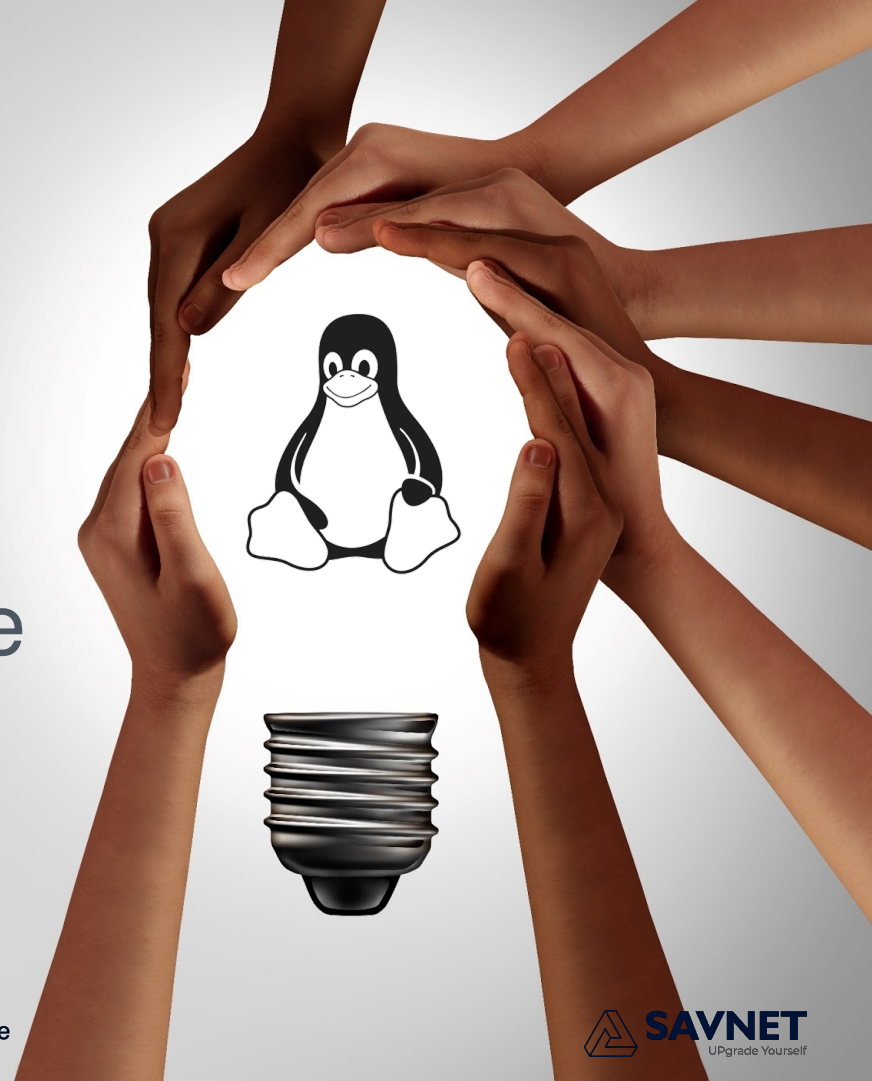
SAVNET
UPgrade Yourself

Linux

Administrare și Securitate



SAVNET
UPgrade Yourself

Ce vom învăța în acest curs?

- Creare, modificare, ştergere directoare şi fişiere
- Globbing
- Permisiuni de bază, proprietari, grupuri
- Arhivare şi comprimare
- Backup
- Introducere in scripting
- Proiect software, realizarea unui script de backup simplu

Capitolul 3: Gestionarea Fișierelor şi Directoarelor

Crearea, Mutarea și Ștergerea Fișierelor

Globbing

Globbing

Atenție!

Linux este sensibil la majuscule

Fișierul hello.txt este diferit de HELLO.txt și Hello.txt

Caracterele glob sunt adesea denumite wild cards (metacaractere).
Acestea sunt caractere simbol care au o semnificație specială pentru shell.

De ce sunt utile?

Glob-urile sunt puternice deoarece vă permit să specificați modele care se potrivesc cu numele de fișiere dintr-un director

În loc să manipulați un singur fișier la un moment dat, puteți executa cu ușurință comenzi care vor afecta multe fișiere.

Globbing - Asteriscul *

Caracterul asterisc este folosit pentru a reprezenta zero sau mai multe caractere de orice tip într-un nume de fișier.

Exemplu:
Să presupunem că doriți să afișați toate fișierele din directorul /etc care încep cu litera t:
Modelul t* se potrivește cu orice fișier din directorul /etc care începe cu caracterul t urmat de zero sau mai multe caractere de orice tip.

```
student@localhost:~$ echo /etc/t*
/etc/terminfo /etc/timezone
```

Globbing - Semnul Întrebării ?

Caracterul semn de întrebare se potrivește exact cu un singur caracter, nici mai mult, nici mai puțin.

Exemplu:
Să presupunem că doriți să afișați toate fișierele din directorul /etc care încep cu litera t și au exact 7 caractere după t:

```
student@localhost:~$ echo /etc/t???????
/etc/terminfo /etc/timezone
```

Exemplu:
Asteriscul și semnul întrebării pot fi folosite împreună pentru a căuta fișiere cu extensii de trei litere:

```
student@localhost:~$ echo /etc/*.???
/etc/blkid.tab /etc/issue.net
```

Globbing - Paranteze Pătrate [ ]

Parantezele sunt folosite pentru a potrivi un singur caracter reprezentând un interval de caractere care sunt posibile caractere de potrivire.

