Linux e terminali

Mattia Graziani, Alice Benatti

04 novembre 2025

>ADM staff

Laboratori fra pari

Perché usare una shell?

I primi computer utilizzavano i terminali, perché dovremmo farlo anche noi quando abbiamo un'ottima interfaccia grafica con delle belle icone?

- Completo controllo di quello che state facendo. Moltissime cose non si possono fare con una GUI
- Maggiore velocità di una GUI
- Operazioni ripetitive automatizzate tramite script
- Accesso a server e configurazione servizi

Aprire un terminale

Per utilizzare una `shell` è necessario disporre di un **emulatore di terminale**.

Per comunicare con la `shell` attraverso l'emulatore di terminale abbiamo bisogno di usare i **comandi**.

Comandi

Un comando rappresenta una richiesta di eseguire una determinata operazione al sistema operativo.

Sintassi: `comando [opzioni] [argomenti]`

```
[mattia@pirandello ~]$ date
Thu Nov 4 05:00:00 PM CEST 2025
[mattia@pirandello ~]$
```

Stampa data, ora e qualche altra informazione come il fuso orario.

Elencare i file

Le operazioni più importanti sono legate alla gestione dei file. Proviamo a digitare il comando `ls` nella shell.

```
[mattia@pirandello ~]$ ls
Desktop Documents Downloads Music Pictures
Public  Templates Video
[mattia@pirandello ~]$
```

Sono gli stessi che vediamo nell'interfaccia

Entrare in una cartella

Per muoverci tra le cartelle con la `shell` possiamo usare il comando `cd`.

Per entrare in una cartella è necessario indicare in quale ci vogliamo muovere visto che potrebbe essercene più di una.

Per farlo è quindi necessario fornire al comando `cd` un **argomento**.

```
[mattia@pirandello ~]$ cd Documents/
[mattia@pirandello Documents]$
```

Uscire da una cartella

Per uscire da una cartella il comando è `cd ..`

```
[mattia@pirandello Documents]$ cd ..
[mattia@pirandello ~]$
```

L'argomento `..` indica sempre la cartella genitore di quella attuale.

Current working directory

In Linux le cartelle si chiamano *directory*. Per stampare il *path* della cartella corrente usiamo il comando: `pwd`.

Il path assume la seguente forma: `/home/mattia/`.

Il carattere / viene utilizzato come separatore, quindi `mattia` è dentro la cartella `home`.

Prima di `home` c'è uno / che in Linux indica la radice del **file-system**. Quindi `home` a sua volta è contenuta in /

Cos'è un file system

Il termine file system assume vari significati

- L'insieme dei file e delle directory che sono accessibili ad una macchina Linux
- L'organizzazione logica utilizzata da un s.o. per gestire un insieme di file in memoria secondaria
- Il termine viene anche utilizzato per indicare una singola unità di memoria secondaria

I file-system di Linux sono gerarchici, cioè organizzati ad albero. La radice è / e directory, sotto-directory e file sono i nodi dell'albero.

I *path* sono unici, non possono esserci quindi file diversi con lo stesso *path*.

Ne consegue che se due file hanno lo stesso *path* sono lo stesso file.

Cartelle di sistema

```
[mattia@pirandello ~]$ cd /
[mattia@pirandello /]$ ls
bin  dev  home  mnt  opt  run   tmp  var   boot  root
[mattia@pirandello /]$ cd
[mattia@pirandello ~]$ cd Documents/prove/
```

Figure: Lista dei file presenti in /

- `/bin`: contiene programmi necessari al sistema per funzionare
- `/boot`: contiene il kernel e altri file necessari al sistema per partire.
- `/etc`: contiene tutti i file di configurazione del sistema.
- `/home`: contiene le cartelle riservate agli utenti
- `/tmp`: contiene file temporanei che vengono cancellati ad ogni spegnimento

Path relativi e assoluti

Per identificare un file è possibile usar due tipi di *path*: assoluto e relativo:

1. Un *path* assoluto è un percorso ad un file che inizia da `/` e termina con il nome di quel file. prende quindi le seguenti forme:
   - `/home/mattia/slides.tex`
   - `/usr/bin/firefox`
   - `/tmp`
2. Un *path* relativo è il percorso necessario per raggiungere un file rispetto alla **cartella corrente**. prende quindi le seguenti forme:
   - `slides.tex`
   - `../`
   - `immagini/greg.png`

