PRIMI PASSI IN LINUX - Consigli, trucchi, suggerimenti per un utente alle prime armi. Appunti per il corso di base di Linux tenuto dal Prato Linux User Group

Lorenzo Masetti

24 febbraio 2003

"Il peggior nemico del software libero è l'ignoranza."
Gianni Ciolli, presidente del Prato Linux User Group

Indice

1 Usare la SHELL

1.1 La situazione attuale

Il mondo dei computer domestici è oggi dominato dalla grafica... tutto si può fare con un click, tantissimi programmi vengono pubblicizzati con slogan del genere “Facile come fare click”. La tastiera sembra che sia diventata uno strumento quasi inutile, usata giusto per chattare. Eppure **non è possibile che i due o tre tasti di un mouse siano potenti come gli oltre 101 tasti di una tastiera :-))**.

1.2 Buoni motivi per usare la shell invece di un’interfaccia grafica

- La shell è più veloce! Anche se il vostro computer ha un processore da un GHz, usare la shell è più veloce che usare un file manager a finestre. Questo non vuol dire che non dovete usare le ottime interfacce grafiche che esistono sotto Linux. Semplicemente è importante capire quando è più opportuno usare una alternativa oppure un’altra. Se poi il computer che avete a disposizione non è il massimo, la velocità diventa ancora più importante.

- La flessibilità (secondo Sergio Ballestrero): Fra i programmi GUI - Graphical User Interface e i programmi da linea di comando, c’è la stessa differenza che c’è fra un trapano multifunzione da hobbista, e l’attrezzatura di frese, troncatrici, pialle elettriche di un professionista. Sì, probabilmente riuscite a rabberciare qualcosa anche con il primo, ma se volete fare sul serio, velocemente e bene, servono le cose serie. Avete tanti strumenti e li potete combinare come volete, e fare quello che vi serve - non quello che un programmatore ha un giorno pensato che vi sarebbe servito.

- Operazioni complesse possono essere realizzate efficientemente. Esempio: una directory contiene immagini e documenti html. Le immagini hanno estensione *gif* o *jpg*. Vogliamo spostare tutte le immagini in una nuova sottodirectory “immagini”. Per questa semplice operazione su Windows dobbiamo usare il menù per creare una nuova cartella e manualmente, tenendo premuto control selezionare tutte le immagini poi cliccare sull’iconcina di taglia, aprire la nuova cartella e cliccare su incollla... Non è più facile scrivere

```
mkdir immagini
mv *.gif *.jpg immagini
```

- La shell vi permette di usare un computer remoto cioè che si trova anche dall'altra parte del mondo nello stesso modo in cui usate il computer che avete davanti agli occhi. Sui server internet non è installata nessuna interfaccia grafica, ma si può connettersi dall'Italia a un server giapponese (se si ha la password) e lavorare in maniera trasparente.
- Sicurezza. La shell è (quasi) sempre disponibile. Scheda grafica preistorica e sconosciuta - niente driver video. E adesso? Al limite potete fare ripartire il computer con un dischetto d'avvio e la vostra shell sarà sempre al vostro servizio... è bene quindi sapere cosa fare in questi casi!
- Usare la shell vi aiuta a capire meglio come funziona il sistema operativo, vi insegna anche a stare attenti a quello che fate visto che con un solo comando potreste cancellare tutto il filesystem (bello no?) :-))

1.3 Cos'è una shell e come si presenta

Una shell è quindi il modo più semplice di comunicare con il kernel del sistema operativo. Una shell permette di

- Usare i comandi interni della shell
- Lanciare qualsiasi tipo di programmi

Ad esempio *ls* è un comando interno della shell che scrive il contenuto della directory corrente, emacs è un programma esterno però si invocano entrambi scrivendo il nome del comando o del programma.

La shell si presenta come un **prompt** che invita l'utente a scrivere un comando, ad esempio

```
guest@curzio:/home/guest/$
```

Nel prompt vengono scritte delle informazioni utili. Tipicamente vengono scritti il nome dell'utente, la directory corrente e a volte il nome del computer su cui si opera. Nell'esempio si vede che l'utente guest è sul computer curzio nella directory `/home/guest`.

Ogni utente ha una sua home directory. Appena fatto il login per ogni utente viene aperta una shell e la directory corrente è la sua home.

1.4 Filosofia di base di UNIX

Opposta a Windows.

- Ogni programma fa una cosa sola e la fa bene
- L'output di un programma può diventare l'input di un altro
- I nomi sono succinti: meglio cp di copy per ridurre il tempo di battitura
- I comandi hanno una forma tipica:

```
<nome comando> <opzioni (precedute da -)> <parametri>
```

1.5 Chi va là? Il login

Per iniziare a interagire con il sistema dovete effettuare il login cioè dovete dire chi siete e provare che siete proprio voi. Esistono molte interfaccie grafiche per effettuare il login ed entrare subito in un ambiente grafico, qui vedremo il login in modalità testuale che è più standard. Il login grafico è del tutto analogo.

```
Red Hat Linux release 7.1 (Seawolf)
Kernel 2.4.2-2 on an i686

curzio login: guest
Password: guest
Last login: Sat Nov 21 10:12:48 on tty2
Il System Manager ti saluta
You have mail.
guest@curzio ~$
```

Questo significa che vi identificate e autenticate con il sistema con una conversazione del genere:
Io sono un sistema Linux Red Hat versione 7.1 soprannominato Seawolf, lupo di mare. La mia versione di kernel è 2.4.2-2 e giro su un computer che ha un processore della famiglia intel versione 686. Il mio nome è curzio.
chi va là ? sono quello che tu chiami guest
parola d'ordine ? guest

Naturalmente quando scrivete la password non appare niente sul video, per proteggervi da sguardi indiscreti, neanche degli asterischi (che rivelerebbero la lunghezza della password)

il sistema (in realtà un ben preciso programma che, guarda il caso, si chiama login) controlla che esista un utente con il nome dato (username, non nome effettivo), e che la password corrisponda a quella registrata nel file /etc/passwd

Quando il login ha successo viene visualizzata di solito la data dell'ultimo login (così si può controllare se qualcuno è entrato con il nostrto account), un messaggio che l'amministratore di sistema ha preparato per voi (naturalmente può variare) e eventualmente un messaggio che vi informa che avete della posta nella vostra casella di posta locale.

