Esercizio 3: Navigare nel Filesystem Linux e Impostazioni dei Permessi

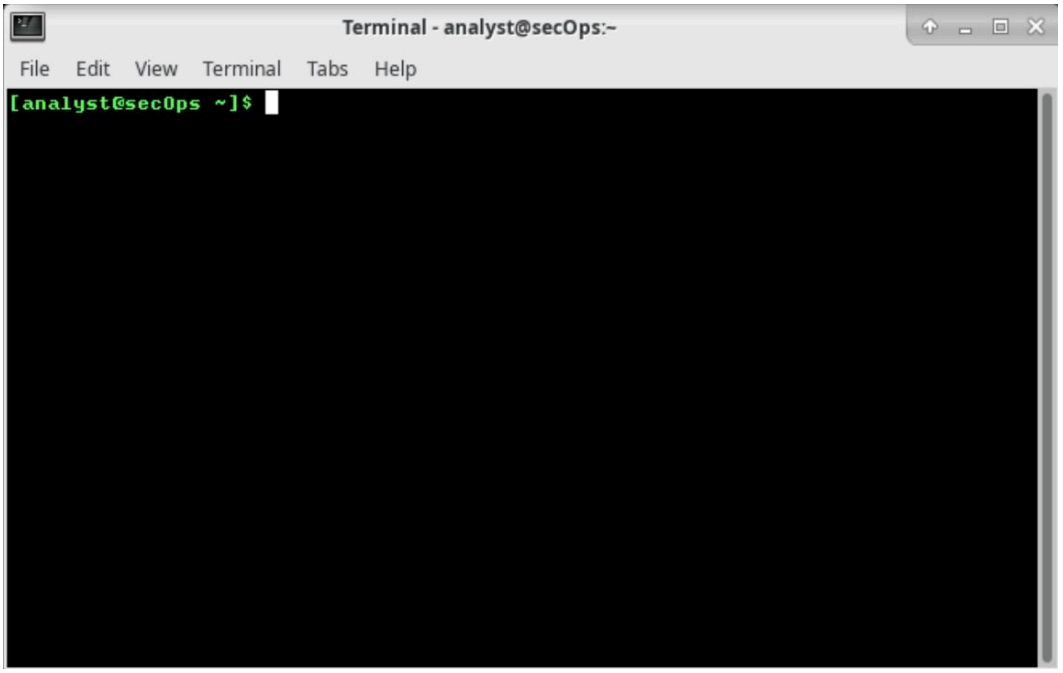
Obiettivi

In questo laboratorio ci esercitiamo a navigare il filesystem di Linux, esplorare le partizioni, montare e smontare dispositivi, e infine gestire permessi e proprietà di file e directory. Lavoriamo in ambiente virtuale, utilizzando la VM **CyberOps Workstation**.

Parte 1: Esplorare i Filesystem in Linux

PASSO 1: ACCEDERE ALLA RIGA DI COMANDO

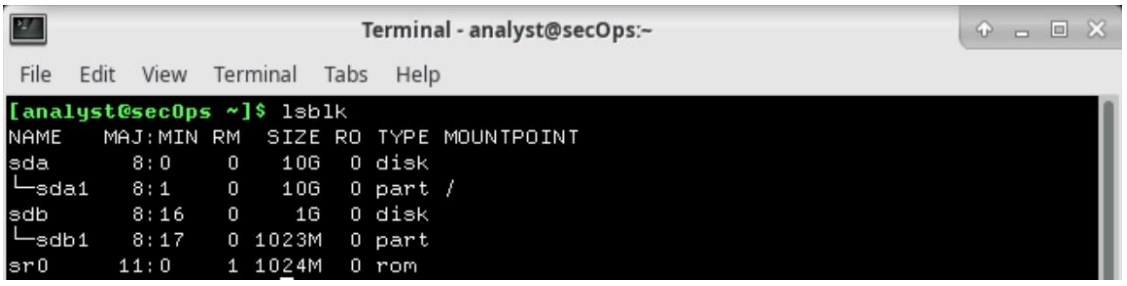
Per iniziare, accediamo alla **VM CyberOps Workstation** e apriamo una **finestra di terminale**. Tutte le operazioni verranno eseguite da qui.



