Informazioni generali

Organizzazione laboratori

- Gruppo (intervallo cognomi): A-Dig
- Lezione: sempre il giovedì (13.30-15.45 o altre soluzioni...) in Aula 307
- Ricevimento: giovedì 11.00-12.00 (**scrivermi in anticipo** per disponibilità)
- Materiale didattico generale: su Ariel
- Materiale didattico aggiuntivo: su `upload.di.unimi.it`

IMPORTANTE:

- **Rispettare** la **suddivisione** in **gruppi**
- Se **non iscritti** $\rightarrow$ andare comunque nel gruppo giusto...
- **PRIORITÀ** agli studenti di **MATEMATICA**

Contatti

- Ufficio 3018, via Celoria 18
- email: `michele.barbato@unimi.it`

La riga di comando in GNU/Linux: una guida pratica

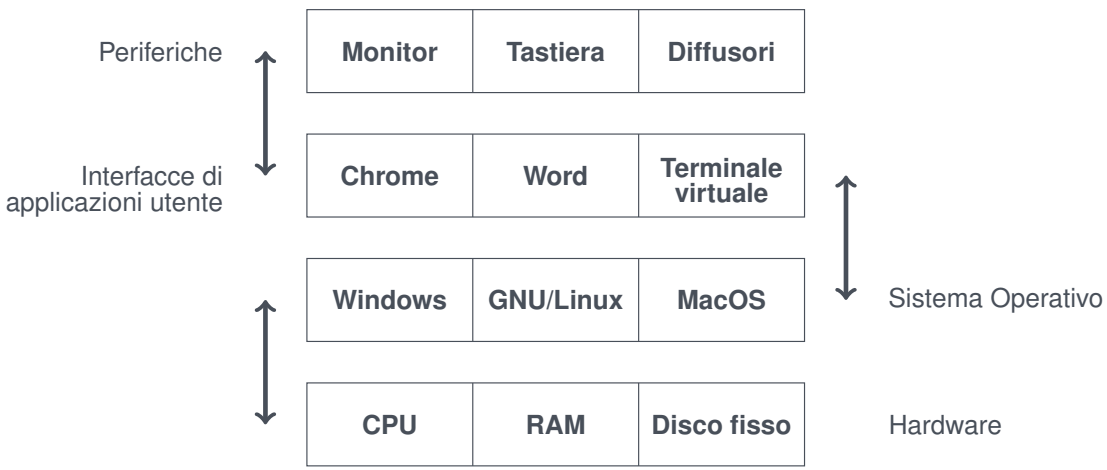
Michele Barbato

Dipartimento di Informatica *“Giovanni degli Antoni”* – Università degli Studi di Milano

UNIVERSITÀ
DEGLI STUDI
DI MILANO

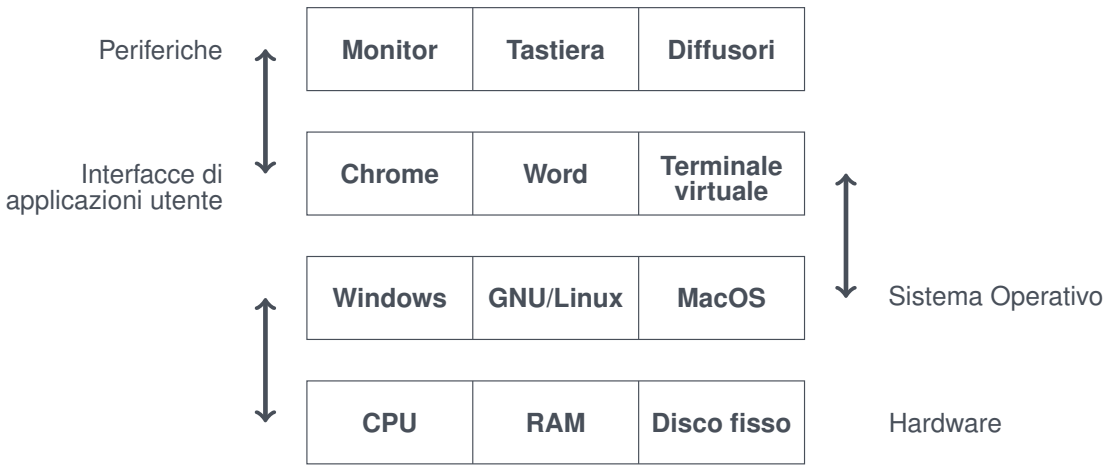
Interazione Utente-Macchina

- **Sistema operativo**: permette di eseguire **programmi** su un **hardware** (parte fisica del computer) su richiesta dell'**utente**.
- Le **due componenti** principali di un sistema operativo sono:
  - **kernel:** gestisce e comunica con l'hardware
  - **interfacce utente:** intermediarie tra il kernel e l'utente



Interazione Utente-Macchina

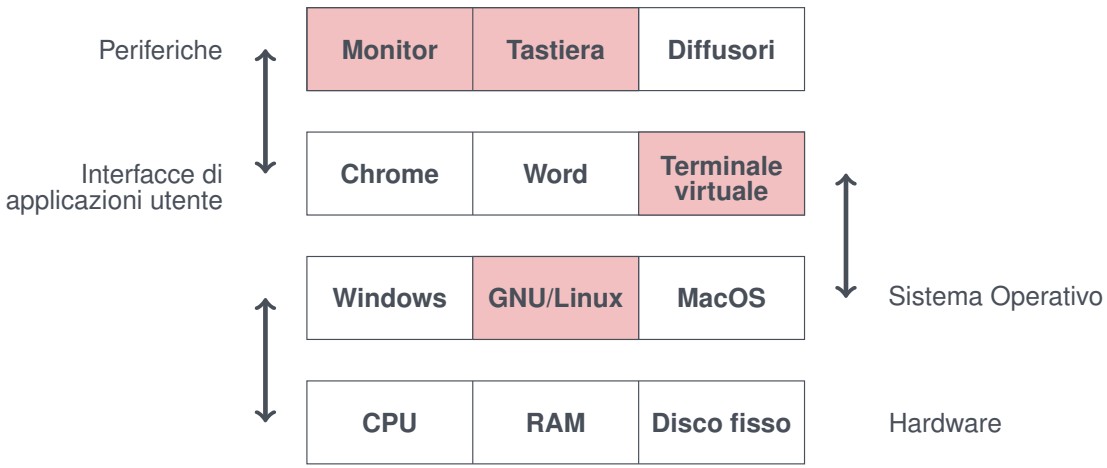
- **Sistema operativo**: permette di eseguire **programmi** su un **hardware** (parte fisica del computer) su richiesta dell'**utente**.
- Le **due componenti** principali di un sistema operativo sono:
  - **kernel:** gestisce e comunica con l'hardware
  - **interfacce utente:** intermediarie tra il kernel e l'utente
- **Windows e MacOS** sono **esempi** di sistemi operativi



Interazione Utente-Macchina

- **Sistema operativo**: permette di eseguire **programmi** su un **hardware** (parte fisica del computer) su richiesta dell'**utente**.
- Le **due componenti** principali di un sistema operativo sono:
  - **kernel:** gestisce e comunica con l'hardware
  - **interfacce utente:** intermediarie tra il kernel e l'utente
- **Windows e MacOS** sono **esempi** di sistemi operativi

In questa lezione:



L'interprete dei comandi

- **Shell:** permette di eseguire **operazioni** digitando **comandi** con **tastiera**
- in questa lezione: **BASH** (**B**ourne **A**gain **Sh**ell)

L'interprete dei comandi

- **Shell:** permette di eseguire **operazioni** digitando **comandi** con **tastiera**
- in questa lezione: **BASH** (**B**ourne **A**gain **Sh**ell)

Come “lancio” BASH?

- Usiamo BASH attraverso un “**terminale virtuale**”

Come “lancio” BASH?

- Usiamo BASH attraverso un “**terminale virtuale**”
- In **laboratorio** (ambiente desktop **`MATE`**):
  - **Menù** delle **applicazioni**
    (**`Applications`** → **`System Tools`** → **`MATE Terminal`**)
  - **Icona** a forma di **terminale** nel **pannello** superiore
  - nella **cartella** `/usr/bin` fare doppio click su `mate-terminal`

Come “lancio” BASH?

- Usiamo BASH attraverso un “**terminale virtuale**”
- In **laboratorio** (ambiente desktop **`MATE`**):
  - **Menù** delle **applicazioni**
    (**`Applications`** → **`System Tools`** → **`MATE Terminal`**)
  - **Icona** a forma di **terminale** nel **pannello** superiore
  - nella **cartella** `/usr/bin` fare doppio click su `mate-terminal`

Nota:

- su altre distribuzioni GNU/Linux la procedura potrebbe essere diversa

Come si usa BASH?

In breve:

- **Digitare i nomi dei comandi** da eseguire
- **Digitare** `Invio` per eseguire il comando

Come si usa BASH?

In breve:

- **Digitare i nomi dei comandi** da eseguire
- **Digitare** `Invio` per eseguire il comando

Struttura generale dei **comandi**:

- `<nome comando> [opzioni] <argomenti>`

Come si usa BASH?

In breve:

- **Digitare i nomi dei comandi** da eseguire
- **Digitare** `Invio` per eseguire il comando

Struttura generale dei **comandi**:

- `<nome comando> [opzioni] <argomenti>`

Esempio (poi lo vediamo meglio)

- **Mostra** il **contenuto** di una cartella: `ls -l /home`
  - `ls` è il **nome del comando** (sta per "**l**i**s**t")
  - `-l` è un'**opzione** (sta per "**l**ong format")
  - `/home` è un **argomento** (in questo caso, una cartella)

Un utile strumento: il manuale

Siete in **difficoltà** con qualche comando?

- `man <nome_comando>` apre il **manuale**
  - **argomenti** tra **parentesi quadre** nella **sinossi** sono **opzionali**
  - **scorrere** il documento con le **frecce direzionali**
  - **uscire** digitando `q` (**q**uit)

Un utile strumento: il manuale

Siete in **difficoltà** con qualche comando?

- `man <nome_comando>` apre il **manuale**
  - **argomenti** tra **parentesi quadre** nella **sinossi** sono **opzionali**
  - **scorrere** il documento con le **frecce direzionali**
  - **uscire** digitando `q` (**q**uit)
- `<nome_comando> --help` mostra una **schermata d'aiuto** (di solito **breve** e **non navigabile**)

Un utile strumento: il manuale

Siete in **difficoltà** con qualche comando?

- `man <nome_comando>` apre il **manuale**
  - **argomenti** tra **parentesi quadre** nella **sinossi** sono **opzionali**
  - **scorrere** il documento con le **frecce direzionali**
  - **uscire** digitando `q` (**q**uit)
- `<nome_comando> --help` mostra una **schermata d'aiuto** (di solito **breve** e **non navigabile**)

Esempi:

- `man touch`: mostra il manuale del comando `touch`
- `touch --help`: mostra la pagina d'aiuto del comando `touch`

N.B. Il comando `touch` sarà presentato in seguito

Esercizio (5 min)

- Leggete il manuale del comando `who` e poi provate tale comando
- Leggete il manuale del comando `echo` e poi provate tale comando (alcune opzioni potrebbero non funzionare)

I nomi di file come input dei comandi

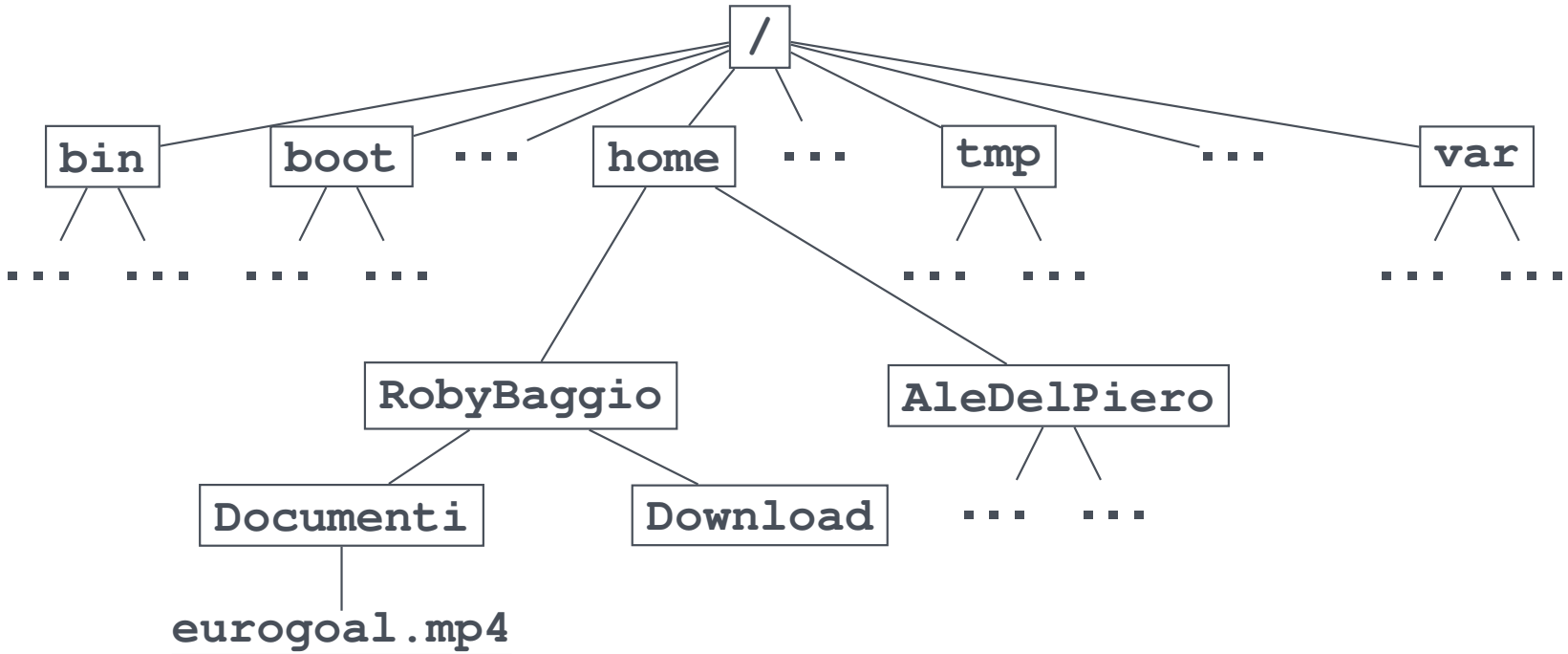
- I **comandi BASH** sono orientati alla **manipolazione** (creazione, lettura e modifica) di **file** e **directory** (cioè le "**cartelle**" del linguaggio comune)
- La maggior parte dei **comandi** prende in input (come **argomenti**) **nomi** di **file** e **directory**

I nomi di file come input dei comandi

- I **comandi BASH** sono orientati alla **manipolazione** (creazione, lettura e modifica) di **file** e **directory** (cioè le "**cartelle**" del linguaggio comune)
- La maggior parte dei **comandi** prende in input (come **argomenti**) **nomi** di **file** e **directory**
- Possono esistere **più file/directory** con lo **stesso nome**. . . come specificarli in **modo univoco**?

