

# File System e Interprete Comandi

---

Marco Alberti

Programmazione e Laboratorio, A.A. 2016-2017

Dipartimento di Matematica e Informatica - Università di Ferrara

Ultima modifica: 29 settembre 2016

- File e cartelle
- Interprete comandi
  - Comandi su cartelle
  - Comandi su file
  - Redirezione e piping

# File System

---

- Sequenza di dati omogenei a cui è associato un nome
- E' compito delle applicazioni associare informazione ai dati
- Esempio: programma, file di testo, documento, immagine (in vari formati), film...
- Fisicamente memorizzato in modo diverso a seconda del supporto di memorizzazione, ma il file system rende le differenze trasparenti all'utente o all'applicazione
- FAT, NTFS, ext?, HFS<sup>+</sup>

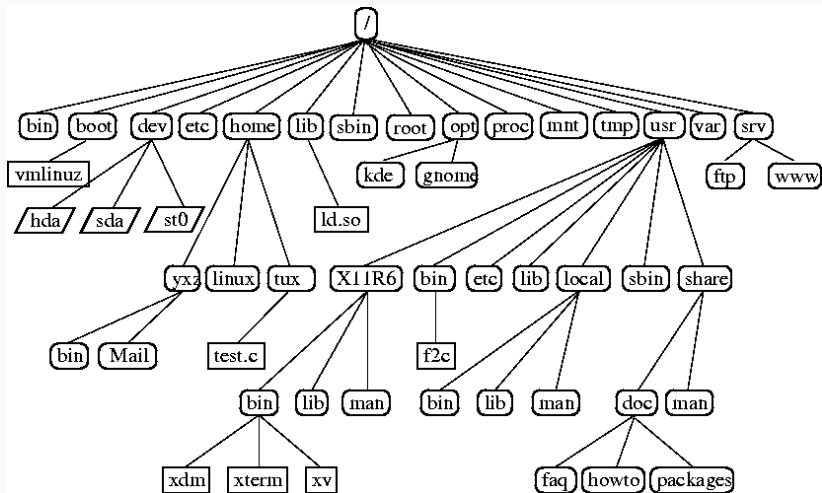
## Nomi ed estensioni

- I nomi di file sono sequenze di caratteri (lettere, numeri, simboli)
- In Linux è proibito il carattere / (slash), utilizzato come separatore nei percorsi
- Alcuni caratteri che hanno un significato speciale devono essere "escaped", cioè preceduti dal carattere \ (backslash) se inseriti in un nome non racchiuso da virgolette(""); tra questi ' ' (spazio), backslash, \* (asterisco).
- In UNIX (e quindi Linux) i nomi sono case sensitive ("Pippo.txt"  $\neq$  "pippo.txt")
- I file il cui nome inizia con . (punto) sono convenzionalmente nascosti

- La parte del nome che segue l'ultimo . (punto) è detta estensione, e convenzionalmente indica il tipo di file (`esempio.txt` è *probabilmente* un file di testo)
- In Linux i file eseguibili (codice macchina) non hanno estensione; in Windows hanno solitamente estensione `exe` o `dll`.

- Un computer contiene milioni di file.
- Non è pensabile tenerli tutti insieme in un unico contenitore
- Possibile soluzione: dividere i file in gruppi
- Tuttavia anche i gruppi diventano presto numerosi e il problema si ripresenta
- Soluzione (migliore): **cartella** (in inglese "folder" o "directory"), contenitore di file o *cartelle*.
- In realtà è un file, opportunamente contrassegnato dal sistema operativo.

# Tipica struttura di un filesystem Linux





# Struttura ad albero

La definizione ricorsiva di cartella definisce la **struttura ad albero** (gerarchica) di ogni file system.

**Nodo**: ogni file o cartella

Padre di un nodo: il nodo della cartella in cui è contenuto.

Nodo senza padre: **root** (/ , radice).

