Utenti e File System

C. Baroglio

a.a. 2002-2003

1 Gestione degli utenti

Unix è un sistema multiutente ovvero più utenti possono lavorare su di una stessa macchina, anche contemporaneamente. Tutti gli utenti di una macchina Unix meno uno hanno la stessa dignità. Quest'unico che fa eccezione è l'amministratore di sistema (o superuser), che non ha restrizioni d'accesso.

Cambiando prospettiva, in una macchina Unix il sistema operativo deve gestire processi e file/directory generati da utenti diversi in modo tale che non vi siano interferenze. Ciascun utente deve avere sempre l'impressione di essere l'unico utilizzatore della macchina. In questo contesto, il primo problema che i realizzatori di Unix hanno affrontato è stato definire un modo per identificare gli utenti; associando poi a ciascun processo e a ciascuna componente del file system l'identificatore dell'utente "proprietario" il sistema operativo può mantenere contemporaneamente gli stati dell'elaborazione di tutti gli utenti collegati.

Ogni utente Unix, per accedere al computer su cui intende lavorare, deve identificarsi per mezzo di un "account". L'account è un nome convenzionale (una stringa di caratteri), utilizzabile oltre che per l'oprazione di login anche negli indirizzi di posta elettronica, per conversare con talk o altri programmi per la comunicazione e così via[1]. L'identificazione dell'utente è completata dall'inserimento di una password di almeno sei caratteri, che può contenere oltre agli alfanumerici anche simboli di punteggiatura e caratteri di controllo.

Internamente, il sistema operativo non utilizza gli account per identificare gli utenti bensì un sistema più semplice: dei *numeri*. Ogni utente ha uno User IDentifier (UID), un numero intero associato al suo account. Possiamo quindi dire che ogni utente ha due identificatori, uno interno ed utilizzato dal solo sistema operativo (UID) e l'altro esterno ed utilizzato dagli utenti per identificarsi l'un l'altro (account).

Oltre a gestire singolarmente gli utenti, Unix consente all'amministratore di definire dei *gruppi di lavoro*, che raggruppano dei sottoinsiemi di utenti. Vedremo parlando del file system l'utilità dei gruppi; per il momento basti dire che ogni gruppo ha un nome (identificatore esterno) e un identificatore interno,

[1] L'account del superuser è "root".

numerico detto Group IDentifier o GID. A seconda della versione di Unix, un utente può o meno fare parte di più di un gruppo contemporaneamente.

2 File System: panoramica

Il file system di Unix è gerarchico e quindi strutturato ad *albero*. La radice dell'albero si chiama "root" ed è identificata dal simbolo '/' (slash). I nodi interni dell'albero sono *directory*; i nodi appartenenti alla frontiera corrispondono a directory vuote oppure a file. Questa struttura, consueta a tutti in quanto adottata da tutti i sistemi operativi di maggiore diffusione, è nata con Unix e si è mantenuta pressoché inalterata nel tempo. Il file system contiene file appartenenti a utenti differenti; in particolare esso contiene sia i file e le directory di sistema (il cui proprietario è l'amministratore) sia i file e le directory dei diversi utenti. Ciascun *utente* è *proprietario* di un sottoalbero del file system.

Di norma i sottoalberi dei diversi utenti sono raggruppati nella directory '/home'. Supponiamo, per fissare le idee, che il computer in questione abbia solo tre utenti. Nella directory '/home' saranno contenute tre directory, ciascuna avente nome identico all'account del suo proprietario e radice del sottoalbero da questi gestito. Se per esempio, uno degli utenti ha account "pippo", esisterà una directory identificata dal cammino assoluto '/home /pippo' (il simbolo '/ ' è usato anche come separatore dei nomi di directory nella scrittura di un cammino). Tale directory è detta *home directory* dell'utente pippo e viene mantenuta come valore di una variabile di ambiente che si chiama HOME. Un utente può accedere alla propria home directory utilizzando il simbolo '∼' mentre può identificare la home directory di un altro utente facendo precedere il simbolo '∼' all'account dell'utente. Per esempio, se pippo vuole accedere alla home directory dell'utente pluto può identificarla utilizzando '∼pluto'. L'idea di poter accedere alle porzioni di file system teoricamente riservate ad altri utenti porta in luce l'esigenza di definire dei meccanismi di protezione che consentano ai singoli utenti di guardarsi le spalle da colleghi pasticcioni o malevoli. A tal fine Unix consente agli utenti di modificare i *diritti di accesso* ai propri documenti in un modo che vedremo un po' più avanti.