PASSO 2: VISUALIZZARE I FILESYSTEM ATTUALMENTE MONTATI

Visualizziamo i dispositivi a blocchi collegati con il comando lsblk

Questo comando ci mostra i dispositivi a blocchi attualmente rilevati dal sistema, come dischi rigidi, SSD, chiavette USB, ecc. Vediamo che la VM dispone di tre dispositivi principali: sr0, sda e sdb. Di particolare interesse sono sda (disco da 10 GB) e sdb (disco da 1 GB), che rappresentano due hard disk.



Ora, con il comando mount vediamo un elenco di tutti i filesystem montati. Tra questi, ci concentriamo su: /dev/sda1 on / type ext4 (rw,relatime,data=ordered)

Significa che il filesystem principale (radice /) si trova sulla partizione /dev/sda1 e utilizza il formato ext4.

Per isolare il filesystem montato sulla partizione sda1, usiamo grep: mount | grep sda1
L'output conferma che /dev/sda1 è montato su /, la directory radice del sistema operativo.

```
[analyst@secOps ~]$ mount | grep sda1
/dev/sda1 on / type ext4 (rw,relatime,data=ordered)
```

Ci spostiamo nella directory radice e visualizziamo i file: cd / poi → ls -l

```
[analyst@secOps ~]$ cd /
[analyst@secOps /]$ ls -l
total 52
lrwxrwxrwx   1 root root     7 Jan  5  2018 bin -> usr/bin
drwxr-xr-x   3 root root  4096 Apr 16  2018 boot
drwxr-xr-x  19 root root  3120 Jun 16 05:26 dev
drwxr-xr-x  58 root root  4096 Apr 17  2018 etc
drwxr-xr-x   3 root root  4096 Mar 20  2018 home
lrwxrwxrwx   1 root root     7 Jan  5  2018 lib -> usr/lib
lrwxrwxrwx   1 root root     7 Jan  5  2018 lib64 -> usr/lib
drwx------   2 root root 16384 Mar 20  2018 lost+found
drwxr-xr-x   2 root root  4096 Jan  5  2018 mnt
drwxr-xr-x   2 root root  4096 Jan  5  2018 opt
dr-xr-xr-x 127 root root     0 Jun 16 05:26 proc
drwxr-x---   9 root root  4096 Jun 12 11:09 root
drwxr-xr-x  17 root root   480 Jun 16 05:26 run
lrwxrwxrwx   1 root root     7 Jan  5  2018 sbin -> usr/bin
drwxr-xr-x   6 root root  4096 Mar 24  2018 srv
dr-xr-xr-x  13 root root     0 Jun 16 05:26 sys
drwxrwxrwt   8 root root   200 Jun 16 05:28 tmp
drwxr-xr-x   9 root root  4096 Apr 17  2018 usr
drwxr-xr-x  12 root root  4096 Apr 17  2018 var
```

Domanda 1: Qual è il significato dell'output?
R: L'output del comando ls -l mostra **l'elenco dei file e delle cartelle presenti nella directory home dell'utente attualmente connesso**. L'output riflette quindi il **contenuto attuale della nostra home directory**, che si trova nel filesystem del disco principale.

Domanda 1.2: Dove sono fisicamente memorizzati i file elencati?
R: I file elencati sono **fisicamente memorizzati sul disco interno del sistema**, ovvero il disco principale, identificato come **/dev/sda** (più precisamente, la partizione root, ad esempio /dev/sda1).

Poiché la directory home (/home/analyst) fa parte del filesystem principale, tutto il suo contenuto risiede su quel disco.

In sintesi:

- Posizione logica: /home/analyst/
- Posizione fisica: sulla partizione principale (es. /dev/sda1), nel disco interno.

Domanda 1.3: Perché /dev/sdb1 non viene mostrato nell'output sopra?
R: Perché il comando ls -l elenca solo il **contenuto della directory corrente** (in questo caso, la home dell'utente), **non mostra i dispositivi o le partizioni del sistema**.