A questo punto, avete una identità: lo username, a cui corrisponde uno userid (o uid) numerico; e appartenete ad alcuni gruppi, a cui corrispondono dei groupid (o gid) numerici.

Nota: Questa identità spesso viene chiamata account : conto nel senso di conto bancario o simili - il nome deriva dai tempi in cui i grandi computer venivano utilizzati in batch o in time sharing, e agli utilizzatori veniva fatto pagare in proporzione a quanto tempo di CPU veniva utilizzato dall'utente - quindi a ogni utente corrispondeva appunto un conto a cui addebitare l'utilizzo.

Potete controllare la vostra identità con il comando `whoami` (chi sono io?). Naturalmente è possibile cambiare utente senza effettuare il login da capo ma ne parleremo nel paragrafo 2.2.

Una volta terminato il lavoro ricordatevi di uscire dalla shell con il comando *logout* o semplicemente premendo *CTRL-D*.

1.6 Comandi fondamentali

1.6.1 Listare le directory con ls

Il comando **ls** dà in output la lista dei file presenti nella directory corrente.

```
guest@curzio ~$ ls
autosave  Desktop  GNUstep
guest@curzio ~$
```

Il comando **ls** può prendere come parametro una directory e allora scriverà la lista di quella directory. Oppure può prendere come directory un file semplice (o più file) e allora listerà solo quel (quei) file. Questo può avere senso con l'opzione -l oppure con il meccanismo di espansione dei comandi (cfr. paragrafo 1.10)

```
guest@curzio ~$ ls /
bin   dev  home  lost+found  mnt  proc  sbin     shlib10  usr
boot  etc  lib   misc        opt  root  shbin10  tmp      var
guest@curzio ~$
```

Alcune opzioni da provare

- -l (long) scrive molte più informazioni
- -F distingue le directory
- -a (all) mostra anche i file nascosti, cioè quelli che cominciano con il punto (.)

Ricordati:

- Le opzioni possono essere scritte anche attaccate. Esempio: *ls -la*.
- Su molti sistemi è definito un *alias* (cioè un nome alternativo) per ls -l, *ll*.
- Linux, a differenza di Windows e del DOS, è case-sensitive cioè distingue le lettere maiuscole dalle minuscole, quindi Curzio.txt e curzio.txt non sono lo stesso file!
- Su Linux i nomi dei file non possono contenere spazi, infatti lo spazio è il carattere che divide una parola dall'altra nel linguaggio della shell. In realtà gli spazi vengono tollerati dal filesystem ma è buona norma non usarli perché complicano l'esecuzione dei comandi. Gli spazi possono tranquillamente essere sostituiti dal trattino basso (_)
- Su Linux l'estensione non è essenziale come su Windows. In particolare i programmi eseguibili non hanno nessuna estensione (non esistono .exe). Tuttavia è una buona norma dare ai file un'estensione appropriata per sapere subito cosa stiamo maneggiando. In ogni caso possiamo avere informazioni sul tipo di file con il comando **file**.
- Gli eseguibili non si lanciano cliccandoci sopra da un file manager ma scrivendo il loro nome in una shell. Perchè il programma sia trovato deve stare in una directory che fa parte del path. I vari menù delle interfacce grafiche non sono altro che scorciatoie per lanciare dei comandi molto frequenti.

1.6.2 Cambiare la directory corrente con cd

Il comando *cd* (*change directory*) permette di cambiare la directory corrente. Prende come parametro il percorso della directory dove spostarsi. Il percorso può essere assoluto o relativo. Infatti il filesystem di unix ha una radice che si chiama `/`, tutte le directory discendono da `/`, ad esempio `/home/guest`. Se il percorso non è assoluto si intende a partire dalla directory attuale. Per risalire nell'albero delle directory si usa `..` (mentre il punto indica la directory corrente); la tilde ( ) indica la directory home dell'utente. Scrivendo solo `cd` si ritorna alla propria home. La directory corrente può essere visualizzata con il comando **pwd** (present working directory).

Ricordatevi la differenza tra percorsi (path) assoluti e percorsi relativi: i percorsi assoluti iniziano con `/` e identificano il percorso di un file muovendosi tra le varie directory. Le directory sono divise dallo slash `/` e non dal backslash (\) come in DOS. Invece i percorsi relativi identificano il percorso da seguire a partire dalla directory attuale.

Esempi

```
guest@curzio ~$ cd /
guest@curzio /$ ls
bin   dev  home  lost+found  mnt  proc  sbin     shlib10  usr
boot  etc  lib   misc        opt  root  shbin10  tmp      var
guest@curzio /$ cd /usr/
guest@curzio /usr$ cd local/bin/
guest@curzio /usr/local/bin$ cd
guest@curzio ~$ pwd
/home/guest
guest@curzio ~$
```

Esempi di percorsi relativi e corrispondente path assoluto

pwd	**relativo**	**assoluto**
/home/guest	prova/	/home/guest/prova
/home/guest	../carlo	/home/carlo
/	usr/bin	/usr/bin
/usr/local	bin/	/usr/local/bin/
/usr/local/bin	../../lib	/usr/lib

1.6.3 Autocompletamento e storico dei comandi

Una delle caratteristiche più belle della shell è l'**autocompletamento**. Quando dovete scrivere un comando o un nome di un file molto lunghi, potete scrivere le prime lettere e premere TAB. Se c'è solo un comando (o un file, dipende dal contesto) che inizia con quelle lettere, automaticamente il nome viene completato. Se invece ci sono più possibilità e quindi il completamento non è unico, vengono mostrate tutti e si deve specificare una (o più di una) ulteriore lettera per togliere l'ambiguità.

Lo storico dei comandi: usando le frecce in su e in giu si possono scorrere i comandi precedenti.