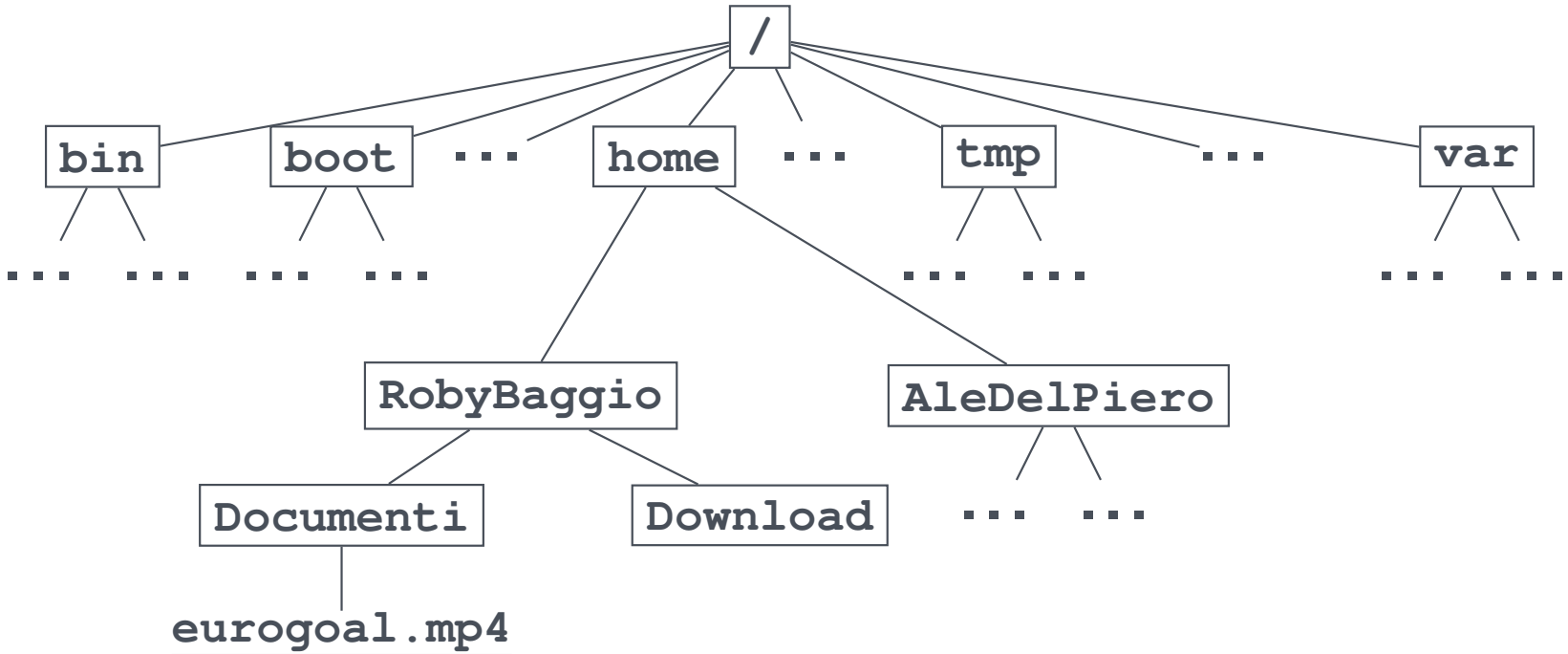
Il filesystem

- Il **filesystem** è una **rappresentazione ad albero** dell'insieme dei file



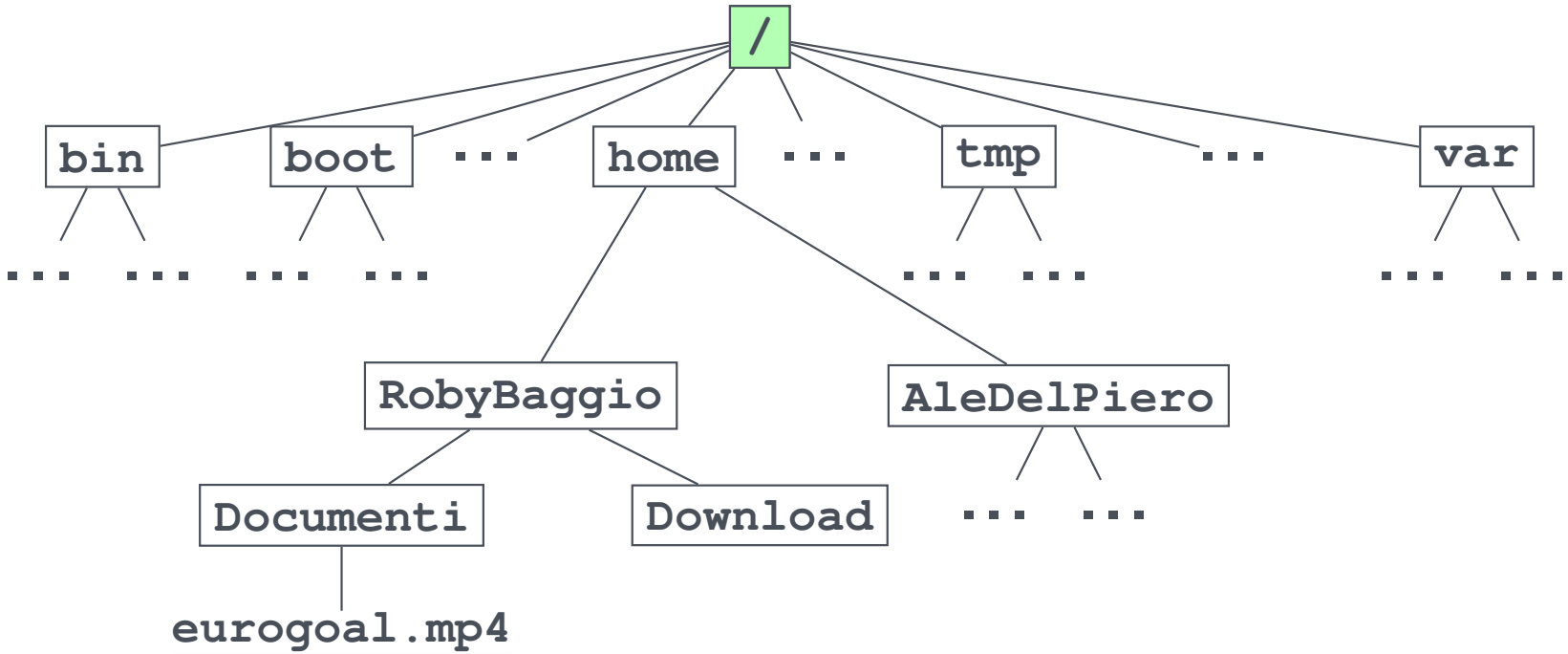
Il filesystem

- Il **filesystem** è una **rappresentazione ad albero** dell'insieme dei file
  - I **nodi intermedi** sono **directory** contenenti i nodi successivi
  - Le **foglie** corrispondono ai **file** e alle **directory vuote**



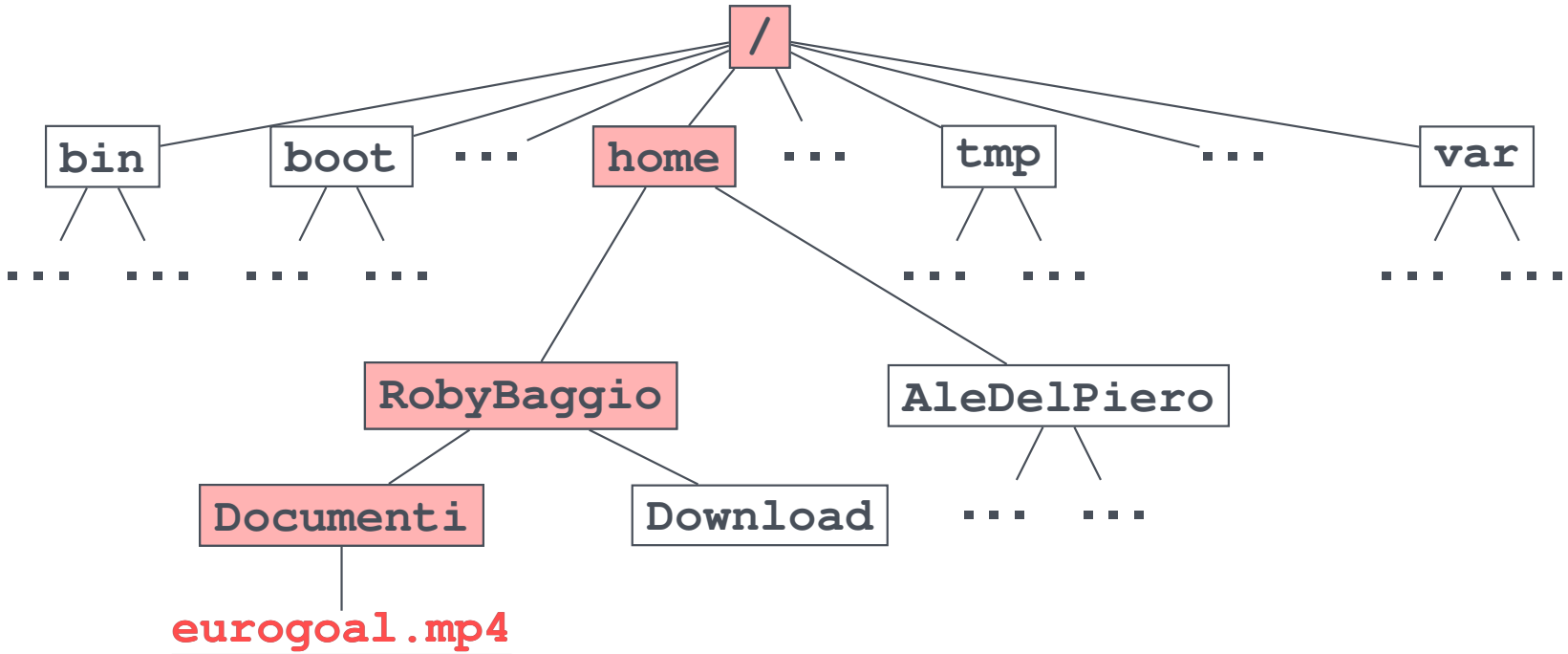
Il filesystem

- Il **filesystem** è una **rappresentazione ad albero** dell'insieme dei file
  - I **nodi intermedi** sono **directory** contenenti i nodi successivi
  - Le **foglie** corrispondono ai **file** e alle **directory vuote**
- La **root** ("radice") è la directory che **contiene tutte le altre**
  - si indica con `/` (slash)
  - **Esempio:** il comando `ls -l /` mostra il contenuto della root



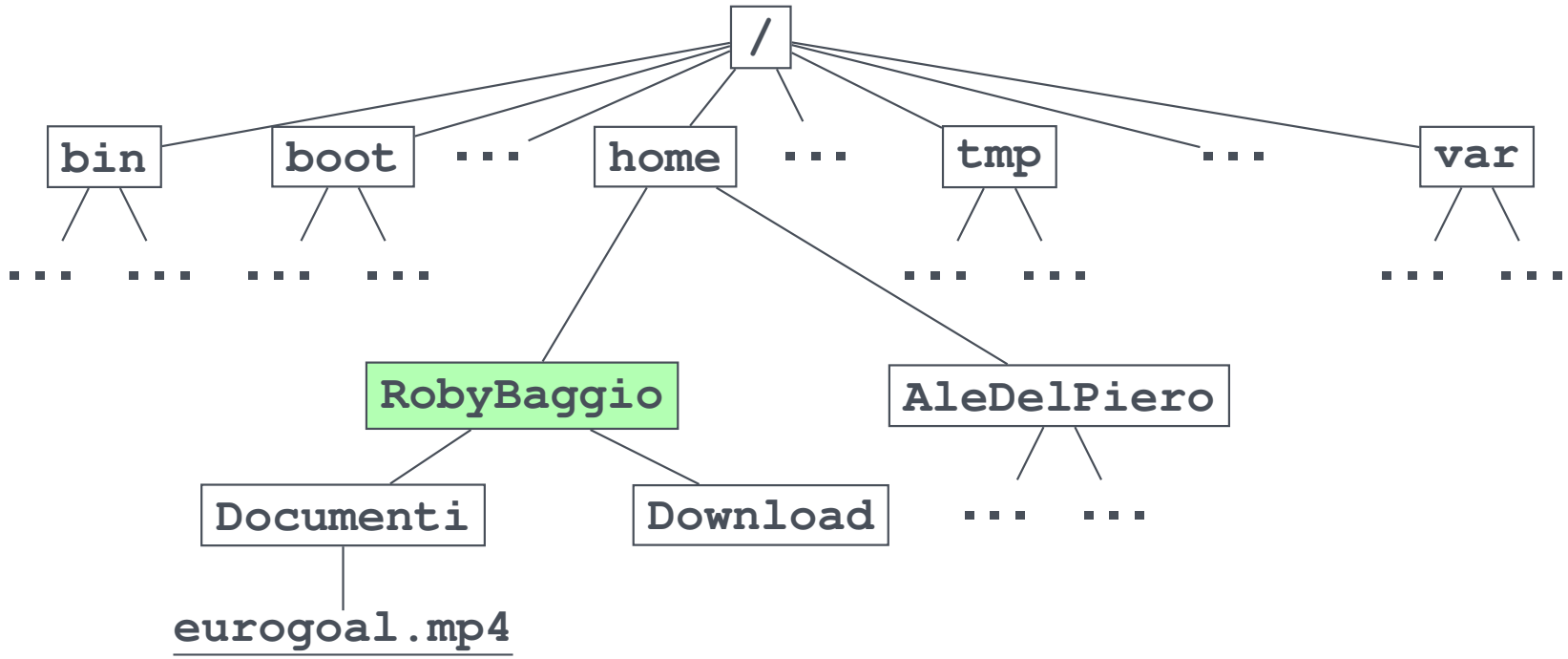
Il filesystem

- Il **filesystem** è una **rappresentazione ad albero** dell'insieme dei file
  - I **nodi intermedi** sono **directory** contenenti i nodi successivi
  - Le **foglie** corrispondono ai **file** e alle **directory vuote**
- La **root** ("radice") è la directory che **contiene tutte le altre**
  - si indica con `/` (slash)
  - **Esempio:** il comando `ls -l /` mostra il contenuto della root
- **Path** (percorso) **assoluto**: **sequenza** di directory **dalla root** (inclusa) al file/directory di interesse, separate da `/` (slash)
  - **Esempio:** `/home/RobyBaggio/Documenti/eurogoal.mp4`