Inoltre, /dev/sdb1:

- **Non è montato** in alcuna directory al momento del comando, quindi **non fa parte del file system visibile all'utente**.
- Anche se fosse montato, ls -l nella home non lo mostrerebbe a meno che non fosse montato *dentro* una sottodirectory della home (es. ~/second_drive).

Per vedere i dispositivi collegati, servirebbero comandi come lsblk, sudo fdisk -l, o df -h.

PASSO 3: MONTARE E SMONTARE MANUALMENTE I FILESYSTEM

Verifichiamo che esista la directory second_drive in /home/analyst/: ls -l second_drive/
Notiamo che la directory è vuota.

```
[analyst@secOps ~]$ ls -l second_drive/
total 0
```

Montiamo /dev/sdb1 su second_drive

sudo mount /dev/sdb1 ~/second_drive/

```
[analyst@secOps ~]$ sudo mount /dev/sdb1 ~/second_drive/
[sudo] password for analyst:
```

Il comando ha successo: non compare, infatti, alcun messaggio dopo l'esecuzione.

Verifichiamo che ora la directory contenga dei file: ls -l second_drive/

```
[analyst@secOps ~]$ ls -l second_drive/
total 20
drwx------ 2 root    root    16384 Mar 26  2018 lost+found
-rw-r--r-- 1 analyst analyst   183 Mar 26  2018 myFile.txt
```

Vediamo, ad esempio, myFile.txt e lost+found.

Domanda 1: Perché la directory non è più vuota?
R: La directory second_drive, che inizialmente era vuota, non è più vuota perché abbiamo montato un nuovo filesystem all'interno di essa.
Il comando usato nel passo precedente è: sudo mount /dev/sdb1 ~/second_drive

Questo comando sovrappone (mount) il contenuto del dispositivo /dev/sdb1 — che contiene un filesystem già formattato e probabilmente con dei file — alla directory ~/second_drive.

Importante: quando montiamo un dispositivo su una directory, qualsiasi contenuto precedente di quella directory (anche se c'erano file) diventa temporaneamente nascosto e viene sostituito dal contenuto del filesystem montato.

Dunque, dopo il mount, se elenchiamo il contenuto della directory second_drive, vedremo i file e le cartelle che sono presenti nella partizione /dev/sdb1, e non quelli (eventualmente) originali della directory.

Domanda 1.1: Dove sono fisicamente memorizzati i file elencati?
R: I file elencati nella directory ~/second_drive/ **non si trovano sul disco principale (es. /dev/sda)**, ma sono **fisicamente memorizzati sulla seconda unità disco**, ovvero **sulla partizione /dev/sdb1**.

In pratica, ~/second_drive/ è solo un **punto di accesso (mount point)**: permette al sistema di accedere ai dati presenti su /dev/sdb1 **come se** fossero nella directory locale, ma in realtà sono salvati fisicamente sul secondo disco (quello che stiamo montando nel laboratorio).

Usiamo il comando: mount | grep /dev/sd
Conferma che sdb1 è montato sulla directory corretta.

```
[analyst@secOps ~]$ mount | grep /dev/sd
/dev/sda1 on / type ext4 (rw,relatime,data=ordered)
/dev/sdb1 on /home/analyst/second_drive type ext4 (rw,relatime,data=ordered)
```

Ci assicuriamo di uscire dalla directory montata, quindi smontiamo: sudo umount /dev/sdb1

```
[analyst@secOps ~]$ sudo umount /dev/sdb1
[sudo] password for analyst:
```

Verifichiamo che second_drive sia di nuovo vuota: ls -l second_drive/

```
[analyst@secOps ~]$ ls -l second_drive/
total 0
```

Parte 2: Permessi dei File

PASSO 1: VISUALIZZARE E MODIFICARE I PERMESSI DEI FILE

Navigare nella cartella scripts con comando: cd~/lab.support.files/scripts/ → poi ls -l

Analizziamo i permessi dei file elencati, ad esempio:

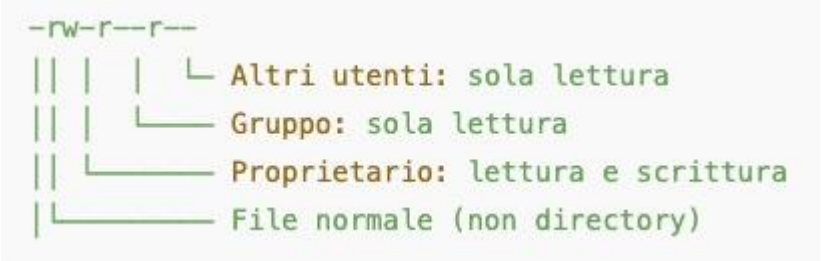
-rw-r--r-- 1 analyst analyst 2871 Apr 28 11:27 cyops.mn

```
[analyst@secOps ~]$ cd ~/lab.support.files/scripts/
[analyst@secOps scripts]$ ls -l
total 60
-rwxr-xr-x 1 analyst analyst  952 Mar 21  2018 configure_as_dhcp.sh
-rwxr-xr-x 1 analyst analyst 1153 Mar 21  2018 configure_as_static.sh
-rwxr-xr-x 1 analyst analyst 3459 Mar 21  2018 cyberops_extended_topo_no_fw.py
-rwxr-xr-x 1 analyst analyst 4062 Mar 21  2018 cyberops_extended_topo.py
-rwxr-xr-x 1 analyst analyst 3669 Mar 21  2018 cyberops_topo.py
-rw-r--r-- 1 analyst analyst 2871 Mar 21  2018 cyops.mn
-rwxr-xr-x 1 analyst analyst  458 Mar 21  2018 fw_rules
-rwxr-xr-x 1 analyst analyst   70 Mar 21  2018 mal_server_start.sh
drwxr-xr-x 2 analyst analyst 4096 Mar 21  2018 net_configuration_files
-rwxr-xr-x 1 analyst analyst   65 Mar 21  2018 reg_server_start.sh
-rwxr-xr-x 1 analyst analyst  189 Mar 21  2018 start_ELK.sh
-rwxr-xr-x 1 analyst analyst   85 Mar 21  2018 start_miniedit.sh
-rwxr-xr-x 1 analyst analyst   76 Mar 21  2018 start_pox.sh
-rwxr-xr-x 1 analyst analyst  106 Mar 21  2018 start_snort.sh
-rwxr-xr-x 1 analyst analyst   61 Mar 21  2018 start_tftpd.sh
```

Domanda 1: Chi è il proprietario del file cyops.mn?
R: Il proprietario è indicato nella **terza colonna** dell'output del comando ls -l. In questo caso, il proprietario è **analyst**.

Domanda 1.2: E il gruppo?
R: Il gruppo è mostrato nella quarta colonna dell'output di ls -l. Qui, il gruppo associato al file è **analyst.**

Domanda 2: I permessi per cyops.mn sono –rw-r–r–. Cosa significa?
R: Questa stringa di permessi si legge così:



Significato dettagliato:

Simbolo	Significato
`-`	È un file regolare (non una directory)
`rw-`	Il **proprietario** può leggere e scrivere
`r--`	Il **gruppo** può solo leggere
`r--`	**Tutti gli altri** possono solo leggere

In pratica, **solo il proprietario** (analyst) **può leggere e scrivere il file** cyops.mn, mentre gruppo e altri possono **solo leggerlo**.

Creiamo un file di testo nella directory /mnt con il comando touch /mnt/myNewFile.txt

Il comando fallisce con un errore: **Permission denied**.

```
[analyst@secOps scripts]$ touch /mnt/myNewFile.txt
touch: cannot touch '/mnt/myNewFile.txt': Permission denied
```

Domanda 3: Perché il file non è stato creato? Elenca i permessi, la proprietà e il contenuto della directory /mnt e spiega cosa è successo.

R: Per rispondere, esaminiamo il contenuto della directory **/mnt** con il comando: **ls -l /mnt**e osserviamo il risultato:

```
[analyst@secOps scripts]$ ls -ld /mnt
drwxr-xr-x 2 root root 4096 Jan  5  2018 /mnt
```

- **Permessi**: drwxr-xr-x →
  - Il proprietario (root) può leggere, scrivere ed entrare nella directory.
  - Il gruppo (root) e gli altri possono solo leggere ed entrare (non scrivere).
- **Proprietario**: root
- **Gruppo**: root

Al momento, siamo loggati come utente analyst, quindi **non siamo root**: quindi **non abbiamo i permessi di scrittura** in /mnt. **Il file non è stato creato perché l'utente non ha i permessi per scrivere dentro la directory /mnt**, che è di proprietà di root.