Esempio:

```
guest@curzio ~$ cd /u                 (premere tab)
```

```
guest@curzio ~$ cd /usr/              (premere l e poi tab)
guest@curzio ~$ cd /usr/l
lib      libexec  local
guest@curzio ~$ cd /usr/l             (premere o e poi tab)
guest@curzio ~$ cd /usr/local/
guest@curzio /usr/local$
```

Nota: Non potete pretendere che la bash vi legga nel pensiero. Se state creando un nuovo file di nome *precipitevolissimevolmente* non potete sperare di scrivere *prec* e poi premere Tab :-)

1.6.4 Creare le directory con mkdir

```
mkdir nomedirectory1 [ nomedirectory2 ... nomedirectoryN ]
```

1.6.5 Copiare i file con cp

Due modelli da usare:

```
cp file_sorgente file_destinazione
cp file_sorgente1 ... file_sorgenteN directory_destinazione
```

Esempio:

```
guest@curzio ~$ mkdir prova
guest@curzio ~$ cd prova
guest@curzio ~/prova$ cp /etc/passwd .
guest@curzio ~/prova$ ls
passwd
guest@curzio ~/prova$ cp passwd curzio
guest@curzio ~/prova$ ls
curzio  passwd
guest@curzio ~/prova$ mkdir malaparte
guest@curzio ~/prova$ cp curzio passwd malaparte/
guest@curzio ~/prova$ ls malaparte/
curzio  passwd
guest@curzio ~/prova$
```

Cosa succede se dò più di due parametri ma l'ultimo non è una directory?

```
guest@curzio ~/prova$ cp curzio passwd pippo
cp: copying multiple files, but last argument ‘pippo’ is not a directory
Usare ‘cp --help’ per ulteriori informazioni.
```

Opzioni:

-i Interattivo, chiede conferma prima di sovrascrivere un file già esistente

-R Ricorsivo: per copiare le directory con tutto il loro contenuto. Esempio

```
guest@curzio ~/prova$ ls -F
curzio  malaparte/  passwd
guest@curzio ~/prova$ cp -R malaparte/ benigni
```

```
guest@curzio ~/prova$ ls -F
benigni/  curzio  malaparte/  passwd
guest@curzio ~/prova$ ls benigni/
curzio  passwd
```

1.6.6 Eliminare i file con rm

Il comando **rm** (da *remove*) elimina i file passati come parametro

L'utilizzo di rm è pericoloso perchè i file cancellati non possono essere recuperati in alcun modo. Bisogna quindi stare attenti prima di premere invio. Questo comando non è amichevole, non solo non chiede conferma prima di cancellare ma se gli passiamo una lista di file e alcuni nomi sono sbagliati, cancellerà lo stesso i file di cui abbiamo scritto bene il nome:

```
guest@curzio ~/prova$ ls
curzio  roberto  yuri
guest@curzio ~/prova$ rm roberto yuri curzzio
rm: cannot remove 'curzzio': File o directory inesistente
guest@curzio ~/prova$ ls
curzio
guest@curzio ~/prova$
```

Non si può cancellare una directory in modo normale: se la directory è vuota si usa il comando **rmdir**. Se la directory non è vuota si usa **rm -r** che cancella tutto il suo contenuto. Vediamo un esempio (il comando touch serve per creare un file vuoto)

```
guest@curzio ~/prova$ mkdir chechi
guest@curzio ~/prova$ rm chechi
rm: 'chechi' is a directory
guest@curzio ~/prova$ rmdir chechi
guest@curzio ~/prova$ mkdir rossi
guest@curzio ~/prova$ touch rossi/paolo
guest@curzio ~/prova$ rmdir rossi
rmdir: 'rossi': La directory non e' vuota
guest@curzio ~/prova$ rm -r rossi
guest@curzio ~/prova$
```

Opzioni utili per rm

-i interattivo, chiede conferma prima di eliminare

-f (force) cancella i file anche se sono protetti dalla scrittura e non dà messaggi d'errore se i file non esistono

1.6.7 Spostare e rinominare i file con mv

Il comando **mv** (*move*) serve per spostare i file. Come il comando cp anche mv ha due modalità d'uso. La prima serve per rinominare i file, la seconda per spostare i file da una directory a un'altra.

```
mv vecchio_nome nuovo_nome
mv file1 file2 ... fileN directory
```

- Il comando `mv` non controlla se il file destinazione esiste già. Se esiste già lo sovrascrive senza dire nulla!
- Anche per mv esistono l'opzione **-i**. In quel caso mv chiede conferma prima di sovrascrivere un file

1.6.8 Per sapere tutto su un comando: man

Il comando **man** (per *manual*) serve per consultare l'help in linea di tutti i comandi e anche di molti programmi.

Quando un principiante di Linux (o più in generale di Unix) chiede a un utente più esperto delucidazioni su come funziona un qualche comando, la risposta scortese ma sicuramente azzeccata è spesso RTFM - "Read The Fucking Manual!". Tutti i comandi sono ampiamente documentati, non si vede perché bisognerebbe perdere tempo a rispiegare cose che qualcuno (spesso l'autore del programma) ha già spiegato esaurientemente....

Quindi provate sempre a scrivere

```
man <nomecomando>
```

e avrete pagine e pagine di documentazione... per consultarle scorrete le pagine con lo spazio e uscite premendo q.

Ci sono più sezioni del manuale, ad esempio quella con i riferimenti per i comandi C. Per specificare esplicitamente la sezione scrivete il numero della sezione prima del comando.

Come si leggono i manuali che visualizziamo con man? Prima regola: i manuali NON SI LEGGONO, si consultano. I manuali sono molto spesso estremamente dettagliati, la stragrande maggior parte delle cose che ci sono scritte non vi serviranno mai o forse vi serviranno tra due mesi, non ora... ricordatevi che, per quanto vuota possa essere la vostra vita, avrete sempre di meglio da fare che mettervi a leggere il manuale di ls. Usate i manuali quando non vi ricordate il nome di un'opzione o a cosa serve, scorreteli velocemente alla ricerca della parte che vi interessa. Ricordatevi che potete fare delle ricerche di testo all'interno del manuale scrivendo lo slash (/) e poi il testo da cercare.

Suggerimento: Se scrivendo `man <comando>` il sistema risponde "No manual entry for..." provate con `<comando> --help` o `<comando> -h`. Spesso funziona.

1.7 Visualizzare, scrivere manipolare file di testo

Divagazione....

I file di testo sono importantissimi. I file di testo in formato ASCII sono un formato standard che dovrebbe essere utilizzato più spesso e meglio. La Microsoft cerca di imporci un modo di pensare secondo cui anche la lista della spesa va scritta in Word con fonts eccezionali e con dieci colori diversi, dove le email sono in html e contengono sfondi, testi colorati o lampeggianti e altre bestialità di questo tipo.

