

CAPITOLO 8

LAVORARE CON I FILES

Pietro Buffa

A questo punto ciascuno di voi dovrebbe riuscire a muoversi con una certa disinvoltura tra gli intricati anfratti del proprio disco. Del resto è anche vero che un filesystem senza files è tanto inutile quanto una tavola apparecchiata senza niente da mangiare. E' arrivato quindi il momento di imparare a sgranocchiare files, capire come crearli, copiarli, spostarli, rinominarli e cancellarli a nostro piacimento, tutto utilizzando l'ormai familiare riga di comando.

Come ben sapete i files possono essere di vario tipo: un file può essere un semplicissimo testo (una lettera, un libro, un file di configurazione di qualche programma), ma anche un'immagine, un filmato, un file sonoro, altri files possono invece contenere sequenze di comandi comprensibili soltanto al computer (files binari o eseguibili). In Windows (e nelle interfacce grafiche in genere), la differenza tra i vari files viene evidenziata associando a ogni diverso tipo un'icona differente: un file di testo sarà probabilmente rappresentato da un foglio, un file immagine da una tavolozza con un pennello e così via. Anche all'interno dei vecchi sistemi Ms-DOS, in assenza di interfaccia grafica e di icone, esisteva un modo chiaro e preciso per individuare la funzione di ogni file: la famosa estensione, un suffisso di tre lettere che indica il tipo di file.

In GNU/Linux, come vi sarete accorti curiosando nel sistema, le estensioni sono molto meno utilizzate, alcuni files le hanno, altri no e in ogni caso, il sistema non obbliga l'utente a dotare di estensione i propri files.

Se desiderate chiamare un file con un nome tipo:

Mio_file_di_testo.Carlo le lettere dopo il punto non hanno nessun significato particolare.

ATTENZIONE: Prima di cominciare a conoscere i comandi che ci consentiranno di organizzare i nostri files, dobbiamo essere sicuri di ricordarci una peculiarità "caratteriale" di Linux. Non bisogna lasciarsi trarre in inganno dall'aspetto mite del simpatico pinguino rotondetto che rappresenta la mascotte di questo sistema operativo, è necessario ricordarsi sempre che Linux non perdona gli errori. Se cancellate un file, lo cancellate sul serio e non ci sarà nessun cestino, almeno a livello testuale, che vi permetterà di recuperarlo in seguito. Inoltre difficilmente il sistema si complimenterà con voi se fate qualcosa in modo giusto, il più delle volte l'unica cosa che otterrete dall'esecuzione corretta di un comando, sarà un laconico silenzio.

Per i primi tempi è bene quindi prestare molta attenzione mentre si digitano i comandi.

8.1 IL COMANDO file

GNU/Linux ha un comando apposito per capire esattamente con che tipologia di files abbiamo a che fare anche in assenza di estensioni. Questo comando, che dimostra la scarsità di fantasia di alcuni programmatori, si chiama appunto **file**, ed il suo utilizzo è molto semplice:

> **file** [nome_file]

Il sistema risponderà fornendo delle informazioni sul contenuto del file. Il comando non si lascia ovviamente "ingannare" dalle eventuali estensioni presenti: se date ad un file immagine una estensione ".mp3", esso continuerà ad individuarne correttamente il tipo. In GNU/Linux è quindi possibile dare nomi ai files in modo molto libero. Un'ultima considerazione: tutti i files che cominciano con un punto, sono considerati dal sistema **files nascosti** e non vengono visualizzati con il comando **ls** (a meno che non specifichiate l'opzione **-a**). Questa caratteristica viene spesso utilizzata per i files di configurazione, che di solito interessano poco ad un utente e in alcuni casi gli causerebbero soltanto fastidio.

8.2 IL COMANDO touch

Se volete creare files con l'unico scopo di fare delle prove potete utilizzare questo comando:

```
> touch [nome_file]
```

Naturalmente il fatto che abbiate creato un file vuoto non significa granchè, ma non c'è niente di meglio che creare qualche file per i vostri esperimenti senza rischiare di perdere importanti documenti.

