Laboratorio di Sistemi Operativi

Daniele Ronzani
Email: dronzani@math.unipd.it

Claudio E. Palazzi
Email: cpalazzi@math.unipd.it

Argomenti e obiettivi

1. **La shell e le sue funzioni**
    a. Cos'è la shell
    b. Comandi principali

Obiettivi: acquisire familiarità con il terminale e i principali comandi tipici del sistema linux

1. **Esercitazione con i simulatori**
    a. SGPEM
    b. SiGeM

Obiettivi: analizzare le politiche di ordinamento e gestione della memoria sotto diverse condizioni

La Shell

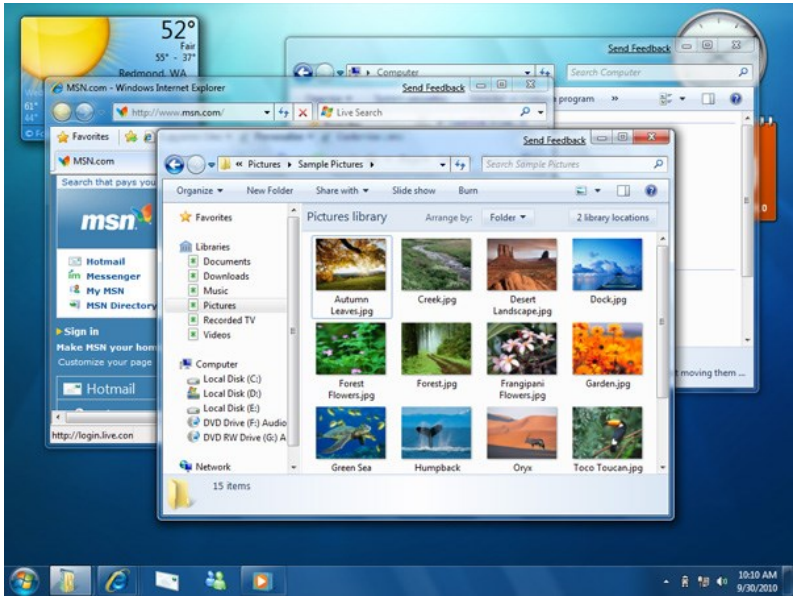
La **shell** è la parte del sistema operativo che permette all'utente di interagire con il sistema stesso.

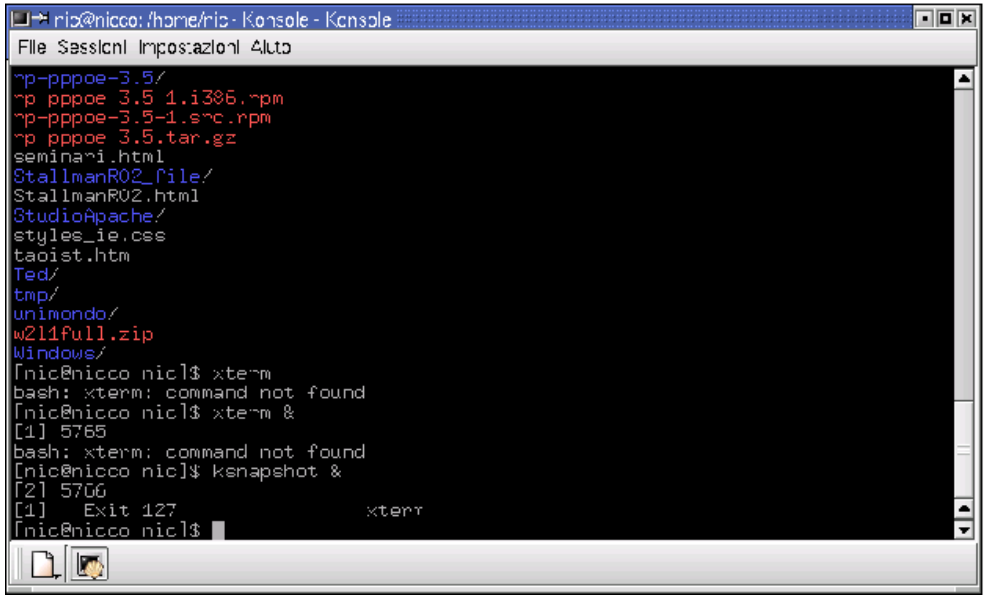
Dall'inglese ***shell*** (guscio), è una delle principali componenti di un sistema operativo, insieme al kernel, e rappresenta **l'interfaccia utente** che rende possibile l'interazione utente-macchina, nascondendone i dettagli.