Questo modo di pensare ha un costo. Per esempio prendiamo un messaggio che potreste scrivere a un amico

```
Ciao oggi sono stato al corso di linux del PLUG.
Era molto interessante!
```

Tutta l'informazione che volevate comunicare, memorizzata in un file di testo, occupa esattamente 73 byte (uno per carattere). Questo è il minimo di memoria con cui potete esprimere questo (banale) pensiero. Se mandate l'email con outlook tutto questo messaggio viene inserito in una pagina html automaticamente generata che occupera tra i 300 e i 600 byte. Per non parlare di un documento word con questo stesso messaggio... Moltiplicate per il numero di email e di documenti che circolano su internet e scoprirete quanti terabyte di informazioni assolutamente inutili viaggiano sulla rete.

Nota: Un romanzo di medie proporzioni, "La coscienza di Zeno", occupa in formato testo 834 Kbytes. Poi scrivete una lettera con word e vi occupa 100 Kb (ma voi non avete assolutamente scritto un ottavo di romanzo, forse un trecentesimo!). La domanda che sorge spontanea è: ma cosa ci scrive Microsoft in un file word? Questo è un buon motivo per preferire i programmi ed i formati liberi.

In ambiente Linux i file di testo sono importantissimi, perchè tutti i file di configurazione sono semplici file di testo. Sapere fare bene alcune cose con i file di testo semplifica molto la vita.

Una velocissima panoramica per vedere cosa si può fare con un file di testo

cat Questo comando butta sullo schermo (per essere precisi sullo standard output, vedi dopo) il contenuto di un file. In realtà funziona anche per i file binari ma in quel caso non ha molta utilità. Per i file di testo ha la scomoda caratteristica di non fermarsi alla fine di una schermata. In realtà cat è un programma che con*cat*ena più file. Se passate a cat più di un argomento, scriverà i vari file uno dopo l'altro.

echo scrive quello che volete. Esempio

```
guest@curzio ~/prova$ echo W IL PLUG
W IL PLUG
```

more e less Due comandi che permettono di leggere un file scorrendolo se è più lungo di una schermata. Meglio less perché permette di tornare indietro! Esempio (solo per fare la squallida battuta):

```
guest@curzio ~$ cd pub/
guest@curzio ~/pub$ more beer
I think I'll have another beer
```

1.8 Standard input, standard output e ridirezione

I comandi e i programmi che non hanno un'interfaccia grafica comunicano attraverso due file: l'input dal quale prendono le informazioni necessarie, e l'output sul quale scrivono i risultati della loro esecuzione.

Normalmente per i comandi lanciati dalla shell l'input e l'output sono associati con due file speciali: lo **standard input**, che coincide con la tastiera e lo **standard output**, che coincide con lo schermo. Questo comportamento può essere cambiato facilmente.

La ridirezione serve per cambiare l'input o l'output di un comando. Il < ridirige l'input, il > ridirige l'output. Questi caratteri vanno visti come se fossero frecce in entrata e in uscita.

Per esempio abbiamo visto che il comando `echo` scrive una stringa sullo schermo (standard output). Lo stesso comando può essere usato per scrivere su un file.

```
guest@curzio ~/prova$ echo Il Signore degli Anelli > yurichechi.txt
guest@curzio ~/prova$ cat yurichechi.txt
Il Signore degli Anelli
guest@curzio ~/prova$
```

Ad esempio il comando sort ordina quello che gli viene passato nell'input. Nell'esempio vediamo che una volta abbiamo ridiretto solo l'input e il risultato viene scritto sullo standard output, la seconda volta abbiamo ridiretto anche l'output su un nuovo file.

```
guest@curzio ~/prova$ cat elenco
yuri
roberto
curzio
paolo
guest@curzio ~/prova$ sort < elenco
curzio
paolo
roberto
yuri
guest@curzio ~/prova$ sort < elenco > elenco_ordinato
guest@curzio ~/prova$ cat elenco_ordinato
curzio
paolo
roberto
yuri
```

Oltre allo standard input e allo standard output esiste anche lo **standard error** che è associato allo schermo come lo standard output. Quando si ridirige lo standard output però i messaggi d'errore non sono ridiretti e compaiono ancora sullo schermo. Lo standard error può essere ridiretto con $2 >$.

Quando si ridirige l'output su un file tutto il contenuto del file viene perso, se invece si vuole aggiungere l'output in fondo al file si può usare $>>$. Invece $\& >$ ridirige sia l'output che l'errore sullo stesso file. Naturalmente è possibile scrivere anche $2 >>$ e $\& >>$ con l'effetto che vi potete immaginare (provare per credere).

Suggerimento: La ridirezione apre delle strade nuove, ad esempio, poichè tutto in Unix è un file, anche le periferiche (ad esempio il cdrom è associato al file speciale `/dev/cdrom`), per creare l'immagine di un cd in un file basta digitare

```
cat /dev/cdrom > fileimmagine.img
```

1.9 La pipeline

Una bellissima funzione della Bash è la **pipeline**. La pipeline è un modo molto semplice di far funzionare insieme comandi diversi. In una pipeline infatti **l'output del primo comando viene inviato all'input del secondo comando**.

Tutto diviene più chiaro facendo un esempio. Supponete di avere una directory che contiene un gran numero di file. Per vederli tutti senza che scappino dallo schermo e per poterli scorrere facilmente visualizzando tutte le informazioni potete usare il comando

```
ls -l | less
```

In questo esempio invece prendiamo un elenco di nomi, lo ordiniamo con sort estraiamo le prime quattro righe con il comando *head -4* e da questo risultato estraiamo le ultime due righe con il comando *tail -2*. Quindi le pipeline possono essere concatenate!

```
guest@curzio ~/prova$ cat elenco
yuri
roberto
curzio
paolo
gaetano
francesco
gianni
guest@curzio ~/prova$ sort < elenco | head -4 | tail -2
gaetano
gianni
guest@curzio ~/prova$
```

1.10 Espansione dei nomi

Questo sarebbe un argomento molto vasto, ma vediamo solo una funzione molto utile. A volte capita di dover eseguire lo stesso comando su molti file che hanno alcune caratteristiche comuni nel nome (tipicamente la stessa estensione). Un modo facile per ottenere questo risultato è usare l'asterisco. L'asterisco si espande con 0 o più caratteri per comporre il nome dei file. Ad esempio

```
guest@curzio ~/prova$ ls
curzio.txt  paolorossi  yuri.txt
guest@curzio ~/prova$ ls *.txt
curzio.txt  yuri.txt
```

L'ultimo comando si è espanso il *ls curzio.txt yuri.txt*.