Creare directory

Per creare una directory esiste il comando `mkdir`.

```
[matti@pirandello ~]$ cd Documents/
[matti@pirandello Documents]$ mkdir prove
[matti@pirandello prove]$ cd prove/
[matti@pirandello prove]$ mkdir castoro
[matti@pirandello prove]$ ls
castoro
[matti@pirandello prove]$
```

Figure: esempi del comando mkdir

Sintassi: `mkdir path/to/directory`

Creare file

Per creare un file esistono molti modi, ma il più semplice è il comando `touch`.

```
[matti@pirandello prove]$ touch lontra
[matti@pirandello prove]$ ls
castoro lontra
[matti@pirandello prove]$
```

Figure: esempi del comando touch

Sintassi: `touch path/to/file`

Il comando `touch` in realtà serve per cambiare la data di modifica di un file, ma se non esiste allora viene creato.

A chi serve Nautilus?

Per sapere di che tipo è un file possiamo usare il comando `file`:

```
[mattia@pirandello prove]$ ls
castoro lontra
[mattia@pirandello prove]$ file castoro
castoro: directory
[mattia@pirandello ~]$ file lontra
lontra: empty
// Se lontra contenesse del testo
// lontra: ASCII text
[mattia@pirandello ~]$
```

Figure: esempi del comando file

In linux i file non hanno bisogno di un'estensione, è quindi molto utile questo comando.

Copia, incolla e interruzione

Nel terminale non funziona il classico copia e incolla da tastiera eseguito con `ctrl + c` e `ctrl + v`. Queste combinazioni di tasti hanno il loro scopo e non sono fatti per copiare.

Per copiare e incollare dovete usare `shift + ctrl + c` e `shift + ctrl + v`.

`ctrl + c` serve per interrompere un processo in esecuzione. Ma se volessimo copiare un intero file? Non è aprirlo, selezionare tutto e copiarlo altrove...

Copiare file

Il comando per copiare dei file è `cp`. il suo utilizzo è principalmente: `cp file/da/copiare destinazione`

```
[mattia@pirandello prove]$ ls
castoro lontra
[mattia@pirandello prove]$ cp lontra criceto
[mattia@pirandello prove]$ ls
castoro criceto lontra
[mattia@pirandello prove]$
```

Figure: esempi del comando cp

Sono ammessi sia *path* assoluti sia relativi

Spostare file

Il comando per spostare (tagliare) dei file è `mv`. Si usa come il comando di copia: `mv file/da/muovere destinazione`

```
[mattia@pirandello prove]$ ls
castoro criceto lontra
[mattia@pirandello prove]$ cd castoro/
[mattia@pirandello castoro]$ mv ../lontra .
[mattia@pirandello castoro]$ ls
lontra
[mattia@pirandello castoro]$
```

Figure: esempi del comando mv

La destinazione `.` indica la directory corrente.

Rinominare file

Il comando `mv` permette anche di rinominare i file:

```
[mattia@pirandello castoro]$ touch cubo
[mattia@pirandello castoro]$ ls
cubo lontra
[mattia@pirandello castoro]$ mv cubo triangolo
[mattia@pirandello castoro]$ ls
lontra triangolo
[mattia@pirandello castoro]$
```

Figure: esempi del comando mv

Now I am become death, the destroyer of files

Il comando più pericoloso in linux è indubbiamente `rm`.

`rm path/to/file` elimina il file passato come argomento.

Non si può tornare indietro, una volta eliminato un file è perso per sempre!

`rm` non funziona come il cestino di Windows con cui puoi ripristinare file, la *shell* si aspetta che voi sappiate esattamente quello che state facendo e non si preoccupa se questo può distruggere il sistema.

Copiare una cartella

Per copiare le directory il comando `cp` deve funzionare in modalità *ricorsiva*, per permettere la copia di tutti gli elementi all'intero della directory.

Per copiare una cartella è quindi necessario aggiungere la *flag* `-r`

```
[mattia@pirandello prove]$ ls
castoro criceto
[mattia@pirandello prove]$ cp castoro capybara
cp: -r not not specified; omitting directory 'castoro'
castoro criceto
[mattia@pirandello prove]$ cp -r castoro capybara
[mattia@pirandello prove]$ ls
capybara castoro criceto
```

Figure: esempi del comando cp ricorsivo

Flags

Le *flag* sono un modo per estendere le funzionalità di un comando.

