

A hand-drawn decorative border in black ink, featuring various loops, swirls, and flourishes that frame the central text. The border is irregular and artistic, with some elements resembling calligraphy or doodles.

# **Introduzione al Terminale**

# Obiettivi della Lezione

- **Comprendere il concetto di terminale:** Cos'è il terminale e perché è uno strumento fondamentale per sviluppatori e programmatori.
- **Familiarizzare con i comandi base:** Saper navigare nel filesystem, gestire file e directory.
- Preannuncio che sarà una bella carrellata di roba!
  - Non importa ricordare ogni comando



# Cos'è il Terminale e Perché Usarlo?

## Definizione:

- Il terminale (o shell) è un'interfaccia testuale che consente di interagire direttamente con il sistema operativo, eseguendo comandi e script.



# Cos'è il Terminale e Perché Usarlo?

Vantaggi dell'uso del terminale:

- **Efficienza e velocità:** Comandi brevi per operazioni complesse.
- **Automazione:** Possibilità di creare script per operazioni ripetitive e complesse.
- **Accesso a strumenti avanzati:** Molti strumenti di sviluppo, come Git, interagiscono principalmente tramite terminale.
- **Controllo preciso:** Permette di comprendere meglio il funzionamento interno del sistema.

# Panoramica dei Terminali negli OS

## Ambienti Unix-based (Linux e macOS)

- **Terminale e Shell:**
  - Di default, Linux e macOS utilizzano shell come **Bash** o, in alcuni casi, **Zsh**.
- **Caratteristiche:**
  - Ampia disponibilità di strumenti nativi e compatibilità con la maggior parte dei comandi esposti nei tutorial.
  - L'uso del comando **man** (manuale) per ottenere informazioni dettagliate su ogni comando.

# Panoramica dei Terminali negli OS

## Ambiente Windows

- **Opzioni disponibili:**

- **Prompt dei Comandi (cmd):** L'interfaccia classica, con sintassi e comandi differenti (es. `dir` al posto di `ls`).
- **PowerShell:** Più potente e versatile rispetto al cmd, offre cmdlet e funzionalità avanzate per la gestione del sistema.
- **Windows Terminal:** Un'app moderna che permette di utilizzare in parallelo diverse shell (cmd, PowerShell, Git Bash o WSL).

# Panoramica dei Terminali negli OS

## Ambiente Windows

- **Soluzioni ibride:**
  - **Git Bash:** Installato insieme a Git per Windows, offre un ambiente simile a quello Unix e rende più facile il passaggio tra sistemi.
  - **Windows Subsystem for Linux (WSL):** Consente di eseguire una distribuzione Linux nativa all'interno di Windows, migliorando la compatibilità con script e comandi tipici di Linux.



# Terminologia di Base

**Shell:** Il programma che interpreta i comandi inseriti nel terminale

**Command Line:** La riga dove si digitano i comandi

**Prompt:** Il simbolo che indica che il terminale è pronto a ricevere comandi

**Working Directory:** La cartella in cui si sta lavorando attualmente

**Path:** Il percorso che identifica la posizione di un file o cartella

**Flag/Option:** Modificatori che alterano il comportamento di un comando



# Navigare nel Filesystem



Visualizzare la directory corrente:

- `pwd` (Print Working Directory) - Mostra il percorso completo della directory attuale

Listare i contenuti di una directory:

- `ls` (Linux/macOS) o `dir` (Windows) - Elenca file e cartelle
- `ls -l` - Formato lungo con dettagli (permessi, dimensioni, date)
- `ls -a` - Mostra anche i file nascosti (quelli che iniziano con `.`)
- `ls -la` - Combina le opzioni precedenti

Cambiare directory:

- `cd percorso` - Si sposta nella directory specificata
- `cd ..` - Sale di un livello (directory padre)
- `cd ~` o semplicemente `cd` - Torna alla home directory
- `cd -` - Torna alla directory precedente

# Percorsi Assoluti e Relativi

## Percorsi Assoluti:

- Partono dalla radice del filesystem
- Iniziano sempre con " / " (Linux/macOS)
  - a. o con una lettera di unità come "C:\ " (Windows)
- Esempio: `/home/utente/documenti/file.txt` o `C:\Users\utente\Documents\file.txt`
- Funzionano indipendentemente dalla posizione corrente