La home

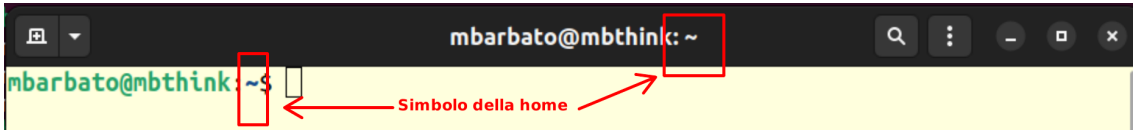
- La **home** è una directory riservata ai file “personali” degli utenti
  - Nel filesystem in basso la posizione è: `/home/<nome-utente>`
  - **Esempio:** `/home/RobyBaggio`



La home

- La **home** è una directory riservata ai file “personali” degli utenti
  - Nel filesystem in basso la posizione è: `/home/<nome-utente>`
  - **Esempio:** `/home/RobyBaggio`

  - **Attenzione:** in BASH la home si può **abbreviare** con `~` (tilde: ottenuta con `AltGr+ì`)
  - **In tal caso:** per l’utente `RobyBaggio` i seguenti comandi sono **equivalenti**:
    - `ls ~`
    - `ls /home/RobyBaggio`



Muoversi nel filesystem: il comando `cd` – Parte 1

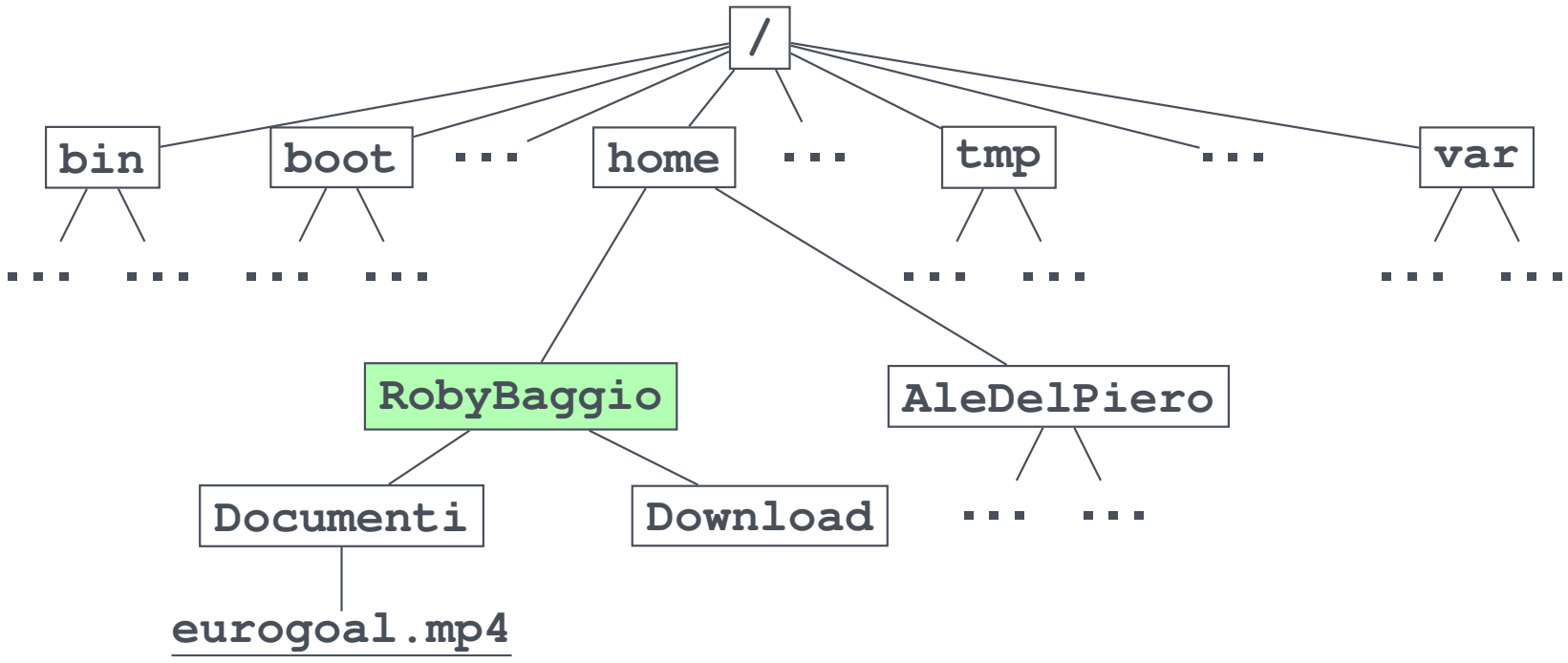
- BASH è **sempre "posizionata"** in una directory del filesystem
- **Directory di lavoro** ("working directory"): **posizione attuale** di BASH

IMPORTANTE

- il comando `pwd` mostra la directory di lavoro
- `<comando> <file>` → **esegue** `comando` su `file` nella **directory di lavoro**
- **omettere** il percorso di `file` ≡ percorso della **directory di lavoro**

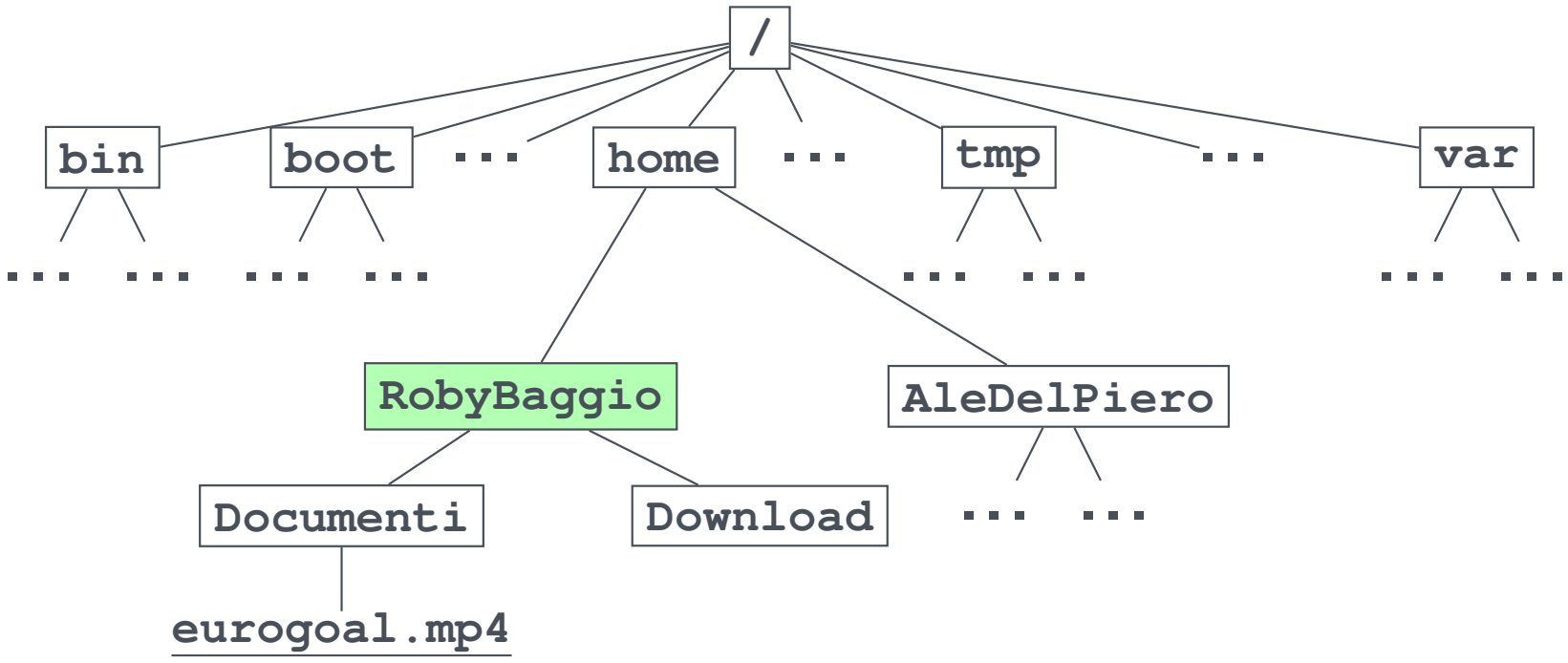
Muoversi nel filesystem: il comando `cd` – Parte 1

- BASH è **sempre "posizionata"** in una directory del filesystem
- **Directory di lavoro** ("working directory"): **posizione attuale** di BASH
- All'**avvio** è "posizionata" nella **home**



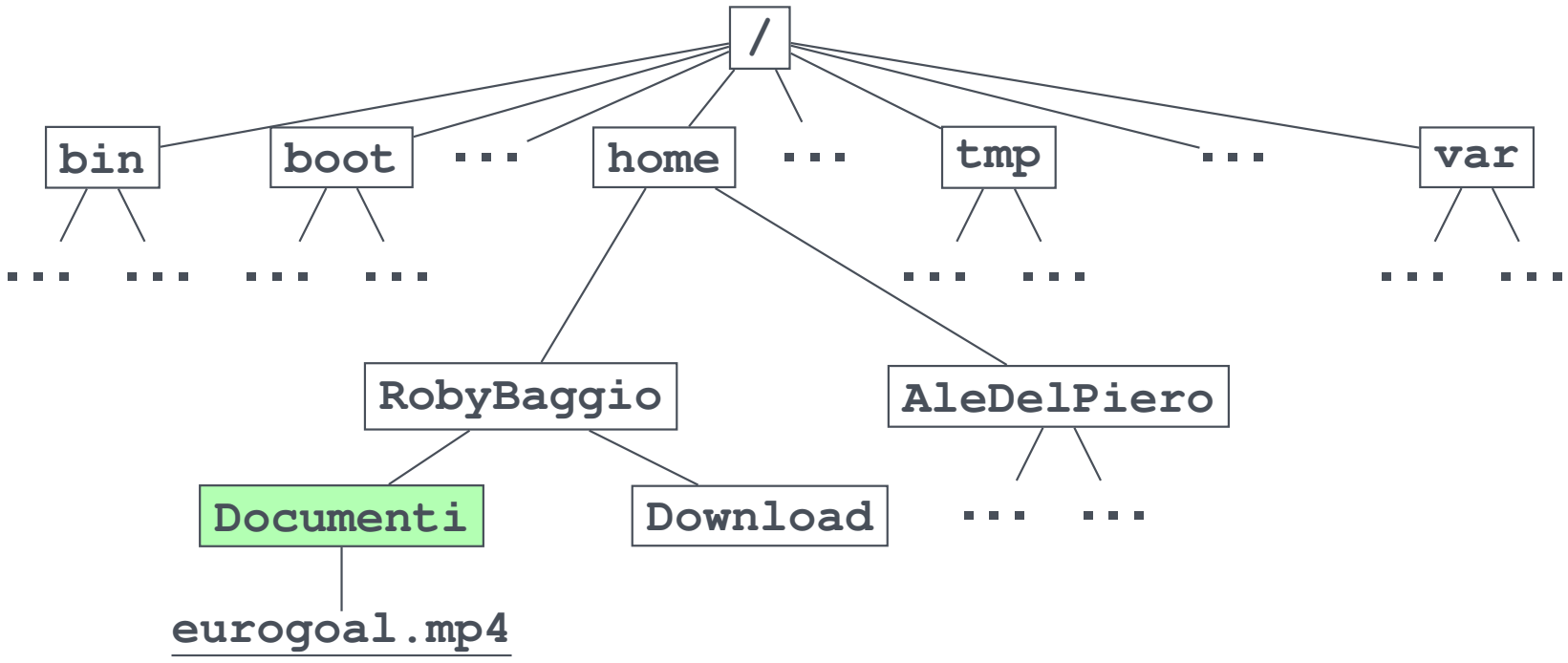
Muoversi nel filesystem: il comando `cd` – Parte 1

- BASH è **sempre "posizionata"** in una directory del filesystem
- **Directory di lavoro** ("working directory"): **posizione attuale** di BASH
- All'**avvio** è "posizionata" nella **home**
- Si può "**riposizionare**" col comando `cd` ("**c**hange **d**irectory")
  - **Sintassi:** `cd <percorso/assoluto/nuova/directory>`



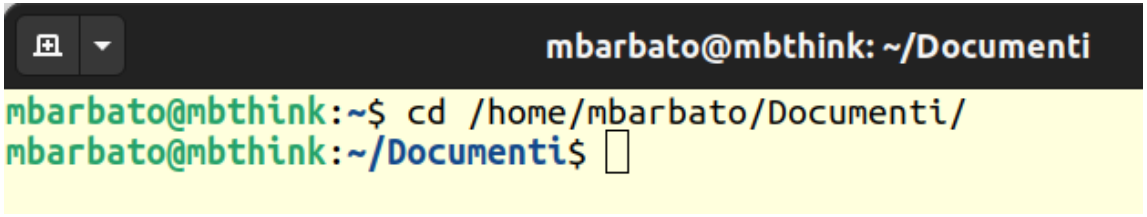
Muoversi nel filesystem: il comando `cd` – Parte 1

- BASH è **sempre "posizionata"** in una directory del filesystem
- **Directory di lavoro** ("working directory"): **posizione attuale** di BASH
- All'**avvio** è "posizionata" nella **home**
- Si può "**riposizionare**" col comando `cd` ("**c**hange **d**irectory")
  - **Sintassi:** `cd <percorso/assoluto/nuova/directory>`
  - **Esempio:** `cd /home/RobyBaggio/Documenti`