2 Utenti, gruppi e permessi nel filesystem di unix

2.1 User e Superuser - una questione psicanalitica

Se volete sempre usare il superuser, avete un problema - e ne avrete ancora di più in seguito. Abbiamo visto che al login ci viene assegnata una identità. Abituati ai PC Windows, dove ai diversi login corrispondono, tutt'al più diverse configurazioni[1], ci troviamo invece di fronte a un grosso cambiamento: diversi account hanno diversi diritti - per quanto riguarda l'accesso ai file, alle periferiche.

Esistono, su tutti i sistemi Unix, delle identità che non appartengono a persone reali, ma a certi "ruoli". Ad esempio i programmi che gestiscono la posta sono eseguiti con una identità speciale, mail, che gli dà il diritto di scrivere nelle caselle di posta degli utenti.

Esiste un account che ha tutti i diritti su tutto, senza alcuna limitazione, ed è l'account del system manager o superuser: root.

Root ha diritto di accesso, in lettura e scrittura, a qualsiasi cosa sul sistema: dalle configurazioni, alle periferiche, la posta di tutti gli utenti, persino ai programmi che sono in esecuzione. Il system manager ha poteri illimitati, e gli utenti devono avere illimitata fiducia nella sua affidabilità e discrezione.

Come utenti Linux, probabilmente sarete voi stessi i system manager del vostro PC. Non dimenticatevi mai che il superuser ha possibilità illimitate - e quindi capacità illimitata di causare danni. La prima regola, quindi, è: non fidatevi di voi stessi. Quando avete l'identità di root, pensate sempre due volte al comando che state scrivendo, o preparatevi a sopportarne le conseguenze.

```
# rm -rf /*
```

Questo comando apparantemente innocuo è sufficiente (premendo solo dieci tasti compreso l'invio) a cancellare qualsiasi file sulle vostre partizioni... e altri possono facilmente fare polpette di tutto l'hard disk.

Se avete appena installato il Linux, come prima contromisura, createvi un utente normale col vostro nome:

```
# adduser mario
```

e assegnategli una password:

```
# passwd mario
```

e usate sempre questo user. Usate root solo quando non ne potete fare a meno: per configurare il sistema, per installare del nuovo software. La tentazione di usare sempre root è forte - soprattutto se in precendenza avete avuto la sfortuna di scontrarvi con un system manager che faceva il bello e il cattivo tempo; non fosse altro per la sensazione di potenza, o per contrastare la sensazione di impotenza ("ma come, non posso neanche scrivere un file in quella directory? ma è il MIO computer!" == paura della castrazione), ma è bene resistere più che si può, anche per evitare le fobie a livello di timor panico che si sviluppano dopo aver distrutto il sistema per la terza volta consecutiva...

2.2 Cambiare utente

Abbiamo appena detto che i vari utenti non corrispondono necessariamente a persone fisiche diverse, piuttosto rappresentano i diversi ruoli che si possono avere nella gestione di un computer. In particolare abbiamo visto che nel vostro computer domestico dovete fare la parte dell'amminsitratore di sistema, e vi succederà spesso di dover diventare l'amministratore per installare un nuovo

[1] questo era vero per Windows 95 e Windows 98. In Windows XP la Microsoft ha introdotto una gestione degli utenti e dei permessi simile a Unix presentando come un'assoluta novità quello che su Unix funziona già da trent'anni

programma. Vediamo quindi come fare a cambiare utente senza fare un nuovo login.

Il comando da usare è **su** che significa *set user*. La sua sintassi è quella che vi aspettate:

```
su <username>
```

Siccome è estremamente comune usare il comando *su* per diventare l'amministratore di sistema, si può evitare di scrivere *su root* e scrivere solo *su*.

Quando eseguite il comando *su* vi viene chiesta la password dell'utente in cui volete trasformarvi. Se però già siete root potete trasformarvi in qualsiasi utente senza inserire nessuna password (secondo la filosofia secondo cui il root può fare tutto!).

Cambiando utente così spesso si può andare incontro a crisi di identità, in quel caso è utile il comando *whoami* (chi sono io?!?). Esempio:

```
guest@curzio ~$ whoami
guest
```

2.3 Come vedere i permessi

Usate il comando `ls -l` (spesso è definito anche l'alias `ll`).

```
guest@curzio ~/prova$ ls -l
totale 20
-rw-------    1 guest    users          18 giu  5 23:46 curzio
-rw-r--r--    1 guest    users          31 giu  5 23:46 paolorossi
drwxrwxrwx    2 guest    users        4096 giu  5 23:48 pratesi
-rwxr-xr--    1 guest    users          20 giu  5 23:46 roberto
-rw-rw-rw-    1 guest    users          24 giu  5 23:46 yuri
guest@curzio ~/prova$
```

In questo modo avete molte informazioni sui file che listate. Sulla prima colonna vengono scritti i permessi associati a ciascun file.

Ogni file in un filesystem unix appartiene ad un utente e ad un gruppo. L'utente e il gruppo sono scritti nella terza e nella quarta colonna dell'output di ls -l. Ogni file può avere diversi permessi per tre categorie di utenti: l'utente a cui appartiene il file, gli utenti che appartengono al gruppo a cui appartiene il file, e tutti gli altri. Questo modo di gestire i permessi è estremamente flessibile e permette di introdurre categorie di utenti con permessi diversi.

Nella prima colonna dell'output di `ls -l` si possono leggere i permessi: il primo carattere è - per i file ordinari, d per le directory. Seguono tre triplette di caratteri che indicano i permessi dell'utente (user), del gruppo (group) e degli altri (others). I permessi possono essere di lettura (r), di scrittura (w) e di esecuzione (x), in questo ordine (se non si ha un permesso nella posizione corrispondente, al posto della lettera viene scritta una lineetta, `-`). Per le directory il permesso di esecuzione corrisponde al permesso di listarne il contenuto.