Exemplu:
Comanda echo /etc/[gu]* va afișa orice fișier care începe cu g sau u și conține zero sau mai multe caractere suplimentare:

```
student@localhost:~$ echo /etc/[gu]*
/etc/gai.conf /etc/groff /etc/group…
```

Exemplu:
Parantezele pot fi folosite pentru a reprezenta un interval de caractere folosind caracterul - (de exemplu, orice literă între și inclusiv a și d):

```
student@localhost:~$ echo /etc/[a-d]*
```

Globbing - Semnul Exclamării !

Semnul exclamării este folosit împreună cu parantezele pătrate pentru a nega un interval.

Exemplu:
Comanda echo /etc/[!a-t]* va afișa orice fișier care nu începe cu o literă de la a la t:

```
student@localhost:~$ echo /etc/[!a-t]*
/etc/ucf.conf /etc/udev /etc/ufw /etc/update-motd.d...
```

Globbing - Listare

Când comanda ls vede un director ca argument, va afișa conținutul directorului, nu doar numele directorului.

Soluție:
Folosiți opțiunea -d, care îi spune comenzii ls să afișeze numele directoarelor, în loc de conținutul lor:

```
student@localhost:~$ ls -d /etc/e*
/etc/encript.cfg /etc/environment /etc/ethers...
```

Crearea Fișierelor și Directoarelor

Crearea Fișierelor

Pentru a crea un fișier gol, folosiți comanda touch, așa cum este demonstrat mai jos:

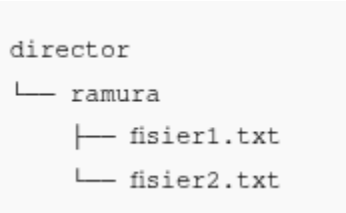
```
student@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates Videos

student@localhost:~$ touch sample

student@localhost:~$ ls -l sample
-rw-rw-r-- 1 student student 0 Nov  9 16:48 sample

student@localhost:~$ ls
Desktop    Downloads  Pictures  Templates  sample
Documents  Music      Public    Videos

student@localhost:~$ touch director/ramura/{fisier1.txt,fisier2.txt}
```



Crearea Directoarelor

Pentru a crea un director, folosiți comanda mkdir:

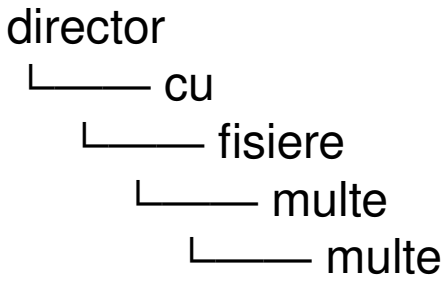
```
student@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates sample.txt

student@localhost:~$ mkdir test

student@localhost:~$ ls
Desktop    Downloads  Pictures  Templates   test
Documents  Music      Public    sample.txt


student@localhost:~$ mkdir -p /director/cu/fisiere/multe/multe

student@localhost:~$ mkdir -p director/ramura/{folder1,folder2}
```




```
director
└── ramura
    ├── folder1
    └── folder2
```

Copierea Fișierelor și Directoarelor

Copierea Fișierelor

Comanda cp este folosită pentru a copia fișiere. Necesită o sursă și o destinație.

Structura comenzii:

```
cp [sursa] [destinatia]
```

Sursa este fișierul de copiat. Destinația este locul unde va fi plasat copia.

Exemplu:
Următoarea comandă va copia fișierul /etc/hosts în directorul dvs. Home:

```
student@localhost:~$ cp /etc/hosts ~
student@localhost:~$ ls
Desktop    Downloads  Pictures  Templates  hosts
```

Copierea Fișierelor - Modul Verbose

Opțiunea -v va determina comanda cp să producă output dacă operația este reușită.
Opțiunea -v înseamnă verbose (detaliat).

Exemplu:

```
student@localhost:~$ cp -v /etc/hosts ~
'/etc/hosts' -> '/home/student/hosts'
```

Observație importantă:
Când destinația este un director, noul fișier va avea același nume ca fișierul original.
Dacă doriți ca noul fișier să aibă un nume diferit, trebuie să furnizați noul nume ca parte a destinației.

Copierea Fișierelor - Evitarea Suprascrierii

Comanda cp poate fi distructivă pentru datele existente dacă fișierul de destinație există deja.

Opțiunea -i (interactiv):
Cu opțiunea -i, cp va întreba înainte de a suprascrie un fișier (y (da) sau n (nu)):

```
student@localhost:~$ cp -i /etc/hosts ~/example.txt
cp: overwrite '/home/student/example.txt'? N
```

Observație:
Opțiunea -i necesită răspuns y sau n pentru fiecare copiere, ceea ce poate fi obositor dacă sunt multe fișiere.
Dacă doriți să răspundeți automat n la fiecare solicitare, folosiți opțiunea -n.
Aceasta înseamnă 'no rewrite' (fără rescriere).

Copierea Directoarelor

Utilizarea comenzii cp pentru a copia directoare va rezulta într-un mesaj de eroare:

```
student@localhost:~$ cp -n /etc/skel/.* ~
cp: omitting directory '/etc/skel/.'
cp: omitting directory '/etc/skel/..'
```

Soluție - Opțiunea -r (recursiv):
Opțiunea -r pentru comanda cp va copia atât fișierele, cât și directoarele.

⚠ Atenție!!!
Fiți atenți cu această opțiune. Întreaga structură de directoare va fi copiată. Acest lucru ar putea duce la copierea multor fișiere și directoare!