Più semplicemente, la *shell*, o anche "terminale", viene solitamente intesa come una interfaccia testuale dove l'utente interagisce attraverso l'immissione di comandi sotto forma di stringhe di testo.

La Shell

La shell può essere grafica (GUIs) o testuale (Terminale)





La shell su Unix

Nei sistemi Unix e Unix-like, le shell sono molto più popolari e utilizzate, rispetto ai sistemi Microsoft Windows.

Non esiste una sola shell, ce ne sono molte. Alcune delle più note shell per Unix sono:

- **Bash** (Bourne-Again Shell - `bash / sh`), la prima shell e la più usata
- **Korn shell** (`ksh`), una shell sviluppata dalla AT&T
- **C shell** (`csh`), creata in Unix-BSD, ha un linguaggio di programmazione differente da Bash
- **Tenex C shell** (`tcsh`), una evoluzione di C shell

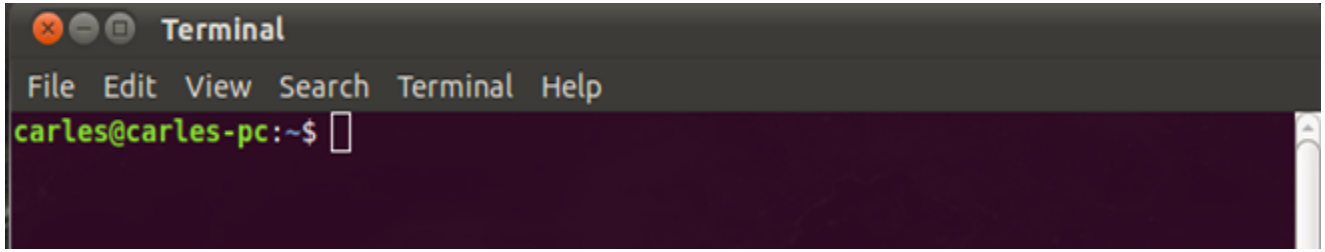
Le shell testuali dei sistemi Unix integrano un linguaggio di scripting.

In Windows: "*prompt dei comandi*"

La shell su Unix

Dal punto di vista dell'utente, la shell è caratterizzata da un terminale, nella quale l'utente può inserire dei comandi testuali.

Una volta aperto il terminale, l'utente inizialmente si trova di fronte ad una riga testuale, seguita da un cursore lampeggiante detto ***prompt***.



Il prompt fornisce all'utente alcune informazioni utili, ad esempio:

- Il nome della macchina
- Il nome dell'utente attuale
- La directory corrente

I comandi nella shell

Nella shell si possono impartire dei comandi.

```
carles@carles-pc:~$ command
```

Il comando può essere:

- Interno (built-in): viene interpretato ed eseguito internamente dalla shell stessa. Ad esempio con comandi come **cd**, **exit**, etc..
- Esterno: denota il nome di un programma eseguibile che viene eseguito dalla shell, lanciando un altro processo; quando il programma termina, il controllo ritorna alla shell. Esempi: **dueffe**, **sigem**, ...

I comandi nella shell

Il comando può essere eseguito da solo oppure insieme a degli argomenti (o parametri), delimitati da uno spazio.

```
carles@carles-pc:~$ command arg1 arg2
```

Una volta che il comando ed eventuali parametri (opzioni) vengono digitati, l'intera linea di comando viene passata ad una funzione main come un array di stringhe.

```
int main(int argc, char** argv)
```

Quando il comando viene eseguito, il valore di ritorno della funzione main denota lo stato di uscita del comando, che può assumere un valore:

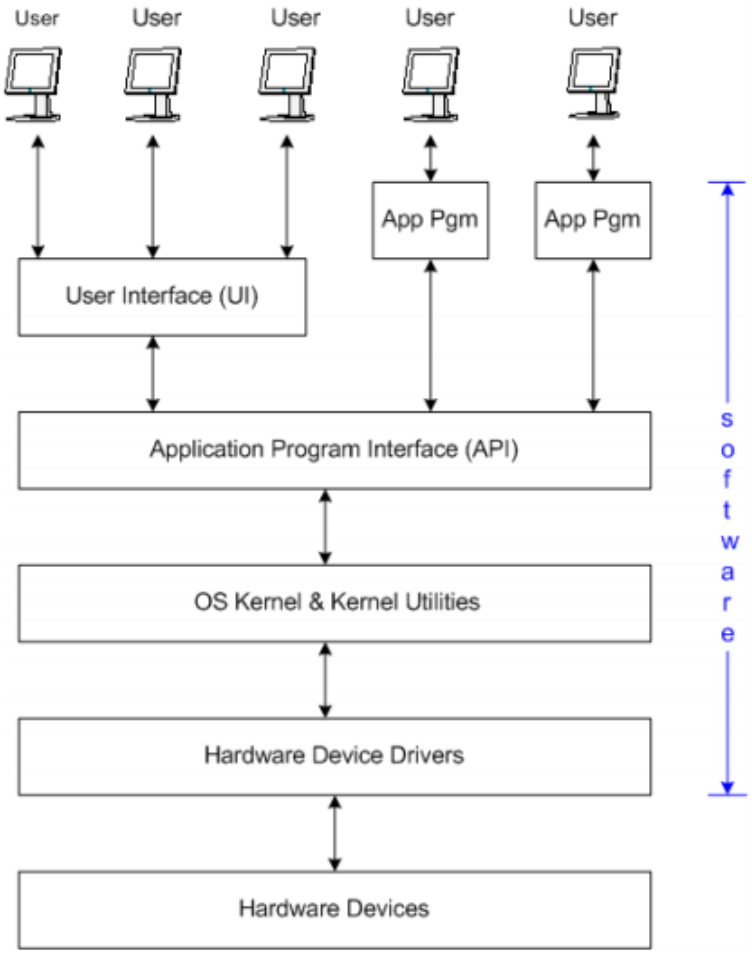
- **0 (zero)**: esecuzione avvenuta con successo senza alcun errore
- **> 0 (non zero)**: uno o più errori sono avvenuti durante l'esecuzione (in generale, ad ogni errore corrisponde un valore di uscita associato)

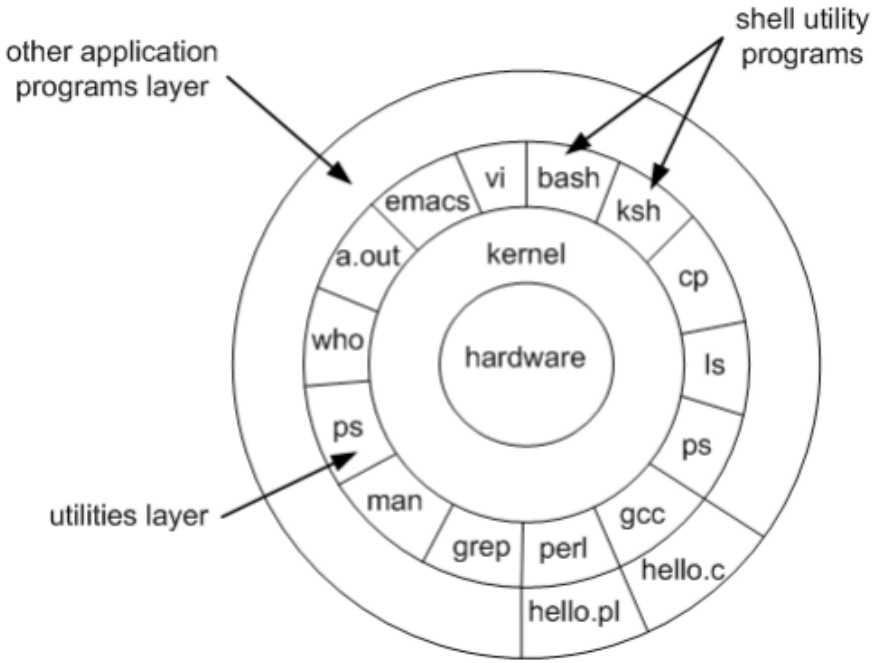
Esecuzione di un comando: esempio

```
carles@carles-pc:~$ ls
```