Nell'esempio:

- Tutti i file appartengono all'utente guest e al gruppo users. Molto spesso non si ha questa situazione, e l'utente e il gruppo hanno lo stesso nome. In quel caso vuol dire che il gruppo è formato solo da quell'utente.

- Il primo file (curzio) può essere letto e scritto dall'utente guest. Tutti gli altri utenti non hanno nessun permesso.
- Il secondo file (paolorossi) può essere scritto e letto dall'utente guest e solo letto da tutti gli altri, anche gli appartenenti al gruppo users.
- Il terzo file (pratesi) è una directory, come si vede dal fatto che la prima lettera è una d, e tutti possono listarla, leggerla e scriverla.
- Il quarto file roberto ha questi permessi: l'utente può leggerlo, scriverlo ed eseguirlo, il gruppo può leggerlo ed eseguirlo, gli altri possono solo leggerlo.
- Il quinto file (yuri) può essere letto e scritto da tutti.

2.4 Cambiare i permessi con chmod

Il comando chmod serve a cambiare i permessi di uno o più file.

Prima sintassi di chmod

Il primo modo di usarlo ha questa sintassi

```
chmod <triads> filename1 [filename2 ... filenameN]
```

Al posto di triads si può scrivere una cosa di questa forma: *chi azione permessi.*

chi	**azione**	**permessi**
u = user (utente) g = group (gruppo) o = other (altri) a = all (tutti) = user	+ = aggiungi - = togli = = assegna	r = read (lettura) w = write (scrittura) x = execute (esecuzione)

Esempi:

- `chmod a+r documento.txt`

  Dà a tutti il permesso di leggere il file documento.txt
- `chmod o-r documentosegreto.txt`

  Non permette a tutti gli altri utenti che non appartengono al gruppo di leggere il file
- `chmod ug+x programma`

  Aggiunge il permesso di esecuzione all'utente e al gruppo
- `chmod +x documento.txt`

  Aggiunge il permesso di esecuzione all'utente (u è sottointeso)
- `chmod a=rw documento.txt`

  Setta i permessi a -rw-rw-rw-.

Seconda sintassi di chmod

Una sintassi alternativa è

```
chmod xyz documento.txt
```

dove x, y e z sono tre cifre (da 0 a 7) che indicano i permessi per utente, gruppo e altri. La cifra si determina ricordando che

- r è rappresentato da 4
- w è rappresentato da 2
- x è rappresentato da 1

e sommando questi valori. Si ottengono quindi i seguenti significati:

- 0 = `---` (nessun permesso)
- 1 = `--x`
- 2 = `-w-`
- 3 = `-wx` (2+1)
- 4 = `r--`
- 5 = `r-x` (4+1)
- 6 = `rw-` (4+2)
- 7 = `rwx` (4+2+1)

Ad esempio se avete un file riservato che non volete fare leggere a nessun altro potete settare i permessi a `rw-------` con il comando

```
chmod 600 filesegreto
```

A questo punto, mi sembra necessario ricordare che sono molto importanti le protezioni assegnate ad una directory (che si gestiscono con gli stessi comandi utilizzati per i file): in particolare, se assegnate ad un gruppo i diritti di scrittura su una directory, tutti gli utenti appartenenti a quel gruppo potranno non solo scrivere dei file in quella directory (che probabilmente è ciò che volete) ma anche cancellare qualsiasi file in quella directory (cosa che probabilmente NON volete). L'assegnazione dei diritti su un file o su una directory, quindi, in un sistema multiutente, è una cosa piuttosto delicata, e bisogna sempre pensare ai possibili effetti collaterali di quello che si fa.

2.5 Cambiare il proprietario e il gruppo

Per cambiare l'utente proprietario di un file si usa il comando

```
chown nuovo_proprietario filename
```

e per cambiare il gruppo

```
chgrp nuovo_gruppo filename
```

Per cambiare il proprietario o il gruppo di un file bisogna essere il super-user (root).

3 Montare e smontare i filesystem

3.1 Mount e Umount

Scordatevi i dischi etichettati con `A:`, `C:`,`D:`. In Unix il filesystem cioè la struttura ad albero secondo la quale sono organizzate la directory è unica. Perché la struttura delle directory si chiama ad albero? Immaginatevi un albero rovesciato (in informatica si disegnano gli alberi con la radice in alto, contro ogni evidenza biologica, con l'eccezione forse della Nuova Zelanda...[2]). In alto troviamo la radice di tutto che si chiama `/`. Come in un albero genealogico, la radice ha vari figli (/usr, /tmp, /home ecc...) i quali a loro volta possono contenere altre directory, formando così una struttura ad albero in cui ogni nodo può avere dei figli.

Come si fa allora ad accedere al floppy, al cdrom, ad altri hard disk? La risposta di Unix è molto semplice, un filesystem va montato in una directory. Montare un filesystem significa sostituire la porzione di albero che comincia da un directory con il filesystem contenuto in un altro dispositivo.

Per esempio provate a inserire un dischetto (formattato) nel drive e a scrivere

```
guest@curzio ~$ mount /floppy
```

(questo funziona su un sistema Debian. Nelle distribuzioni Red Hat e Mandrake di solito la directory giusta dove montare il floppy disk è `/mnt/floppy`).

A questo punto potete entrare nella directory `/floppy` e vedere il contenuto del dischetto, copiarci file e fare tutto quello che volete come se fosse una directory qualunque. Quando avete finito di usare il dischetto, uscite dalla directory `/floppy` e scrivete

```
guest@curzio ~$ umount /floppy
```

Solo a questo punto potete estrarre il dischetto. Ricordatevi sempre di smontare i dischetti prima di estrarli. Con i cd-rom non c'è questo problema, perché di solito Linux impedisce di estrarre il cdrom se è ancora montato.