Mutarea Fișierelor

Mutarea Fișierelor

Pentru a muta un fișier, folosiți comanda mv.

Sintaxa comenzii:

```
mv [sursa] [destinatia]
```

Sintaxa pentru comanda mv este similară cu cea a comenzii cp.

Diferența față de cp:
Când un fișier este mutat, fișierul este eliminat din locația originală și plasat într-o nouă locație.

⚠ Notă:
Dacă nu aveți permisiunile corecte, veți primi un mesaj de eroare 'Permission denied' (Permisiune refuzată).

Mutarea Fișierelor - Redenumirea

Comanda mv nu este folosită doar pentru a muta un fișier, ci și pentru a-l redenumi.

Cum funcționează:
Numele fișierului se va schimba doar dacă este specificat și un nume de fișier de destinație.

Dacă nu este specificat un director de destinație, fișierul va fi redenumit folosind numele de fișier de destinație și va rămâne în directorul sursă.

Exemplu:
Următoarele comenzi vor redenumi fișierul newexample.txt în myexample.txt:

```
student@localhost:~/Videos$ mv newexample.txt myexample.txt
```

Mutarea Fișierelor - Opțiuni Suplimentare

La fel ca comanda cp, comanda mv oferă următoarele opțiuni:

Opțiuni disponibile:

- -i (Interactive): Întreabă dacă un fișier trebuie suprascris.
- -n (No Clobber): Nu suprascrie conținutul unui fișier de destinație.
- -v (Verbose): Arată rezultatul mutării.

⚠ Important:
Nu există opțiunea -r deoarece comanda mv va muta directoarele în mod implicit.

Ștergerea Fișierelor și Directoarelor

Ștergerea Fișierelor

Pentru a șterge un fișier, folosiți comanda rm:

```
student@localhost:~$ ls
Desktop    Downloads  Pictures  Templates  sample
Documents  Music      Public    Videos

student@localhost:~$ rm sample

student@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates Videos
```

⚠ Atenție:
Utilizarea rm poate cauza probleme atunci când ștergeți mai multe fișiere folosind caractere glob.
Ca măsură de precauție, utilizatorii ar trebui să folosească opțiunea -i atunci când șterge mai multe fișiere.

Ștergerea Directoarelor

Comanda rm poate fi folosită pentru a șterge directoare. Cu toate acestea, utilizarea implicită (fără opțiuni) a comenzii rm nu va reuși să șteargă un director:

```
student@localhost:~$ rm Videos
rm: cannot remove 'Videos': Is a directory
```

-rf NU sterge limba franceza?

rm -rf /

vs

rm -rf ./

NU faceti copy/paste la comenzi fara sa verificati

Soluție - Opțiunea -r (recursiv):

Pentru a șterge un director, folosiți opțiunea -r pentru comanda rm:

```
student@localhost:~$ rm -r Videos
```

⚠ Important:

Când un utilizator șterge un director, toate fișierele și subdirectoarele sunt șterse fără nicio întrebare interactivă.

Este cel mai bine să folosiți opțiunea -i cu comanda rm.

Permisiuni de baza, proprietari, grupuri

Proprietatea Fișierelor

Proprietatea fișierelor este critică pentru securitate.

Reguli de bază:
• În mod implicit, utilizatorii vor deține fișierele pe care le creează. Proprietatea poate fi schimbată de administrator.
• Fiecare fișier are și un proprietar de grup. În mod implicit, grupul principal al utilizatorului care creează fișierul va fi proprietarul de grup al oricăror fișiere noi.

Memorați:
UID-urile și GID-urile sunt asociate cu numele de utilizator și numele de grup.
Comanda id poate fi folosită pentru a vizualiza UID, GID, numele de utilizator și numele grupului(grupurilor).

Proprietatea Fișierelor

Când un utilizator creează un fișier cu comanda touch, acesta va aparține utilizatorului curent și grupului său principal.

Proprietatea fișierului poate fi confirmată folosind opțiunea de listare lungă -l a comenzii ls.

```
student@localhost:~$ touch /tmp/filetest1
student@localhost:~$ ls -l /tmp/filetest1
-rw-rw-r--. 1 student student 0 Oct 21 10:18 /tmp/filetest1
```

Schimbarea Grupurilor

Pentru a crea un fișier sub un grup diferit, folosiți comanda newgrp pentru a schimba grupul principal curent.

Folosiți comanda groups pentru a vizualiza informațiile despre grupurile utilizatorului. Verificați noul grup principal folosind comanda id:

```
student@localhost:~$ id
uid=502(student) gid=503(student) groups=503(student),10001(research)...

student@localhost:~$ newgrp research
student@localhost:~$ id
uid=502(student) gid=10001(research) groups=503(student),10001(research)…
```

Observație:
Comanda newgrp deschide un shell nou. Pentru a reveni la grupul original, rulați comanda exit.

Schimbarea Proprietarului de Grup

Pentru a schimba proprietarul de grup al unui fișier existent, folosiți comanda chgrp.