**Foglia**: nodo senza figli. I file sono necessariamente foglie.

# Shell

---

# Interprete comandi

Programma che ripete la sequenza:

1. attendi un comando
2. esegui il comando

Spesso chiamato **shell** (guscio) per differenziarlo dal **kernel** (nucleo)

Esempi:

- Unix: sh, bash, csh, zsh
- Windows: cmd ("Prompt dei comandi"), PowerShell

Eseguito in un **emulatore di terminale**

## Sintassi di base

```
nome_comando [-opzione_breve{opzione_breve}*]  
{--opzione_lunga }* {argomento}*
```

## Esempio

```
ls
```

```
ls -l
```

```
ls -la
```

```
ls --color
```

```
ls -l --color /bin/
```

## Comandi interni ed esterni

- Comandi interni (**shell builtin**): implementati come parte del programma shell
- Comandi esterni: qualsiasi programma eseguibile

# Documentazione

La prossima rassegna di comandi è tutt'altro che esaustiva. I comandi di shell sono numerosissimi, potenti e, fortunatamente, ben documentati.

- molti comandi hanno a disposizione una sintetica **guida in linea** (cioè integrata nel comando), che si invoca solitamente con le opzioni `-h` o `--help` e che contiene la sintassi del comando e l'elenco delle opzioni disponibili
- **man pages**: guida che spiega nel dettaglio gli effetti del comando e tutte le opzioni. Si apre invocando il comando `man __comando__`.
- **info**: veri e propri manuali di comandi e pacchetti complessi, dove si trovano spiegazioni più approfondite. Si apre invocando il comando `info __comando__`. Se non sono installate sul sistema sono comunque disponibili online.

## **Percorso (path) assoluto**

di una cartella o di un file `f`: elenco dei nodi dell'albero del filesystem da attraversare per arrivare alla cartella o al file, compresi `/` e `f`, separati da `/` a partire dal secondo.

## **Esempio**

`/home/marco/programmazione/esempi`

# Percorsi relativi

In ogni momento, la shell si trova in una cartella del file system, detta **cartella corrente**.

## **Percorso relativo**

di una cartella o di un file **f**: come percorso assoluto, ma partendo dal nodo della cartella corrente anziché dal nodo root.

## **Esempio**

Se la cartella corrente è `/home/marco`, il percorso relativo della cartella dell'esempio precedente è `programmazione/esempi`.



## Visualizzazione

`pwd` stampa il percorso assoluto della cartella corrente.

## Cambio

`cd __path__`, dove `__path__` è il percorso (relativo o assoluto) di una cartella, imposta a `__path__` la cartella corrente.

# Cartelle speciali

- `.`: la cartella corrente
- `..`: la cartella padre (sic) della cartella corrente; utilizzabile nei percorsi
- `~`: la cartella **home** dell'utente (normalmente `/home/__nome_utente__`, ad esempio `/home/marco`).

## Esempio

- `../..` è la cartella superiore di due livelli rispetto a quella corrente.
- `~/..` è solitamente la cartella `/home`
- `~/../luigi` è `/home/luigi`

Se `__path__` è un percorso relativo o assoluto:

## **Creazione**

```
mkdir __path__
```

crea la cartella `__path__`.

## **Eliminazione**

```
rmdir __path__
```

 elimina la cartella `__path__` (se vuota, cioè non contenente altri file o cartelle).

## ls

mostra il contenuto della cartella corrente.

## ls `__path__`

mostra il contenuto di `__path__`:

- se `__path__` è un file, mostra il file stesso.
- se `__path__` è una cartella, mostra i suoi figli.

## Opzioni

- `-l`: mostra dettagli su file e cartelle
- `-a`: mostra tutti i file e le cartelle (compresi quelli nascosti)

# Permessi file

Ogni file e cartella ha **proprietario** (owner) e **gruppo** (group).