3.2 Il file di configurazione /etc/fstab

In realtà se la sintassi di mount e umount è così semplice perché il sistema è opportunamente configurato. Altrimenti per montare il floppy avreste dovuto scrivere qualcosa del tipo

```
# mount -t vfat /dev/fd0 /mnt/floppy/
```

e avreste dovuto essere root per farlo. Infatti di norma l'operazione di mount è consentita solo al root e il comando mount deve specificare la periferica (in questo caso `/dev/fd0` è il file associato al drive A:), la directory dove montare e il tipo di filesystem (`vfat` indica che il floppy è formattato per il DOS).

Per evitare di dover scrivere tutte queste cose tutte le volte, esiste un apposito file di configurazione, che come molti altri file di configurazione si trova nella directory `/etc`. Questo file si chiama `/etc/fstab` (file system table).

Ecco come si presenta questo file in una distribuzione Debian:

[2] questa orribile battuta non è mia, l'ho trovata sul libro *Practical OO Development with C++ and Java* di Cay S. Horstmann.

```
# /etc/fstab: static file system information.
#
# <file system> <mount point>   <type>  <options>                       <dump>  <pass>
/dev/hda6       /               ext2    defaults,errors=remount-ro      0       1
/dev/hda5       none            swap    sw                              0       0
proc            /proc           proc    defaults                        0       0
/dev/fd0        /floppy         auto    defaults,user,noauto            0       0
/dev/hda1       /boot           ext2    rw                              0       2
/dev/scd0       /cdrom          iso9660 defaults,ro,user,noauto,exec    0       0
/dev/hda2       /win            vfat    rw,user                         0       0
```

La prima colonna di ogni riga di questa tabella indica la periferica da montare, la seconda il punto di montaggio, la terza il tipo di filesystem, la quarta le opzioni. Le altre due colonne le vedremo meglio dopo.

Nell'esempio si vede che il filesystem principale viene montato in `/`. Il filesystem principale è un filesystem Linux ed è quindi di tipo `ext2` (recentemente è stato introdotto anche il filesystem `ext3`), `/dev/hda6` è la partizione che contiene il root filesystem. Le partizioni su linux corrispondono a file speciali di nome `/dev/hd`*xn* dove ad hd segue una lettera che indica l'hard disk (a=master, b=slave) e un numero che indica la partizione.

La seconda riga è quella che indica dove si trova la partizione di swap, che viene usata per la memoria virtuale e quindi non deve essere montata. La terza riga riguarda un filesystem speciale per la gestione dei processi. La quarta riga indica che il floppy deve essere montato nella directory floppy, il tipo di filesystem deve essere rilevato automaticamente, le opzioni user e noauto indicano che non deve essere eseguito il mounting automatico all'avvio e che il floppy può essere montato anche da utenti normali. Sulla riga successiva vediamo un filesystem che viene montato nella directory `/boot` e che contiene il kernel.

Il cdrom ha un filesystem di tipo `iso9660`, le opzioni indicano che deve essere montato in sola lettura (ro = read-only), che può essere montato da tutti (user), che non deve essere montato automaticamente all'avvio, e che i file al suo interno possono essere eseguiti (sul filesystem dei cdrom i file non hanno gli usuali permessi di unix perché usano un filesystem diverso).

Infine sull'ultima riga vediamo come si può accedere da Linux a una partizione Windows. Il tipo di filesystem usato da Windows è `vfat`, si deve indicare rw=read-write per poterci scrivere e user per poterlo montare senza essere root. Il problema di poter scrivere come utente comune (non root) su una partizione window è molto comune quindi lo descrivo nella sezione 3.3

Due parole sul significato delle ultime due colonne. La prima di queste indica al programma dump quali filesystem sottoporre al backup (di solito nessuno quindi si scrive 0), la seconda comunica al programma fsck per determinare l'ordine nel quale deve essere fatta la verifica dei file system in fase di reboot (quando raggiungono un certo numero di mount o quando il computer non viene chiuso correttamente). Specificare 1 per il filesystem radice (quello montato in `/`) e 2 per gli atri filesystem linux. Per i filesystem non linux specificare 0 perché non vanno controllati.

Per informazioni più dettagliate, naturalmente... `man fstab`.

3.3 Come montare un filesystem FAT32 (windows) in modo da poterci scrivere senza essere root

In questa sezione viene spiegato come impostare la fstab in modo da poter accedere in modo facile alla partizione windows, anche da utente non root. Queste indicazioni si riferiscono a filesystem di tipo FAT, quello usato da Windows 95 e Windows 98. Windows 2000 di default usa NTFS, anche se al momento dell'installazione di Windows 2000 è possibile chiedere che il disco venga formattato con FAT32.

Il problema da affrontare è che, quando i filesystem vengono montati all'avvio l'operazione di mount viene effettuata da utente root e quindi di default tutti i file del filesystem di Windows vengono assegnati all'utente root. È possibile modificare questo comportamento di default specificando opportuni parametri nella fstab.

Come prima cosa creiamo un gruppo di utenti che avranno il permesso di scrivere sulla partizione di Windows. Chiamiamo il gruppo ad esempio `fatwriters`.

```
# addgroup fatwriters
```

Aggiungiamo a questo gruppo gli utenti che avranno il permesso di scrivere sulla partizione di Windows, ad esempio supponiamo che gli utenti a cui vogliamo accordare questo permesso siano `marco` e `sara`.

```
# adduser marco fatwriters
# adduser sara fatwriters
```

Faremo in modo che i file all'interno della partizione windows appartengano all'utente root e al gruppo fatwriters. Per fare questo è necessario conoscere l'uid di root (è sempre 0!!) e il gid di fatwriters. Per sapere il gid di fatwriters, fate il login come uno degli utenti che appartengono al gruppo fatwriters (ricordatevi che è necessario rifare il login perchè l'utente venga effettivamente aggiunto al gruppo!) e usate il comando id.

```
marco@curzio:~$ id
uid=1001(marco) gid=1001(marco) gruppi=1001(marco),104(fatwriters)
```

In questo caso l'id del gruppo `fatwriters` è 104. Tra le opzioni di montaggio del filesystem di windows dovremo quindi aggiungere le opzioni che indicano l'uid e il gid con i quali effettuare l'operazione di montaggio. Inoltre si deve specificare la maschera dei permessi in modo che il gruppo abbia i permessi di scrittura (questo si fa con l'opzione `umask=1003` per permettere solo a chi appartiene al gruppo di leggere e scrivere mentre a chi non appartiene al gruppo è permesso l'accesso in sola lettura) Nel nostro caso si avrà `uid=0,gid=104,umask=1003`. Altre opzioni utili per filesystem di tipo FAT sono `rw` che dice che il filesystem è in lettura e scrittura, `user` che permette a tutti di montarlo e smontarlo, exec che permette di eseguire file contenuti nel filesystem. L'opzione `codepage=850` determina il modo di visualizzare i caratteri speciali e può essere utile se i caratteri accentati d un filesystem windows non vengono visti correttamente da Linux.