A livello utente, i file sono contraddistinti da un nome. Il nome di un file è una sequenza di caratteri, alfanumerici e non alfanumerici, fatta eccezione per il simbolo '/', il cui significato è già stato spiegato. L'organizzazione gerarchica del file system consente agli utenti di mantenere file diversi con lo stesso nome, a patto che essi siano contenuti in directory diverse.

La directory root contiene altre cartelle, oltre a home. Fra queste la sottodirectory mnt (mount) viene utilizzata in maniera particolare. '/mnt' è il punto di aggancio/distacco dei supporti di memoria ausiliari (CD, floppy, unità di backup, dischi esterni, compresi quelli di macchine fotografiche e videocamere digitali). L'aggancio di un disco esterno, tecnicamente il suo mount, aggiunge all'albero del file system un sottoalbero. La radice del sottoalbero è una di-

rectory contenuta in '/mnt' e corrispondente all'unità in questione, così come la home directory corrisponde a un utente. Il suo contenuto è il contenuto della memoria esterna agganciata, compresa una sua eventuale organizzazione in sottodirectory.

Altre directory speciali sono '/bin', contenente degli eseguibili, '/lib', contenente librerie di sistema e '/etc' contenente configurazioni di sistema e password.

Oltre a file e directory il file system Unix può contenere anche dei *link*. Un link può essere immaginato come un puntatore di comodo a un file (o a una directory) contenuta in qualche altra parte del file system. I link hanno vita autonoma rispetto agli oggetti che puntano: la cancellazione di un link non comporta la cancellazione del file puntato e viceversa la cancellazione di un file non comporta l'automatica cancellazione dei suoi link.

3 Comandi per la gestione delle directory

Quando un utente esegue l'operazione di login viene creato un ambiente di lavoro dedicato, costituito da varie informazioni. Fra le varie cose l'utente viene *posizionato* in un punto preciso del file system, che all'inizio corrisponde alla sua home directory e in seguito può essere modificato dall'utente medesimo. La direcotry all'interno della quale un utente è posizionato è detta *working directory* e viene mantenuta in una variabile di ambiente chiamata PWD.

I comandi Unix possono avere come argomento file (o directory) posizionati nella working directory oppure in altre parti del file system, individuati grazie alla specifica di un *cammino*. In generale un cammino può essere *assoluto* oppure *relativo*. È assoluto quando inizia da '/' (root). Quindi, per esempio '/home /pippo /documenti' identifica in modo univoco la cartella documenti dell'utente pippo, indipendentemente dal valore di PWD per l'utente in questione.

I cammini sono detti *relativi* quando utilizzano come punto di partenza la working directory anziché la radice. Supponiamo che l'utente pippo sia attualmente posizionato nella sua home directory ('/home /pippo'), in tal caso potrà spostarsi nella sua cartella 'documenti' eseguendo il comando *cd documenti* anziché *cd /home/pippo/documenti* ('cd' sta per "change directory" ed è il comando Unix per spostarsi all'interno del file system). Tutti quei cammini che non iniziano con il simbolo di radice '/' sono intesi essere relativi, ovvero aventi come punto di partenza il valore di PWD. Ogni directory contiene infine due directory speciali, indicate dai simboli '.' e '..', corrispondenti rispettivamente alla directory stessa e al suo genitore nell'albero. '..' consente di costruire cammini che risalgono l'albero. Per esempio, supponendo che PWD abbia come valore '/home /pippo /documenti', il cammino '.. /..' corrisponde alla directory '/home'. Naturalmente è possibile scrivere cammini che in parte risalgono e in parte discendono l'albero, es. '.. /.. /pluto /programmi'.