## Percorsi Relativi:

- Relativi alla directory di lavoro corrente
- Non iniziano con / o una lettera di unità
- Esempio: `documenti/file.txt` (dalla directory home)
  - a. Più concisi ma dipendono dalla posizione attuale



# Percorsi Assoluti e Relativi

Simboli speciali nei percorsi:

- `.` - Directory corrente
- `..` - Directory padre
- `~` - Home directory dell'utente (Linux/macOS)



# Manipolazione di File e Directory

## Creazione:

- `mkdir nome_directory` - Crea una nuova directory
- `touch nome_file` (Linux/macOS) - Crea un file vuoto o aggiorna timestamp
- `echo > nome_file` (Windows) - Crea un file vuoto

## Copia:

- `cp origine destinazione` (Linux/macOS) - Copia file
- `cp -r origine destinazione` - Copia directory ricorsivamente
- `copy origine destinazione` (Windows)

# Manipolazione di File e Directory

## Spostamento/Rinomina:

- `mv origine destinazione` (Linux/macOS) - Sposta o rinomina
- `move origine destinazione` o `ren origine destinazione` (Windows)

## Eliminazione:

- `rm nome_file` (Linux/macOS) - Elimina un file
- `rm -r nome_directory` - Elimina una directory e il suo contenuto
- `del nome_file` o `erase nome_file` (Windows)
- `rmdir nome_directory` o `rd nome_directory` (Windows)

(occhio a non usare `rm -rf /*` !!!!!)

# Visualizzazione del Contenuto dei File

## Visualizzazione completa:

- `cat nome_file` (Linux/macOS) - Mostra l'intero contenuto
- `type nome_file` (Windows)

## Visualizzazione parziale:

- `head nome_file` - Mostra le prime 10 righe (default)
- `head -n 20 nome_file` - Mostra le prime 20 righe
- `tail nome_file` - Mostra le ultime 10 righe
- `tail -n 20 nome_file` - Mostra le ultime 20 righe
- `tail -f nome_file` - Mostra le ultime righe e continua a mostrare gli aggiornamenti (utile per i log)

# Visualizzazione del Contenuto dei File

Visualizzazione interattiva:

- `less nome_file` - Visualizzatore interattivo (navigabile con frecce, PgUp/PgDown)
  - Premi `q` per uscire
  - Premi `/` seguito da testo per cercare
  - Premi `n` per trovare l'occorrenza successiva



# Visualizzazione del Contenuto dei File

Editor di testo da terminale:

- `nano nome_file` - Editor semplice, ideale per principianti
- `vim nome_file` o `vi nome_file` - Editor potente ma con curva di apprendimento ripida
- `notepad nome_file` (Windows)



# Wildcards e Pattern Matching

Le wildcards sono caratteri speciali che permettono di selezionare più file contemporaneamente

**Caratteri jolly principali:**

- **\*** - Corrisponde a qualsiasi sequenza di caratteri
  - `ls *.txt` - Elenca tutti i file con estensione .txt
  - `rm test*` - Elimina tutti i file che iniziano con "test"
- **?** - Corrisponde a un singolo carattere qualsiasi
  - `ls file?.txt` - Trova file1.txt, fileA.txt, ma non file10.txt
- **[]** - Corrisponde a uno qualsiasi dei caratteri tra parentesi
  - `ls file[123].txt` - Trova file1.txt, file2.txt, file3.txt
  - `ls file[a-z].txt` - Trova file con un carattere minuscolo dopo "file"

# Wildcards e Pattern Matching



Esempi pratici:

- `cp *.jpg /backup` - Copia tutti i file JPEG nella directory /backup
- `ls report-????-???.pdf` - Trova report con pattern specifico di date
- `rm -i test[0-9].*` - Elimina con conferma i file test seguiti da un numero

# Reindirizzamento dell'Input/Output

Il reindirizzamento permette di controllare da dove provengono gli input e dove vanno gli output dei comandi:

## Reindirizzamento dell'output:

- `comando > file.txt` - Reindirizza l'output in un file (sovrascrive)
  - `ls > elenco_file.txt` - Salva l'elenco dei file in un documento
- `comando >> file.txt` - Aggiunge l'output alla fine del file
  - `echo "nuova riga" >> note.txt` - Aggiunge testo a un file esistente

# Reindirizzamento dell'Input/Output

## Reindirizzamento dell'input:

- `comando < file.txt` - Usa il contenuto del file come input
  - `sort < nomi.txt` - Ordina il contenuto di nomi.txt

