

## Uso basilare di Shell Linux

# Obiettivi

- ▶ Insegnare le basi della shell di Linux e i comandi principali
- ▶ Concetti fondamentali per eseguire programmi da shell saranno utilizzati anche in sede d'esame

# Cos'è la shell?

Permette l'interazione tra l'utente e il sistema operativo.

- ▶ GUI (Graphic User Interface) - interfaccia grafica
- ▶ CLI (Command Line Interface) - interfaccia testuale (riga di comando)

In alcuni sistemi operativi, ad es. Windows, l'interfaccia grafica è una **componente integrata**.

In alcuni sistemi operativi, ad es. Linux, l'interfaccia grafica è una **componente opzionale** (e ce ne sono diverse).

- ▶ KDE
- ▶ GNOME
- ▶ XFCE
- ▶ ...

L'interfaccia testuale è l'unica disponibile per alcuni programmi

# Shell Linux

In ambiente Linux, con **shell** si indica la CLI usata, mentre si usa **Desktop Environment** per indicare la GUI.

La shell più comune in ambiente Linux si chiama **Bash** (ma ne esistono molte altre).

- ▶ sh
- ▶ zsh
- ▶ csh

Bash fornisce accesso a programmi attraverso la CLI (incluso comandi basilari per la gestione del filesystem e go tools)

Include un linguaggio di programmazione “semplificato” - per automatizzare operazioni sul OS, ma non per fare codice efficiente, veloce, portabile

## Shell - Come funziona?

La shell tipicamente mostra un *messaggio* (**prompt**) seguito da un cursore che indica dove è possibile inserire un comando.

Il prompt può essere un semplice simbolo come \$, oppure può mostrare informazioni aggiuntive, come ad esempio il nome dell'utente connesso e il nome del PC.

Chiameremo il testo inserito per invocare un comando: **riga di comando**.

Negli esempi useremo \$ per indicare il prompt, segnalando che ciò che segue è la riga di comando.

# Perché usare la shell?

*La GUI permette una interazione più semplice e immediata rispetto alla CLI. Perché dovrei utilizzare la CLI?*

Lista (non esaustiva) dei possibili motivi per cui utilizzare la shell:

- ▶ mancanza di risorse per eseguire una GUI;
- ▶ ambiente in cui la GUI non è necessaria;
- ▶ possibilità di scrivere una sequenza di comandi all'interno di un singolo file (*script*) per eseguirli ripetutamente e in modo automatico;
- ▶ alcune operazioni risultano essere più complesse (o impossibili) se fatte da GUI.

# Struttura di un comando

Una riga di comando in bash (e la maggior parte di shell) è composta da:

`nome_comando [argomenti]`

Gli argomenti sono input inseriti da riga di comando

La loro interpretazione è definita dallo stesso programma

In genere la semantica in ambiente linux è la seguente:

- ▶ opzioni: iniziano per `-`(carattere) (es `-v`) o `--`(parola) (es: `--verbose`)
- ▶ parametri: seguono opzioni e ne definiscono il valore (es: `-f input.txt` o `--file=input.txt`)
- ▶ sottocomandi: eseguono una specifica funzionalità del comando (es: `go build file_esempio.go`)

# Completamento automatico comandi

Spesso non è necessario scrivere per esteso il nome di un file ma basta

1. iniziare a scriverne il nome
  2. premere il tasto di tabulazione
- ▶ Se i caratteri scritti individuano uno e un solo file nella directory corrente, il suo nome viene automaticamente completato dalla shell.
  - ▶ Altrimenti premendo una seconda volta il tasto di tabulazione si otterrà un elenco dei file compatibili con i caratteri specificati.

Il completamento automatico funziona anche con i nomi di comandi della shell.



## Storico dei comandi

Premendo i tasti *freccia su* e *freccia giù* è possibile scorrere lo storico dei comandi utilizzati nella shell.

# File system

Il file system è la componente del sistema operativo preposta alla gestione delle informazioni memorizzate permanentemente, che risiedono tipicamente su periferiche di memorizzazione (disco, chiavette di memoria, ...).

Le componenti principali di un file system sono:

- ▶ i file: memorizzano una serie di informazioni aventi unità logica, ad esempio un file audio, un'immagine, un'applicazione. . .
- ▶ le directory (o cartella, o folder): contenitori che possono includere al suo interno file o altre directory.

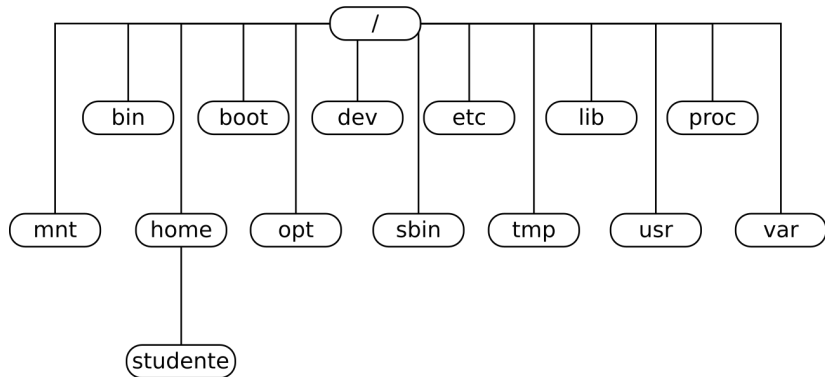
In linux anche l'interazione con le periferiche avviene attraverso file!

