

Report: Gestione dei Permessi in Linux

1. Introduzione

Lo scopo di questo esercizio è comprendere il funzionamento dei permessi in Linux, attraverso la creazione di file e directory, la modifica dei permessi con `chmod` e la verifica pratica con test di scrittura e creazione di file. La gestione corretta dei permessi è un aspetto fondamentale per la sicurezza e l'organizzazione in ambienti multiutente.

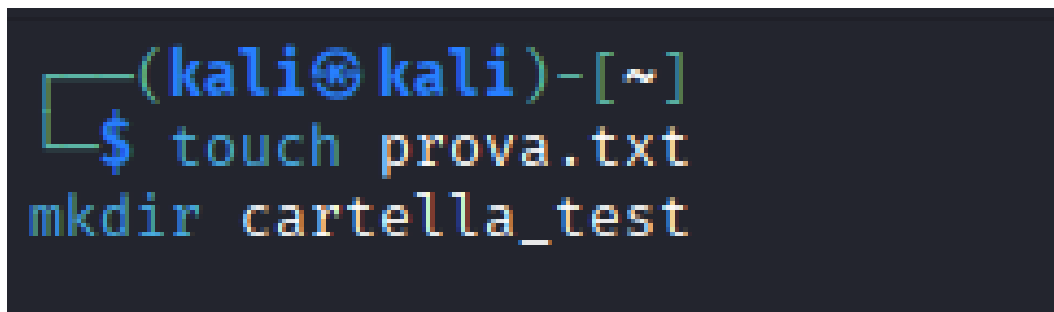
2. Creazione di file e directory

Comandi eseguiti:

```
touch prova.txt  
mkdir cartella_test
```

Questi comandi creano rispettivamente un file vuoto (`prova.txt`) e una nuova directory (`cartella_test`). Sono la base per testare i permessi.

Ho deciso di creare un file vuoto (`prova.txt`) e una directory (`cartella_test`) per avere un ambiente di prova sicuro. In questo modo i test sui permessi non rischiano di alterare file o cartelle di sistema. Questa è una buona pratica in laboratorio e nei contesti di test.

A screenshot of a terminal window with a dark background. The prompt is `(kali@kali)-[~]`. The user has entered two commands: `$ touch prova.txt` and `mkdir cartella_test`.

```
(kali@kali)-[~]  
$ touch prova.txt  
mkdir cartella_test
```

3. Verifica dei permessi iniziali

Comandi eseguiti:

```
ls -l prova.txt  
ls -l cartella_test
```

Con `ls -l` possiamo vedere i permessi iniziali del file e della directory. Per default, il file ha permessi di lettura e scrittura per l'utente, e lettura per gruppo e altri. La cartella appare inizialmente vuota.

Prima di procedere con modifiche è fondamentale sapere quali sono i permessi iniziali. Il comando `ls -l` mostra in dettaglio chi può leggere, scrivere o eseguire un file o una

cartella. Questo passaggio permette di avere un punto di partenza per confrontare i risultati delle modifiche.

```
(kali㉿kali)-[~]  
$ ls -l prova.txt  
ls -l cartella_test  
  
-rw-rw-r-- 1 kali kali 0 Sep 16 05:43 prova.txt  
total 0
```

```
(kali㉿kali)-[~]  
$ ls -l cartella_test  
  
total 0
```

4. Modifica dei permessi

Comandi eseguiti:

```
chmod u=rw,g=r,o=r prova.txt  
chmod u=rwx,g=rx,o=rx cartella_test
```

Con chmod abbiamo modificato i permessi:

- prova.txt: l'utente può leggere e scrivere, mentre gruppo e altri solo leggere.
- cartella_test: l'utente può leggere, scrivere ed entrare, mentre gruppo e altri possono solo leggere ed entrare.

Questa configurazione permette al proprietario di gestire pienamente i file, ma limita le azioni degli altri.

Sul file prova.txt ho scelto i permessi u=rw,g=r,o=r per permettere solo al proprietario di scrivere e modificare il file, mentre gli altri possono solo leggerlo.