Vengono specificate dopo il comando e sono precedute da un trattino –

Per comodità sono di una sola lettera, ma in certi casi possono essere anche più verbose es. `--recursive`.

Si possono combinare più *flag* concatenando le lettere dopo il trattino: `-r -t` è equivalente a `-rt`

Man

Esiste un comando per leggere il *manuale* di un comando: `man`.

Potreste trovarvi in situazioni in cui non saprete come usare un comando, non avrete accesso a internet per cercare o non avrete interfacce grafiche... Ii il comando man vi tornerà molto utile.

Sintassi: `man comando`

Si apre un visualizzatore di testo integrato nel terminale chiamato `less`. Per navigare sono usati i seguenti comandi:

- `j`: Muoversi verso il basso
- `k`: Muoversi verso l'alto
- `g`: Inizio del file
- `/name`: Cerca la stringa `name`
- `n`: Va all'occorrenza successiva
- `q`: Esce e torna al prompt

Less

Esiste un lettore di testo integrato nella *shell* chiamato `less`. (evoluzione di `more` presente nei primi s.o. Unix)

`less` è in realtà un comando che permette di leggere file di testo.

Sintassi: `less path/to/file`

`less` è molto veloce a leggere file di testo di grandi dimensioni.

Nano e Vim

Su Linux esistono due principali editor di testo: nano e vim.

Nano è un editor semplice che ha i comandi scritti a schermo per evitare di scordarseli.

La filosofia di vim invece è diversa. Facciamo qualche osservazione:

- Quando si programma la maggior parte del tempo è passato a *modificare* il codice, non a scriverlo
- Modificare il codice include molto altro oltre a scrivere: eliminare, sostituire, riordinare, duplicare, formattare, ecc.
- Ha senso facilitare tutta la parte di modifica del codice, più che di scrittura effettiva
- Il mouse è una perdita di tempo quando si deve scrivere, se si può fare tutto da tastiera in modo efficiente è meglio
- Un'operazione usa il minor numero di tasti possibili.

Approfondiremo vim più avanti, per il momento usiamo nano.

Esempio di nano

```
[mattia@pirandello castoro]$ ls
lontra triangolo
[mattia@pirandello castoro]$ nano lontra
```

```
  GNU nano 4.8                      lontra                      Modified

  Hello World!
~
~
~
~
~
~
^G Help      ^O Write Out    ^W Where Is     ^K Cut      ^T Execute
^X Exit      ^R Read File    ^\ Replace      ^U Text     ^J Justify
```

Figure: Scrivere "Hello World!" nel file `lontra` usando nano.

Scrivere in un file di testo

Per scrivere in un file di testo esiste il comando `nano`.

Sintassi: `nano path/to/file`

Una volta dentro l'editor si può scrivere normalmente. Per salvare e uscire si usa la combinazione di tasti `ctrl + x`, poi `y` per confermare il salvataggio e infine `invio` per salvare con lo stesso nome.

Per uscire senza salvare si usa sempre `ctrl + x` ma poi si preme `n` per non salvare.

Contare caratteri, linee, ecc. - wc

Esiste un comando per contare i caratteri, le linee e altre informazioni all'interno di un file di testo: `wc`

Sintassi: `wc path/to/file`

Senza nessuna *flag* stampa:

- Il numero di righe
- Il numero di parole
- Il numero di bytes

Per stampare solo il numero di byte: `wc -c path/to/file`.
Per stampare il numero di linee: `wc -l path/to/file`.

Ricerca di una stringa - grep

Uno dei comandi più potenti per la ricerca di stringhe in un file è `grep`

Sintassi: `grep "string" file`

Esiste la *flag* `-i` per la ricerca case-insensitive.

```
[mattia@pirandello castoro]$ grep -i "hello" lontra
Hello World!
[mattia@pirandello castoro]$
```

Figure: Ricerca di una stringa in un file

Concatenazione o lettura? - cat

Il comando `cat` è nato per concatenare più file.

Per renderlo però completamente funzionante abbiamo bisogno dell'operatore di ridirezione che verrà spiegato più avanti.