# Struttura del file system

Il file system è quindi una struttura gerarchica (ad albero) nella quale:

- ▶ esiste una directory principale (detta radice, o *root*) cui tutta la struttura fa capo;
- ▶ le altre directory rappresentano nodi intermedi dell'albero;
- ▶ i file sono le foglie dell'albero.

## Struttura ad albero - Esempio



## Path assoluti

Ogni file/directory all'interno del file system è individuato/a in base alla sua posizione nell'albero, cioè al percorso (**path**) che si deve percorrere per raggiungerlo/a partendo dalla radice (path **assoluto**).

Il path assoluto che porta ad un file/directory parte dalla root directory (/)

## Path relativi

Per accedere a file o directory si possono utilizzare anche path **relativi**: che portano ad un file/directory partendo dalla directory nella quale ci si trova.

Per questo esistono due *directory* speciali:

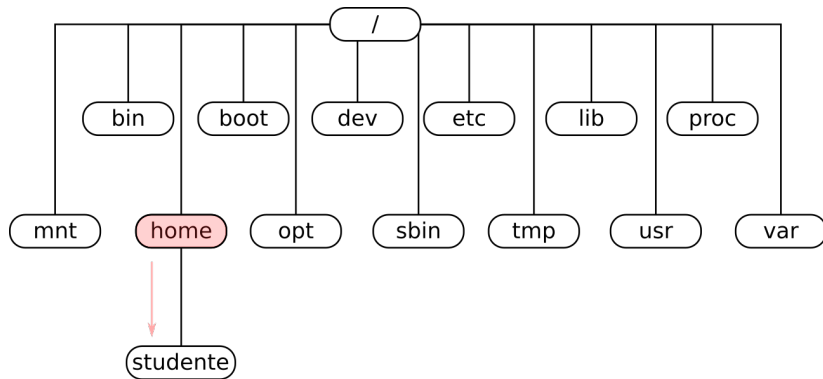
- ▶ `.` che indica la directory corrente (in genere omissa);
- ▶ `..` che indica la directory parent che contiene la directory corrente.

# Comandi per la navigazione delle directory

Apriamo una shell e proviamo i seguenti comandi:

- ▶ `pwd`: visualizza il path della directory corrente;
- ▶ `ls`: visualizza il contenuto di una directory;
- ▶ `cd <nome directory>`: apre una directory avente per nome `<nome directory>`.

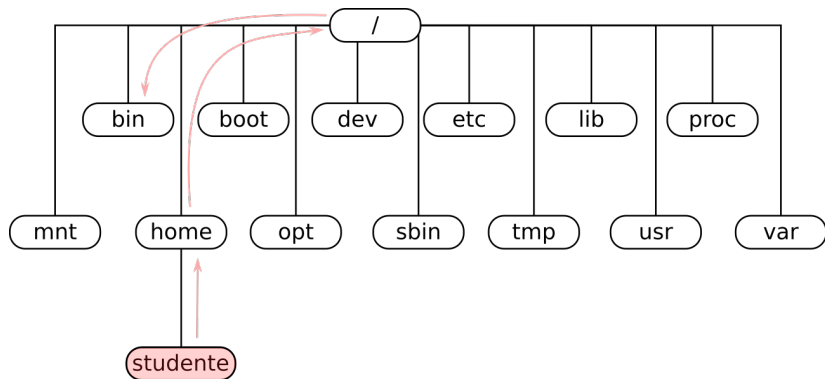
## Path assoluti e relativi - Esempio 1



- ▶ accesso con path assoluto `$ cd /home/studente`
- ▶ con path relativo: `$ cd studente`
- ▶ con directory speciale: `$ cd ~`, o semplicemente `$ cd`



## path assoluti e relativi - Esempio 2



```
$ pwd
```

```
/home/studente
```

```
$ cd /bin
```

oppure con path relativo: 

```
$ cd ../../bin
```

# La shell è case sensitive

**Attenzione:** la shell di Linux è **case sensitive**.

scrivere:

```
$ cd Documenti
```

è diverso da:

```
$ cd documenti
```

# Attenzione agli spazi

**Attenzione:** in un comando i diversi argomenti sono separati da spazi.

Per inserire un argomento che include spazi, molti comandi prevedono l'uso di virgolette o richiedono di mettere il carattere \ prima del carattere spazio (o altri caratteri problematici)