Domanda 3.1: Con l'aggiunta dell'opzione -d, elenca i permessi della directory genitore:
R: Comando: ls -ld /mnt

Output registrato: drwxr-xr-x 2 root root 4096 Jan 5 2018 /mnt

Significato:

- d → è una directory
- rwx (proprietario root) → può leggere, scrivere ed entrare
- r-x (gruppo root) → può leggere ed entrare
- r-x (altri) → può leggere ed entrare → Nessuno tranne root può **scrivere** nella directory /mnt.

Domanda 4: Cosa si può fare affinché il comando '*touch*' abbia successo?
R: Abbiamo 3 opzioni:

Soluzione 1: Utilizzare sudo

Si può eseguire il comando con sudo (possibile perché l'utente analyst è nel gruppo sudo):

sudo touch /mnt/myNewFile.txt

Soluzione 2: Cambiare i permessi (non consigliato su sistemi multiutente o in ambienti reali)

Possiamo permettere la scrittura a tutti (molto rischioso): sudo chmod o+w /mnt

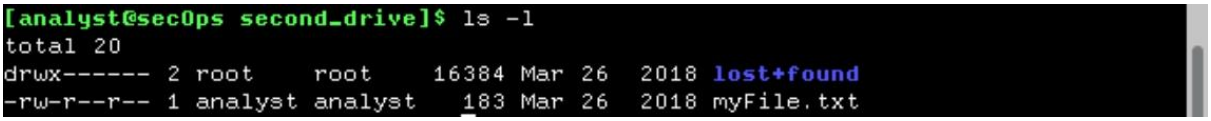
Soluzione 3: Cambiare proprietario o gruppo della directory

Possiamo rendere la directory scrivibile per il nostro utente analyst: sudo chown analyst /mnt

oppure: sudo chgrp analyst /mnt && sudo chmod g+w /mnt

Montiamo di nuovo /dev/sdb1 su second_drive con il comando:

sudo mount /dev/sdb1 ~/second_drive/ poi → cd ~/second_drive poi → ls -l



Domanda 5: Quali sono i permessi del file myFile.txt?

R: Analizzando la stringa da sinistra verso destra, il primo carattere (-) ci dice che si tratta di un **file regolare**, quindi non è una directory, né un collegamento simbolico o un file speciale. myFile.txt è un file leggibile da chiunque, ma modificabile solo dal proprietario.

È un'impostazione tipica per i file di testo che devono essere consultabili da altri, ma la cui modifica deve restare sotto controllo.

Categoria	Permessi	Azioni consentite
Proprietario	`rw-`	Legge, scrive, non esegue
Gruppo	`r--`	Solo lettura
Altri	`r--`	Solo lettura

Cambiamo i permessi con il comando **sudo chmod 665 myFile.txt** **poi** → **ls -l**

```
[analyst@secOps second_drive]$ sudo chmod 665 myFile.txt
[sudo] password for analyst:
[analyst@secOps second_drive]$ ls -l
total 20
drwx------ 2 root    root    16384 Mar 26  2018 lost+found
-rw-rw-r-x 1 analyst analyst   183 Mar 26  2018 myFile.txt
```

Domanda 6: **I permessi sono cambiati? Quali sono i permessi di myFile.txt?**
R: Il file ora ha i permessi **-rw-rw-r-x**, cioè:

- Proprietario: lettura e scrittura
- Gruppo: lettura e scrittura

Altri: lettura ed esecuzione

Domanda 7: **Quale comando cambierebbe i permessi di myFile.txt a rwxrwxrwx, garantendo a qualsiasi utente nel sistema pieno accesso al file?**
R: Il comando chmod 777 myFile.txt.