Possiamo però usarlo per leggere file generalmente corti. Spesso è più rapido da usare di `less`, anche perchè stampa il file completo sul terminale.

```
[mattia@pirandello castoro]$ cat lontra
Hello World!
[mattia@pirandello castoro]$
```

Figure: Stampa di un file tramite cat

Testa e coda - head, tail

Il comando `cat` prende uno o più file in input e li stampa sul terminale.

Esiste anche `head` che stampa solo le prime 10 righe di un file (si possono modificare con la *flag* `-n numero`).

Da notare che la *flag* precedente ha preso un parametro.

Esiste anche `tail` che fa esattamente la stessa cosa di `head`, ma partendo dalla fine del file.

Complementi di comandi base

- Per pulire il terminale esiste il comando `clear`

- Per resettare il terminale esiste il comando `reset`. Sarà molto utile quando lavorerete al progetto di programmazione e romperete tutto con la libreria grafica.

- Per vedere i vecchi comandi eseguiti esite il comando `history`. Digitando solamente history vedrete gli ultimi comandi usati, idendificati dal numero del comando.

- Per riprendere un comando eseguito di recente basta utilizzare la freccetta verso l'alto.

Wildcard

Nella *shell* l'asterisco `*` fa da "segnaposto" per una qualsiasi altra sequenza di caratteri.

Esempio: `torr*` si espande in: `torr`, `torra`, `torrb`, `···`, `torraa`, `torrab`, `torrac`. Valgono ovviamente anche i numeri e altri caratteri oltre alle lettere.

Questo permette di indicare più file con parti comuni nel nome. Si possono combinare anche più asterischi: `c*a*` fa match con tutte le parole che iniziano per `c` e hanno almeno una `a` nel nome. `*pila*` fa match con tutte le parole che hanno `pila` nel nome.

Wildcard esempi

```
[mattia@pirandello castoro]$ cd ..
[mattia@pirandello prove]$ ls
capybara  castoro  criceto
```

Proviamo a stampare solo le cartelle che iniziano con `ca`:

```
[mattia@pirandello prove]$ ls -d ca*
capybara  castoro
```

La flag `-d` del comando `ls` permette di elencare solo le directory

Perché dobbiamo metterla se vogliamo elencare solo le cartelle? Cosa stamperebbe altrimenti?

Sistema di permessi

Linux è un sistema multi-utente, quindi più utenti posso usare simultaneamente la stessa macchina.

Per questo è necessario avere un sistema di permessi adeguato.

Il sistema di permessi utilizzato da Linux non è semplice, e di seguito sarà data soltanto un'introduzione.

Root

In tutti i sistemi Linux esiste un unico utente che ha i permessi per seguire qualsiasi operazione: **root**

È l'amministratore del sistema.

NON deve essere MAI usato come utente se non per le operazioni strettamente necessarie.

Per aprire una *shell* come utente `root` è possibile digitare il comando: `su`

Il comando chiederà quindi la password di `root` (che generalmente è impostata durante l'installazione del sistema) e se corretta aprirà una *shell* con i privilegi di amministratore.

Permessi su un file

Torniamo ad usare il nostro utente (se siamo root facciamo `exit`). Dentro la cartella `/etc` esiste un file chiamato `shadow`. Se proviamo a leggerlo con `less` otteniamo:
/etc/shadow: Permission denied

Questo significa che il nostro utente non ha i permessi per leggere il file. Ma come potevamo saperlo a priori senza tentare di leggerlo?

Una *flag* molto usata per il comando `ls` è `-l`. (o `-al`) che permettono di vedere molte più informazioni sui file presenti in una directory.

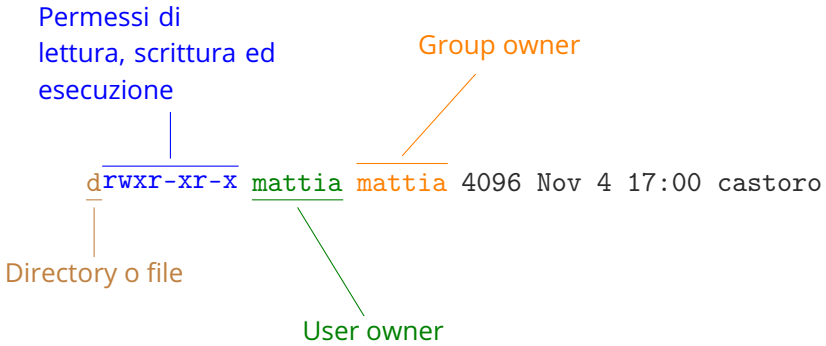
Cambiare owner e group

Per cambiare l'owner di un file e il gruppo possiamo usare il comando `chown`.