8.3 IL COMANDO cp

Una delle operazioni più comuni che è necessario saper compiere con i files è la copia. Il comando si chiama **cp** (copy) e la sintassi è la seguente:

```
> cp [file_sorgente] [destinazione]
```

Facciamo un esempio:

Proviamo a creare due file di prova nella nostra home directory dal nome "prova1" e "prova2":

```
bash:~> touch prova1 prova2
```

Controllate se ci sono lanciando un **ls -l**

Adesso copiamo sul Desktop il primo file, cogliamo così l'occasione per mettere in evidenza come il Desktop sia una normalissima directory all'interno della personale directory home.

```
bash:~> cp prova1 Desktop
```

Adesso copiamo anche il secondo ma cambiandogli nome, chiamiamolo "prova_due":

```
bash:~> cp prova2 Desktop/prova_due
```

Dopo l'esecuzione dei due comandi, il Desktop contiene un file di nome "prova1" ed un file di nome "prova_due".

ATTENZIONE: Se copiamo o spostiamo uno o più file dandogli come "destinazione" una directory, bash capisce che il file deve essere copiato o spostato lì, ma se erroneamente la "destinazione" non è una directory ma un file, il file "sorgente" verrà copiato o spostato all'interno del file "destinazione", ed il contenuto di quest'ultimo cancellato.

E cosa succede se per caso esiste già in Desktop un file dal nome prova1? GNU/Linux lo cancella sostituendolo con quello sorgente. E' una buona idea quindi, prima di fare operazioni di questo tipo, controllare con un **ls -l** che non esista già un file che abbia lo stesso nome di quello che andiamo a copiare.

Si può però chiedere al comando **cp** di avvisarci se per caso esiste già un eventuale file con nome uguale a quello che andiamo a copiare, questo si ottiene utilizzando l'opzione **-i**.

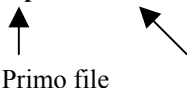
OPZIONI:

- i Chiede conferma per sovrascrivere files di destinazione eventualmente presenti.
- r Ricorsività. Effettua una copia ricorsiva di files, directory e sottodirectory.
- f Rimuove i files di destinazione preesistenti, se necessario.

```
bash:~> cp -i prova1 Desktop
cp: sovrascrivo 'Desktop/prova1'?
```

E' possibile inoltre copiare più files contemporaneamente, come mostra l'esempio:

```
bash:~> cp -i prova1 prova2 Desktop
```



Primo file Secondo file

8.4 IL COMANDO **mv**, SPOSTARE FILES E RINOMINARLI

Con una sintassi del tutto simile si può usare il comando **mv** (move), che serve a spostare i files (in maniera simile al comando “taglia ed incolla”, tipico di molti editor), la sintassi è la seguente:

```
> mv [file_sorgente] [destinazione]
```

E' vivamente consigliato anche per questo comando l'uso dell'opzione **-i**, in modo da ricevere un avviso se per caso esiste già un eventuale file con nome uguale a quello che andiamo a spostare.

E' possibile usare il comando **mv** anche per rinominare un file, scrivendo semplicemente:

```
> mv [vecchio_nome] [nuovo_nome]
```

Usando **mv** in questo modo si otterrà di rinominare il file “vecchio_nome” in “nuovo_nome”, ovviamente nel file in se stesso non verrà modificato nemmeno un bit.

8.5 IL COMANDO **rm**

I files si possono anche cancellare utilizzando il comando **rm** (remove), la sintassi è la seguente:

```
> rm [nome_file_da_cancellare]
```

Fate attenzione perché **rm** funziona come un inceneritore, non come un cestino, quindi anche per questo comando sarebbe buona cosa utilizzare l'opzione **-i** che chiederà conferma prima di effettuare la rimozione di un file.

E' possibile cancellare anche una lista di files, basta scrivere:

```
bash:~> rm -i file1 file2 file3
```

8.6 GUARDARE DENTRO UN FILE: **cat**, **more** e **less**

Fino ad ora abbiamo giocato a “taglia e cuci” con i files, ma non avete ancora appreso gli strumenti utili per vedere cosa essi contengono.

Bisogna premettere che Linux riconosce cinque tipi fondamentali di files:

1. **Files ordinari** (generalmente di testo, contenenti caratteri alfanumerici).
2. **Files eseguibili** (contenenti codici speciali generati in seguito alla compilazione del codice sorgente di un programma e che si presentano incomprensibili quando vengono visualizzati sullo schermo).
3. **Files device** (che si riferiscono ad una periferica).
4. **Directory** (files contenitori che vedremo nel prossimo capitolo, abbiate pazienza).
5. **Link simbolici** (abbiate ancora più pazienza).

Sono due i programmi fondamentali che permettono di visualizzare il contenuto di un file: **more** e **less**.