In definitiva la riga della fstab che regola il montaggio del filesystem di windows sarà qualcosa del genere.

```
/dev/hda1  /win vfat user,rw,exec,codepage=850,uid=0,gid=104,umask=1003
0 0
```

Per controllare che tutto funzioni, rimontate il filesystem di windows e controllate che i file appartengano all'utente root e al gruppo fatwriters, poi provate a scrivere dentro al filesystem di Windows come uno degli utenti appartenenti al gruppo fatwriters.

4 Installazione di programmi

La maggior parte del software su Linux è software libero, che quindi viene distribuito attraverso i sorgenti, scritti di solito in C o in C++. Quindi installare un programma su linux può voler dire dover ricompilare i sorgenti per ottenere un file binario eseguibile dalla macchina.

Per fortuna non è sempre necessario ricompilare tutti i programmi, perché questa operazione non è sempre facile. Le varie distribuzioni hanno introdotto vari formati di pacchetti già compilati.

4.1 Gestione da linea di comando di pacchetti RPM

Il formato RPM (RedHat Package Manager) è il sistema di gestione dei pacchetti della distribuzione Red Hat che viene usato anche dalla Mandrake (che è derivata dalla Red Hat) e da altre distribuzioni, quali ad esempio la SUSE.

Ci sono molti programmi grafici per la gestione dei pacchetti RPM, ad esempio *rpmdrake* della Mandrake oppure *KPackage* e *GnoRPM*.

Tuttavia è importante anche sapere come installare gli rpm da linea di comando, un po' per le solite ragioni (è più veloce, ecc...), ma anche perché vi potreste trovare in una situazione in cui il programma grafico non è disponibile (crash della scheda video...).

Per installare un pacchetto rpm dovete essere l'amministratore di sistema (diventatelo con il comando su, cfr. 2.2) e usare il comando **rpm**. Portatevi nella directory dove si trova il pacchetto da installare e scrivete

```
[root /root]# rpm -i pacchetto.rpm
```

Opzioni interessanti sono `v` che vi permette di vedere alcune informazioni aggiuntive sui pacchetti e `h` che dà una rappresentazione grafica del progresso nell'installazione. Invece l'opzione `U` (update), nel caso che il pacchetto sia già installato lo aggiorna, se non è installato lo installa. Quindi spesso la scelta migliore è

```
[root /root]# rpm -Uvh pacchetto.rpm
```

mentre

```
[root /root]# rpm -F pacchetto.rpm
```

aggiorna il pacchetto solo se una versione più vecchia del pacchetto è già installata. In questo modo è facile aggiornare tutti i pacchetti che già abbiamo installato: dalla directory che contiene tutti i nuovi pacchetti lanciate il comando

```
[root /root]# rpm -F *.rpm
```

Si può controllare se un pacchetto è installato con il comando `rpm -q`. Esempio

```
guest@curzio ~$ rpm -q emacs
emacs-20.7-34
```

Per disinstallare un pacchetto usare il comando

```
# rpm -e nomepacchetto
```

4.2 Installazione dai sorgenti di un pacchetto tgz

A volte può capitare di non trovare il pacchetto rpm di un programma e di dovere compilare direttamente i sorgenti.

Vediamo qual è la procedura standard, che a volte funziona, a volte incontra qualche problema se il nostro sistema non ha tutte le caratteristiche giuste per quel pacchetto.

I sorgenti vengono distribuiti di solito nel formato compresso tgz. Il file che scaricate avrà estensione `tgz` o `tar.gz`. In ogni caso potete estrarre il contenuto dell'archivio con il comando

```
tar zxvf nomepacchetto.tgz
```

Di solito il pacchetto crea una directory in cui sono contenuti i sorgenti del programma. Entrate nella directory. La procedura standard per installare un programma è

```
$ ./configure
$ make
```

e, da root,

```
# make install
```

Il primo comando controlla le caratteristiche del vostro sistema e prepara la compilazione con i parametri appropriati. Il comando make lancia la compilazione dei sorgenti secondo le modalità descritte nel file di nome `Makefile`, make install copia i file nelle directory giuste. Quando avete installato con successo il programma potete cancellare la directory creata al momento dell'estrazione dell'archivio, non cancellate invece il pacchetto che avete scaricato perché vi potrebbe servire per installarlo nuovamente.

Naturalmente ci sono delle eccezioni, a volte `./configure` non serve, a volte ha bisogno di alcuni parametri... con un po' di pratica imparerete a gestire anche i casi più complicati.

5 Ringraziamenti e indirizzi di riferimento

Per scrivere questi appunti mi sono ispirato a vari altri documenti analoghi, tra i quali le lezioni del dott. Pugliese per il laboratorio di sistemi operativi del corso di laurea in informatica, gli appunti linux di Daniele Giacomini e la lezione introduttiva a Linux di Sergio Ballestrero (dalla quale ho tratto la magnifica metafora del trapano e, soprattutto l'intero paragrafo 2.1 più qualche frase qua e là). Riporto i riferimenti per trovare questo materiale su internet.

- Appunti Linux di Daniele Giacomini:
  http://a2.swlibero.org/
- Lezione di Sergio Ballestrero:
  http://www.firenze.linux.it/docs/corso/sergio/index.html
- Home Page di Rosario Pugliese:

  `http://www.dsi.unifi.it/~pugliese/`

Devo inoltre ringraziare Marco Nenciarini che mi ha spiegato il metodo descritto nella sezione 3.3 per montare il filesystem di Windows con i permessi opportuni.

Gli appunti che state leggendo si possono trovare on line in vari formati (ps, pdf e anche i sorgenti LaTeX) all'indirizzo

`http://www.prato.linux.it/~lmasetti/seriamente/lezionebash/`