Sintassi: `chown user:group file`

Per eseguire il comando sono necessari i privilegi di root.

Esempio di ls -l



Read, write, execute

Ogni file ha un stringa formata da **9 bit** che determina quali permessi specifici ha quel file rispetto all'*utente*, il *gruppo* e gli *altri*.

I 9 bit sono suddivisi in **gruppi di 3**: il primo è specifico per l'*utente*, il secondo è specifico per il *gruppo* e i rimanenti sono per tutti gli *altri*.

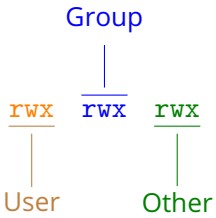


Tabella con i permessi

	File	Directory
r	Permette la lettura di un file	Permette di vedere il contenuto se anche `x` è segnato
w	Permette di scrivere sul file	Permette di creare ed eliminare file dentro la directory se `x` è segnato
x	Permette di eseguire un file	Permette di entrare nella directory

Table: Significato dei permessi per file e directory

Cambiare permessi wrx

Per cambiare i permessi lettura, scrittura ed esecuzione si utilizza il comando `chmod`.

Sintassi: `chmod [PART][ACTION][PERMISSION] file`

Al posto di `[PART]` è necessario specificare la parte che si vuole modificare:

- user: si utilizza `u`
- group: si utilizza `g`
- others: si utilizza `o`
- all: si utilizza `a`

Per eseguire il comando sono necessari i privilegi di root.

Cambiare permessi wrx

Per cambiare i permessi lettura, scrittura ed esecuzione si utilizza il comando `chmod`.

Sintassi: `chmod [PART][ACTION][PERMISSION] file`

Al posto di `[ACTION]` è necessario specificare la l'azione da compiere

- +: Aggiunge il permesso
- –: Rimuove il permesso
- =: Assegna esattamente quel permesso

Per eseguire il comando sono necessari i privilegi di root.

Cambiare permessi wrx

Per cambiare i permessi lettura, scrittura ed esecuzione si utilizza il comando `chmod`.

Sintassi: `chmod [PART][ACTION][PERMISSION] file`

Al posto di `[PERMISSION]` è necessario specificare la il permesso o i permessi da moficare:

- `r`: Read
- `w`: Write
- `x`: Execute

Per eseguire il comando sono necessari i privilegi di root.

Esempi `chmod`

- `chmod u+x pippo` Rende pippo un file eseguibile per l'utente
- `chmod o-w pippo` Rimuove la possibilità a tutti gli utenti diversi dall'owner del file e non appartenenti al gruppo del file di scrivere su pippo.
- `chmod g+r pippo` Rende pippo leggibile al gruppo
- `chmod g+x pippo` Rende pippo eseguibile dal gruppo
- `chmod u=rwx,g=,o= pippo` Rende pippo leggibile, scrivibile ed eseguibile per l'utente. Inoltre rimuove tutti i permessi dal gruppo e altri.

Esempi `chmod` con bit

- `chmod +100 pippo` Rende pippo un file eseguibile per l'utente
- `chmod -002 pippo` Rimuove la possibilità a tutti gli utenti diversi dall'owner del file e non appartenenti al gruppo del file di scrivere su pippo.
- `chmod +040 pippo` Rende pippo leggibile al gruppo
- `chmod +010 pippo` Rende pippo eseguibile dal gruppo
- `chmod 700 pippo` Rende pippo leggibile, scrivibile ed eseguibile per l'utente. Inoltre rimuove tutti i permessi dal gruppo e altri.
- `chmod 777 pippo` Rende pippo leggibile, scrivibile ed eseguibile per tutti.
- `chmod 644 pippo` Rende pippo leggibile e scrivibile per l'utente, leggibile per il gruppo e gli altri.
- `chmod 600 pippo` Rende pippo leggibile e scrivibile solo per l'utente.

