txt
```

```
!! Aceasta metoda  
functioneaza doar daca nu  
exista spatii pe linie
```

```
#!/bin/bash
```

```
F=$1
```

```
SUM=0
```

```
N=0
```

```
for C in `cat $F`; do
```

```
    K=`echo $C | wc -c`
```

```
    SUM=`expr $SUM + $K`
```

```
    N=`expr $N + 1`
```

```
done
```

```
echo "S=$SUM N=$N M=`expr $SUM / $N`"
```



Exemple

```
sh d.sh a.txt
```

```
-----
```

```
Afiseaza numarul de  
caractere, numarul de  
linii si numarul mediu de  
caractere / linie din  
fisierul a.txt
```

```
!! Aceasta metoda  
functioneaza chiar daca  
exista spatii pe linie
```

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

```
F=$1
```

```
SUM=0
```

```
N=0
```

```
while read C; do
```

```
    K=`echo $C | wc -c`
```

```
    SUM=`expr $SUM + $K`
```

```
    N=`expr $N + 1`
```

```
done < $F
```

```
echo "S=$SUM; N=$N M=`expr $SUM / $N`"
```



Exemple

```
sh f.sh
```

Afiseaza un text la
modificarea structurii de
fisiere si directoare

```
#!/bin/bash
PREV=""
while true; do
    if [ -z "$PREV" ]; then
        PREV=`ls -Rl`
    fi
    sleep 1
    NOW=`ls -Rl`
    if [ ! "$PREV" = "$NOW" ]; then
        echo cineva a schimbat ceva
    fi
    PREV=$NOW
done
```



LINUX SHELL PROGRAM MING

- Funcții
 - `function FUNCTION { COMMANDS; }`
 - `FUNCTION () { COMMANDS; }`
- Se apelează
 - `FUNCTION argumente`



LINUX SHELL PROGRAM MING

```
#!/bin/sh
add_a_user()
{
    USER=$1
    PASSWORD=$2
    shift; shift;
    # Having shifted twice, the rest is now comments ...
    COMMENTS=$@
    echo "Adding user $USER ..."
    echo useradd -c "$COMMENTS" $USER
    echo passwd $USER $PASSWORD
    echo "Added user $USER ($COMMENTS) with pass $PASSWORD"
}

echo "Start of script..."
add_a_user bob letmein Bob Holness the presenter
add_a_user fred badpassword Fred Durst the singer
add_a_user bilko worsepassword Sgt. Bilko the role model
echo "End of script..."
```



LINUX SHELL PROGRAM MING

```
FUNCTII RECURSIVE:
#!/bin/sh

factorial()
{
    if [ "$1" -gt "1" ]; then
        i=`expr $1 - 1`
        j=`factorial $i`
        k=`expr $1 \* $j`
        echo $k
    else
        echo 1
    fi
}

while :
do
    echo "Enter a number:"
    read x
    factorial $x
done
```