Cambiamo il proprietario del file con il comando sudo chown analyst myFile.txt poi → ls -l

```
[analyst@secOps second_drive]$ sudo chown analyst myFile.txt
[analyst@secOps second_drive]$ ls -l
total 20
drwx------ 2 root    root    16384 Mar 26  2018 lost+found
-rw-rw-r-x 1 analyst analyst   183 Mar 26  2018 myFile.txt
```

Scriviamo all'interno del file con echo test >> myFile.txt **poi** → cat myFile.txt

```
[analyst@secOps second_drive]$ echo test >> myFile.txt
[analyst@secOps second_drive]$ cat myFile.txt
This is a file stored in the /dev/sdb1 disk.
Notice that even though this file has been sitting in this disk for a while, it
couldn't be accessed until the disk was properly mounted.
test
```

Domanda 8: **L'operazione è riuscita? Spiega.**
R: L'operazione è riuscita perché ora abbiamo i permessi per scrivere nel file.

PASSO: DIRECTORY E PERMESSI

Per visualizzazione il contenuto della directory lab.support.files digitiamo: cd ~/lab.support.files poi → ls -l

```
[analyst@secOps second_drive]$ cd ~/lab.support.files
[analyst@secOps lab.support.files]$ ls -l
total 580
-rw-r--r-- 1 analyst analyst    649 Mar 21  2018 apache_in_epoch.log
-rw-r--r-- 1 analyst analyst    126 Mar 21  2018 applicationX_in_epoch.log
drwxr-xr-x 4 analyst analyst   4096 Mar 21  2018 attack_scripts
-rw-r--r-- 1 analyst analyst    102 Mar 21  2018 confidential.txt
-rw-r--r-- 1 analyst analyst   2871 Mar 21  2018 cyops.mn
-rw-r--r-- 1 analyst analyst     75 Mar 21  2018 elk_services
-rw-r--r-- 1 analyst analyst    373 Mar 21  2018 h2_dropbear.banner
drwxr-xr-x 2 analyst analyst   4096 Apr  2  2018 instructor
-rw-r--r-- 1 analyst analyst    255 Mar 21  2018 letter_to_grandma.txt
-rw-r--r-- 1 analyst analyst  24464 Mar 21  2018 logstash-tutorial.log
drwxr-xr-x 2 analyst analyst   4096 Mar 21  2018 malware
-rwxr-xr-x 1 analyst analyst    172 Mar 21  2018 mininet_services
drwxr-xr-x 2 analyst analyst   4096 Mar 21  2018 openssl_lab
drwxr-xr-x 2 analyst analyst   4096 Mar 21  2018 pcaps
drwxr-xr-x 7 analyst analyst   4096 Mar 21  2018 pox
-rw-r--r-- 1 analyst analyst 473363 Mar 21  2018 sample.img
-rw-r--r-- 1 analyst analyst     65 Mar 21  2018 sample.img_SHA256.sig
drwxr-xr-x 3 analyst analyst   4096 Mar 21  2018 scripts
-rw-r--r-- 1 analyst analyst  25553 Mar 21  2018 SQL_Lab.pcap
```

Domanda 1: Qual è la differenza tra la parte iniziale della riga di malware e la riga di mininet_services?
R: Analizziamo e confrontiamo i permessi delle due voci:

rwx (proprietario: può leggere, scrivere e accedere alla directory)
r-x (gruppo: può leggere e accedere ma non scrivere)
r-x (altri: idem)

La prima lettera dei permessi è **diversa**:

- d per **malware** → indica una **directory**
- - per **mininet_services** → indica un **file regolare**

Questa differenza è fondamentale: anche se i permessi per utente, gruppo e altri sono gli stessi (rwxr-xr-x), la **natura dell'oggetto** (directory vs file) cambia il significato pratico dei permessi.

Parte 3: Link simbolici e altri tipi di file speciali

PASSO 1: ESAMINARE I TIPI DI FILE IN /home/analyst

Digitiamo il comando: ls -l /home/analyst

```
[analyst@secOps lab.support.files]$ cd
[analyst@secOps ~]$ ls -l
total 32
drwxr-xr-x 2 analyst analyst 4096 Jun 12 13:58 cyops_folder2
drwxr-xr-x 3 analyst analyst 4096 Jun 12 11:52 cyops_folder3
drwxr-xr-x 2 analyst analyst 4096 Mar 22  2018 Desktop
drwxr-xr-x 3 analyst analyst 4096 Mar 22  2018 Downloads
-rw-r--r-- 1 analyst analyst   24 Jun 12 12:37 fileditesto.txt
drwxr-xr-x 9 analyst analyst 4096 Jul 19  2018 lab.support.files
drwxr-xr-x 3 root    root    4096 Mar 26  2018 second_drive
-rw-r--r-- 1 analyst analyst  312 Jun 12 08:37 space.txt
```

Osservazioni:

- Le righe che iniziano con d sono directory.
- Le righe che iniziano con - sono file regolari.
- Nessun file speciale in questa directory (per ora).