Il potere della shell

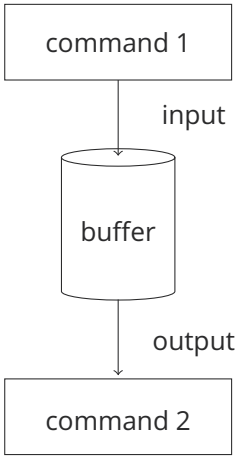
Finora abbiamo visto soltanto comandi base e usati singolarmente. Spesso sono utili usati così come abbiamo visto, ma diventano molto potenti se usati in combinazione con altri comandi.

Come si fa però a combinare più comandi?

Ricordiamoci che:

- Un comando restituisce sempre qualcosa sullo standard output
- Lo standard output è considerato come un file dal sistema
- La maggior parte dei comandi visti fino ad adesso hanno la possibilità di prendere in input lo standard output invece che un file classico

Cosa possiamo fare dalla shell



Pipe

Per prendere l'output di un comando e riderizionarlo in input verso un altro comando si usa la **pipe** |

Per esempio se vogliamo vedere tutti i file presenti in `/bin` il nostro terminale si riempie di scritte.

Possiamo visualizzare il lungo output con il comando `less`: `ls /bin | less`

Pipe - grep

Come abbiamo visto la lista di file presenti in `/bin` è molto lunga. Se volessimo trovarne uno specifico?

Nonostante esiste un comando apposito per cercare file, con le conoscenze che abbiamo al momento possiamo costruire una soluzione alternativa:

```
[mattia@pirandello prove]$ ls /bin | grep "firefox"
firefox
[mattia@pirandello prove]$
```

dove al posto di *firefox* può andarci una qualsiasi stringa.

Pipe - esempi

Di seguito una lista di esempi di utilizzo della `pipe`:

- `cat file1 file2 | grep "word"` cerca una stringa in più file
- `ls /bin | wc -l` conta quanti programmi sono presenti in `/bin`
- `ls /bin | grep "zip" | wc -l` conta quanti programmi hanno la stringa "*zip*" al loro interno nella cartella `/bin`
- `grep "castoro" animali | wc -l` conta le occorrenze di `castoro` trovate nel file `animali`
- `grep "the" book | less` mostra le occorrenze di `the` trovate in `book` attraverso il lettore `less`

Ridirezione su file

Visto che è possibile mandare l'output di un comando nell'input di un altro comando, come possiamo salvare l'output di un comando su un file?
Esiste l'**operatore di ridirezione** >

Sintassi: `comando > file`

ATTENZIONE: Alla *shell* non interessa se il file esiste già, quindi se esiste lo SOVRASCRIVE COMPLETAMENTE.

Esempi ridirezione distruttiva su file

```
[mattia@pirandello prove]$ ls
capybara  castoro  criceto
[mattia@pirandello prove]$ ls > lista
[mattia@pirandello prove]$ ls
capybara  castoro  criceto lista
[mattia@pirandello prove]$ cat lista
capybara  castoro  criceto lista
[mattia@pirandello prove]$ echo "Hello␣World" > lista
[mattia@pirandello prove]$ cat lista
// cosa contiene ora lista?
```

Ridirezione su file non distruttiva

Esiste anche un operatore per indirizzare su file l'output di un comando senza sovrascrivere il contenuto del file, ma "appendendo" alla fine del file il contenuto scritto.

Sintassi: `comando » file`

Si usa nello stesso modo dell'operatore classico

Esempi ridirezione non distruttiva su file

```
[mattia@pirandello prove]$ ls
capybara  castoro  criceto lista
[mattia@pirandello prove]$ cat lista
Hello World
[mattia@pirandello prove]$ ls | wc -l >> lista
[mattia@pirandello prove]$ cat lista
// cosa contiene ora lista?
```

More work to do

I comandi presentati sono soltanto una piccolissima parte dell'infinità di comandi presenti in una sistema Linux.

Non possiamo ovviamente includerli tutti, ma di seguito lasceremo alcuni comandi che potete approfondire:

- `ssh`
- `sudo`
- `apt`, `yum` e `pacman` (Dipende dalla distribuzione Linux)
- `top` e `htop`
- `kill`
- `touch`, `locate` e `find`
- `nano` e `vim`