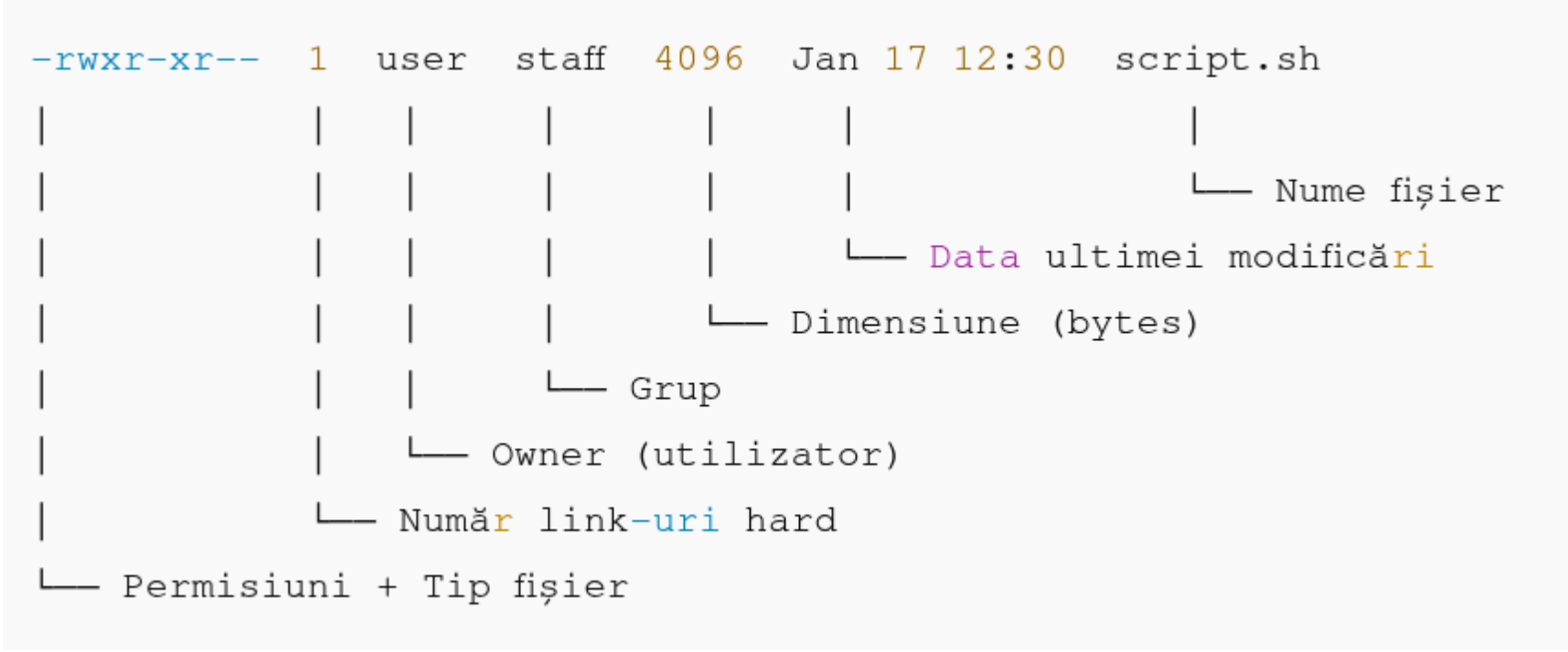
Reguli:
- Utilizatorul root poate folosi comanda chgrp pentru a schimba proprietarul de grup al oricărui fișier.
- Un utilizator obișnuit poate schimba proprietarul de grup al fișierului la un grup din care este membru.

```
student@localhost:~$ touch sample
student@localhost:~$ ls -l sample
-rw-rw-r-- 1 student student 0 Dec 10 00:44 sample

student@localhost:~$ chgrp research sample
student@localhost:~$ ls -l sample
-rw-rw-r--. 1 student research 0 Oct 23 22:12 sample
```

ls -l script.sh

Pentru a afișa tipul fișierului și permisiunile unui fișier, folosiți comanda ls -l:



Permisiuni - Tipul Fișierului

Pentru a afișa tipul fișierului și permisiunile unui fișier, folosiți comanda ls -l:

```
root@localhost:~# ls -l /etc/passwd
-rw-r--r--. 1 root root 4135 May 27 21:08 /etc/passwd
```

Tipul Fișierului:
Primul caracter al fiecărei linii indică tipul fișierului. Valori posibile:

- fișier obișnuit
c fișier caracter
d director
p fișier pipe
l link simbolic
s fișier socket
b fișier bloc

Permisiuni - Grupuri de Permisiuni

Următoarele 9 (nouă) caractere demonstrează permisiunile fișierului.
Acestea determină nivelul de acces pe care un utilizator îl va avea asupra fișierului.

```
-rw-r--r--. 1 root root 4135 May 27 21:08 /etc/passwd
```

Proprietar Utilizator (User Owner):
Caracterele 2-4 indică permisiunile pentru utilizatorul care deține fișierul.

Proprietar Grup (Group Owner):
Caracterele 5-7 indică permisiunile pentru grupul care deține fișierul.

Permisiuni Others (Altele):
Caracterele 8-10 indică permisiunile pentru alții sau ceea ce este uneori numit permisiunile globale (world's permissions).


```
-rw-r--r--. 1 root root 4135 May 27 21:08 /etc/passwd
```

Tipuri de Permisiuni

Fiecare grup are atribuite trei tipuri de permisiuni: read (citire), write (scriere) și execute (executare).

User Owner			Group Owner			Other		
Read	Write	Execute	Read	Write	Execute	Read	Write	Execute
r	w	-	r	-	-	r	-	-

Read (r) - Citire:

• Fișier: Permite procesului să citească conținutul fișierului, ceea ce înseamnă că conținutul poate fi vizualizat și copiat.
• Director: Numele din director sunt listate, dar nu sunt disponibile alte detalii.