MORE: La sintassi è la seguente:

```
> more [opzioni] [nome_file_da_visualizzare]
```

Visualizza il contenuto di un file, una schermata alla volta. Se il file è lungo dà la possibilità di leggerlo facendo scorrere in avanti le schermate, ma non dà la possibilità di tornare indietro.

Non sono presenti opzioni di rilievo.

COMANDI:

q Esce
spazio Visualizza la prossima schermata
invio Visualizza la prossima riga

LESS: La sintassi è la seguente:

> **less [opzioni] [nome_file_da_visualizzare]**

Fu sviluppato in risposta alla scarsa flessibilità di more. Visualizza il contenuto di un file e se questo è troppo lungo da la possibilità di leggerlo facendo scorrere in avanti ed indietro le schermate.

OPZIONI:

-n -N Aggiunge al bordo sinistro i numeri di righe.
-m Include la % di file letto.
-C Pulisce lo schermo prima di mostrare il file.
-M Visualizza un prompt dettagliato con N° di righe, %file letto, righe totali etc.

COMANDI:

q Esce.
spazio Visualizza la prossima schermata.
invio Visualizza la prossima riga.
freccia su / freccia giù Sposta la pagina su e giù di una riga alla volta
v Invoca l'editor "Vi" se configurato (oppure quello specificato nella variabile d'ambiente \$VISUAL o \$EDITOR), per consentire di apportare modifiche al file.

CAT: La sua sintassi è la seguente:

> **cat [opzioni] [nome_file_da_visualizzare]**

Il programma cat nonostante la sua bontà ha una pecca, se il file da visualizzare è lungo, ci scorre via davanti agli occhi senza darci la possibilità di leggere nulla.

Più che per leggere files, esso è infatti utilizzato per "concatenare" files, da qui il suo nome (conCATenate). E' infatti possibile, usando opportuni operatori, reindirizzare l'output di cat per creare un file, aggiungere in coda ad un file il contenuto di un altro file oppure combinare più files in un unico nuovo file. Vediamo alcuni esempi e prestate molta attenzione:

```
bash:~> cat > frutta
```

Contrariamente al comando **touch**, usato all'inizio per creare file vuoti, cat usato in questo modo, consente di creare anch'esso un file, ma da la possibilità di poter inserire al suo interno ciò che vogliamo. Abbiamo quindi appena creato un file dal nome "frutta" all'interno della nostra directory home, il prompt dei comandi è sparito per consentirci di inserire ciò che vogliamo all'interno del file. Scriviamo quanto segue:

```
banane  
mele  
fragole  
pesche
```

Premere la combinazione CTRL + Z per tornare al prompt ed uscire dal programma.

Andate adesso a controllare il file appena creato, con il comando:

```
bash:~> cat frutta
```

Comodo vero? Alla stessa maniera creiamo un file "verdura" sempre all'interno della nostra directory home:

```
bash:~> cat > verdura
```

Il prompt dei comandi è sparito per consentirci di inserire ciò che vogliamo all'interno del file. Scriviamo quanto segue:

```
cipolle
sedano
indivia
zucchine
```

Premere la combinazione CTRL + Z per tornare al prompt.

Andate a controllare il file appena creato, con il comando:

```
bash:~> cat verdura
```

Adesso attenzione!

Scrivendo il seguente comando:

```
bash:~> cat frutta > verdura
```

Stiamo trasferendo il contenuto del file "frutta" all'interno del file "verdura" cancellando però ciò che sta all'interno del file "verdura". Adesso abbiamo due file dal contenuto uguale ma dal nome diverso.

Provate invece a scrivere:

```
bash:~> cat frutta >> verdura
```

Stiamo trasferendo il contenuto del file "frutta" all'interno del file "verdura", combinando il contenuto dei due files, cioè il contenuto del file "frutta" è stato aggiunto in coda al contenuto del file "verdura".

Provate adesso a scrivere:

```
bash:~> cat frutta verdura > frutta_e_verdura
```

Stiamo trasferendo il contenuto del file "frutta" e del file "verdura", combinando il contenuto dei due files, all'interno di un nuovo file chiamato "frutta_e_verdura", provate a dargli un'occhiata.

OPZIONI:

-n Aggiunge al bordo sinistro i numeri di righe.