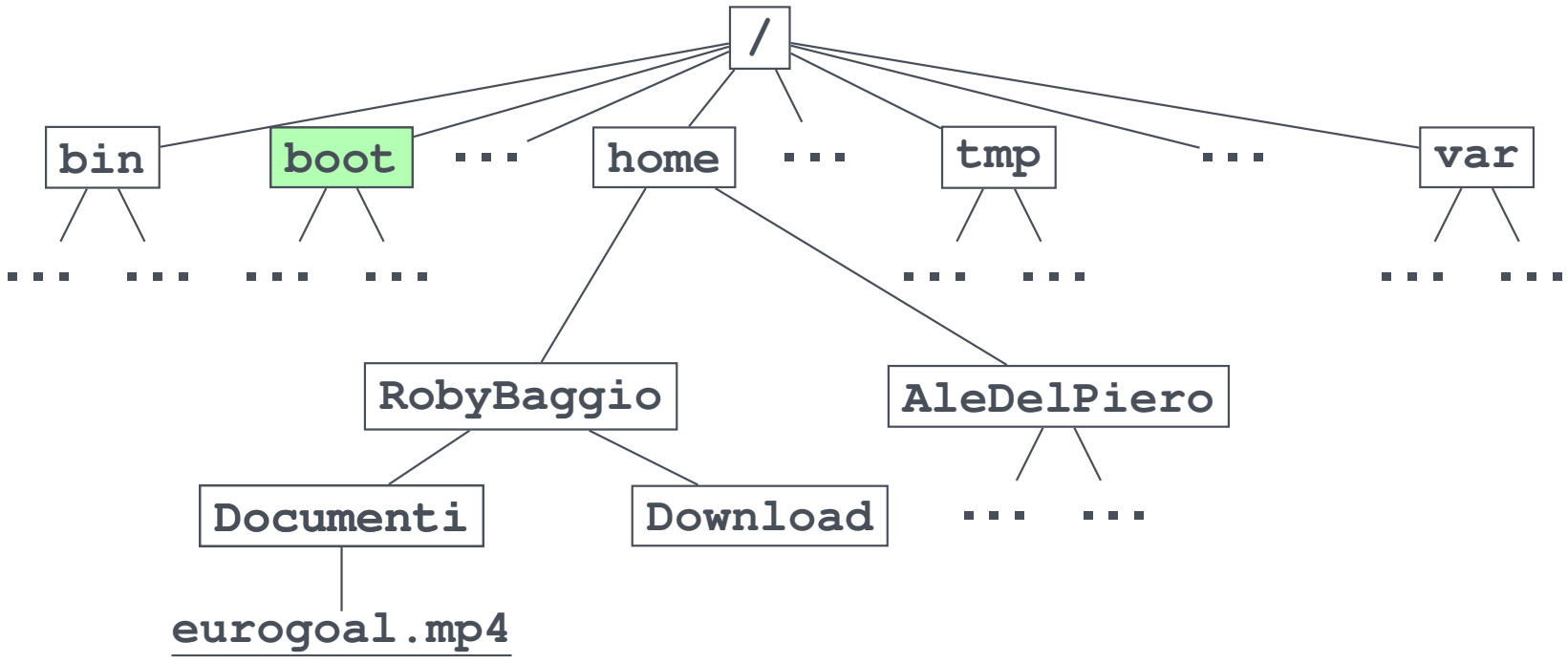
Muoversi nel filesystem: il comando `cd` – Parte 1

- BASH è **sempre "posizionata"** in una directory del filesystem
- **Directory di lavoro** ("working directory"): **posizione attuale** di BASH
- All'**avvio** è "posizionata" nella **home**
- Si può "**riposizionare**" col comando `cd` ("**c**hange **d**irectory")
  - **Sintassi:** `cd <percorso/assoluto/nuova/directory>`



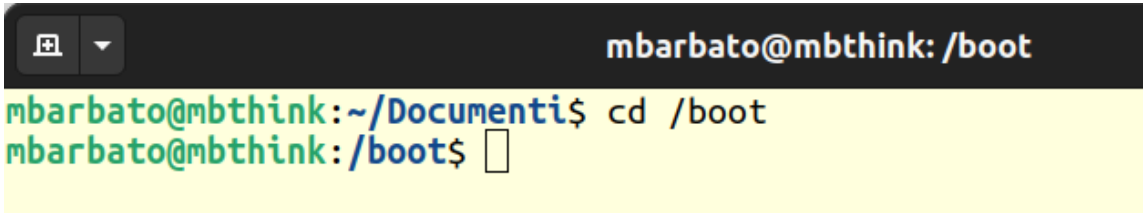
Muoversi nel filesystem: il comando `cd` – Parte 1

- BASH è **sempre "posizionata"** in una directory del filesystem
- **Directory di lavoro** ("working directory"): **posizione attuale** di BASH
- All'**avvio** è "posizionata" nella **home**
- Si può "**riposizionare**" col comando `cd` ("**c**hange **d**irectory")
  - **Sintassi:** `cd <percorso/assoluto/nuova/directory>`
  - **Esempio:** `cd /boot`



Muoversi nel filesystem: il comando `cd` – Parte 1

- BASH è **sempre "posizionata"** in una directory del filesystem
- **Directory di lavoro** ("working directory"): **posizione attuale** di BASH
- All'**avvio** è "posizionata" nella **home**
- Si può "**riposizionare**" col comando `cd` ("**c**hange **d**irectory")
  - **Sintassi:** `cd <percorso/assoluto/nuova/directory>`
  - **Esempio:** `cd /boot`



La directory di lavoro e i path relativi: il comando `cd` – Parte 2

- **Path relativo**: **sequenza** di directory o file a partire dalla **dir. di lavoro** separati da `/` (slash)

La directory di lavoro e i path relativi: il comando `cd` – Parte 2

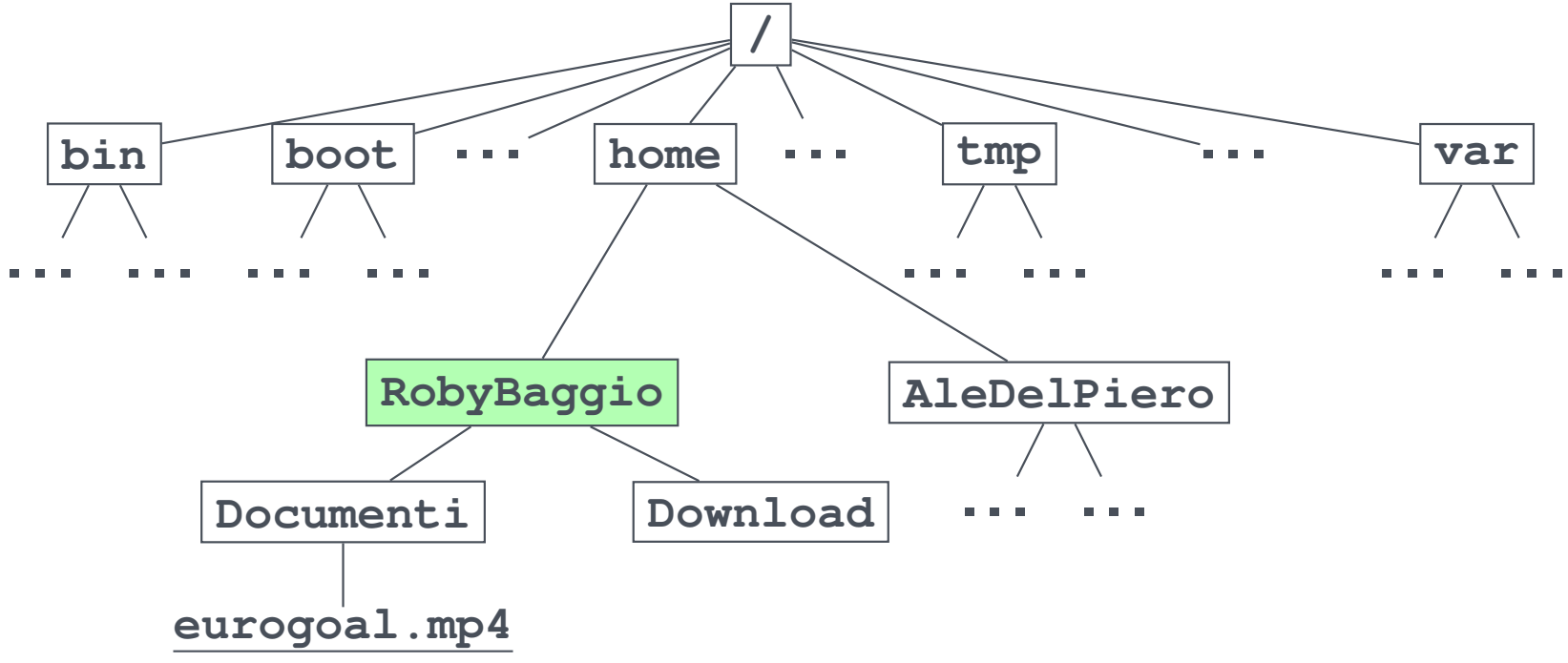
- **Path relativo**: **sequenza** di directory o file a partire dalla **dir. di lavoro** separati da `/` (slash)
  - **path relativo** della **directory di lavoro**: `.` (punto)
  - **path relativo** della **directory genitore**: `..` (punto-punto)

La directory di lavoro e i path relativi: il comando `cd` – Parte 2

- **Path relativo**: **sequenza** di directory o file a partire dalla **dir. di lavoro** separati da `/` (slash)
  - **path relativo** della **directory di lavoro**: `.` (punto)
  - **path relativo** della **directory genitore**: `..` (punto-punto)

  Esempio: se sono in `/home/RobyBaggio/` il comando

  `cd ./Documenti` sposta BASH in `/home/RobyBaggio/Documenti`

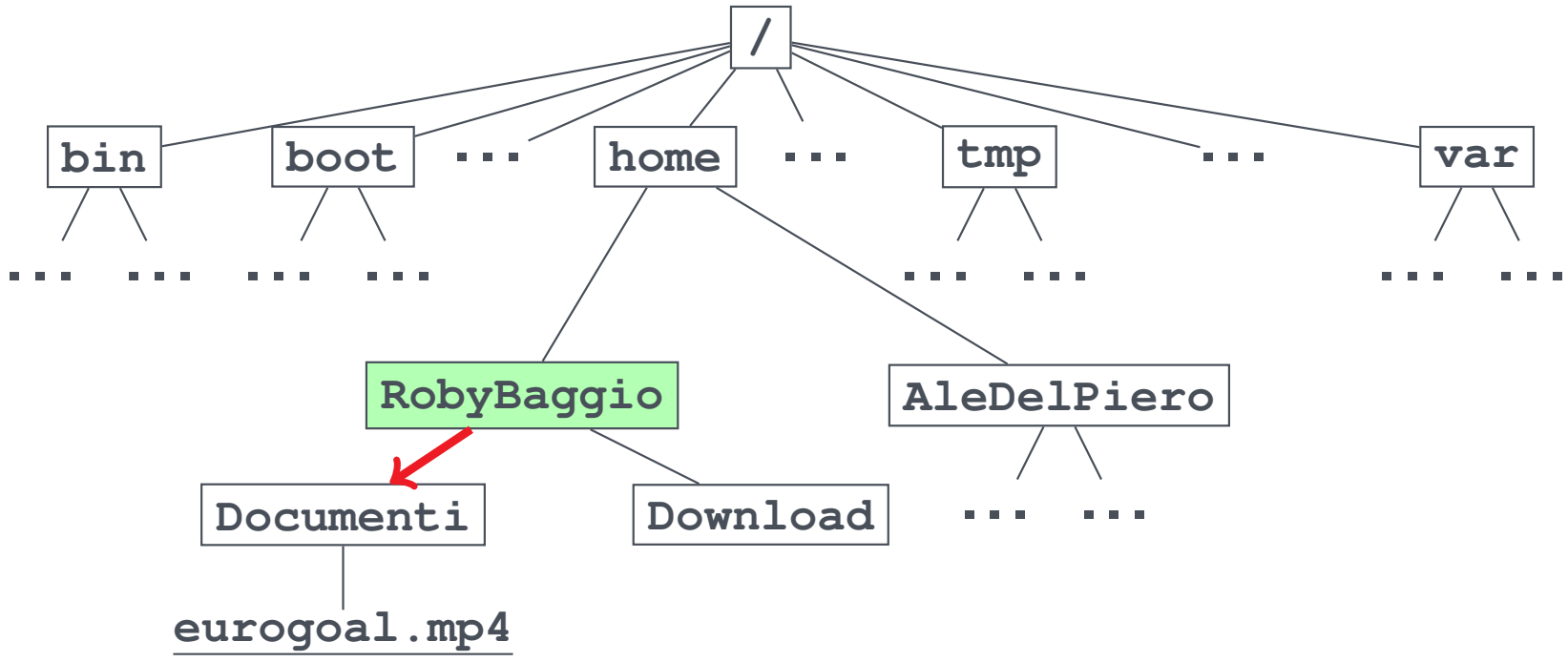


La directory di lavoro e i path relativi: il comando `cd` – Parte 2

- **Path relativo**: **sequenza** di directory o file a partire dalla **dir. di lavoro** separati da `/` (slash)
  - **path relativo** della **directory di lavoro**: `.` (punto)
  - **path relativo** della **directory genitore**: `..` (punto-punto)

  Esempio: se sono in `/home/RobyBaggio/` il comando

  `cd ./Documenti` sposta BASH in `/home/RobyBaggio/Documenti`

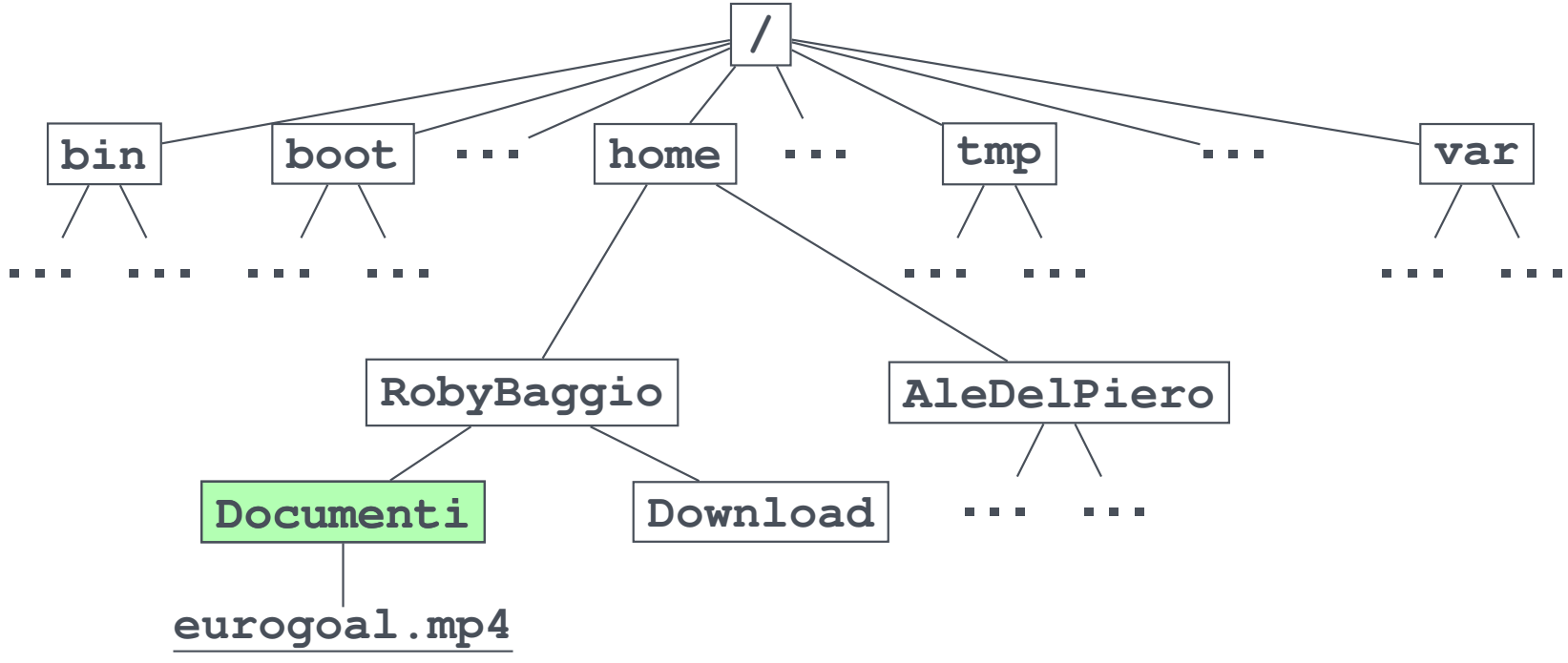


La directory di lavoro e i path relativi: il comando `cd` – Parte 2

- **Path relativo**: **sequenza** di directory o file a partire dalla **dir. di lavoro** separati da `/` (slash)
  - **path relativo** della **directory di lavoro**: `.` (punto)
  - **path relativo** della **directory genitore**: `..` (punto-punto)

