

## Modulo 2: Controllo degli accessi e autenticazione Breve introduzione a Linux

### 1 Breve storia

Con il termine Linux si fa riferimento ad una famiglia di sistemi operativi di tipo Unix-like, rilasciati sotto varie possibili distribuzioni, aventi la caratteristica comune di utilizzare come nucleo il kernel Linux.

Il kernel Linux fu programmato nell'agosto 1991 dal giovane studente finlandese della Helsinki University Linus Torvalds che, appassionato di programmazione, era insoddisfatto del sistema operativo Minix (sistema operativo Unix-like destinato alla didattica, scritto da Andrew Tanenbaum, professore ordinario di Sistemi di rete all'università di Amsterdam), poiché supportava male la nuova architettura i386 a 32 bit, all'epoca economica e popolare. Inizialmente, Linux per girare utilizzava, oltre al kernel di Torvalds, l'userspace di Minix. Successivamente, Linus decise di rendere il sistema indipendente da Minix in quanto la licenza di Minix lo rendeva liberamente utilizzabile solo a fini didattici e decise, quindi, di sostituire quella parte del sistema operativo col software del progetto GNU. Per fare ciò, Torvalds cambiò la licenza e adottò la GPL.

Il suo sviluppo crebbe enormemente nel corso degli anni e col passare del tempo aumentarono anche coloro che erano interessati a questo progetto. Si unirono a Torvalds migliaia di sviluppatori tra studenti e ricercatori da tutto il mondo. Ad oggi Linux è disponibile in circa 300 diverse distribuzioni che si distinguono per veste grafica e diverse applicazioni, anche se hanno tutte il medesimo kernel Linux.

### 2 Linux: il Filesystem

In Linux esistono tre diverse tipologie di file:

- **Ordinari**: archivi di dati, comandi, programmi sorgente, eseguibili, ecc.
- **Directory** (o **Cartelle**): un file speciale contenente riferimenti ad altri file (file ordinari, speciali o cartelle).
- **Speciali**: Interfaccia al driver di dispositivi hardware, memoria centrale, hard disk, ecc.

I file sono organizzati in una struttura gerarchica ad albero (come in Figura 1). La radice corrisponde ad una directory, mentre i nodi successivi possono essere directory, file di dati o file di altro genere.

Per identificare un nodo (file o directory) all'interno della gerarchia del file system, si definisce il percorso (*path* o *pathname*). Il percorso è espresso da una sequenza di nomi di nodi che devono essere attraversati, separati da una barra obliqua (/). Si distinguono due tipi di percorsi: **relativo** (rispetto alla directory corrente) e **assoluto** (rispetto alla radice dell'albero). Va ricordato che Linux è *case sensitive*: ovvero è sensibile alla differenza tra le lettere maiuscole e minuscole. Ad esempio eventuali file di nome `Prova`, `pRova`, `prova`, `PROVA` per Linux sono tutti diversi.

Alcuni nomi particolari di directory:

- . la directory corrente

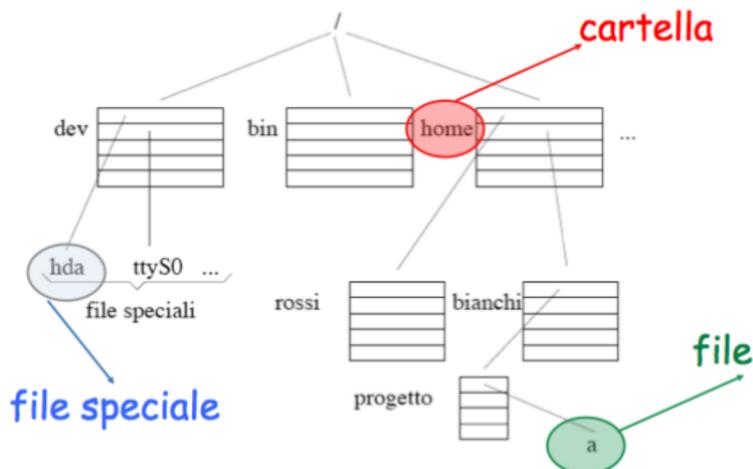


Figura 1: Esempio di struttura ad albero del filesystem

- ... la directory madre
- ~ la propria home utente

Inoltre, alcune directory sempre presenti sono:

- Directory radice /
- Directory /usr dove viene riposto il software applicativo;
- Directory /home dove vengono riposti i dati degli utenti;
- Directory /dev dove ci sono le interfacce ai driver dei vari dispositivi.