Sulla directory cartella_test ho applicato i permessi u=rwx,g=rx,o=rx: il proprietario può creare e modificare file, mentre gli altri possono solo accedere e leggere. Questa configurazione è molto comune in ambienti condivisi, perché bilancia sicurezza e accessibilità.

```
(kali@kali)-[~]  
$ chmod u=rw,g=r,o=r prova.txt  
  
(kali@kali)-[~]  
$ chmod u=rwx,g=rx,o=rx cartella_test
```

5. Test dei permessi

Comandi eseguiti:

```
echo "Verifica dei permessi" > prova.txt  
touch cartella_test/nuovo_doc.txt  
ls -l cartella_test/nuovo_doc.txt
```

Abbiamo verificato i permessi effettuando due prove pratiche:

- Scrittura dentro il file prova.txt, che è permessa perché l'utente ha rw.
- Creazione di un nuovo file (nuovo_doc.txt) dentro cartella_test, che funziona perché l'utente ha rwx.

Infine, abbiamo controllato i permessi del nuovo file.

Ho verificato i permessi con prove pratiche: scrivendo dentro prova.txt per controllare che il proprietario possa effettivamente modificarlo e creando un nuovo file dentro cartella_test per confermare che il proprietario abbia i permessi di scrittura. Infine, ho usato `ls -l` per controllare i permessi del nuovo file creato.

```
(kali@kali)-[~]  
$ echo "Verifica dei permessi" > prova.txt
```

```
(kali@kali)-[~]  
$ touch cartella_test/nuovo_doc.txt  
  
(kali@kali)-[~]  
$ ls -l cartella_test/nuovo_doc.txt  
  
-rw-rw-r-- 1 kali kali 0 Sep 16 05:46 cartella_test/nuovo_doc.txt
```

6. Conclusioni

Questo esercizio dimostra come in Linux la gestione dei permessi sia fondamentale per controllare l'accesso a file e directory. Permessi troppo permissivi possono rappresentare un rischio di sicurezza, mentre permessi troppo restrittivi possono impedire il lavoro degli utenti.

Scenari reali di applicazione:

- Protezione di file sensibili (chiavi SSH, configurazioni di database).
- Gestione di directory condivise in ambienti multiutente.
- Sicurezza in contesti di pentesting e hardening del sistema.

Una corretta configurazione dei permessi garantisce che solo chi è autorizzato possa leggere, scrivere o eseguire file e directory.

7. Consultazione del manuale Linux

Per documentarmi sui comandi utilizzati, ho consultato il manuale integrato di Linux tramite il comando `man`.

In particolare:

`man chmod` per capire la sintassi corretta e le diverse modalità di assegnazione dei permessi.

Questa scelta dimostra l'importanza di utilizzare la documentazione ufficiale, sempre disponibile direttamente nel sistema, per avere spiegazioni affidabili e complete.

Ovviamente ho chiesto anche aiuto a chat gpt per conoscere meglio la funzione dei comandi.

check the underlying system behavior.

For directories **chmod** preserves set-user-ID and set-group-ID bits unless you explicitly specify otherwise. You can set or clear the bits with symbolic modes like **u+s** and **g-s**. To clear these bits for directories with a numeric mode requires an additional leading zero like **00755**, leading minus like **-6000**, or leading equals like **=755**.

RESTRICTED DELETION FLAG OR STICKY BIT

The restricted deletion flag or sticky bit is a single bit, whose interpretation depends on the file type. For directories, it prevents unprivileged users from removing or renaming a file in the directory unless they own the file or the directory; this is called the restricted deletion flag for the directory, and is commonly found on world-writable directories like **/tmp**. For regular files on some older systems, the bit saves the program's text image on the swap device so it will load more quickly when run; this is called the sticky bit.

OPTIONS

Change the mode of each FILE to MODE. With **--reference**, change the mode of each FILE to that of RFILE.

-c, --changes

like verbose but report only when a change is made

-f, --silent, --quiet

suppress most error messages

-v, --verbose

output a diagnostic for every file processed

--dereference

affect the referent of each symbolic link, rather than the symbolic link itself

-h, --no-dereference

affect each symbolic link, rather than the referent