A meno di altre indicazioni esplicite, tutte i comandi eseguiti dall'utente e finalizzati a creare/cancellare nodi del file system avranno effetto sulla working

directory. L'utente può navigare il file system modificando la propria working directory per mezzo del già citato comando cd:

$$> \ cd\ path$$

L'argomento *path* è opzionale. Quando è presente si tratta di un cammino che identifica un'altra directory; in tal caso, il cammino *assoluto* che identifica tale directory diviene il nuovo valore della viariabile PWD. Quando l'argomento *path* è assente, il comando riposiziona l'utente nella sua home directory.

Creazione e cancellazione di una directory vuota vengono eseguite tramite i comandi:

$$> mkdir\ nome\_directory$$

$$> rmdir\ nome\_directory$$

dove *nome_directory* è il nome della directory da creare/rimuovere. Si osservi che è possibile creare/cancellare directory in posizioni del file system diverse dalla working directory, facendo precedere un cammino al nome della directory in questione: 'path /nome_directory'.

Per visualizzare il contenuto di una directory si utilizza il comando ls. Questo comando ha diversi usi. Se utilizzato senza argomenti 'ls' visualizza il contenuto della working directory, in ordine alfabetico. Attenzione: ls non visualizza tutti quei nodi del file system il cui nome inizia con il simbolo '.' (punto), che risultano quindi *nascosti* ad una visualizzazione standard. Gli argomenti di ls possono essere uno o più file e/o directory. In tal caso il comando elencherà solo i file specificati e/o i contenuti delle directory specificate. 'ls' ha molte opzioni che modificano la presentazione del listato di una directory. Fra queste le più utili sono 'a' (all, visualizza anche i file nascosti), 't' (time, ordina i nodi in base al tempo di ultima modifica visualizzando per primi quelli modificati più di recente), 'l' (long, visualizza informazioni estese comprendenti i diritti d'accesso a file e directory, il proprietario e il gruppo, la dimensione, il tempo di ultima modifica). Le opzioni possono essere combinate; per esempio:

$$> ls\ -lt\ pippo$$

visualizzerà tutte le proprietà dei contenuti della directory 'pippo', ordinando le varie voci secondo il tempo di ultima modifica.

Fino ad ora abbiamo parlato della working directory, come se fosse unica, in realtà ogni utente può creare e utilizzare uno *stack* di working directory, utilizzando i comandi pushd e popd. Per comprendere l'utilità di questi comandi consideriamo la seguente situazione: un utente, posizionato nella directory '/home/pippo/documenti/progetti/2003/TDK31', sta scrivendo un documento per il quale gli occorre consultare spesso dei dati contenuti nella directory '/home/pippo/lavoro/2003/TDK31/dati'. Cambiare directory ogni volta con il

comando cd è piuttosto disagevole, in quanto i cammini da scrivere sono piuttosto lunghi. L'utente può in alternativa memorizzare il cammino corrispondente alla posizione dei dati da consultare in cima allo stack delle working directory utilizzando il comando pushd:

$$> pushd\ ../../../../lavoro/2003/TDK31/dati$$

A questo punto lo stack conterrà il cammino indicato come argomento di pushd in cima, tale valore verrà anche assegnato alla variabile PWD; subito sotto lo stack conterrà il valore precedente di PWD. A questo punto è possibile eseguire degli switch veloci di directory utilizzando ancora il comando 'pushd' ma senza argomenti. L'effetto è di *scambiare* i due elementi in cima allo stack delle working directory senza doverli specificare per esteso, modificando di conseguenza anche il vaolore di PWD. Per togliere dallo stack l'elemento che si trova in cima si utilizza il comando popd (senza argomenti).