8.7 RICERCHE: I COMANDI *find* e *grep*

Esistono due tipi di ricerche possibili: quelle fatte all'interno di un filesystem alla ricerca di file o directory e quelle fatte all'interno di un file per identificare delle stringhe che contengano al loro interno un determinato lemma. Per questo si utilizzano due programmi di servizio fondamentali: **find** e **grep**.

Accade spesso di non riuscire più a trovare uno specifico file che tanto ci interessava. Prima di entrare nel panico, lasciate che sia la shell a cercarlo per voi, digitando il comando:

```
> find [directory_di_partenza] -name [nome_file_da_trovare] -print
```

Directory nella quale desiderate cercare il file. Il comando *find* ricerca in modo ricorsivo, ossia inizia dalla directory indicata ed entra, se si hanno i permessi, in tutte le eventuali sottodirectory presenti.

Indica al programma che state ricercando un file con un nome specifico.

Stampa a video il risultato.

Nome file da trovare (è possibile utilizzare anche caratteri jolly).

Esempio:

```
bash:~> find . -name file1 -print
```

In questo caso il comando **find** cerca il file indicato dopo l'opzione **-name** a partire dalla directory corrente, che in questo caso, corrisponde alla home directory dell'utente bios.

```
bash:~> find / -name file1 -print
```

In questo caso il comando **find** cerca il file indicato dopo l'opzione **-name** a partire dalla directory base “ / ”, quindi cerca praticamente nell'intero filesystem principale. Se avete eventuali partizioni montate (Windows ad esempio), provvedete prima a smontarle dal filesystem principale altrimenti la ricerca del vostro file verrà estesa anche all'interno di tali partizioni.

ATTENZIONE: Il comando **find** non permette di cercare nelle aree per le quali non disponete di autorizzazioni. Se nel corso della ricerca il comando attraversa zone protette, riceverete un messaggio del tipo “Permission denied” ogni volta che vi entra.

Adesso che sappiamo come leggere un file, vorremmo anche essere in grado di cercarvi all'interno ciò che ci interessa, per esempio vorremmo individuare tutte le righe di una nostra ipotetica relazione di Biologia in cui compare la parola “DNA”. A questo scopo ci potrebbe essere utile il comando **grep** il quale è in grado di trovare un determinato lemma all'interno del file indicato.

La sua sintassi è la seguente:

```
> grep [opzioni] [parola_da_cercare] [file_in_cui_cercare]
```



Rappresenta il lemma da cercare. Nel caso in cui la stringa sia costituita da più parole, bisogna utilizzare le virgolette.

OPZIONI:

-i case non sensitive (ignora cioè la differenza tra maiuscole e minuscole).

Esempio:

```
bash:~> grep DNA Desktop/Relazione_Biologia_Molecolare
```

Stiamo cercando il lemma “DNA” all'interno del file “Relazione_Biologia_Molecolare” che si trova sul Desktop.

```
bash:~> grep “DNA ricombinante” Desktop/Relazione_Biologia_Molecolare
```

Stiamo adesso cercando la stringa “DNA ricombinante” all'interno dello stesso file (notate l'uso delle virgolette quando si ricercano più parole).

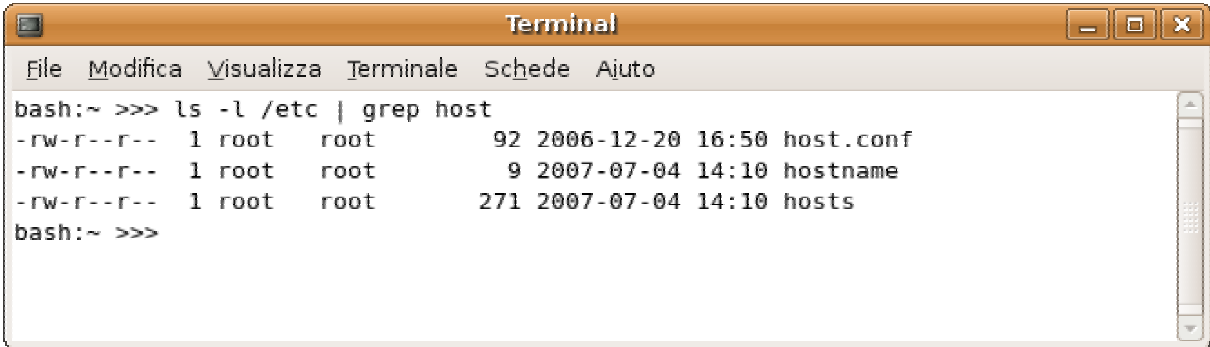
Scriviamo adesso:

```
bash:~> grep cielo Divina_commedia.txt | less
```