Write (w) - Scriere:

• Fișier: Poate fi scris de proces. Permisiunea w necesită permisiunea r pentru a funcționa.
• Director: Fișierele pot fi adăugate sau eliminate din director. Permisiunea w necesită permisiunea x pentru a funcționa.

Execute (x) – Executare:

• Fișier: Un fișier poate fi executat sau rulat ca proces.
• Director: Utilizatorul poate folosi comanda cd pentru a intra în director și poate folosi calea pentru a accesa fișierele din director.

Exemplu de Scenariu

Pe baza următoarelor informații, ar avea utilizatorul bob, acces la fișierul abc.txt?

```
drwxr-xr-x. 17 root root 4096 23:38 /
drwxr-xr--. 10 root root 128  03:38 /data
-rwxr-xr--.  1 bob  bob  100  21:08 /data/abc.txt
```

Exemplu de Scenariu

Pe baza următoarelor informații, ar avea utilizatorul bob, acces la fișierul abc.txt?

```
drwxr-xr-x. 17 root root 4096 23:38 /
drwxr-xr--. 10 root root 128  03:38 /data
-rwxr-xr--.  1 bob  bob  100  21:08 /data/abc.txt
```

Răspuns: Nu.
Pentru a face orice cu fișierul, utilizatorul trebuie mai întâi să 'intre' în directorul /data.

Permisiunile pentru bob pentru directorul /data sunt permisiunile pentru 'alții' (r--), ceea ce înseamnă că bob nu poate folosi nici măcar comanda cd pentru a intra în director.

Dacă permisiunea de executare (--x) ar fi setată pentru director, atunci utilizatorul bob ar putea 'intra' în director, ceea ce înseamnă că s-ar aplica permisiunile fișierului în sine.

Schimbarea Permisiunilor - Metoda Simbolică

Există două tehnici care pot fi folosite cu această comandă: simbolică și numerică.

Comanda chmod (change mode) este folosită pentru a schimba permisiunile unui director. Caracterele indică la ce grup de permisiuni (user, group, others) să aplicați modificările:

• u = user (utilizator) g = group (grup) o = others (alții) a = all (toți)

Apoi, alegeți un indicator pentru a indica cum să modificați permisiunile:
• + (plus) = adaugă permisiune
• - (minus) = elimină permisiune
• = (egal) = setează exact aceste permisiuni

În cele din urmă, specificați tipul de permisiune:
• r = read w = write x = execute

Exemplu:
Pentru a acorda utilizatorului proprietar permisiunea de citire pe un fișier numit abc.txt:

```
root@localhost:~# chmod u+r abc.txt
```

Schimbarea Permisiunilor - Metoda Numerică

Bazată pe sistemul de numerotare octal unde fiecare tip de permisiune primește o valoare numerică.

Valori numerice:
4 = Read (Citire) 2 = Write (Scriere) 1 = Execute (Executare)

Combinații posibile (0-7):
- 7 = rwx 6 = rw- 5 = r-x 4 = r--
- 3 = -wx 2 = -w- 1 = --x 0 = ---

Exemplu:
Pentru a seta permisiunile unui fișier numit abc.txt la rwxr-xr--:

```
root@localhost:~# chmod 754 abc.txt
```

Permisiuni Implicite

Comanda umask este folosită pentru a determina permisiunile implicite care sunt setate când un fișier sau director este creat.
Valoarea umask este scăzută din permisiunile implicite maxime permise.

Valori maxime implicite:
- Fișier = rw-rw-rw- (666)
- Director = rwxrwxrwx (777)

```
student@localhost:~$ umask
0002
```

Explicație cifre umask (0002):
- Primul 0: umask este dat ca număr octal
- Al doilea 0: ce permisiune să scadă din proprietarul utilizator
- Al treilea 0: ce permisiune să scadă din proprietarul de grup
- Ultimul 2: ce permisiune să scadă din alții

Permisiuni Implicite - Cum Funcționează umask?

Să presupunem că umask este setat la 027:

```
Implicit fișier      666
Umask              - 027
                   -----
Rezultat             640
```

Umask-ul 027 înseamnă că, în mod implicit, fișierele noi vor primi permisiunile 640 sau rw-r-----:

```
student@localhost:~$ umask 027
student@localhost:~$ touch sample
student@localhost:~$ ls -l sample
-rw-r-----. 1 student student 0 Oct 28 20:14 sample
```

Arhivarea și Comprimarea Fișierelor

Ce este Comprimarea? Ce este Arhivarea?

Comprimarea reduce cantitatea de date necesare pentru a stoca sau transmite un fișier, păstrându-l într-o formă din care poate fi restaurat.
Fișiere mai mici → transmisie mai rapidă → economie de spațiu

Tipuri de comprimare:
• Lossless (fără pierderi): Nu se elimină informații din fișier
• Lossy (cu pierderi): Informații pot fi eliminate din fișier

Arhivarea combină mai multe fișiere sau directoare într-un singur fișier.
Avantaje:
• Elimină overhead-ul din fișierele individuale
• Facilitează transmiterea mai multor fișiere deodată
• Organizare mai bună a datelor

Observație importantă:
Arhivarea (spatiu ocupat mai mare) ≠ Comprimarea (economie spatiu / se poate combina cu arhivarea)

Diferența și Când să le Folosești?

Comprimare (gzip):
• Un singur fișier → fișier mai mic
• Exemplu: raport.txt → raport.txt.gz

Arhivare (tar):
• Mai multe fișiere → un singur fișier arhivă
• Exemplu: 100 fișiere → proiect.tar

Combinație (tar + gzip):
• Mai multe fișiere → un singur fișier arhivă comprimată
• Exemplu: 100 fișiere → proiect.tar.gz
• Cel mai eficient pentru backup și transmisie!