Il comando pwd visualizza il valore della working directory o, qualora lo stack delle working directory contenga più di un elemento, lo stack intero.

4 Comandi per la gestione dei file

Le principali operazioni che possono essere compiute sui file sono: *creazione*, *rinomina*, *spostamento*, *modifica del contenuto*, *modifica delle proprietà*, *cancellazione*. Creazione e modifica del contenuto vengono solitamente eseguite utilizzando dei programmi appositi (editor). Le varie versioni di Unix forniscono, oggi come oggi, molti editor diversi, alcuni specialistici (es. "quanta" è dedicato alla realizzazione di pagine web) altri general purpose. Fra questi ultimi i due principali sono "vi" (letto all'inglese, "vi ai") e "emacs" (letto "imacs"). Per quel che riguarda vi, on-line nelle mie pagine del corso sono disponibili sia una breve guida al suo utilizzo sia alcuni link a guide più complete. La difficoltà principale nell'uso di vi è che tutti i comandi sono inviati da tastiera, quindi a seconda della modalità in cui si sta lavorando una lettera può diventare parte del contenuto del file oppure corrispondere a un comando. Superato questo scoglio, vi si rivela essere un editor semplice e molto completo. In questo corso non studieremo emacs, che lascio all'intrapprendenza dei più curiosi.

Un file può essere rinominato utilizzando il comando mv (move):

$$> mv\ file1\ file2$$

rinomina il file *file1*, attribuendogli come nuovo nome *file2* (il simbolo di maggiore indica il prompt della linea di comandi di Unix, non fa parte del comando). Lo stesso comando può essere usato per spostare un file in una directory diversa:

$$> mv\ file\ path$$

In questo caso se path termina con il nome di una directory esistente, il file verrà spostato in essa, se invece termina con il nome di un file esistente oppure con

un nome inesistente, il file verrà contemporaneamente spostato e rinominato. Quindi, per esempio:

$$> mv\ doc1.txt\ documenti/progetti/2003/relazione\_annuale.txt$$

supponendo che la directory 'documenti/progetti/2003' non contenga alcun nodo avente nome 'relazione_annuale.txt' il file 'doc1.txt' verrà spostato in essa e contemporaneamente rinominato. Si osservi che anche il file da spostare (rinominare) non deve necessariamente essere contenuto nella working directory: è possibile fare riferimento a un qualsiasi file del file system utilizzando alternativamente il suo cammino assoluto oppure il cammino relativo che conduce dalla working directory al file in questione.

Per cancellare uno o più file si utilizza il comando rm:

$$> rm\ file1\ file2\ \ldots$$

Come nel caso precedente e come valido in generale, non è necessario che i file in questione siano contenuti nella working directory. Il comando rm ha diverse opzioni, fra queste particolarmente utile e, a causa della sua distruttività, da usare "cum grano salis" l'opzione R (o anche r):

$$> rm\ -r\ nome\_directory$$

elimina il sottoalbero avente come radice la directory il cui nome è specificato come argomento. Un'altra opzione utile è 'i', che indica la modalità interattiva: quando si usa questa opzione il sistema chiede conferma per ogni nodo da cancellare. Si osservi che una directory non viene rimossa se non è vuota: utilizzando congiuntamente le opzioni 'i' ed 'r' è possibile scegliere quale porzione di un sottoalbero si desidera salvare.

Altri comandi utili sono file e touch. Il primo consente di eseguire dei test su di un file passato come argomento. In particolare restituisce il tipo del file in questione (per esempio se si tratta di un eseguibile e di che tipo piuttosto che un file sorgente scritto in un certo linguaggio). I tipi di file riconosciuti sono elencati nel file '/usr/share/magic' (per "magic number" si intende un numero che identifica un tipo di file). Il secondo modifica alcune proprietà del file passato come argomento: il tempo di ultima modifica e il tempo di ultimo accesso, ai quali viene assegnato il tempo corrente. Questo comando è spesso utile quando si utilizza make, che studieremo più avanti nel corso.