## Reindirizzamento degli errori:

- `comando 2> errori.log` - Reindirizza solo gli errori
- `comando > output.txt 2> errori.log` - Separa output normale ed errori
- `comando > output.txt 2>&1` - Reindirizza sia output che errori nello stesso file

# Pipe: Combinare i Comandi

Le pipe (|) permettono di collegare l'output di un comando all'input di un altro, creando sequenze potenti:

**Sintassi:** `comando1 | comando2`

**Esempi comuni:**

- `ls -la | grep "pdf"` - Elenca i file e filtra solo quelli contenenti "pdf"
- `cat file.txt | sort` - Legge un file e ne ordina il contenuto
- `ps aux | grep firefox` - Mostra i processi e filtra solo quelli relativi a Firefox
- `history | grep git` - Cerca comandi git nella cronologia

# Pipe: Combinare i Comandi

Le pipe (|) permettono di collegare l'output di un comando all'input di un altro, creando sequenze potenti:

**Sintassi:** `comando1 | comando2`

**Esempi avanzati:**

- `ls -la | sort -k5 -n` - Elenca i file e li ordina per dimensione
- `cat log.txt | grep ERROR | wc -l` - Conta il numero di errori in un log
- `find . -name "*.txt" | xargs cat | wc -l` - Conta le righe in tutti i file .txt

# Comando Find: Cercare File

Il comando `find` è uno strumento potente per cercare file e directory in base a vari criteri:

Sintassi base: `find [percorso] [espressione]`

Esempi di ricerca per nome:

- `find . -name "documento.txt"` - Cerca un file specifico
- `find . -name "*.pdf"` - Cerca tutti i file PDF
- `find . -iname "*.jpg"` - Come sopra ma ignora maiuscole/minuscole

Ricerca per dimensione:

- `find . -size +10M` - File più grandi di 10 MB
- `find . -size -1k` - File più piccoli di 1 KB
- `find . -empty` - File o directory vuoti

# Comando Find: Cercare File

Il comando `find` è uno strumento potente per cercare file e directory in base a vari criteri:

Sintassi base: `find [percorso] [espressione]`

Ricerca per tipo:

- `find . -type f` - Cerca solo file
- `find . -type d` - Cerca solo directory
- `find . -type l` - Cerca solo link simbolici

Ricerca per tempo:

- `find . -mtime -7` - File modificati negli ultimi 7 giorni
- `find . -mtime +30` - File non modificati da più di 30 giorni
- `find . -mmin -60` - File modificati nell'ultima ora



# Grep: Ricerca di Testo

`grep` è uno strumento potente per cercare pattern di testo all'interno di file:

Sintassi base: `grep [opzioni] pattern [file...]`

Esempi base:

- `grep "errore" log.txt` - Trova le righe contenenti "errore"
- `grep -i "warning" *.log` - Trova "warning" ignorando maiuscole/minuscole
- `grep -r "TODO" .` - Cerca ricorsivamente in tutti i file della directory

# Grep: Ricerca di Testo

## Opzioni utili:

- `-i` - Ignora maiuscole/minuscole
- `-v` - Inverte la selezione (righe che NON contengono il pattern)
- `-n` - Mostra i numeri di riga
- `-l` - Mostra solo i nomi dei file (non il contenuto)
- `-c` - Conta le occorrenze in ogni file

## Pattern avanzati:

- `grep -E "error|warning" log.txt` - Cerca più pattern (OR)
- `grep -E "^[A-Z]" file.txt` - Righe che iniziano con lettera maiuscola
- `grep -E "\b[0-9]{3}-[0-9]{4}\b" contatti.txt` - Cerca numeri telefonici

# Gestione dei Processi

I terminali permettono di gestire i processi in esecuzione:

## Esecuzione in background:

- `comando &` - Esegue il comando in background
- `nohup comando &` - Esegue in background anche dopo la chiusura del terminale

## Ctrl+Z e controllo dei job:

- `Ctrl+Z` - Sospende il processo in esecuzione
- `jobs` - Elenca i processi sospesi o in background
- `fg` - Riprende l'ultimo processo sospeso
- `fg %n` - Riprende il processo numero n
- `bg` - Continua l'ultimo processo sospeso in background