  Esempio: se sono in `/home/RobyBaggio/` il comando

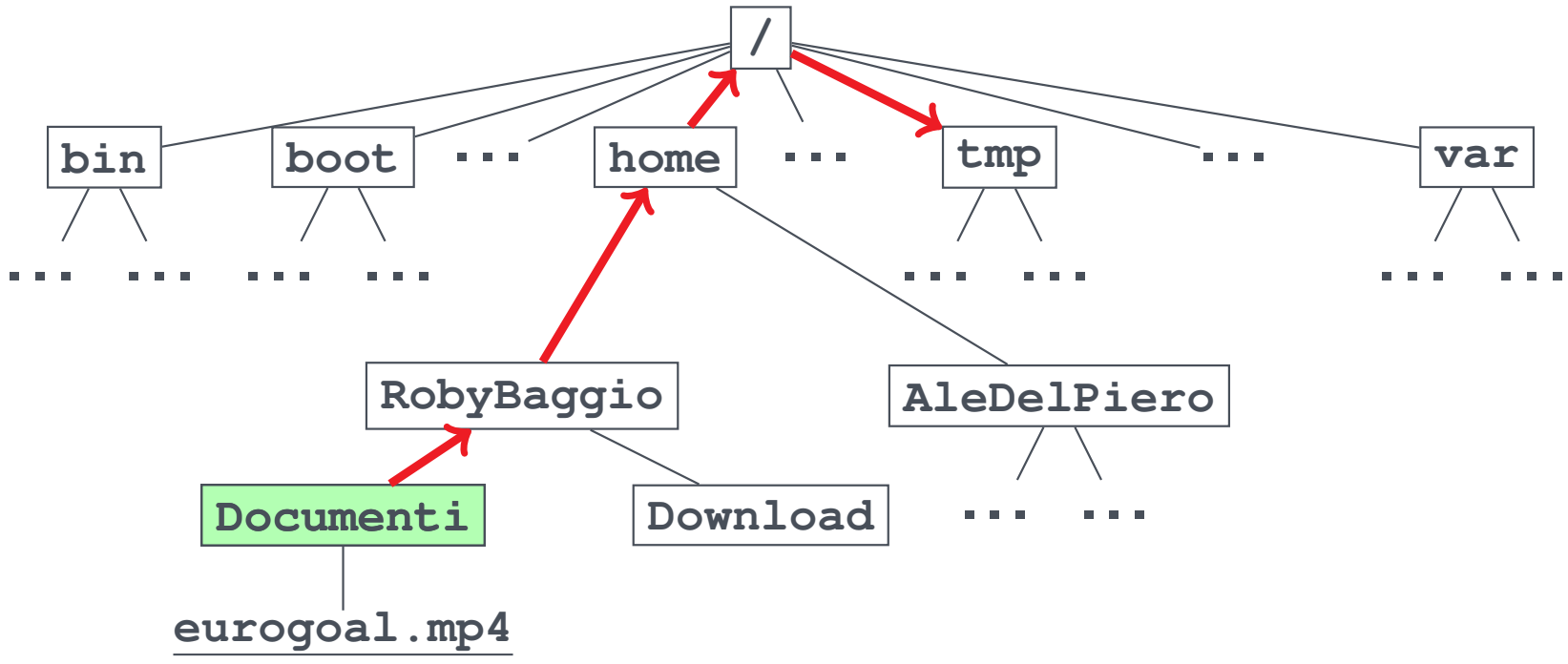
  `cd ./Documenti` sposta BASH in `/home/RobyBaggio/Documenti`



La directory di lavoro e i path relativi: il comando `cd` – Parte 2

- **Path relativo**: **sequenza** di directory o file a partire dalla **dir. di lavoro** separati da `/` (slash)
  - **path relativo** della **directory di lavoro**: `.` (punto)
  - **path relativo** della **directory genitore**: `..` (punto-punto)

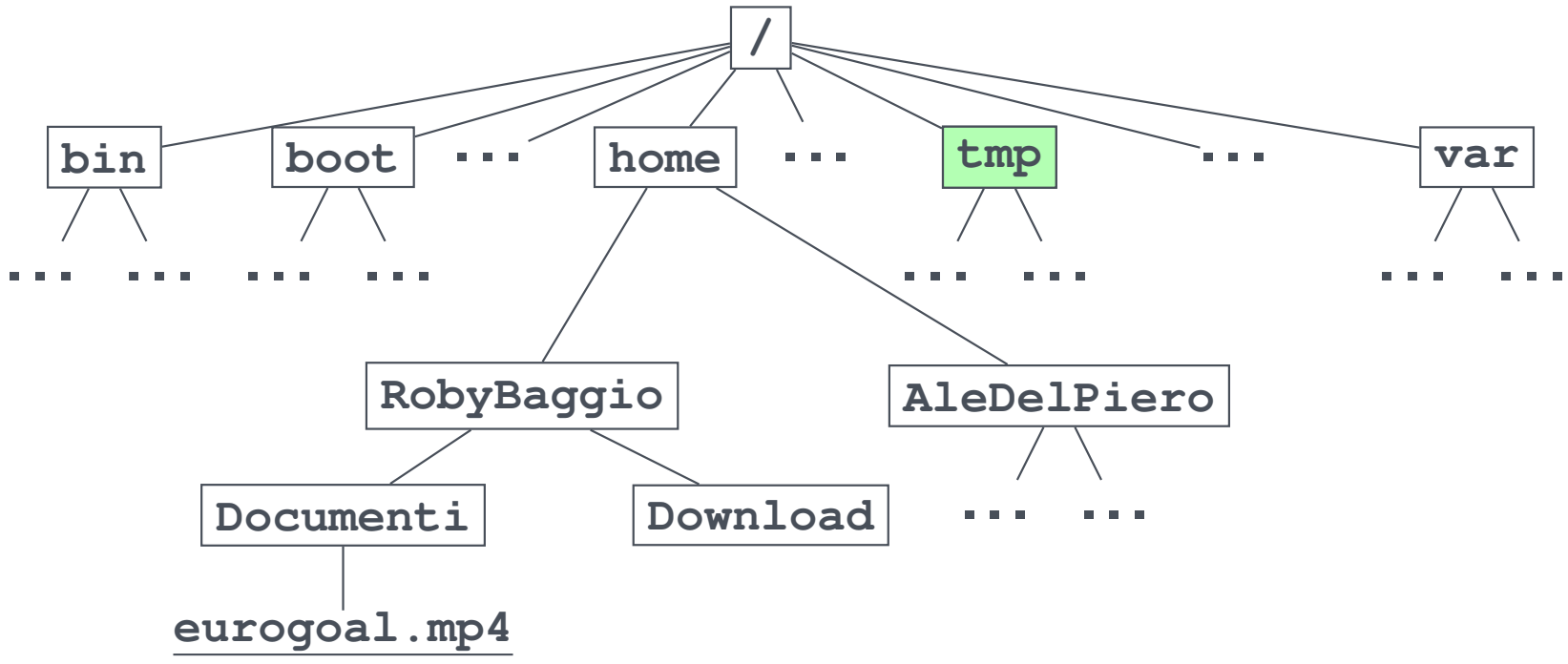
  Esempio: se sono in `/home/RobyBaggio/Documenti` il comando `cd ./../../../tmp` sposta BASH in `/tmp`



La directory di lavoro e i path relativi: il comando `cd` – Parte 2

- **Path relativo**: **sequenza** di directory o file a partire dalla **dir. di lavoro** separati da `/` (slash)
  - **path relativo** della **directory di lavoro**: `.` (punto)
  - **path relativo** della **directory genitore**: `..` (punto-punto)

  Esempio: se sono in `/home/RobyBaggio/Documenti` il comando `cd ./../../../tmp` sposta BASH in `/tmp`



La directory di lavoro e i path relativi: il comando `cd` – Parte 2

- **Path relativo**: **sequenza** di directory o file a partire dalla **dir. di lavoro** separati da `/` (slash)
  - **path relativo** della **directory di lavoro**: `.` (punto)
  - **path relativo** della **directory genitore**: `..` (punto-punto)

    Esempio: se sono in `/home/RobyBaggio/Documenti` il comando `cd ./../../../tmp` sposta BASH in `/tmp`

Nota 1

- Con molti comandi (tra cui `cd`), il simbolo `.` si può **omettere**

  Esempio: i due comandi seguenti sono equivalenti

  ```
  cd ./..
  cd ..
  ```

Nota 2

- Il comando `cd` (senza argomenti) riposiziona sempre BASH in `~`.

Esercizio (5 min)

- Immettete il comando per posizionare BASH nella vostra home
- Posizionate BASH nella directory genitore della vostra home
- Posizionate BASH nella cartella `/home`, specificandone il percorso assoluto
- Posizionate BASH nella cartella root, specificando un percorso relativo rispetto a `/home`
- Riposizionate BASH nella vostra home

Suggerimenti

Ricordare che:

- `cd percorso` $\rightarrow$ sposta BASH nel percorso
- `~`: abbreviazione della home
- `/`: simbolo della root
- `.` e `..`: simbolo di directory di lavoro (omettibile con i comandi precedenti) e directory genitore

Esercizio (5 min)

- Immettete il comando per posizionare BASH nella vostra home
- Posizionate BASH nella directory genitore della vostra home
- Posizionate BASH nella cartella `/home`, specificandone il percorso assoluto
- Posizionate BASH nella cartella root, specificando un percorso relativo rispetto a `/home`
- Riposizionate BASH nella vostra home

Possibile soluzione:

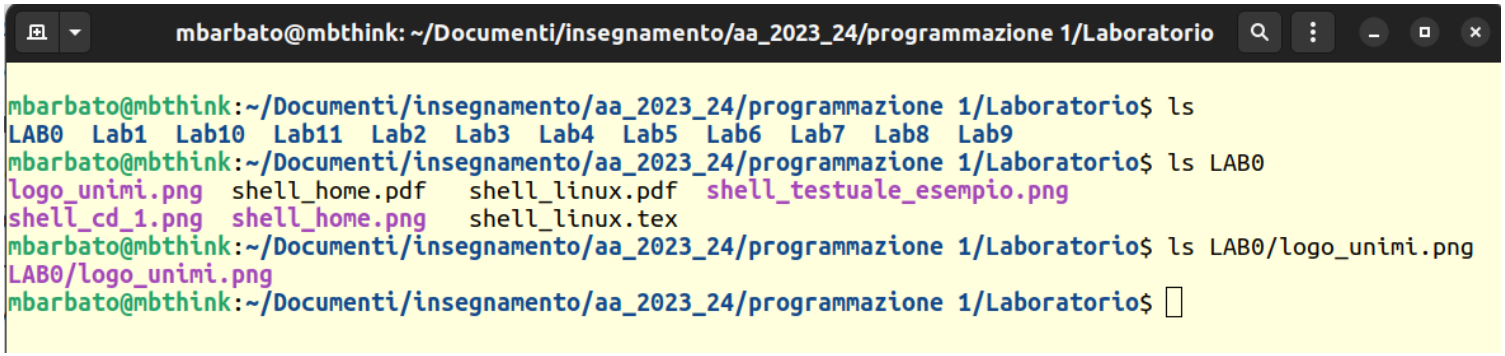
- `cd` (oppure `cd ~`)
- `cd ..`
- `cd /home`
- `cd ..`
- `cd` (oppure `cd ~`)

Primi comandi: il comando `ls`

- `ls percorso/file` **mostra i dettagli** del file in **argomento**:
  - se `percorso/file` è **assente** → dettagli dei file nella **dir. di lavoro**
  - se `percorso/file` è una **directory** → dettagli dei **file nella directory**
  - se `percorso/file` è un **file** → dettagli del **file**

Primi comandi: il comando `ls`

- `ls percorso/file` **mostra i dettagli** del file in **argomento**:
  - se `percorso/file` è **assente** → dettagli dei file nella **dir. di lavoro**
  - se `percorso/file` è una **directory** → dettagli dei **file nella directory**
  - se `percorso/file` è un **file** → dettagli del **file**

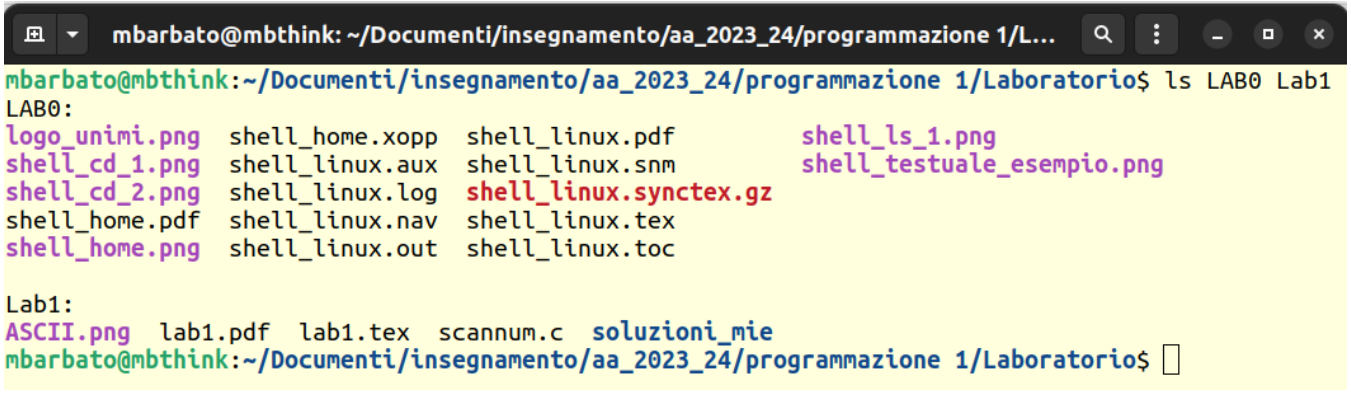


Primi comandi: il comando `ls`

- `ls percorso/file` **mostra i dettagli** del file in **argomento**:
  - se `percorso/file` è **assente** → dettagli dei file nella **dir. di lavoro**
  - se `percorso/file` è una **directory** → dettagli dei **file nella directory**
  - se `percorso/file` è un **file** → dettagli del **file**