Utilitare Principale

gzip / gunzip:
• Comprimare și decomprimare de fișiere individuale
• Extensie: .gz

```
gzip fisier.txt → fisier.txt.gz
gunzip fisier.txt.gz → fisier.txt
```

tar (Tape Archive):
• Arhivare fișiere și directoare
• Create (-c), Extract (-x), List (-t)

```
tar -czf arhiva.tar.gz director/
```

zip / unzip:
• Arhivare + comprimare în format Windows-compatibil
• Extensie: .zip

Exemple Practice

Backup director proiect

```
tar -czf proiect_backup.tar.gz proiect/
# Rezultat: Un fișier arhivă comprimat
```

Verificare conținut arhivă

```
tar -tzf proiect_backup.tar.gz
# Afișează lista fișierelor din arhivă
```

Restaurare arhivă

```
tar -xzf proiect_backup.tar.gz
# Extrage toate fișierele
```

Opțiuni tar importante:

-c (create), -x (extract), -t (list), -z (gzip), -j (bzip2), -f (file), -v (verbose)

Backup în Linux Concepte Esențiale

Ce este Backup-ul?

Backup = Copie de siguranță a datelor

De ce avem nevoie?
- Hard disk-ul se strică
- Ștergi fișiere din greșeală
- Virus/malware
- Furt/incendiu

Ideea principală:
Datele importante trebuie să existe în MAI MULTE locuri!

Exemplu simplu:
Proiectul tău → Copiază pe HDD extern → Copiază în Cloud

Ce înseamnă Automatizare backup?

Problema generala:
"Uităm să facem backup manual"

Cum rezolvam?

- Scriem un script (comandă sau set de comenzi)
- Programăm computerul să ruleze scriptul automat
- Exemplu: Backup în fiecare zi la ora 2:00 AM

Tool-uri necesare:
- tar sau rsync - fac backup-ul
- cron - programează când să ruleze

tar - Comenzi de Bază

Creează backup (arhivă comprimată):

```
tar -czf nume-backup.tar.gz folder-de-salvat
```

Vezi ce e în backup:

```
tar -tzf nume-backup.tar.gz
```

Restaurează din backup:

```
tar -xzf nume-backup.tar.gz
```

Optiuni:

-c = create (creează)
-z = gzip (comprimă)
-f = file (fișier)
-t = list (listează)
-x = extract (extrage)

rsync

Copiază fișiere, dar DOAR ce s-a schimbat (nu tot)

- Backup frecvent (zilnic/de mai multe ori pe zi)
- Fișiere mari care se modifică puțin
- Backup rapid (nu arhivă comprimată)

Diferența față de TAR:
- tar - Face o arhivă (pachet) cu TOTUL
- rsync - Copiază doar DIFERENȚELE

Exemplu:
Ai 1000 de fișiere. Modifici doar 5.
- tar → Creează o arhivă nouă cu toate 1000
- rsync → Copiază doar cele 5 modificate

rsync - Comenzi de Bază

Copiază folder local în backup:

```
rsync -av sursa destinatie        #rsync -av ce-copiez unde-copiez
```

Opțiuni:

- -a = păstrează tot (permisiuni, date, etc.)
- -v = arată ce face (verbose)

Șterge din backup ce nu mai există în sursă:

```
rsync -av --delete sursa backup
```

Backup pe alt computer (prin SSH):

```
rsync -avz sursa user@server:/backup/
```

-z = comprimă în timpul transferului

tar vs rsync

tar:
• Backup complet periodic (săptămânal/lunar)
• Arhivare de lungă durată
• Transfer între computere
• Fișier .tar.gz compact
Exemplu: Backup săptămânal al proiectelor

rsync:
• Backup frecvent (zilnic/continuu)
• Fișiere mari cu modificări mici
• Mirror/sincronizare
• Copie exactă, nu arhivă
Exemplu: Sincronizare zilnică documente

Se pot folosi AMBELE:
tar pentru backup săptămânal complet + rsync pentru sincronizare zilnică

cron - Ce este?

Program care rulează în fundal și execută comenzi la ore programate

Componentele CRON:

- cron (serviciul): Rulează non-stop în fundal, verifică la fiecare minut: "Trebuie să fac ceva?"
- cron job: Comanda/scriptul pe care vrei să-l ruleze (ex: "Fă backup la fiecare zi la 2 AM")
- crontab: Fișierul unde scrii ce comenzi vrei să ruleze și când

cron - Cum programam?

Format CRON:

```
* * * * * comandă
| | | | |
| | | | └── zi din săptămână (0-7) (0 sau 7 = duminică)
| | | └──── lună (1-12)
| | └────── zi din lună (1-31)
| └──────── oră (0-23)
└────────── minut (0-59)
```

Exemple simple:

- 0 2 * * * → Zilnic la 2:00 AM
- 0 */6 * * * → La fiecare 6 ore
- 0 0 * * 0 → Duminica la miezul nopții

Wildcard (*) = orice valoare */N = la fiecare N unități

cron - Comenzi Practice

Editează programările tale: `crontab -e`

Vezi ce ai programat: `crontab -l`

Exemplu complet:

```
# Deschide crontab
crontab -e

# Adaugă această linie:
0 2 * * * tar -czf /backup/daily.tar.gz /home/user/documents

# Salvează și închide
# → Acum se face backup zilnic la 2 AM automat!
```

Verificare că funcționează: `grep CRON /var/log/syslog`

Scripting de Bază

Introducere

Descrierea Obiectivului:
Transformarea comenzilor repetitive în script-uri

Ce este un Shell Script?
Un shell script este un fișier de comenzi executabile care au fost stocate într-un fișier text. Când un fișier script este executat, fiecare comandă din el este executată.

