C. Gaibisso
B. Martino

Programmazione e Laboratorio di Programmazione

Lezione XII.IV
Gestione dei file
Uso della *shell*
I permessi di accesso

C. Gaibisso
B. Martino

Permessi associati ai files

- Unix è un sistema multiuser per cui appare naturale che incorpori un sistema di protezione per prevenire accessi non autorizzati ai files dei singoli utenti.

- Il sistema riconosce quattro classi di accesso ad un file:
  - Accesso del super utente *(root)*
  - Accesso del proprietario del file (*owner*)
  - Accesso degli appartenenti allo stesso gruppo del proprietario *(group)*
  - Accesso di tutti gli altri utenti del sistema *(others)*

Permessi associati ai files

- Il super utente (root) ha libero accesso a qualunque file appartenente al file system. Per questo motivo è consigliabile evitare di effettuare attività diverse da quelle di amministrazione di sistema con questo account.
  Un errore può costare caro!

- Un generico file ammette tre diversi livelli di protezione:

  - r: Permesso di aprire il file in lettura
  - w: Permesso di aprire il file in scrittura
  - x: Permesso di aprire il file in esecuzione

Permessi associati ai files

- Per esaminare i permessi associati ad un file si usa il comando ls con l'opzione –l; il comando mostra nella prima colonna del flusso di uscita una stringa di 10 caratteri.
- Il primo carattere identifica il tipo di file ("d" per le direttory, "-" per i file ordinari ed altri caratteri per file di tipo speciale).
- I rimanenti nove caratteri identificano i permessi sopra descritti. I primi tre dei nove caratteri relativi ai premessi si riferiscono al proprietario, i secondi tre al gruppo ed i terzi agli altri utenti.

Permessi associati ai files

- Il permesso di lettura implica che è possibile esaminare il contenuto del file o copiarlo. Quasi ogni comando che usa un file esistente ha bisogno del permesso di lettura su quel file. Per esempio anche possedendo il permesso di esecuzione non è possibile eseguire un file senza possedere anche il permesso di lettura.
- Il permesso in scrittura implica che è possibile modificare il contenuto del file cioè creare, alterare o cancellare il contenuto del file (il contenuto, non il nome, per questo si vedranno le interpretazioni dei permessi associate ai direttori).
- Il permesso di esecuzione implica l'eseguibilità del file.

Permessi associati ai files

Per cancellare un file ordinario non occorre possedere alcun permesso su quel file, è sufficiente il permesso in scrittura sulla directory che contiene il file. Il comando di cancellazione rm, se usato interattivamente, procede alla cancellazione solo se si possiede anche il permesso di scrittura sul file, altrimenti richiede conferma (avuta la quale però, effettua comunque la cancellazione).

Permessi associati ai files

Se una directory ha il permesso “r” è possibile accedere in lettura al contenuto del direttorio cioè elencare (con il comando ls senza opzioni) i nomi dei file contenuti. Anche l'espansione di nomi di file (per es. con il metacarattere '*') da parte delle shell ha bisogno di questo permesso per poter operare. Se si desiderano maggiori informazioni sui file (per esempio le informazioni ottenibili con un comando "ls -l") è necessario avere accesso anche in esecuzione.

Permessi associati ai files

- L'accesso ad una directory non è sufficiente a garantire l'accesso al contenuto dei singoli file listati: per esaminare il contenuto di uno specifico file si devono avere i permessi opportuni per quel file.
- Se una directory ha il permesso “w” è possibile modificarne il contenuto cioè inserire o cancellare file. Per modificare il contenuto di un file si deve possedere il permesso di scrittura su quel file.
- Il permesso in esecuzione per un direttorio (“x”) consente ad un utente di effettuare un cd nella directory.
- Come già detto root ha i permessi su tutti i file con una sola eccezione che riguarda il permesso di esecuzione di un file se tutti tre i permessi di esecuzione sono negati. In UNIX non è possibile vietare l'accesso a file o direttori da parte del super-user.

Modifica dei permessi

- I nove permessi o diritti di accesso vengono spesso indicati collettivamente anche con il nome di "**mode**".
- Solo il proprietario ed il super-user possono cambiare il mode di un file. Il comando da impiegare è **chmod** (CHange MODe). La sintassi associata al comando è la seguente:

  chmod <mode> <files>

  Dove <files> rappresenta l'insieme di file di cui si intende cambiare il mode mentre il <mode> indica il nuovo mode richiesto. L'espressione <mode> può essere specificata in due maniere: assoluta (cioè numerica) o simbolica.

Modifica dei permessi

- Il modo più semplice è quello assoluto per chi ha dimestichezza con la rappresentazione ottale. Per usare questa rappresentazione è sufficiente rappresentare i nove permessi in tre gruppi di tre cifre binarie e tradurre ciascun gruppo in ottale.