Comando: ls -l /dev

```
[analyst@secOps ~]$ ls -l /dev
```

```
crw-------  1 root root      108,    0 Jun 16 05:26 ppp
crw-------  1 root root       10,    1 Jun 16 05:26 psaux
crw-rw-rw-  1 root tty         5,    2 Jun 16 10:47 ptmx
drwxr-xr-x  2 root root              0 Jun 16 05:26 pts
crw-rw-rw-  1 root root        1,    8 Jun 16 05:26 random
lrwxrwxrwx  1 root root              4 Jun 16 05:26 rtc -> rtc0
crw-rw----  1 root audio     250,    0 Jun 16 05:26 rtc0
brw-rw----  1 root disk        8,    0 Jun 16 05:26 sda
brw-rw----  1 root disk        8,    1 Jun 16 05:26 sda1
brw-rw----  1 root disk        8,   16 Jun 16 05:26 sdb
brw-rw----  1 root disk        8,   17 Jun 16 05:26 sdb1
drwxrwxrwt  2 root root             40 Jun 16 05:26 shm
crw-------  1 root root       10,  231 Jun 16 05:26 snapshot
drwxr-xr-x  3 root root            180 Jun 16 05:26 snd
brw-rw----+ 1 root optical    11,    0 Jun 16 05:26 sr0
lrwxrwxrwx  1 root root             15 Jun 16 05:26 stderr -> /proc/self/fd/2
lrwxrwxrwx  1 root root             15 Jun 16 05:26 stdin -> /proc/self/fd/0
lrwxrwxrwx  1 root root             15 Jun 16 05:26 stdout -> /proc/self/fd/1
crw-rw-rw-  1 root tty         5,    0 Jun 16 09:55 tty
crw--w----  1 root tty         4,    0 Jun 16 05:26 tty0
crw--w----  1 root tty         4,    1 Jun 16 05:26 tty1
crw--w----  1 root tty         4,   10 Jun 16 05:26 tty10
crw--w----  1 root tty         4,   11 Jun 16 05:26 tty11
crw--w----  1 root tty         4,   12 Jun 16 05:26 tty12
```

Osservazioni:

- b: file a blocchi (es. sda, sdb)
- c: file a caratteri (es. tty, random)
- l: link simbolici (es. stdout -> /proc/self/fd/1)

Per creare il file di testo digitiamo: echo "symbolic" > file1.txt poi → cat file1.txt

```
[analyst@secOps ~]$ echo "symbolic" > file1.txt
[analyst@secOps ~]$ cat file1.txt
symbolic
```

E poi ancora digitiamo → echo "hard" > file2.txt → cat file2.txt

```
[analyst@secOps ~]$ echo "hard" > file2.txt
[analyst@secOps ~]$ cat file2.txt
hard
```

Per creare i link digitiamo:
ln -s file1.txt file1symbolic # Link simbolico
ln file2.txt file2hard # Hard link

```
[analyst@secOps ~]$ ln -s file1.txt file1symbolic
[analyst@secOps ~]$ ln file2.txt file2hard
```