Estensioni:

- **più di un argomento** → effetto identico a prima, per ognuno

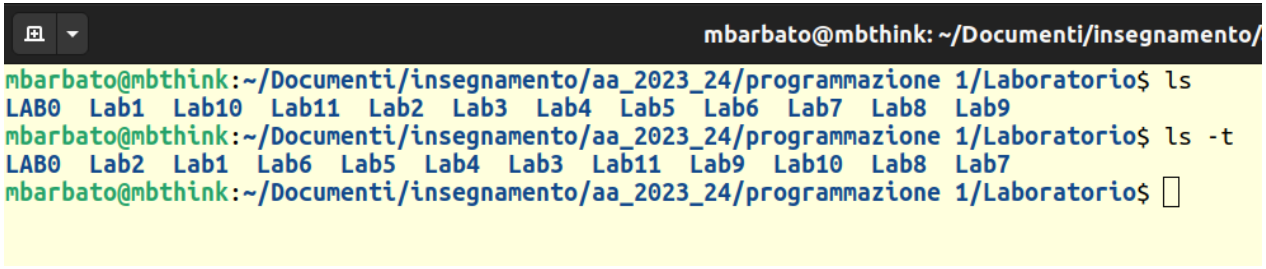


Primi comandi: il comando `ls`

- `ls percorso/file` **mostra i dettagli** del file in **argomento**:
  - se `percorso/file` è **assente** → dettagli dei file nella **dir. di lavoro**
  - se `percorso/file` è una **directory** → dettagli dei **file nella directory**
  - se `percorso/file` è un **file** → dettagli del **file**

Estensioni:

- **più di un argomento** → effetto identico a prima, per ognuno
- **opzione** `-t` → ordina per **modifica** al contenuto **più recente**

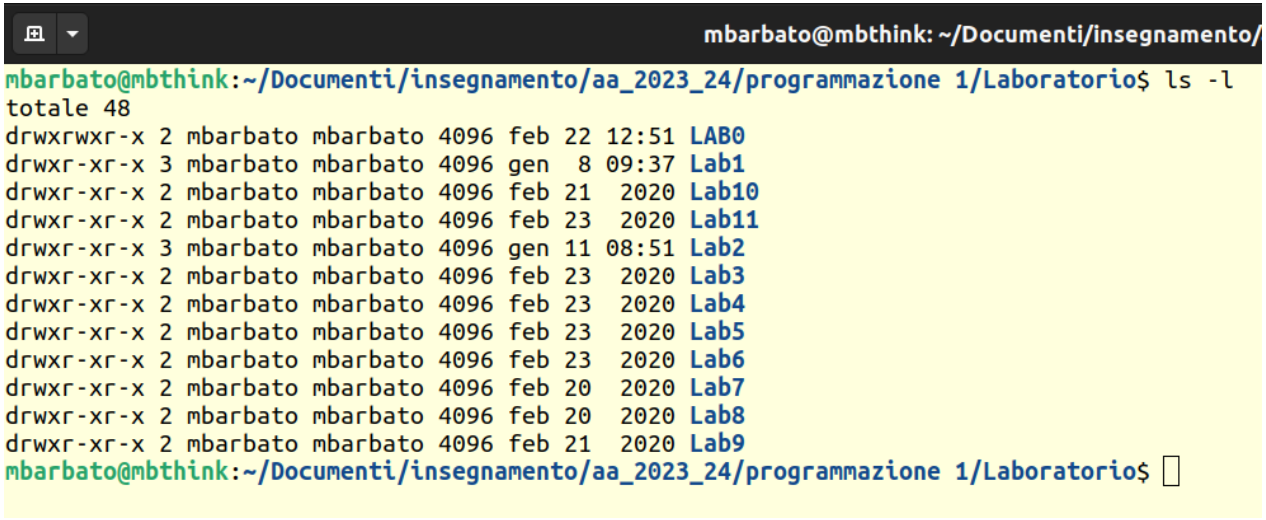


Primi comandi: il comando `ls`

- `ls percorso/file` **mostra i dettagli** del file in **argomento**:
  - se `percorso/file` è **assente** → dettagli dei file nella **dir. di lavoro**
  - se `percorso/file` è una **directory** → dettagli dei **file nella directory**
  - se `percorso/file` è un **file** → dettagli del **file**

Estensioni:

- **più di un argomento** → effetto identico a prima, per ognuno
- **opzione** `-t` → ordina per **modifica** al contenuto **più recente**
- **opzione** `-l` → molte **info aggiuntive** (data, permessi, taglia, ecc)

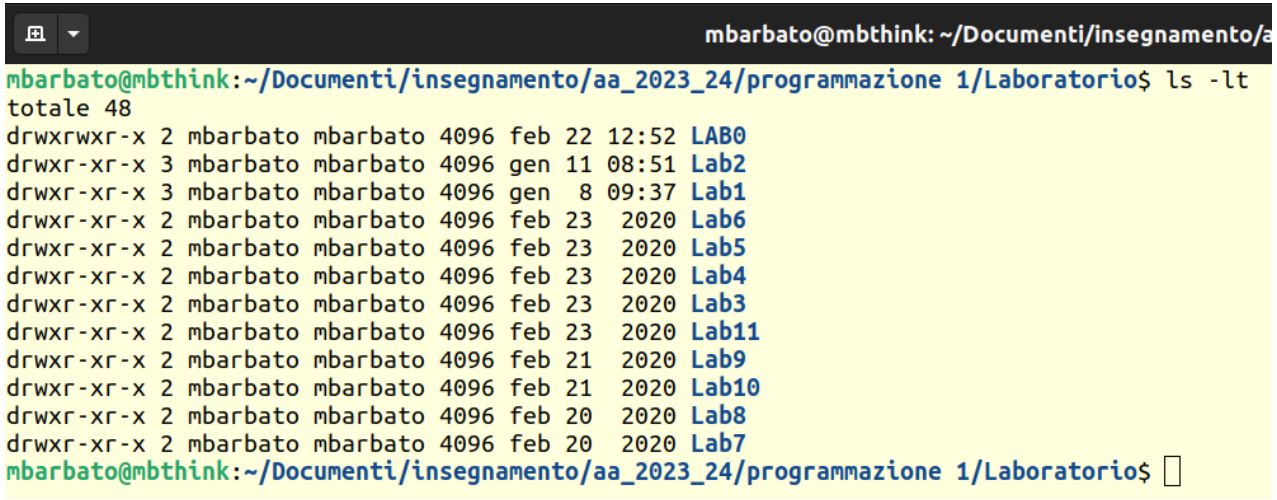


Primi comandi: il comando `ls`

- `ls percorso/file` **mostra i dettagli** del file in **argomento**:
  - se `percorso/file` è **assente** → dettagli dei file nella **dir. di lavoro**
  - se `percorso/file` è una **directory** → dettagli dei **file nella directory**
  - se `percorso/file` è un **file** → dettagli del **file**

Estensioni:

- **più di un argomento** → effetto identico a prima, per ognuno
- **opzione** `-t` → ordina per **modifica** al contenuto **più recente**
- **opzione** `-l` → molte **info aggiuntive** (data, permessi, taglia, ecc)
- **opzioni combinate** `-lt` → molte **info aggiuntive ordinate** per data

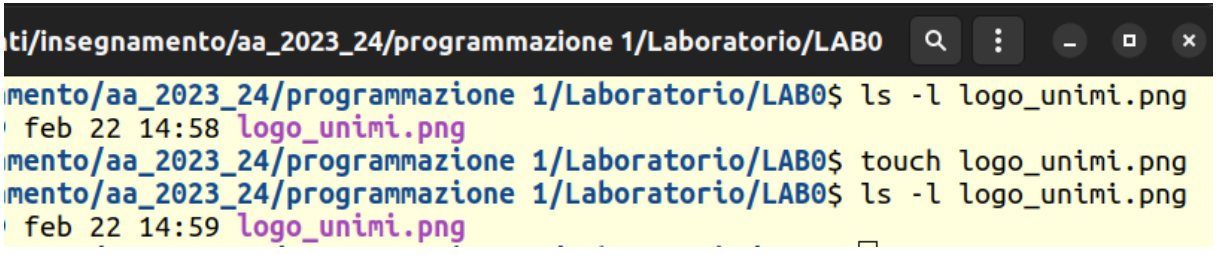


Primi comandi: il comando `touch` e la creazione di file

- `touch percorso/file` aggiorna la **data di modifica** all'orario corrente
  - l'argomento `percorso/file` è **obbligatorio**

Primi comandi: il comando `touch` e la creazione di file

- `touch percorso/file` aggiorna la **data di modifica** all'orario corrente
  - l'argomento `percorso/file` è **obbligatorio**

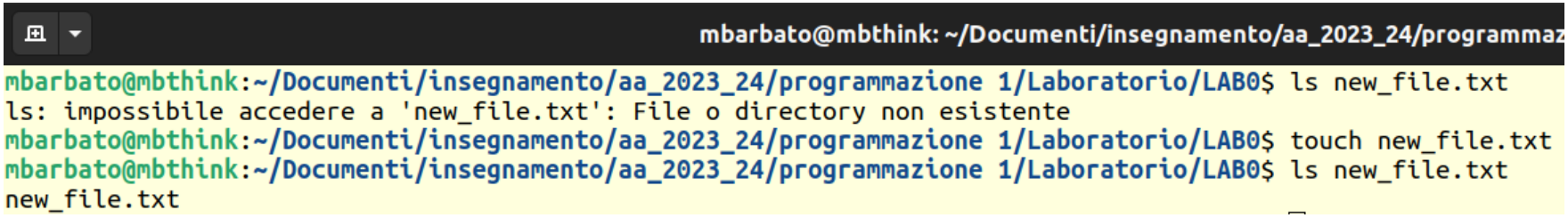


Primi comandi: il comando `touch` e la creazione di file

- `touch percorso/file` aggiorna la **data di modifica** all'orario corrente
  - l'argomento `percorso/file` è **obbligatorio**
  - se `percorso/file` è un nome di file inesistente $\rightarrow$ `percorso/file` è creato (file vuoto)

Primi comandi: il comando `touch` e la creazione di file

- `touch percorso/file` aggiorna la **data di modifica** all'orario corrente
  - l'argomento `percorso/file` è **obbligatorio**
  - se `percorso/file` è un nome di file inesistente → `percorso/file` è creato (file vuoto)



Primi comandi: il comando `touch` e la creazione di file

- `touch percorso/file` aggiorna la **data di modifica** all'orario corrente
  - l'argomento `percorso/file` è **obbligatorio**
  - se `percorso/file` è un nome di file inesistente → `percorso/file` è creato (file vuoto)

Note

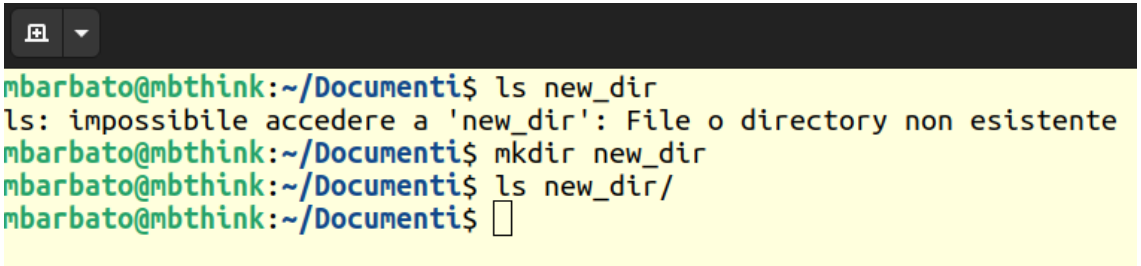
- il file creato sarà sempre un **file "semplice"** (**non directory**)
- se si specificano **più file** → effetto identico, su ognuno

Primi comandi: il comando `mkdir` e la creazione di directory

- `mkdir percorso/nuova/directory` crea una **nuova directory**
  - l'argomento `percorso/nuova/directory` è **obbligatorio**
  - la directory **inizialmente** sarà **vuota**

Primi comandi: il comando `mkdir` e la creazione di directory

- `mkdir percorso/nuova/directory` crea una **nuova directory**
  - l'argomento `percorso/nuova/directory` è **obbligatorio**
  - la directory **inizialmente** sarà **vuota**



Primi comandi: il comando `mkdir` e la creazione di directory

- `mkdir percorso/nuova/directory` crea una **nuova directory**
  - l'argomento `percorso/nuova/directory` è **obbligatorio**
  - la directory **inizialmente** sarà **vuota**

Note:

- Il file creato sarà una **directory** (**non** un **file "semplice"**)
- se si specificano **più directory** $\rightarrow$ effetto identico, su ognuno

Memo:

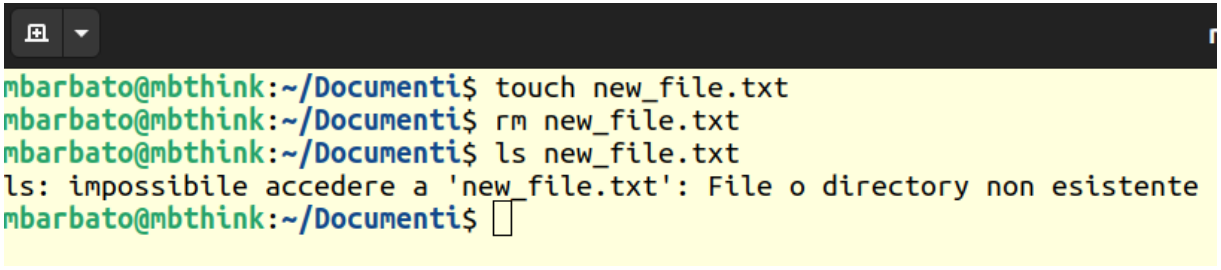
- Ovviamente, `mkdir` sta per "**m**a**k**e **dir**ectory"