Utilizzando questo comando complesso chiediamo a **grep** di ricercare la parola “cielo” all'interno del file “Divina_Commedia.txt” e di voler leggere tutti i versetti che la contengono. Il risultato è il seguente:

```
per quel ch'io di lui nel cielo udito.  
curan di te ne la corte del cielo,  
Non isperate mai veder lo cielo:  
... etc etc
```

Il comando **grep** può anche essere utilizzato per filtrare l'informazione quando, ad esempio, l'output di **ls -l** risulta essere troppo esteso. La **Fig. 8.1** in basso mostra come **grep** riesce a filtrare l'output che ci interessa concatenandosi al comando **ls -l**, lanciato sulla directory */etc*, solitamente piena di file.



```
Terminal
File Modifica Visualizza Terminale Schede Aiuto
bash:~ >>> ls -l /etc | grep host
-rw-r--r-- 1 root root 92 2006-12-20 16:50 host.conf
-rw-r--r-- 1 root root 9 2007-07-04 14:10 hostname
-rw-r--r-- 1 root root 271 2007-07-04 14:10 hosts
bash:~ >>>
```

Fig. 8.1 Esecuzione di un comando complesso che utilizza **grep** per filtrare l'informazione che ci interessa.

8.8 I CARATTERI JOLLY

I caratteri jolly o metacaratteri, sono quei simboli utilizzati per fare facilmente riferimento a gruppi di file o di directory. Essi sono utili quando desiderate lavorare con diversi files dal nome simile o con files di cui non riuscite a ricordare il nome esatto. Nei sistemi Unix è la shell ad occuparsi della traduzione dei caratteri jolly.

I caratteri jolly più usati sono due:

- ? Sostituisce un singolo carattere.
- * Sostituisce un numero imprecisato di caratteri.

Ma vediamo all'opera facendo alcuni esempi ed inserendoli all'interno di comandi già noti:

```
bash:~> ls -l Desktop/???
```

Chiediamo al comando **ls** di avere la lista di tutti i files i cui nomi sono di tre lettere all'interno della directory Desktop.

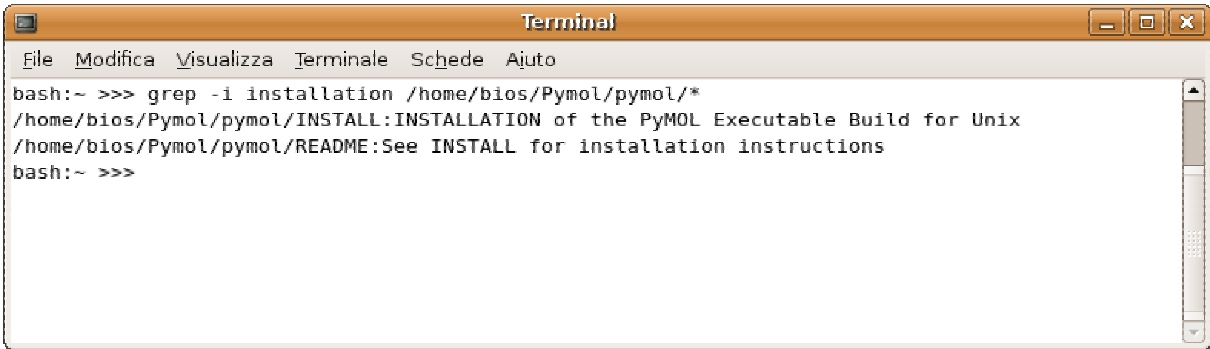
```
bash:~> mv *1996 Desktop/Budget
```

Chiediamo al comando **mv** di spostare tutti i files i cui nomi finiscono con 1996 nella directory Budget che sta nel Desktop.

```
bash:~> mv /home/Desktop/Dati/* .
```

Chiediamo al comando **mv** di spostare tutti i files presenti all'interno della directory Dati (che sta sul Desktop), nella nostra directory di lavoro corrente (in questo caso la nostra home).