I permessi possibili sono

- **read** (lettura)
- **write** (scrittura)
- **execute** (esecuzione<sup>1</sup>)

riferiti a

- **user** (proprietario)
- **group** (utenti del gruppo, escluso il proprietario)
- **others** (altri)

---

<sup>1</sup>Transito in caso di cartella

# Dettagli file

total 304

Tipo	Permessi								
	user	group	others	HardLink	Utente	Gruppo	Dimensione	Ultima modifica	nome
d	rwx	r-x	r-x	3	marco	marco	4096	Sep 17 12:30	bibliography
-	rw-	r--	r--	1	marco	marco	42896	Sep 14 18:26	llncs.cls
-	rw-	r--	r--	1	marco	marco	198444	Sep 20 21:36	main.pdf
-	rw-	r--	r--	1	marco	marco	21128	Sep 20 21:36	main.tex
-	rw-	r--	r--	1	marco	marco	33098	Sep 14 18:26	splncs03.bst

Ogni utente appartiene a zero o più gruppi

## A quali gruppi appartengo?

Comando `groups`

Un'applicazione può eseguire un'operazione su un file se vale almeno una delle seguenti condizioni:

- l'utente che la esegue è il proprietario e l'operazione è consentita per user
- l'utente che la esegue appartiene al gruppo del file e l'operazione è permessa per group
- l'operazione è permessa per others

L'utente `root` (`superutente`) può eseguire qualsiasi operazione, indipendentemente dai permessi.

## Operazioni legate ai permessi

Anch'esse soggette a permesso.

**chown** \_\_utente\_\_ \_\_path\_\_

rende \_\_utente\_\_ proprietario di \_\_path\_\_

**chgrp** \_\_gruppo\_\_ \_\_path\_\_

rende \_\_gruppo\_\_ gruppo di \_\_path\_\_

**chmod** [ugoa][+-][rwx] \_\_path\_\_

aggiunge (se +) o toglie (se -) il permesso di lettura, scrittura o esecuzione (se r, w, x, rispettivamente) a user, group, altri, tutti (se u, g, o, a, rispettivamente).



I file possono essere creati

- dal comando `touch`
- da applicazioni (es. editor di testo)
- per copia

### Esempio

`touch __path__` crea un file vuoto se `__path__` non esiste, o aggiorna la data di ultima modifica se esiste.

`cp __origine__ __destinazione__`

copia il file `__origine__`

- se `__destinazione__` esiste ed è
  - una cartella, come nuovo file dentro `__destinazione__`, con lo stesso nome file di `__origine__`
  - un file, nella cartella padre di `__destinazione__`, sovrascrivendo il file
- se `__destinazione__` non esiste, nella cartella padre di `__destinazione__`, con il nome file di `__destinazione__`

`cp -r __origine__ __destinazione__`

dove `__origine__` è una cartella,

- se `__destinazione__` esiste ed è un cartella, copia `__origine__` e, ricorsivamente, il suo contenuto dentro `__destinazione__`
- se `__destinazione__` non esiste, copia il contenuto di `__origine__` dentro una nuova cartella chiamata `__destinazione__`

`mv __origine__ __destinazione__`

sposta il file o cartella `__origine__` (con tutti i discendenti, se cartella)

- se `__destinazione__` esiste ed è
  - una cartella, come nuovo file o cartella dentro `__destinazione__`, con lo stesso nome di `__origine__`
  - un file (e `__origine__` è un file), nella cartella padre di `__destinazione__`, sovrascrivendo il file
- se `__destinazione__` non esiste, nella cartella padre di `__destinazione__`, con il nome di `__destinazione__`

# Eliminazione

```
rm __file__
```

elimina il file `__file__`

```
rm -r __dir__
```

elimina la cartella `__dir__` e tutti i suoi figli (attenzione!).

Se `__path__` è un file:

`cat __path__`

stampa a video il contenuto di `__path__`

`more __path__`

stampa a video il contenuto di `__path__`, diviso in pagine. Si può scorrere verso il basso ma non verso l'alto.

`less __path__`

stampa a video il contenuto di `__path__`, diviso in pagine. Si può scorrere anche verso l'alto.

In molti sistemi invocando `more` si invoca, in realtà, `less`.

## Wildcards (o caratteri jolly)



Poker di Re!