Primi comandi: il comando `rm` e la rimozione di contenuto

- `rm percorso/file` **elimina** il **file** in **argomento**

Primi comandi: il comando `rm` e la rimozione di contenuto

- `rm percorso/file` **elimina** il **file** in **argomento**



Primi comandi: il comando `rm` e la rimozione di contenuto

- `rm percorso/file` **elimina** il **file** in **argomento**
  - Di default, **non** elimina le **directory**

Primi comandi: il comando `rm` e la rimozione di contenuto

- `rm percorso/file` **elimina** il **file** in **argomento**
  - Di default, **non** elimina le **directory**

Opzioni importanti:

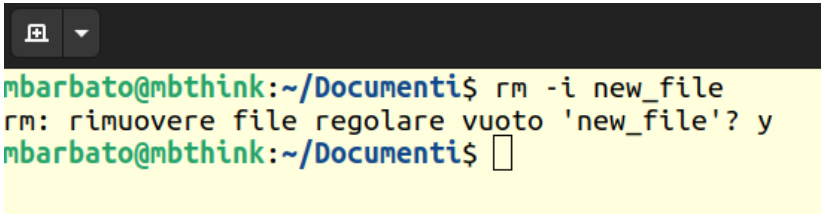
- `rm -d percorso/directory` : elimina **directory vuote**
- `rm -r percorso/directory` : elimina le **directory non vuote** e il loro contenuto
- `rm -i percorso/file/o/directory` : **chiede conferma** (rispondere `y` o `n` per "**yes**" o "**no**")

Primi comandi: il comando `rm` e la rimozione di contenuto

- `rm percorso/file` **elimina** il **file** in **argomento**
  - Di default, **non** elimina le **directory**

Opzioni importanti:

- `rm -d percorso/directory` : elimina **directory vuote**
- `rm -r percorso/directory` : elimina le **directory non vuote** e il loro contenuto
- `rm -i percorso/file/o/directory` : **chiede conferma** (rispondere `y` o `n` per "**yes**" o "**no**")



Esercizio (10 min)

- Nella directory `Documenti` del vostro utente creare una nuova directory chiamata `prog1_23_24`.
- All'interno di quest'ultima creare una directory `lab0`.
- All'interno di quest'ultima creare un file chiamato `hello_world.c`
- Usando il comando appropriato, controllare la correttezza della soluzione

Riassunto dei comandi:

- `ls percorso/file` → elenca i file specificati
- `mkdir percorso/nuova/directory` → crea la directory al percorso specificato
- `touch percorso/nuovo/file` → crea il file al percorso indicato
- `cd percorso` → sposta BASH nel percorso

Riassunto dei path

- `~`: abbreviazione della home
- `/`: simbolo della root
- `.` e `..`: simbolo di directory di lavoro (omettibile con i comandi precedenti) e directory genitore

Esercizio (10 min)

- Nella directory `Documenti` del vostro utente creare una nuova directory chiamata `prog1_23_24`.
- All'interno di quest'ultima creare una directory `lab0`.
- All'interno di quest'ultima creare un file chiamato `hello_world.c`
- Usando il comando appropriato, controllare la correttezza della soluzione

Esempio di soluzione 1

- `cd ~/Documenti`
- `mkdir prog1_23_24`
- `cd prog1_23_24`
- `mkdir lab0`
- `cd lab0`
- `touch hello_world.c`
- `ls`

Esercizio (10 min)

- Nella directory `Documenti` del vostro utente creare una nuova directory chiamata `prog1_23_24`.
- All'interno di quest'ultima creare una directory `lab0`.
- All'interno di quest'ultima creare un file chiamato `hello_world.c`
- Usando il comando appropriato, controllare la correttezza della soluzione

Esercizio (10 min)

- Nella directory `Documenti` del vostro utente creare una nuova directory chiamata `prog1_23_24`.
- All'interno di quest'ultima creare una directory `lab0`.
- All'interno di quest'ultima creare un file chiamato `hello_world.c`
- Usando il comando appropriato, controllare la correttezza della soluzione

Esempio di soluzione 2

- `mkdir ~/Documenti/prog1_23_24`
- `mkdir ~/Documenti/prog1_23_24/lab0`
- `touch ~/Documenti/prog1_23_24/lab0/hello_world.c`
- `ls ~/Documenti/prog1_23_24/lab0/`

Alcuni trucchi

- Per **riportare il cursore** del terminale in **alto**: `Ctrl+L`
- Premere su `TAB` per **completare automaticamente** un nome **digitato parzialmente** (premere **due volte** se esiste **più di una** soluzione)

Eseguire i programmi – Parte 1

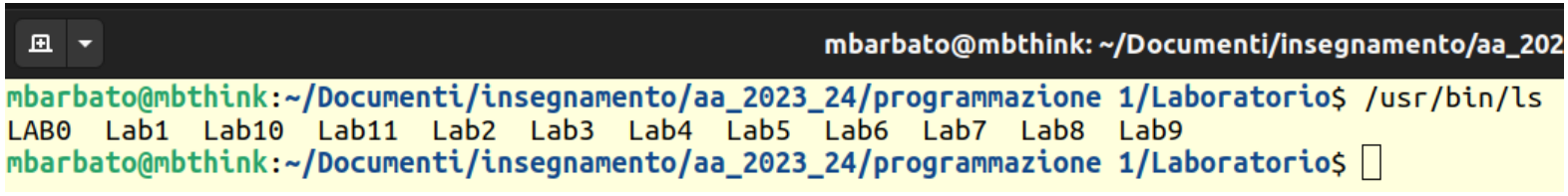
- Quasi tutti i **comandi** sono **“programmi”** (file “eseguibili”) già presenti nel filesystem (directory `/usr/bin`)
- In BASH, la **prima parola** digitata nella **linea di comando** è il **nome del programma** da eseguire (dopo la pressione di `Invio`)

Eseguire i programmi – Parte 1

- Quasi tutti i **comandi** sono **“programmi”** (file “eseguibili”) già presenti nel filesystem (directory `/usr/bin`)
- In BASH, la **prima parola** digitata nella **linea di comando** è il **nome del programma** da eseguire (dopo la pressione di `Invio`)
- Di **default**, BASH **cerca** il file col **nome corrispondente** in `/usr/bin`
  - **se esiste** $\rightarrow$ il programma viene **eseguito** avviene
  - **se non esiste** $\rightarrow$ messaggio d’**errore**

Eseguire i programmi – Parte 1

- Quasi tutti i **comandi** sono **“programmi”** (file “eseguibili”) già presenti nel filesystem (directory `/usr/bin`)
- In BASH, la **prima parola** digitata nella **linea di comando** è il **nome del programma** da eseguire (dopo la pressione di `Invio`)
- Di **default**, BASH **cerca** il file col **nome corrispondente** in `/usr/bin`
  - **se esiste** → il programma viene **eseguito** avviene
  - **se non esiste** → messaggio d’**errore**



Eseguire i programmi – Parte 2

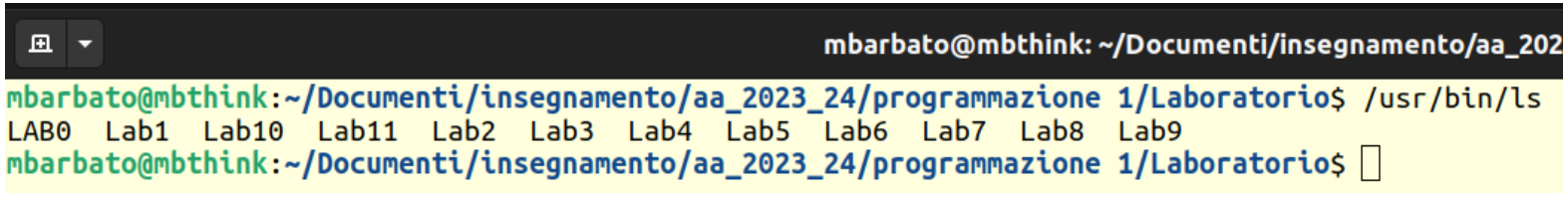
In generale, per eseguire **programmi** (anche in **altre directory**):

- specificare il **path assoluto o relativo** (`.` **obbligatorio**)

Eseguire i programmi – Parte 2

In generale, per eseguire **programmi** (anche in **altre directory**):

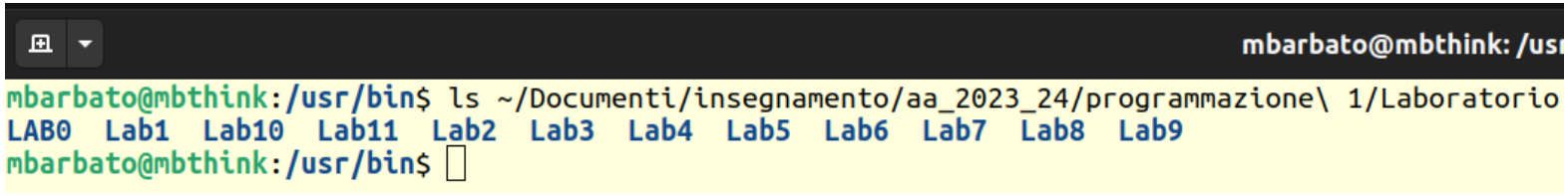
- specificare il **path assoluto o relativo** (`.` **obbligatorio**)



Eseguire i programmi – Parte 2

In generale, per eseguire **programmi** (anche in **altre directory**):

- specificare il **path assoluto o relativo** (`.` **obbligatorio**)



Eseguire i programmi – Parte 2

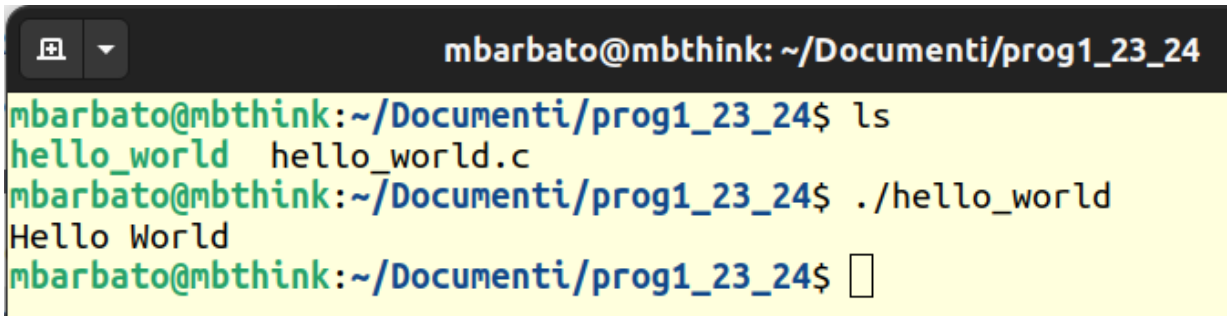
In generale, per eseguire **programmi** (anche in **altre directory**):

- specificare il **path assoluto o relativo** (`.` **obbligatorio**)
  - **Esempio:** se nella dir. di lavoro c'è un programma chiamato `hello_world`, si può eseguirlo con: `./hello_world`

Eseguire i programmi – Parte 2

In generale, per eseguire **programmi** (anche in **altre directory**):

- specificare il **path assoluto o relativo** (`.` **obbligatorio**)
  - **Esempio:** se nella dir. di lavoro c'è un programma chiamato `hello_world`, si può eseguirlo con: `./hello_world`



Eseguire i programmi – Parte 2

In generale, per eseguire **programmi** (anche in **altre directory**):

- specificare il **path assoluto o relativo** (`.` **obbligatorio**)
  - **Esempio:** se nella dir. di lavoro c'è un programma chiamato `hello_world`, si può eseguirlo con: `./hello_world`
  - se il file **non esiste** → messaggio d'**errore**
  - se il file **non è eseguibile** → messaggio d'**errore**

Programmi utili (in `/usr/bin`)

- `pluma` (editor di testo)
  - **Utilizzo base:** `pluma percorso/file` → apre il file per **inserimento di testo** non formattato
  - se `percorso/file` **non esiste** → viene **creato** (ma va **salvato**)
- `gcc` (compilatore `C`)
  - **Utilizzo base:**
    `gcc /percorso/sorgente -o /nome/eseguibile/desiderato`
  - **Esempio:** `gcc hello_world.c -o hello_world`

Esercizio (20 min)

- Scrivere nel file `hello_world.c` un codice `C` che stampi a schermo la frase “Ciao a tutti!”.
  Risolvere l’esercizio usando solo l’editor `pluma` e i comandi di BASH visti finora.
- Usando `gcc`, compilare il file sorgente del punto precedente facendo in modo che il file eseguibile sia chiamato `ciao_mondo`.

Esercizio (20 min)

- Scrivere nel file `hello_world.c` un codice `C` che stampi a schermo la frase “Ciao a tutti!”.
  Risolvere l’esercizio usando solo l’editor `pluma` e i comandi di BASH visti finora.
- Usando `gcc`, compilare il file sorgente del punto precedente facendo in modo che il file eseguibile sia chiamato `ciao_mondo`.