Guardiamo adesso la **Fig. 8.2** in basso che mostra il comando **grep** all'opera:



```
Terminal
File Modifica Visualizza Terminale Schede Aiuto
bash:~ >>> grep -i installation /home/bios/Pymol/pymol/*
/home/bios/Pymol/pymol/INSTALL:INSTALLATION of the PyMOL Executable Build for Unix
/home/bios/Pymol/pymol/README:See INSTALL for installation instructions
bash:~ >>>
```

Fig. 8.2 Esecuzione del comando **grep -i**, dando come argomento un percorso contenente un carattere jolly.

L'asterisco evidenzia il fatto che desideriamo cercare all'interno di tutti i files presenti nella directory */home/bios/pymol*, la parola "installation". L'output mostrato evidenzia il fatto che sono stati trovati due files che contengono al loro interno la parola "installation".

Esiste un altro carattere jolly abbastanza importante, ma meno utilizzato:

[set] Sostituisce esattamente un carattere appartenente al gruppo di caratteri indicati fra parentesi quadre, rappresentato dalla stringa "set":

```
bash:~> ls -l file_di_prova[1-9]
```

verranno visualizzati i files:

```
file_di_prova1
file_di_prova2
file_di_prova3
.
.
.
file_di_prova9
```

8.9 I COMANDI **cmp** E **diff**

Vi sarete resi conto che le operazioni con i file sono pressoché infinite, continuiamo il capitolo facendo attenzione ad un'altra caratteristica che la shell ci mette a disposizione, il confronto tra file. I comandi più usati sono praticamente due: **cmp** e **diff**.

La sintassi del comando **cmp** è la seguente:

```
> cmp [opzioni] file1 file2
```

Il comando confronta "file1" e "file2" e non dice nulla se i due file sono uguali, altrimenti avvisa l'utente che è stata trovata almeno una differenza.

L'unica OPZIONE di rilievo è **-s**, che fa lavorare il programma in silenzio.

La seguente stringa mostra una personalizzazione di tale comando:

```
> cmp -s file1 file2 && echo 'i file sono identici'
```

Questo è un esempio di "comando complesso" ossia fatto da più istruzioni legate insieme, la shell interpreta tale comando nel seguente modo:

- Confronta i due file
- Esegue il secondo comando (echo) solo se il primo ha successo, quindi se i due file sono uguali stampa a video il messaggio "i file sono identici".

Più complesso rispetto al precedente per via delle sue molteplici opzioni, il comando **diff** presenta la seguente sintassi generale:

```
> diff [opzioni] file1 file2
```

Il comando confronta due file e restituisce le righe differenti indicando anche il contesto di ognuno dei due file, con il testo di file1 contrassegnato da un simbolo "<" e quello di file2 da un simbolo ">".

OPZIONI:

- a tratta tutti i file come file di testo, utile per verificare se file binari sono identici.
- b ignora le sequenze di caratteri "blank" e "end of line".
- B ignora le righe vuote nei file.
- i case non sensitive.
- w ignora tutti i caratteri "white space" contenuti nei file confrontati.

-y produce un output su due colonne.

8.10 IL COMANDO **sort**

Ordina il contenuto dei file riga per riga prendendo in considerazione il primo carattere di ogni riga. La sua sintassi è:

> **sort [opzioni] file_in -o file_out**

↑
Nome file da ordinare.

↙
Nome del file nel quale verranno messi i risultati dell'ordinamento.

Esso presenta alcune opzioni di rilievo

OPZIONI:

- f ordina senza priorità del maiuscolo sul minuscolo
- r ordina inversamente
- n ordina i numeri

8.11 IL COMANDO **wc**

La sua sintassi è la seguente:

> **wc [opzioni] file**

Stampa informazioni sul numero di caratteri, parole, righe di un file.

OPZIONI:

- c stampa solamente il conteggio caratteri.
- l stampa solamente il conteggio righe.
- w stampa solamente il conteggio parole.

8.12 IL COMANDO **whereis**, DOVE E' FINITO QUEL PROGRAMMA ?

Molte volte capita di cercare in che directory è contenuto un nostro file eseguibile. Il comando che andremo a vedere consente di trovare binari, sorgenti e manuali per un determinato file desiderato.

Il comando **whereis** indica l'ubicazione di un file tentando di localizzarlo nelle directory standard:

/bin /etc /usr/bin /usr/local/bin

la sua sintassi è la seguente:

> **whereis [opzioni] nome_file**

OPZIONI:

- b ricerca solamente file binari.
- m ricerca solamente le sessioni di manuale.
- s ricerca solo i sorgenti.