Comando: ls -

```
[analyst@secOps ~]$ ls -l
total 44
drwxr-xr-x 2 analyst analyst 4096 Jun 12 13:58 cyops_folder2
drwxr-xr-x 3 analyst analyst 4096 Jun 12 11:52 cyops_folder3
drwxr-xr-x 2 analyst analyst 4096 Mar 22  2018 Desktop
drwxr-xr-x 3 analyst analyst 4096 Mar 22  2018 Downloads
lrwxrwxrwx 1 analyst analyst    9 Jun 16 10:58 file1symbolic -> file1.txt
-rw-r--r-- 1 analyst analyst    9 Jun 16 10:54 file1.txt
-rw-r--r-- 2 analyst analyst    5 Jun 16 10:55 file2hard
-rw-r--r-- 2 analyst analyst    5 Jun 16 10:55 file2.txt
-rw-r--r-- 1 analyst analyst   24 Jun 12 12:37 fileditesto.txt
drwxr-xr-x 9 analyst analyst 4096 Jul 19  2018 lab.support.files
drwxr-xr-x 3 root    root    4096 Mar 26  2018 second_drive
-rw-r--r-- 1 analyst analyst  312 Jun 12 08:37 space.txt
```

Osservazioni:

- file1symbolic inizia con l, mostra -> file1.txt
- file2hard appare come file regolare, ma con lo **stesso inode** di file2.txt (quinta colonna = 2)

Comandi: mv file1.txt file1new.txt poi → mv file2.txt file2new.txt

```
[analyst@secOps ~]$ mv file1.txt file1new.txt
[analyst@secOps ~]$ mv file2.txt file2new.txt
```

Verifichiamo il funzionamento link digitando: cat file1symbolic poi → cat file2hard

```
[analyst@secOps ~]$ cat file1symbolic
cat: file1symbolic: No such file or directory
```

```
[analyst@secOps ~]$ cat file2hard
hard
```

Osservazioni:

- file1symbolic fallisce: **link simbolico rotto**.
- file2hard funziona: **hard link rimane valido** perché punta all'inode.

Domanda: <u>Cosa pensi succederebbe a file2hard se aprissi un editor di testo e cambiassi il testo in file2new.txt?</u>
R: Poiché **file2hard** e **file2new.txt** condividono lo stesso inode, qualsiasi modifica a uno si rifletterà anche nell'altro. Sono **fisicamente** lo stesso file.

Osservazioni:

- I **link simbolici** sono **puntatori al nome** di un file: se il file viene rinominato o spostato, il link si rompe.
  Gli **hard link** sono **puntatori all'inode**: rimangono validi finché l'inode esiste, anche se il file viene rinominato.

Conclusioni

In questo laboratorio abbiamo imparato a:

- Esplorare i filesystem Linux e identificarne il tipo e il punto di montaggio.
- Montare e smontare manualmente partizioni.
- Interpretare e modificare i permessi di file e directory.
- Usare i comandi chmod, chown, touch, mount, umount, lsblk, ls -l, grep.

Abbiamo visto come Linux gestisce la sicurezza del filesystem attraverso permessi e proprietà, fornendo un controllo granulare su chi può fare cosa con ogni singolo file o directory.

Inoltre, è stato visto in modo approfondito il funzionamento dei **diversi tipi di file** in Linux, con particolare attenzione a quelli **speciali** e ai **link simbolici e hard link**. Le principali conclusioni emerse sono:

- Il **primo carattere** mostrato dal comando ls -l è fondamentale per identificare il tipo di file: file regolari, directory, dispositivi di sistema, link, socket, pipe, ecc.

- I **link simbolici** (indicati con l) si comportano come scorciatoie e **puntano al nome** di un altro file. Questo li rende flessibili ma anche vulnerabili: se il file di destinazione cambia nome o viene rimosso, il link simbolico smette di funzionare (diventa “rotto”).

- Gli **hard link** (apparentemente indistinguibili da file normali) **condividono lo stesso inode** del file originale. Questo significa che continuano a funzionare anche se il file originale cambia nome o viene spostato, poiché puntano direttamente ai dati sul disco.

- Abbiamo visto che modificare il contenuto di un file attraverso un hard link **modifica anche il file originale**, poiché entrambi accedono agli stessi blocchi dati.

Dal punto di vista della **cybersecurity**, è fondamentale:

- Saper interpretare correttamente i permessi e la struttura dei file.
  Riconoscere i link simbolici e gli hard link, che possono essere utilizzati anche in modo malevolo per **mascherare o duplicare contenuti** nei sistemi.

- Capire come i file speciali in /dev rappresentano i **punti di accesso ai dispositivi hardware**, un aspetto critico per il controllo delle autorizzazioni.