# Gestione dei Processi

I terminali permettono di gestire i processi in esecuzione:

## Visualizzare processi:

- `ps` - Elenca i processi dell'utente
- `ps aux` - Elenca tutti i processi in esecuzione
- `top` o `htop` o `bttop` (da installare) - Visualizzazione interattiva dei processi

## Terminare processi:

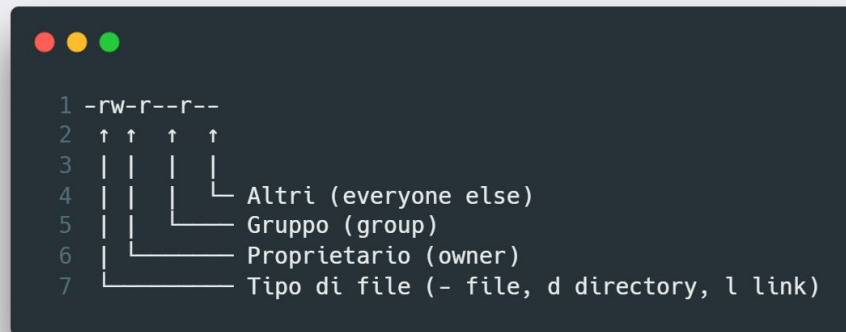
- `kill PID` - Invia un segnale di terminazione al processo
- `kill -9 PID` - Forza la chiusura del processo
- `killall nome_processo` - Termina tutti i processi con un certo nome

# Permessi dei File (Unix)

Nei sistemi Unix-like, ogni file ha permessi che controllano chi può leggerlo, scriverlo o eseguirlo:

**Visualizzazione dei permessi:**

- `ls -l file.txt` - Mostra i permessi in formato simbolico



```
1 -rw-r--r--
2 ↑ ↑ ↑ ↑
3 | | | |
4 | | | | Altri (everyone else)
5 | | | | Gruppo (group)
6 | | | | Proprietario (owner)
7 | | | | Tipo di file (- file, d directory, l link)
```

# Permessi dei File (Unix)

Modificare i permessi (modo simbolico):

- `chmod u+x script.sh` - Aggiunge permesso di esecuzione al proprietario
- `chmod g+w file.txt` - Aggiunge permesso di scrittura al gruppo
- `chmod o-r file.txt` - Rimuove permesso di lettura agli altri
- `chmod a+x file` - Aggiunge permesso di esecuzione a tutti

Modificare i permessi (modo ottale):

- `chmod 755 script.sh` - Imposta rwx per il proprietario, r-x per gruppo e altri
- `chmod 644 file.txt` - Imposta rw- per il proprietario, r-- per gruppo e altri

Cambiare proprietario/gruppo:

- `chown utente file.txt` - Cambia il proprietario
- `chgrp gruppo file.txt` - Cambia il gruppo
- `chown utente:gruppo file.txt` - Cambia entrambi

# Variabili d'Ambiente

Le variabili d'ambiente sono impostazioni globali che influenzano il comportamento del sistema e dei programmi:

**Visualizzare variabili d'ambiente:**

- `env` o `printenv` - Mostra tutte le variabili d'ambiente
- `echo $NOME_VARIABILE` - Mostra il valore di una variabile specifica

**Variabili d'ambiente importanti:**

- `PATH` - Elenco delle directory dove cercare i comandi
- `HOME` - Percorso della home directory dell'utente
- `USER` o `USERNAME` - Nome dell'utente corrente
- `SHELL` - Shell in uso (Linux/macOS)

**Impostare variabili temporaneamente:**

- `VARIABILE=valore` - Imposta per il comando attuale
- `export VARIABILE=valore` - Imposta per tutti i processi figli

# Alias e Funzioni della Shell

Gli alias e le funzioni permettono di creare comandi personalizzati:

Alias:

- `alias` - Mostra tutti gli alias definiti
- `alias ls='ls --color=auto'` - Crea un alias per un comando
- `unalias nome_alias` - Rimuove un alias

Esempio di alias utili:

- `alias ll='ls -la'` - Shortcut per listare file con dettagli
- `alias c='clear'` - Pulisce lo schermo
- `alias ..='cd ..'` - Risale di una directory
- `alias gs='git status'` - Shortcut per Git