Soluzione

- nella **prossima lezione** (forse... :-P)

Copia e spostamento: i comandi `cp` e `mv`

- `cp percorso/file_semplice/ percorso/file` **copia** il primo file nel secondo file
  - secondo argomento è una **directory** $\rightarrow$ il primo file viene **copiato nella directory**
  - secondo argomento è un **file semplice** $\rightarrow$ viene **sovrascritto**
  - secondo argomento è un nome di file **inesistente** $\rightarrow$ viene **creato** con contenuto identico al primo
- `mv percorso/file_uno/ percorso/file_due`
  - Come `cp` ma il primo file **non** viene **mantenuto**

Pattern in BASH – Parte 1

- Un **pattern** è una parola speciale che **rappresenta** un **gruppo di parole**
- I seguenti pattern sono **sempre riconosciuti** da BASH:
  - `{<lista (separata da virgole)>}` → corrisponde a **tutte le stringhe** nella **lista**

Pattern in BASH – Parte 1

- Un **pattern** è una parola speciale che **rappresenta** un **gruppo di parole**
- I seguenti pattern sono **sempre riconosciuti** da BASH:
  - `{<lista (separata da virgole)>}` → corrisponde a **tutte le stringhe** nella **lista**
    - **Esempio:** `C{ia,ar}o` corrisponde alle due stringhe: `Ciao` e `Caro`

Pattern in BASH – Parte 1

- Un **pattern** è una parola speciale che **rappresenta** un **gruppo di parole**
- I seguenti pattern sono **sempre riconosciuti** da BASH:
  - `{<lista (separata da virgole)>}` → corrisponde a **tutte le stringhe** nella **lista**
    - **Esempio:** `C{ia,ar}o` corrisponde alle due stringhe: `Ciao` e `Caro`
  - `{x..y}` con `x` e `y` interi o caratteri singoli → corrisponde a tutte le stringhe `x`, `x+1`, `x+2`,..., `y`

Pattern in BASH – Parte 1

- Un **pattern** è una parola speciale che **rappresenta** un **gruppo di parole**
- I seguenti pattern sono **sempre riconosciuti** da BASH:
  - `{<lista (separata da virgole)>}` → corrisponde a **tutte le stringhe** nella **lista**
    - **Esempio:** `C{ia,ar}o` corrisponde alle due stringhe: `Ciao` e `Caro`
  - `{x..y}` con `x` e `y` interi o caratteri singoli → corrisponde a tutte le stringhe `x`, `x+1`, `x+2`,..., `y`
    - **Esempio:** `a{1..4}f` corrisponde alle stringhe `a1f`, `a2f`, `a3f`, `a4f`
    - **Esempio:** `5{a..d}f` corrisponde alle stringhe `5af`, `5bf`, `5cf`, `5df`

Pattern in BASH – Parte 1

- Un **pattern** è una parola speciale che **rappresenta** un **gruppo di parole**
- I seguenti pattern sono **sempre riconosciuti** da BASH:
  - `{<lista (separata da virgole)>}` → corrisponde a **tutte le stringhe** nella **lista**
    - **Esempio:** `C{ia,ar}o` corrisponde alle due stringhe: `Ciao` e `Caro`
  - `{x..y}` con `x` e `y` interi o caratteri singoli → corrisponde a tutte le stringhe `x`, `x+1`, `x+2`,..., `y`
    - **Esempio:** `a{1..4}f` corrisponde alle stringhe `a1f`, `a2f`, `a3f`, `a4f`
    - **Esempio:** `5{a..d}f` corrisponde alle stringhe `5af`, `5bf`, `5cf`, `5df`
  - `{x..y..incr}` → come il precedente, con incremento pari a `incr`

Pattern in BASH – Parte 1

- Un **pattern** è una parola speciale che **rappresenta** un **gruppo di parole**
- I seguenti pattern sono **sempre riconosciuti** da BASH:
  - `{<lista (separata da virgole)>}` → corrisponde a **tutte le stringhe** nella **lista**
    - **Esempio:** `C{ia,ar}o` corrisponde alle due stringhe: `Ciao` e `Caro`
  - `{x..y}` con `x` e `y` interi o caratteri singoli → corrisponde a tutte le stringhe `x`, `x+1`, `x+2`,..., `y`
    - **Esempio:** `a{1..4}f` corrisponde alle stringhe `a1f`, `a2f`, `a3f`, `a4f`
    - **Esempio:** `5{a..d}f` corrisponde alle stringhe `5af`, `5bf`, `5cf`, `5df`
  - `{x..y..incr}` → come il precedente, con incremento pari a `incr`
    - **Esempio:** `a{1..6..2}f` corrisponde a `a1f`, `a3f`, `a5f`
    - **Esempio:** `5{a..d..2}f` corrisponde a `5af`, `5cf`

Pattern in BASH – Parte 2

- BASH riconosce i seguenti pattern solo se corrispondenti a **file esistenti** (nel path specificato):
  - `*` (asterisco) $\rightarrow$ rappresenta **qualsiasi sequenza** di caratteri (anche **vuota**)

Pattern in BASH – Parte 2

- BASH riconosce i seguenti pattern solo se corrispondenti a **file esistenti** (nel path specificato):
  - `*` (asterisco) $\rightarrow$ rappresenta **qualsiasi sequenza** di caratteri (anche **vuota**)
    - **Esempio:** `ls *.txt` elenca tutti i file esistenti (nella dir. di lavoro) con estensione "`.txt`"

Pattern in BASH – Parte 2

- BASH riconosce i seguenti pattern solo se corrispondenti a **file esistenti** (nel path specificato):
  - `*` (asterisco) → rappresenta **qualsiasi sequenza** di caratteri (anche **vuota**)
    - **Esempio:** `ls *.txt` elenca tutti i file esistenti (nella dir. di lavoro) con estensione "`.txt`"
  - `?` (punto interrogativo) → rappresenta un **singolo carattere** qualsiasi

Pattern in BASH – Parte 2

- BASH riconosce i seguenti pattern solo se corrispondenti a **file esistenti** (nel path specificato):
  - `*` (asterisco) → rappresenta **qualsiasi sequenza** di caratteri (anche **vuota**)
    - **Esempio:** `ls *.txt` elenca tutti i file esistenti (nella dir. di lavoro) con estensione "`.txt`"
  - `?` (punto interrogativo) → rappresenta un **singolo carattere** qualsiasi
    - **Esempio:** `ls ?olo` elenca i file esistenti (nella dir. di lavoro) `aolo`, `bolo`, `colo`, `dolo`, ecc...

Pattern in BASH – Parte 2

- BASH riconosce i seguenti pattern solo se corrispondenti a **file esistenti** (nel path specificato):
  - `*` (asterisco) → rappresenta **qualsiasi sequenza** di caratteri (anche **vuota**)
    - **Esempio:** `ls *.txt` elenca tutti i file esistenti (nella dir. di lavoro) con estensione "`.txt`"
  - `?` (punto interrogativo) → rappresenta un **singolo carattere** qualsiasi
    - **Esempio:** `ls ?olo` elenca i file esistenti (nella dir. di lavoro) `aolo`, `bolo`, `colo`, `dolo`, ecc... ma anche `1olo`, `2olo`, ecc...

Pattern in BASH – Parte 2

- BASH riconosce i seguenti pattern solo se corrispondenti a **file esistenti** (nel path specificato):
  - `*` (asterisco) → rappresenta **qualsiasi sequenza** di caratteri (anche **vuota**)
    - **Esempio:** `ls *.txt` elenca tutti i file esistenti (nella dir. di lavoro) con estensione "`.txt`"
  - `?` (punto interrogativo) → rappresenta un **singolo carattere** qualsiasi
    - **Esempio:** `ls ?olo` elenca i file esistenti (nella dir. di lavoro) `aolo`, `bolo`, `colo`, `dolo`, ecc... ma anche `1olo`, `2olo`, ecc...
  - `[<sequenza di caratteri>]` (caratteri tra quadre) → rappresenta un **singolo carattere** tra quelli **specificati** nella sequenza

Pattern in BASH – Parte 2

- BASH riconosce i seguenti pattern solo se corrispondenti a **file esistenti** (nel path specificato):
  - `*` (asterisco) → rappresenta **qualsiasi sequenza** di caratteri (anche **vuota**)
    - **Esempio:** `ls *.txt` elenca tutti i file esistenti (nella dir. di lavoro) con estensione "`.txt`"
  - `?` (punto interrogativo) → rappresenta un **singolo carattere** qualsiasi
    - **Esempio:** `ls ?olo` elenca i file esistenti (nella dir. di lavoro) `aolo`, `bolo`, `colo`, `dolo`, ecc... ma anche `1olo`, `2olo`, ecc...
  - `[<sequenza di caratteri>]` (caratteri tra quadre) → rappresenta un **singolo carattere** tra quelli **specificati** nella sequenza
    - `ls [m,p,s,v]olo` elenca i file esistenti (nella dir. di lavoro) `molo`, `polo`, `solo`, `volo`

Pattern in BASH – Parte 2

- BASH riconosce i seguenti pattern solo se corrispondenti a **file esistenti** (nel path specificato):
  - `*` (asterisco) → rappresenta **qualsiasi sequenza** di caratteri (anche **vuota**)
    - **Esempio:** `ls *.txt` elenca tutti i file esistenti (nella dir. di lavoro) con estensione "`.txt`"
  - `?` (punto interrogativo) → rappresenta un **singolo carattere** qualsiasi
    - **Esempio:** `ls ?olo` elenca i file esistenti (nella dir. di lavoro) `aolo`, `bolo`, `colo`, `dolo`, ecc... ma anche `1olo`, `2olo`, ecc...
  - `[<sequenza di caratteri>]` (caratteri tra quadre) → rappresenta un **singolo carattere** tra quelli **specificati** nella sequenza
    - `ls [m,p,s,v]olo` elenca i file esistenti (nella dir. di lavoro) `molo`, `polo`, `solo`, `volo`
  - `[!<sequenza di caratteri>]` (punto esclamativo seguito da caratteri tra quadre) → rappresenta tutti i **singoli caratteri diversi** da quelli **specificati** nella sequenza

Esercizio (15 min)

- In `prog1_23_24/lab0` (primo esercizio), aggiungere la directory `tanti_file`. Creare al suo interno tutti i possibili file il cui nome rispetti il seguente formato:
  - la prima lettera è un carattere tra 'a' e 'd'
  - la seconda lettera è un intero tra 1 e 9
  - la terza lettera è un intero pari tra 1 e 9
  - la quarta lettera è un intero multiplo di 3 tra 1 e 9
  - la quinta lettera è un carattere tra 'a' e 'd'
- creare una nuova directory in `prog1_23_24/lab0` chiamata `alcuni_file`
- copiare da `tanti_file` ad `alcuni_file`:
  - i file appena creati aventi un multiplo di 4 come seconda lettera del nome
  - i file che non hanno '6' come terza o quarta lettera del nome

Esercizio (15 min)

- In `prog1_23_24/lab0` (primo esercizio), aggiungere la directory `tanti_file`. Creare al suo interno tutti i possibili file il cui nome rispetti il seguente formato:
  - la prima lettera è un carattere tra 'a' e 'd'
  - la seconda lettera è un intero tra 1 e 9
  - la terza lettera è un intero pari tra 1 e 9
  - la quarta lettera è un intero multiplo di 3 tra 1 e 9
  - la quinta lettera è un carattere tra 'a' e 'd'
- creare una nuova directory in `prog1_23_24/lab0` chiamata `alcuni_file`
- copiare da `tanti_file` ad `alcuni_file`:
  - i file appena creati aventi un multiplo di 4 come seconda lettera del nome
  - i file che non hanno '6' come terza o quarta lettera del nome

Possibile soluzione:

- `mkdir ~/prog1_23_24/lab0/tanti_file`
- `cd ~/prog1_23_24/lab0/tanti_file`
- `touch {a..d}{1..9}{2..9..2}{3..9..3}{a..d}`

Esercizio (15 min)

- In `prog1_23_24/lab0` (primo esercizio), aggiungere la directory `tanti_file`. Creare al suo interno tutti i possibili file il cui nome rispetti il seguente formato:
  - la prima lettera è un carattere tra 'a' e 'd'
  - la seconda lettera è un intero tra 1 e 9
  - la terza lettera è un intero pari tra 1 e 9
  - la quarta lettera è un intero multiplo di 3 tra 1 e 9
  - la quinta lettera è un carattere tra 'a' e 'd'
- creare una nuova directory in `prog1_23_24/lab0` chiamata `alcuni_file`
- copiare da `tanti_file` ad `alcuni_file`:
  - i file appena creati aventi un multiplo di 4 come seconda lettera del nome
  - i file che non hanno '6' come terza o quarta lettera del nome

Possibile soluzione:

- `mkdir ~/prog1_23_24/lab0/tanti_file`
- `cd ~/prog1_23_24/lab0/tanti_file`
- `touch {a..d}{1..9}{2..9..2}{3..9..3}{a..d}`
- `mkdir ../alcuni_file`

Esercizio (15 min)

- In `prog1_23_24/lab0` (primo esercizio), aggiungere la directory `tanti_file`. Creare al suo interno tutti i possibili file il cui nome rispetti il seguente formato:
  - la prima lettera è un carattere tra 'a' e 'd'
  - la seconda lettera è un intero tra 1 e 9
  - la terza lettera è un intero pari tra 1 e 9
  - la quarta lettera è un intero multiplo di 3 tra 1 e 9
  - la quinta lettera è un carattere tra 'a' e 'd'
- creare una nuova directory in `prog1_23_24/lab0` chiamata `alcuni_file`
- copiare da `tanti_file` ad `alcuni_file`:
  - i file appena creati aventi un multiplo di 4 come seconda lettera del nome
  - i file che non hanno '6' come terza o quarta lettera del nome

Possibile soluzione:

- `mkdir ~/prog1_23_24/lab0/tanti_file`
- `cd ~/prog1_23_24/lab0/tanti_file`
- `touch {a..d}{1..9}{2..9..2}{3..9..3}{a..d}`
- `mkdir ../alcuni_file`
- `cp ?[4,8]* ../alcuni_file`
- `cp *[!6][!6]?  ../alcuni_file`

Redirezione dell'input e dell'output dei comandi

Redirezione output

- Molti comandi producono un **output**: "scrivono a schermo" il risultato
  - **Esempio:**
    `ls` → **mostra** a schermo il contenuto della cartella

Redirezione dell'input e dell'output dei comandi

Redirezione output

- Molti comandi producono un **output**: "scrivono a schermo" il risultato
  - **Esempio:**
    `ls` → **mostra** a schermo il contenuto della cartella
- Per **sovrascrivere** l'output **su un file** (anche nuovo):
  - `nome_comando > nome_file`

Redirezione dell'input e dell'output dei comandi

Redirezione output

- Molti comandi producono un **output**: "scrivono a schermo" il risultato
  - **Esempio:**
    `ls` → **mostra** a schermo il contenuto della cartella
- Per **sovrascrivere** l'output **su un file** (anche nuovo):
  - `nome_comando > nome_file`
- Per **aggiungere** l'output **su un file** (anche nuovo):
  - `nome_comando >> nome_file`

Redirezione input

- Vari comandi leggono un **input**
  - **Esempio:**
    `cat <nome file>` → **scrive** a schermo il **contenuto** di `<nome file>`

Redirezione dell'input e dell'output dei comandi

Redirezione output

- Molti comandi producono un **output**: "scrivono a schermo" il risultato
  - **Esempio:**
    `ls` → **mostra** a schermo il contenuto della cartella
- Per **sovrascrivere** l'output **su un file** (anche nuovo):
  - `nome_comando > nome_file`
- Per **aggiungere** l'output **su un file** (anche nuovo):
  - `nome_comando >> nome_file`

Redirezione input

- Vari comandi leggono un **input**
  - **Esempio:**
    `cat <nome file>` → **scrive** a schermo il **contenuto** di `<nome file>`
- Per **leggere** l'input **da un file**:
  - `nome_comando < nome_file`

Esempio sulla redirezione dell'output

Supponiamo che `new_file.txt` non esista:

- `ls > new_file.txt`: crea `new_file.txt` e ci scrive dentro l'output di `ls`
- `ls >> new_file.txt`: aggiunge l'output di `ls` al file `new_file.txt`
- `ls > new_file.txt`: riscrive l'output di `ls` al file `new_file.txt`