Questo non funziona:

```
$ cd mia cartella
```

Questo sì:

```
$ cd "mia cartella"
```

Questo anche:

```
$ cd mia\ cartella
```

# Operazioni sui file

- ▶ `touch <percorso file>`: crea un nuovo file;
- ▶ `cp <percorso file> <percorso file copia>`: copia un file in una directory diversa o nella stessa directory ma con nome differente;
- ▶ `mv <percorso file> <nuovo percorso file>`: sposta un file o ne modifica il nome;
- ▶ `rm <percorso file>`: rimuove un file;
- ▶ `cat <percorso file>`: visualizza il contenuto di un file;
- ▶ `more <percorso file>`: visualizza il contenuto di un file una porzione alla volta;

**Attenzione!:** non esiste il *cestino* quando si rimuove qualcosa con i comandi da shell, tutto ciò che viene eliminato è perso!

## E per aprire un file?

Se il file è programma eseguibile: `./<nome file>` (i nostri programmi si eseguiranno così).

I file eseguibili installati in alcune cartelle speciali del sistema possono essere usati col solo nome. Per questo scriviamo:

```
pwd
```

e non:

```
/bin/pwd
```

## Aprire file non eseguibili

Per file non eseguibili è necessario aprirli con un'applicazione apposita. Ad esempio, per aprire un file di testo è possibile usare l'editor Pluma: `pluma <nome file>`.

**Attenzione:** quando eseguite un comando da shell non potrete più interagire con la shell fino a quando il comando non sarà terminato. Nel caso dell'esempio precedente significa che non potrete interagire con la shell fino a quando l'editor Pluma non sarà chiuso.

Soluzioni possibili:

- ▶ aprire Pluma dal menu Applicazioni -> Accessori -> Pluma;
- ▶ aprire Pluma da terminale in background `pluma <nome file> &`.

## Operazioni sulle directory

- ▶ `mkdir <percorso directory>`: crea una directory;
- ▶ `mv <percorso directory> <nuovo percorso directory>`: sposta una directory o le cambia il nome;
- ▶ `cp -r <percorso directory> <percorso directory copia>`: copia una directory;
- ▶ `rmdir <percorso directory>`: rimuove una directory (se vuota);
- ▶ `rm -r <percorso directory>`: rimuove una directory e tutto il suo contenuto.

## E se mi dimentico come si usa un comando?

Esiste un comando chiamato `man` che permette di leggere le istruzioni e le opzioni disponibili per eseguire molti comandi della shell.

Ad esempio, `man ls` produce la descrizione del comando `ls` ed elenca tutte le sue possibili opzioni (non le vedremo tutte!).

Un altro modo per avere informazioni sull'uso di un comando è l'utilizzo dell'opzione `--help`, ad esempio: `cd --help`.



## Esercizio

1. Creare una cartella chiamata sorgente all'interno della propria home directory
2. Creare una cartella chiamata destinazione all'interno della propria home directory
3. Spostarsi all'interno della cartella sorgente
4. Creare un file di testo prova.txt
5. Modificare il file di testo appena creato inserendo il testo Questa è una prova! (Si utilizzi l'editor Pluma)

Successivamente:

- ▶ **Copiare** il file di testo presente nella cartella sorgente all'interno della cartella destinazione usando un *path assoluto*
- ▶ **Spostare** il file di testo dalla cartella destinazione alla propria home directory, usando un *path relativo*
- ▶ **Eliminare** il file prova.txt dalla cartella sorgente
- ▶ **Visualizzare** sul terminale il contenuto del file prova.txt

## Redirezione output

La shell permette di salvare l'output di un comando all'interno di un file attraverso la *redirezione dell'output*.

La redirezione può essere fatta in due modi:

- ▶ con > (un singolo carattere di maggiore):

```
$ ls > file_output.txt
```

L'output del comando `ls` viene scritto all'interno del file `file_output.txt`. Se il file non esiste verrà creato. Se il file esiste verrà sovrascritto.

- ▶ con >> (un doppio carattere di maggiore):

```
$ ls >> file_output.txt
```

L'output del comando `ls` viene scritto all'interno del file `file_output.txt`. Se il file non esiste verrà creato. Se il file esiste l'output del comando verrà aggiunto alla fine del file.

# Wildcard

Caratteri speciali che permettono di estendere l'effetto di un comando ad una selezione di file/directory.

## Wildcard - ?

? (punto interrogativo): indica un qualsiasi singolo carattere, ad esempio:

```
$ ls
```

```
prova_1  prova_2  prova_3  prova_4  test_1  test_2  test_3  
test_4   test_10
```

```
$ rm test_?
```

```
$ ls
```

```
prova_1  prova_2  prova_3  prova_4  test_10
```

## Wildcard - \*

\* (asterisco): indica una qualsiasi sequenza di caratteri (anche nulla), ad esempio:

```
$ ls
```

```
prova_1  prova_2  prova_3  prova_4  test_1  test_2  test_3  
test_4   test_10
```

```
$ rm test_*
```

```
$ ls
```

```
prova_1  prova_2  prova_3  prova_4
```