5 Link

In Unix esistono due tipi di link: *hard* e *symbolic* link. Un hard link è un secondo nome per un file; i due nomi condividono lo stesso inode, quindi cancellando il link si cancella il file e vice versa. Solo l'amministratore può creare hard link. I normali utenti possono creare link simbolici: un link simbolico è un file speciale

che in un certo senso *punta* al contenuto di un altro file; a differenza dal caso precedente tuttavia l'inode non è condiviso quindi la cancellazione del link non comporta la cancellazione del file e vice versa. Quando si apre con un editor un file che è un link simbolico si modifica il contenuto del file puntato.

Per creare un link simbolico si utilizza il comando ln con l'opzione 's':

$$> ln \ -s \ file\_target \ nome\_link$$

Il nome del link (ultimo argomento) è opzionale. Qualora non sia indicato viene creato nella working directory un link avente nome identico a quello del file puntato (target). Per rimuovere i link si usa il già visto comando rm.

6 Mount

Il file system di una macchina con sistema operativo Unix è un unico albero i cui elementi possono essere sparsi su più dispositivi di memoria. Il meccanismo di mount consente di "agganciare" il file system presente su vari dispositivi di memoria (Hard Disk, CD, disco esterno, file system di rete o altro) in un unico albero, permettendone l'accesso omogeneo attraverso i comandi per il percorrimento (manipolazione) delle directory e di accesso (manipolazione) ai file che abbiamo già visto. All'avvio (bootstrap) della macchina il kernel esegue una serie di comandi di mount secondo le disposizioni contenute nel file di configurazione '/etc/fstab'. L'amministratore di sistema può modificare il contenuto di tale file adattandolo alla configurazione del computer in questione.

Il mount da prompt di un sottoalbero viene eseguito tramite il comando mount, un comando con molte opzioni, che di base prevede chiamate del tipo:

$$> mount \ -t \ tipofs \ dispositivo \ directory$$

Questo comando aggancia il file system contenuto in *dispositivo* (che è di tipo *tipofs*) all'albero del file system nella posizione specificata da *directory*. Directory di solito è una directory vuota contenuta in '/mnt'. Se non fosse vuota, dopo l'operazione di mount i contenuti precedenti non sarebbero più accessibili. Il tipo di file system assume valori in un insieme predefinito di file system riconosciuti, fra i quali, per esempio, ext3, hpfs, ufs, xenix, ecc. Di norma l'operazione di mount può essere effettuata solo dall'amministratore.

Per visualizzare l'insieme dei dispositivi agganciati si utilizza il comando mount privo di argomenti. Infine, l'operazione inversa alla mount (che sgancia un dispositivo dal file system) è umount.

7 Accesso a floppy disk

L'accesso a floppy disk può avvenire in due modi alternativi. Se l'unità floppy è agganciata al file system attraverso il meccanismo di mount, visto sopra, è

possibile accedere a un dischetto eseguendo semplicemente il comando:

$$> cd\ /mnt/floppy$$

e quindi lavorare sui file e sulle directory in esso contenute utilizzando i comandi visti nelle precedenti sezioni. Però non su tutte le macchine questa opzione è attivata. In questo caso è possibile utilizzare le utility contenute nel pacchetto mtools. Mtools è una collezione di strumenti che consentono ai sistemi Unix di manipolare file system MSDOS, tipicamente contenuti su floppy disk e che non richiedono che sia stata eseguita in precedenza un'operazione di mount. Per visualizzare il contenuto di un floppy si utilizza:

$$> mdir\ a:$$

dove 'a:' indica il device 'floppy disk'. Per copiare si utilizza mcopy; per esempio, per copiare il file 'doc.txt' su dischetto si utilizza:

$$> mcopy\ doc.txt\ a:doc.txt$$

mentre per cancellare dei file contenuti su floppy si utilizza il comando mdel.