  Esempio:

proprietario	gruppo	altri
RWX	RWX	RWX
rw-	r--	---
110	100	000
6	4	0

Modifica dei permessi

- Ad esempio, per assegnare al file std.c permesso il lettura e scrittura al proprietario, lettura per i componenti del gruppo di appartenenza e nessun accesso per gli altri utenti:

```
# ls -l std.c
-rw-r--r--    1 root     root          175 Mar 11 19:34 std.c
# chmod 640 std.c
# ls -l std.c
-rw-r-----    1 root     root          175 Mar 11 19:34 std.c
#
```

- Una modifica dei permessi effettuata in modo assoluto non permette la modifica di un solo permesso lasciando gli altri inalterati, occorre specificarli tutti.

Modifica dei permessi

- Il formato simbolico è un po' più complesso e si usa come segue:

chmod [<chi>] <op> <permessi> <files>

<chi> è una combinazione delle lettere qui elencate:

u: proprietario (owner, **U**ser);

g: **G**ruppo;

o: altri (**O**ther);

a: tutti (**A**ll, cioè "ugo").

Modifica dei permessi

- <op> può essere uno dei caratteri "+", "-" o "=" a seconda che il permesso vada (rispettivamente) aggiunto, tolto o reso identico a quello specificato nei <permessi> e per le categorie di utenti individuati da <chi>.

- <permessi> è una combinazione delle lettere "r" (read), "w" (write) e "x" (esecuzione) che rappresentano i permessi da modificare.

Modifica dei permessi

- Esempi di mode simbolico:

u+x aggiunge il permesso di eseguibilità al proprietario

go-wx vengono tolti i permessi di scrittura ed eseguibilità per gruppo e altri

-x viene tolta l'eseguibilità per tutti gli utenti, proprietario compreso

o=r agli altri viene assegnato il solo permesso di lettura (se non c'era permesso di lettura viene accordato e se c'erano permessi in scrittura o esecuzione vengono negati)

Modifica dei permessi

- il comando *chmod* in realtà ammette un quarto gruppo di permessi. Il quarto gruppo, che in realtà nella specificazione del mode precede gli altri tre, viene impiegato dal super-user e si riferisce esclusivamente a file eseguibili.
- Se il primo bit del quarto gruppo (bit chiamato "**set user-ID**") è alto, il Process Effective User ID diventa uguale allo User ID del file in esecuzione per il tempo dell'esecuzione. Significato analogo ha il secondo bit ("set group-ID") che agisce temporaneamente sul Process Effective Group ID. Quando questi due bit sono presenti, i due convenzionali bit di permesso di esecuzione per proprietario e gruppo, che solitamente vengono indicati con "x", vengono trasformati in "s" o "t".

Modifica dei permessi

- Il terzo bit viene chiamato "**sticky-bit**" ed è usato per gli eseguibili condivisibili da più utenti. Se il bit è attivo l'area di swap del programma non viene rilasciata anche se nessun utente sta usando il programma. Questo in generale migliora la risposta del sistema nell'uso di questi eseguibili.
- Benché ormai non venga più utilizzato per i file, lo *sticky bit* ha invece assunto un uso importante per le directory; in questo caso se tale bit è impostato un file potrà essere rimosso dalla directory soltanto se l'utente ha il permesso di scrittura su di essa ed inoltre è vera una delle seguenti condizioni:
  - l'utente è proprietario del file
  - l'utente è proprietario della directory
  - l'utente è l'amministratore

Modifica dei permessi

- Un classico esempio di directory che ha questo bit impostato è **/tmp**, i permessi infatti di solito sono i seguenti:

```
# ls -l /tmp

drwxrwxrwt 6 root root 1024 Aug 10 01:03 /tmp
```

quindi con lo *sticky bit* bit impostato.

In questo modo qualunque utente nel sistema può creare dei file in questa directory (che, come suggerisce il nome, è normalmente utilizzata per la creazione di file temporanei), ma solo l'utente che ha creato un certo file potrà cancellarlo o rinominarlo. In questo modo si evita che un utente possa, più o meno consapevolmente, cancellare i file temporanei creati degli altri utenti.

Modifica dei permessi

- Esempi con set user/group ID e sticky bit:

```
$ chmod 777 std
$ ls -l std
-rwxrwxrwx 1 rossi 12 Oct 2 10:52 std
$ su                    # diventa super-user
Password:
# chmod 4777 std
# ls -l std -rwsrwxrwx 1 rossi 12 Oct 2 10:52 std
# chmod 6777 std
# ls -l std -rwsrwsrwx 1 rossi 12 Oct 2 10:52 std
# chmod 7777 std
# ls -l std -rwsrwsrwt 1 rossi 12 Oct 2 10:52 std
```

Modifica dei permessi

- I due comandi *chown* (**CH**ange **OWN**ership) e *chgrp* (**CH**ange **GR**ou**P**) consentono di modificare il proprietario (owner) ed il gruppo di un file:

  $ chown <proprietario> <nomefile>

  Nell'esempio il file <nomefile> diventa dell'utente <proprietario>.

- Solo il proprietario del file o il super-user possono cambiare l'owner di un file. Il comando chgrp è impiegato dal super-user e ha la sintassi:

  $ chgrp <gruppo> <nomefile>

- Il comando chown puo' essere usato per cambiare sia il il proprietario che il gruppo di appartenenza con la sintassi:

  $ chown <proprietario>.<gruppo> <nomefile>