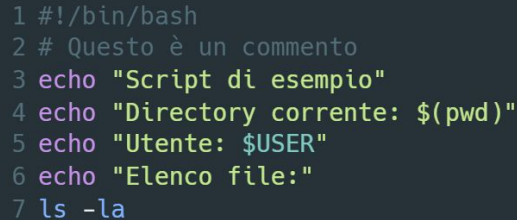


# Script di Shell

Gli script di shell permettono di automatizzare sequenze di comandi:

## Creazione di uno script:

1. Creare un file con estensione `.sh` (convenzione)
2. Aggiungere lo "shebang" come prima riga: `#!/bin/bash`
3. Aggiungere i comandi desiderati
4. Rendere lo script eseguibile: `chmod +x script.sh`
5. Eseguire con `./script.sh`

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It displays a shell script with seven lines of code, where the first three lines are commented out with a '#' character. The code uses 'echo' for output and 'ls' for listing files.

```
1 #!/bin/bash
2 # Questo è un commento
3 echo "Script di esempio"
4 echo "Directory corrente: $(pwd)"
5 echo "Utente: $USER"
6 echo "Elenco file:"
7 ls -la
```

# Script di Shell

Gli script di shell sono un vero e proprio mondo a parte, online trovate script shell per qualsiasi cosa!

## Elementi base degli script:

- Variabili: `NOME="Valore"`
- Argomenti: `$1`, `$2`, `$@` (tutti gli argomenti)
- Condizionali: `if...then...else...fi`
- Cicli: `for`, `while`
- Funzioni

```
1 #!/bin/bash
2 if [ -f "$1" ]; then
3     echo "Il file $1 esiste"
4 else
5     echo "Il file $1 non esiste"
6 fi
```

L'opzione `-f` in questo script verifica due condizioni:

1. Che il percorso esista
2. Che sia un file regolare (non una directory, un link, ecc.)

Bash offre diversi operatori per testare i file:

- `-f`: file esistente e regolare
- `-d`: directory esistente
- `-e`: percorso esistente (tipo non importa)
- `-L`: link simbolico
- `-r`: file esistente e leggibile
- `-w`: file esistente e scrivibile
- `-x`: file esistente ed eseguibile

# SSH e Connessioni Remote


SSH (Secure Shell) permette di connettersi e controllare macchine remote:

## Connessione base:

- `ssh username@hostname` - Connessione a un server remoto
- `ssh username@hostname -p 2222` - Specifica una porta diversa dalla 22
- `ssh username@hostname "cd /var && ls -la"` - Più comandi insieme

## Trasferimento file con SCP:

- `scp file.txt username@hostname:/percorso/` - Copia un file locale su server remoto
- `scp username@hostname:/percorso/file.txt ./` - Copia un file remoto in locale
- `scp -r directory/ username@hostname:/percorso/` - Copia directory ricorsivamente



# Curl e Wget

Strumenti per interagire con risorse web dal terminale:

**curl:**

- `curl https://example.com` - Scarica e mostra il contenuto di una pagina web
- `curl -o file.html https://example.com` - Salva il contenuto in un file

**wget:**

- `wget https://example.com` - Scarica una pagina web
- `wget -O file.html https://example.com` - Specifica il nome del file
- `wget -r -np https://example.com` - Scarica un sito ricorsivamente
- `wget -c https://example.com/file.zip` - Riprende un download interrotto

# Monitoraggio del Sistema

## Informazioni su disco e file:

- `df -h` - Spazio utilizzato e disponibile sui dischi
- `du -sh directory` - Dimensione di una directory
- `du -sh *` - Dimensione di tutti i file e directory
- `ncdu` - Visualizzatore interattivo dell'utilizzo del disco

## Processi e risorse:

- `htop` - Informazioni in tempo reale su processi e risorse con interfaccia interattiva
- `bttop` - Versione migliorata di top con interfaccia interattiva
- `ps aux` - Elenco dettagliato di tutti i processi
- `free -h` - Utilizzo della memoria
- `uptime` - Tempo di attività del sistema e carico medio

## Rete:

- `ifconfig` o `ip addr` - Informazioni sulle interfacce di rete
- `ping hostname` - Verifica connettività verso un host

# Tips & Tricks

## Navigazione efficiente:

- Tab completion: Premi Tab per completare nomi di file e comandi

## Gestione degli errori:

- `comando || echo "Errore"` - Esegue la seconda parte solo se la prima fallisce
- `comando && echo "Successo"` - Esegue la seconda parte solo se la prima ha successo
- `comando ; echo "Sempre"` - Esegue entrambe le parti indipendentemente