- ? corrisponde a **qualsiasi carattere**
- \* corrisponde a **qualsiasi sequenza di caratteri**

Utili (utilissimi) per eseguire la stessa operazione su più file o cartelle con un solo comando.

Indicare con l'uso di caratteri jolly

- Tutti i file nella cartella corrente
- Tutti i file della cartella `/bin`
- Tutti i file il cui nome contiene una `a`
- Tutti i file il cui nome termina con `a`
- Tutti i file il cui nome ha come secondo carattere una `a`
- Tutti i file il cui nome contiene una `a` che non sia l'ultima lettera
- Tutti i file con estensione `doc`



## redirezione

L'output di un comando può essere **rediretto** su un file

```
__comando__ > __file__
```

crea `__file__` contenente l'output di `__comando__`.

### Esempio

`echo "__contenuto__"` stampa `__contenuto__` a video.

`echo "__contenuto__" > __path__` crea un file `__path__` contenente `__contenuto__`.

```
__comando__ >> __file__
```

accoda a `__file__` (creandolo se non esiste) l'output di `__comando__`

Anche l'input di un comando può essere rediretto su un file

```
__comando__ < __file__
```

esegue `__comando__` dandogli come input il contenuto di `__file__`

`__comando1__ | __comando2__`

chiama `__comando2__` dandogli come input l'output di `__comando1__`

## Esempio

`ls -l | less` mostra il contenuto della cartella corrente diviso in pagine.

## Script di shell

Le shell possono interpretare non solo singoli comandi, ma veri e propri programmi, scritti in un linguaggio specifico della shell, in cui i comandi sono le operazioni elementari, detti **shell script**.

Il caso più semplice è la sequenza di comandi, uno per riga.

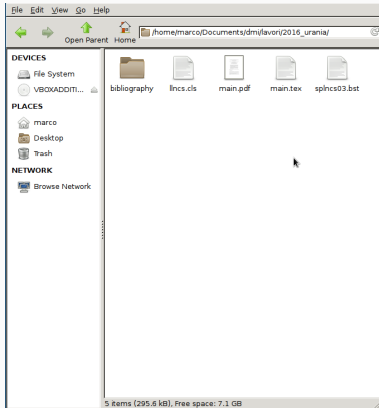
Gli script di shell devono essere resi eseguibili con il comando `chmod +x __nome__`.

Creare uno script di shell che

1. Stampi a video il messaggio "Inizio copia"
2. Crei nella cartella corrente una sottocartella di nome `copie`.
3. Copi in `copie` tutti i file con estensione `txt` della cartella corrente.
4. Stampi a video il messaggio "Fine copia".
5. Stampi a video i nomi dei file copiati.

Verificarne il funzionamento.

# Ambiente grafico



- Esistono interfacce grafiche (**Graphical User Interface**, GUI) per l'esecuzione di operazioni sul file system, dette **file manager**.

## Variabili di ambiente

Informazioni identificate da un nome e accessibili alla shell.

`echo $__VAR__`

stampa il valore della variabile di ambiente `__VAR__`.

`__VAR__ = __VALORE__`

fa sì che `$__VAR__` valga `__VALORE__` *nella shell corrente*.

La variabile `PATH` contiene un elenco di path, separati da `:`, in cui vengono ricercati file eseguibili corrispondenti al comando invocato.

### Esempio

`ls` è in realtà (di solito) `/bin/ls`. La variabile `PATH` contiene, fra gli altri, il percorso `/bin`.

La variabile `HOME`, abbreviata anche in `~`, contiene il percorso della cartella home dell'utente connesso.

La shell più usata è `cmd.exe` (Prompt dei comandi). Molte differenze:

- Un filesystem per ogni unità di memorizzazione, identificato da una lettera (es. `C:`, `F:`); ma in shell di origine Unix (es. Git-Bash) indicati in altro modo (es. `/c`, `/f`).
- il separatore nei path è `\` (ma quasi sempre i comandi accettano anche `/`)
- `cp` è `copy`, `mv` è `move`, `rm` è `del`