Avantaj:
Construirea de shell script-uri ajută la automatizarea părților repetitive ale muncii tale.

Script Simplu

Exemplu de script cu o singură comandă:

```
#!/bin/bash
echo "Hello, World!"
```

Rularea unui script:
Metoda 1: Ca argument pentru shell:

```
bash myscript.sh
```

Metoda 2: Rulare directă (necesită permisiuni):

```
chmod +x myscript.sh
./myscript.sh
```

Eroarea **„Permission denied”** înseamnă că scriptul nu a fost marcat ca fiind executabil.
Comanda chmod poate fi folosită pentru a schimba permisiunile fișierului.

```
student@localhost:~$ ./myscript.sh
-bash: ./myscript.sh: Permission denied
```

Script Simplu - Shebang

Script-urile complexe indică un shell specific:

```
#!/bin/bash
echo "Acest script folosește Bash"
```

Explicație Shebang (#!):
- #! (shebang) este un prefix care marchează textul următor ca executabil
- /bin/bash specifică calea absolută către interpretor

Alte exemple comune:
- #!/bin/sh → Bourne shell
- #!/usr/bin/python3 → Python 3
- #!/usr/bin/env bash → Bash (locație flexibilă)

Editarea Shell Script-urilor

Shell script-urile sunt scrise folosind text simplu (plain text).
Familiarizarea cu editori de text ajută la scrierea shell script-urilor.

Editori populari:
• GNU nano: Editor simplu, potrivit pentru editarea fișierelor text mici
• vim (Visual Editor): Editor remarcabil de puternic și versatil

Exemplu simplu în nano:

```
nano myscript.sh

#!/bin/bash
echo "Acesta este un script simplu"
```

Elementele de Bază ale Scripting-ului

Reamintim: script-ul începe cu linia shebang (#!), indica ca trebuie interpretat de catre bash

3 concepte esențiale în scripting:

- Variables (Variabile): Păstrează informații temporare în script
- Conditionals (Condiționale): Permit executarea diferită bazată pe teste
- Loops (Bucle): Permit repetarea aceleiași acțiuni de mai multe ori

Variabile

Variabilele sunt o parte cheie a oricărui limbaj de programare.

Utilizare simplă a variabilelor:

```
#!/bin/bash
ANIMAL="pisica"
echo "Animalul meu preferat este: $ANIMAL"
```

Explicație:

- Numele variabilei este ANIMAL
- Semnul egal (=) atribuie string-ul "pisica"
- Pentru a accesa conținutul variabilei, se prefixează cu $ (dollar)
- Când interpretorul vede semnul $, recunoaște că va substitui conținutul variabilei(interpolation)

Variabile – atribuire cu output-ul unei comenzi

O altă modalitate de a atribui unei variabile este să folosești output-ul unei alte comenzi:

```
#!/bin/bash
CURRENT_DIR=$(pwd)
echo "Directorul curent este: $CURRENT_DIR"

# Sau cu backtick-uri (stil vechi)
DATE=`date`
echo "Data este: $DATE"
```

- Comanda pwd este executată
- Output-ul comenzii pwd este atribuit variabilei CURRENT_DIRECTORY
- Variabila poate fi apoi folosită în script

Variabile - Input de la Utilizator

Citirea input-ului de la utilizator:

Este de asemenea posibil să obții input de la utilizatorul scriptului și să atribui acel input unei variabile folosind comanda read.

Exemplu:

```
#!/bin/bash

echo -n "What is your name? "
read NAME
echo "Hello $NAME!"
```

Explicație:

- echo -n afișează mesajul fără newline (cursorul rămâne pe aceeași linie)
- read NAME așteaptă input de la utilizator și îl stochează în variabila NAME
- Variabila NAME poate fi folosită apoi în script cu $NAME

Variabile Speciale (argumente)

Variabile Speciale în Script-uri:

- $0 - Conține numele script-ului însuși
- $1, $2, $3... - Argumentele pasate script-ului
- $# - Numărul total de argumente
- $@ - Toate argumentele ca listă
- $? - Exit code-ul comenzii anterioare

Exemplu cu argumente:

```
#!/bin/bash
echo "Numele scriptului: $0"
echo "Primul argument: $1"
echo "Al doilea argument: $2"

# Rulare: ./script.sh Linux Programming
# Output
Numele scriptului: script.sh
Primul argument: Linux
Al doilea argument: Programming
```

Variabile

Exit Codes:

După ce un program rulează, acesta returnează un cod de ieșire (exit code) care este un întreg între 0 și 255.
Acesta poate fi testat prin variabila $? pentru a vedea dacă comanda anterioară s-a finalizat cu succes.
Comanda grep returnează 0 dacă șirul a fost găsit și 1 în caz contrar.

Setarea exit code-ului propriului script:

Poți seta, de asemenea, codul de ieșire al propriului script cu comanda exit.