## 2.1 Esempi di path

Si supponga di voler fare riferimento ad una directory `slide` contenuta nella directory `lab1` nella mia home. In questo caso il path assoluto è:

`/home/braghin/lab1/slide`

Mentre il path relativo (supponendo di trovarsi nella directory `/home/braghin`) è:

`lab1/slide`

## 2.2 Prefissi usati per identificare il tipo di dispositivo in /dev

Quelli che seguono sono i prefissi usati per identificare il tipo di dispositivo in `/dev`:

- `fd`: floppy disk oppure file descriptor;
- `hd`: identifica un disco di tipo IDE (`sd` nel caso sia SCSI);
- `hda`: il primo disco (`hda1`: prima partizione logica);
- `hdb`: il secondo disco;
- `lp`: line printer (stampanti);

- `tty`: terminali.

### 2.3 La shell di Linux

In un sistema operativo, una *shell* (detta anche *terminale* o *interprete di comandi*) è un programma che gestisce l'interazione tra l'utente e il sistema operativo e permette di impartire comandi e avviare altri programmi, oppure di gestire il sistema. È una delle componenti principali di un sistema operativo, insieme al *kernel*.

La shell può essere:

- **Testuale**: l'utente interagisce attraverso un terminale (o un emulatore di terminale) tramite un'interfaccia a riga di comando.
- **Grafica**: ci sono i cosiddetti *desktop environment*, che forniscono agli utilizzatori un ambiente grafico da cui è possibile gestire file e avviare programmi (ad esempio X-Windows).

In caso di interfaccia a riga di comando l'interazione con il sistema aperitivo avviene nel modo seguente:

- l'utente apre un terminale (si veda Figura 2);
- il sistema operativo rimane in attesa di un comando da parte dell'utente dopo il *prompt* (simbolo speciale);
- se l'utente digita un comando seguito da INVIO, il comando/programma viene mandato in esecuzione interagendo o meno con l'utente e mostrando dati sul terminale;
- al termine dell'esecuzione ricompare il *prompt*, in modo da continuare l'interazione con l'utente.

Per terminare la shell si possono digitare i comandi `logout` o `exit` oppure premere `Ctrl-D`.

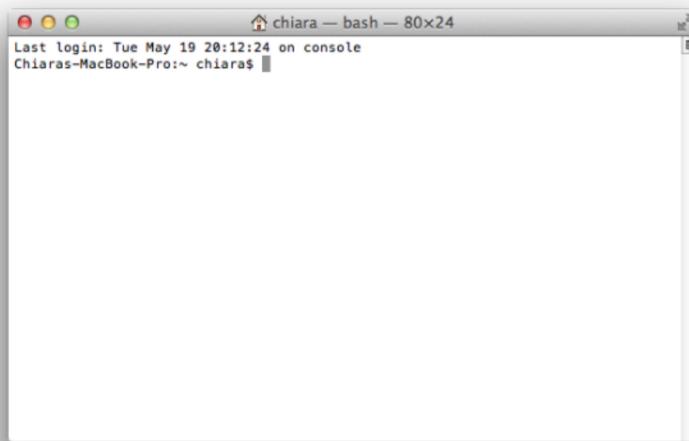


Figura 2: Esempio di utilizzo di terminale

Il simbolo del *prompt* non è sempre lo stesso: il *prompt* di default di un normale utente è il simbolo del dollaro (\$), mentre il *prompt* di default per l'utente di root (amministratore del sistema) è il simbolo cancelletto (#).

In molti sistemi Linux, di default, il simbolo del *prompt* è preceduto da alcune informazioni come il nome del computer, la directory corrente e il nome dell'utente che ha aperto la shell.

## 3 Linux: comandi principali

Tipicamente la sintassi di un comando è:

```
nomeComando opzioni argomenti
```

La convenzione che adotto in questa dispensa nella rappresentazione della sintassi dei comandi (e che utilizza anche il comando `man`) è la seguente:

- se un'opzione o un argomento possono essere omessi, li indico tra parentesi quadre `[opzione]`
- se due opzioni/argomenti sono mutuamente esclusivi, li separo dalla barra verticale `|` come in `arg1 | arg2`
- quando un argomento può essere ripetuto  $n$  volte, aggiungo dei puntini come in `arg...`

### 3.1 Comando importante!

Un comando molto importante è il comando che fa leggere la pagina del manuale relativa al comando per cui è stato consultato. La sintassi è:

```
man nome_comando
```

È consultabile per informazioni su (quasi) ogni comando e indica: il formato del comando (input) e risultato atteso (output), la descrizione delle opzioni, le possibili restrizioni e i file di sistema interessati dal comando. Inoltre specifica anche i comandi correlati ed eventuali bug.

Per uscire dal manuale, digitare `:q` (*quit* per editor tipo `vi`).