## Editing della linea di comando:

- `Ctrl+A` / `Home` - Vai all'inizio della riga
- `Ctrl+E` / `End` - Vai alla fine della riga
- `Ctrl+K` - Taglia dalla posizione corrente fino alla fine
- `Ctrl+U` - Taglia dall'inizio alla posizione corrente
- `Ctrl+W` - Taglia la parola precedente
- `Ctrl+Y` - Incolla il testo tagliato

# Tips & Tricks

Il terminale può essere personalizzato per migliorare produttività e comfort

Framework di personalizzazione:

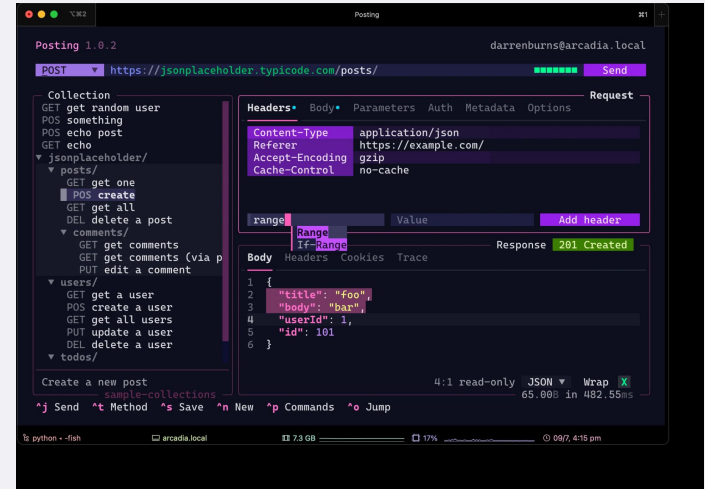
- Oh My Zsh (per Zsh): Temi e plugin
- Oh My Posh (per PowerShell): Prompt personalizzabili
- Bash-it (per Bash): Collezione di script e alias

Qualche risorsa utile:

- <https://www.reddit.com/r/commandline/>

```
ohmyzsh demo

+ projects omz theme use cloud
+ projects take omz-demo && git init
Reinitialized existing Git repository in /Users/robbyrussell/projects/omz-demo/.git/
- omz-demo [main] echo "TODO: This is my new README" > README.md
- omz-demo [main] % git add README.md
- omz-demo [main] % git commit -m "Adding a README to new repo" --quiet
- omz-demo [main] echo "Wow, Oh My Zsh looks neat" >> README.md
- omz-demo [main] % git add -p
diff --git a/README.md b/README.md
index fc97e80..b0939f1 100644
--- a/README.md
+++ b/README.md
@@ -1,2 @@
+TODO: This is my new README
+Wow, Oh My Zsh looks neat
(1/1) Stage this hunk [y,n,q,a,d,e,f]? y
- omz-demo [main] % git commit -m "Updating README in the repo" --quiet
- omz-demo [main] git checkout -b feature/openai-integration
fatal: a branch named 'feature/openai-integration' already exists
- omz-demo [main] git checkout main
Already on 'main'
- omz-demo [main] ..
- projects
```



# Esercizi - BASE

1. Crea una struttura di cartelle per un progetto web che includa le directory "css", "js", "images" e un file index.html nella directory principale.
2. Crea un file chiamato "note.txt" con alcune frasi a tua scelta, poi conta quante parole contiene.
3. Trova tutti i file con estensione .txt nella directory corrente e nelle sue sottodirectory (unix).
4. Crea uno script bash chiamato "saluto.sh" che stampi "Ciao, mondo!" e rendilo eseguibile.
5. Crea un alias temporaneo chiamato "ll" che esegua il comando "ls -la" e testalo.





# Esercizi - INTERMEDI



1. Elenca tutti i processi in esecuzione, filtra solo quelli relativi al browser e conta quanti sono (wc per contare).
2. Crea tre file .txt con contenuto diverso, poi cerca in tutti i file quelli che contengono la parola "progetto".
3. Crea uno script bash che accetti un parametro (nome di una directory) e mostri i 5 file più grandi in quella directory.
4. Crea uno script che visualizzi alcune informazioni sul sistema usando variabili d'ambiente.



FINE



— WWW.DRACO.ME —