Exemplu:

```
#!/bin/bash
grep -q root /etc/passwd
if [ $? -eq 0 ]; then
    echo "root was found"
    exit 0
else
    echo "root was not found"
    exit 1
fi
```

Instrucțiuni condiționale - if

Ramificarea (decizia condițională):
Poți face ca scriptul tău să execute funcții diferite pe baza testelor
Dacă condiția se îndeplinește(codul de ieșire este 0), se execută instrucțiunile din interiorul structurii

if statement:
Instrucțiunea if este operatorul de bază pentru implementarea ramificării.

O instrucțiune if de bază arată astfel:

```
if somecommand; then
    # do this if somecommand has an exit code of 0
fi
```

Instrucțiuni condiționale - if

Comanda test:
Comanda test îți oferă acces ușor la operatori de comparație și de testare a fișierelor.

Exemple:
De exemplu, operatorul -f la comanda test verifică dacă fișierul există:

```
test -f /dev/ttyS0
```
• Va testa dacă fișierul există

```
test -d /tmp
```
• Va testa dacă directorul există

Utilizare în if:

```
if test -f /dev/ttyS0; then
    echo "File exists"
fi
```

Instrucțiuni condiționale - if, elif, else

elif (else if):
Instrucțiunea if are o formă finală care îți permite să faci comparații multiple simultan folosind elif (prescurtare pentru else if).

Exemplu:

```
#!/bin/bash
if [ "$1" = "hello" ]; then
    echo "hello yourself"
elif [ "$1" = "goodbye" ]; then
    echo "nice to have met you"
    echo "I hope to see you again"
else
    echo "I didn't understand that"
fi
```

Explicație:
Dacă primul argument transmis scriptului este hello, primul bloc este executat. Dacă nu, scriptul verifică dacă este goodbye și afișează un mesaj diferit dacă este așa.
Altfel, este trimis un al treilea mesaj.

Instrucțiuni condiționale - case

Instrucțiunea case oferă o modalitate diferită de a face teste multiple mai ușor.

Structura case:

- Instrucțiunea case începe cu o descriere a expresiei testate: `case EXPRESSION in`
- Apoi, fiecare set de teste este executat ca o potrivire de pattern terminată cu o paranteză de închidere.
- Urmează comenzile care trebuie executate dacă pattern-ul returnează true, care sunt terminate cu două puncte și virgulă.
- Pattern-ul * este la fel ca un else deoarece potrivește orice.

```
#!/bin/bash

case "$1" in
  start|run)
    echo "Starting the process..."
    ;;
  stop|end)
    echo "Stopping the process..."
    echo "Process has ended."
    ;;
  status)
    echo "Checking the status..."
    ;;
  *)
    echo "Command not recognized."
    ;;
esac
```

Bucle (loops)

Loop-urile permit ca codul să fie executat în mod repetat pana se indeplineste o conditie.

Două loop-uri principale în shell scripts:
• for
• while

for
Buclele for sunt folosite când ai o colecție finită de elemente, cum ar fi o listă de fișiere, prin care vrei să iterezi (să repeți și să execuți cod pentru fiecare).

while
O buclă while operează pe o listă de dimensiune necunoscută.

Bucla - for

În exemplul de mai jos, o buclă for "loops" intr-o lista de servere:

```
#!/bin/bash
SERVERS="servera serverb serverc"

for S in $SERVERS; do
    echo "Doing something to $S"
done
```

Explicație:
- Variabila SERVERS reprezintă lista de servere (servera serverb serverc).
- La fiecare iterare, variabila S primește numele unui server din această listă.
- Comenzile dintre do și done sunt executate pentru fiecare element din lista .

Bucla - while

O buclă while continuă să ruleze și la fiecare iterație efectuează un test pentru a vedea dacă ar trebui să ruleze din nou.
Cu alte cuvinte, "cât timp o condiție este adevărată, executa instructiunile."

Exemplul de mai jos arată o buclă while care numără de la 0 la 9.

```
#!/bin/bash
i=0
while [ $i -lt 10 ]; do
    echo $i
    i=$(( $i + 1 ))
done
echo "Done counting"
```

Explicație:
- O variabilă contor, i, este inițializată la 0.
- Bucla continuă cât timp i este mai mic decât 10.
- La fiecare iterație, i este incrementat cu 1
  Exemplul afișează numerele de la 0 la 9.

Organizarea unui proiect software, backup simplu – exemplu practic

Punând totul cap la cap - pasi automatizare backup

1. Decizi ce vrei să faci backup
   - Alegi fișierele sau directoarele importante pe care vrei să le salvezi.

2. Stabilești unde să faci backup-ul
   - Selectezi locația unde vor fi salvate copiile de siguranță (ex: /backup).

3. Alegi metoda de backup
   - tar pentru arhivare completă într-un fișier .tar.gz
   - rsync pentru sincronizare incrementală și păstrarea structurii.

4. Scrii scriptul de backup
   - Construiești un script Bash simplu care să execute comanda de backup alesă.
   - Adaugi logică simplă pentru variabile (cale sursă, destinație, dată).
   - Testezi scriptul manual să te asiguri că funcționează corect.

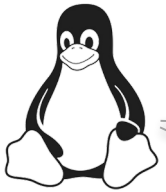
5. Programezi backup-ul automat cu cron
   - Editezi crontab și setezi ora la care să ruleze scriptul zilnic, săptămânal etc.

6. Monitorizezi și verifici backup-ul
   - Verifici periodic fișierele de backup în locația aleasă.
   - Poți adăuga în script scriere în fișier de log pentru a urmări execuțiile.

Q&A

Activități pentru acasă

Ce urmează?



- Înțelegerea Hardware-ului Computerului
- Configurarea Reţelei
-


SAVNET
UPgrade Yourself