## 3.2 Comandi per la gestione di file e cartelle

### 3.2.1 Informazioni su file e directory

```
ls [opzioni...] [file...]
```

Elenca informazioni su file ed il contenuto delle directory. Alcune opzioni possibili sono:

- `l` (long format): per ogni file riporta nome, proprietario, gruppo del proprietario, occupazione di disco (blocchi), data e ora dell'ultima modifica o dell'ultimo accesso, ecc.
- `a` (all files): fornisce una lista completa dei file contenuti in una directory (normalmente i file il cui nome comincia con il punto non vengono visualizzati)
- `t` (time): la lista è ordinata per data dell'ultima modifica
- `u`: la lista è ordinata per data dell'ultimo accesso
- `r` (reverse order): inverte l'ordine alfabetico

### 3.2.2 Creazione/rimozione di directory

```
mkdir d1  
rmdir d1
```

Crea o rimuove la directory `d1`.

### 3.2.3 Copia di file

```
cp f1 d1
```

Copia il file `f1` nella directory `d1`. Come primo argomento può prendere una lista di file; in tal caso il secondo argomento deve essere una directory: `cp f1 f2 f3 d1` come conseguenza copia `f1, f2, f3` nella directory `d1`.

### 3.2.4 Rinomina/spostamento di file o directory

```
mv f1 f2
```

Rinomina il file `f1` in `f2`.

### 3.2.5 Spostamento all'interno del filesystem

```
cd dir
```

Il comando `cd` sta per *change directory* e come effetto sposta l'utente dalla directory corrente alla directory `dir`. Il comando `cd` usato senza argomenti sposta l'utente nella sua home directory, mentre `cd ..` sposta l'utente nella directory madre.

Per sapere in quale directory ci si trova si può utilizzare il comando `pwd`, che sta per *present working directory*.

### 3.2.6 Visualizzazione del contenuto di un file

Comandi per la visualizzazione di file sono: `cat, more, tail e head`. Prendono come argomento il nome del file.

### 3.2.7 Metacaratteri

La shell Linux riconosce alcuni caratteri speciali, chiamati metacaratteri o *wild card*, che possono comparire nei comandi: quando l'utente invia un comando, la shell lo scandisce alla ricerca di eventuali metacaratteri, che processa in modo speciale; una volta processati tutti i metacaratteri, viene eseguito il comando.

I metacaratteri più comuni sono:

- \* stringa di zero o più caratteri in un nome di file
- ? singolo carattere in un nome di file
- [ z f c ] singolo carattere tra quelli elencati, anche range di valori nel caso di [ a - d ]
- {} stringa tra quelle elencate tra le parentesi graffe
- simbolo di *escape*, segnala di non interpretare il carattere successivo come speciale

Esempi:

```
ls [q-s]* lista i file con nomi che iniziano con un carattere compreso tra q e s.
```

```
ls [a-d,0-9]*[c,g,1]? elenca i file i cui nomi hanno come iniziale un carattere compreso tra a e d oppure tra 0 e 9, e il cui penultimo carattere sia c, g o 1.
```

### 3.2.8 Ridirezione dell'I/O

Di default i comandi Linux prendono l'input da tastiera (standard input) e mandano l'output ed eventuali messaggi di errore su video (standard output, error). L'input/output in Linux può essere rediretto da/verso file, utilizzando dei metacaratteri:

```
comando < fileInput
```

Ridirezione dell'input: `comando` legge l'input dal file `fileInput` (file aperto in lettura) e non da tastiera.

```
comando > fileOutput
```

Ridirezione dell'output: `comando` scrive l'output non a video ma nel file `fileOutput` (file aperto in scrittura - nuovo o sovrascritto).

```
comando >> fileOutput
```

Ridirezione dell'output: `comando` scrive l'output non a video ma in coda nel file `fileOutput`.

Ad esempio:

```
ls -l > fileListato
```

`fileListato` conterrà il risultato di `ls -l`.

```
sort < file > file2
```

Ordina il contenuto di `file` scrivendo il risultato su `file2`.

### 3.2.9 Piping

L'output di un comando può esser diretto a diventare l'input di un altro comando (piping) usando il carattere speciale `|` (*pipe*). Il carattere viene usato come costrutto parallelo: l'output del primo comando viene reso disponibile al secondo e consumato appena possibile, non ci sono file temporanei. Suggerimento per l'uso: aggiungere i comandi uno alla volta (per vedere cosa viene prodotto in output da ogni pezzo della pipe).

Esempi:

```
$ who | wc -l > numUtenti
```

Il comando `who` preleva il numero di utenti connessi, `wc` conta il numero di parole nella lista, e il risultato viene scritto sul file `numUtenti`.

```
$ ls | rev
```

Lista dei file nella directory corrente in ordine invertito.

### 3.2.10 Autocompletamento dei comandi

La shell Bash fornisce un utile sistema di autocompletamento ed esplorazione dei comandi. Se scrivo sul terminale una qualsiasi lettera e successivamente premo due volte il tasto tab, viene restituito l'elenco completo dei comandi che iniziano con la lettera che ho digitato.