- Quando viene dato il comando "**ls**", il file eseguibile viene cercato nel sistema (precisamente in `/bin/ls`).
  - `bin`: sottocartella di root contenente gli eseguibili del S/O
- La shell crea un processo figlio che a sua volta lancia in esecuzione l'eseguibile (fork + exec).
- La linea di comando viene passata alla funzione main come un array di stringhe. In questo caso l'array è: `["ls", "/"]`.
- La shell fa una chiamata alla funzione wait, mettendosi in attesa della fine dell'esecuzione del processo figlio appena lanciato.
- Quando **ls** ha terminato, il controllo viene di nuovo dato alla shell che verifica il codice di uscita (valore di ritorno) di **ls**.
  - Output nello Standard Output

La shell nel SO





Vediamo alcuni comandi

Directory corrente e contenuto - 1

Per aprire il terminale abbiamo due modi:

- Attraverso la GUI
- Premendo CTRL + ALT + T

La prima cosa da fare è chiederci in quale directory ci troviamo. Per farlo usiamo il comando **pwd** (Print Working Directory).

```
$ pwd
/home/[username]
```

A questo punto, controlliamo quali file sono presenti nella nostra cartella utente tramite **ls** (List). La risposta è una lista di cartelle e di file presenti nella directory.

```
$ ls
99-006          Images          VTC98          kde.tgz
AAA             LEZIONE         WPERFECT       lib
ANT             LINUX           X11            mp.fvwmrc
....
```

Directory corrente e contenuto - 2

Il comando **ls** può essere chiamato con alcuni argomenti (opzioni). Per leggere il manuale di ogni comando, comprese le opzioni disponibili, si utilizza **man**, seguito dal nome del comando.

L'opzione **-i** permette di vedere l'*i-node number* di ogni file.
L'opzione **-a** permette di visualizzare anche i file nascosti.

```
$ ls -a
..                .nedit          FIGURE          appunti.dvi
..                .neditdb        FSMAGGIO97      appunti.log
.Xdefaults        .netscape       GENNAIO97       appunti.tex
. . . . . .
```

Possiamo anche cercare tutti i file il cui nome ha una parte di testo comune, utilizzando il simbolo *****. Ad esempio, vogliamo trovare tutti i file il cui nome inizia per "app".

```
$ ls app*
appunti.dvi       appunti.log       appunti.tex       appuntamenti.txt
```

History dei comandi e autocompletamento

Un modo comodo per lavorare con la shell è utilizzare la "storia dei comandi".

Utilizzando le **frecce in alto** e **in basso** è possibile scorrere la cronologia degli ultimi comandi dati.

Oppure, digitando il comando **history**, è possibile visualizzare tutti i comandi eseguiti dall'inizio della sessione. Questi comandi sono memorizzati nel file **.bash_history** all'interno della directory **home/[utente]/**.

Un'altra comoda funzione viene fornita dal tasto **TAB**. Quando viene digitato l'inizio di un comando o di un file e poi si preme TAB, il sistema completerà il comando o il nome del file automaticamente, o visualizzerà tutti i comandi o nomi di file che iniziano con quel pattern, se questi sono più di uno.

Creare una directory e spostarsi nel file system

Iniziamo a creare una cartella nella nostra home con il comando **mkdir**.

```
$ mkdir <nome_directory>
```

Usiamo poi **ls** per verificare la corretta creazione.

A questo punto possiamo accedere alla nuova directory usando il comando **cd** (Change Directory).

```
$ cd <nome_directory>
```

e con **pwd** possiamo notare la nuova directory di lavoro corrente.

Per tornare alla cartella padre (la nostra home), si usano due punti **..** dopo il comando **cd**.

```
$ cd ..
```

Creare il primo file

Per creare un file, esistono molti comandi. Uno di questi è **touch**, che permette di creare un file vuoto, di dimensione nulla.

```
$ touch <nome_file>
```

Altri comandi per creare file aprono un editor di testo: **nano**, **pico**, **vi**.

```
$ nano <nome_file>
```

Per visualizzare il contenuto di un file possiamo usare il comando **cat**. Proviamo con il file appena creato.

```
$ cat <nome_file>
```

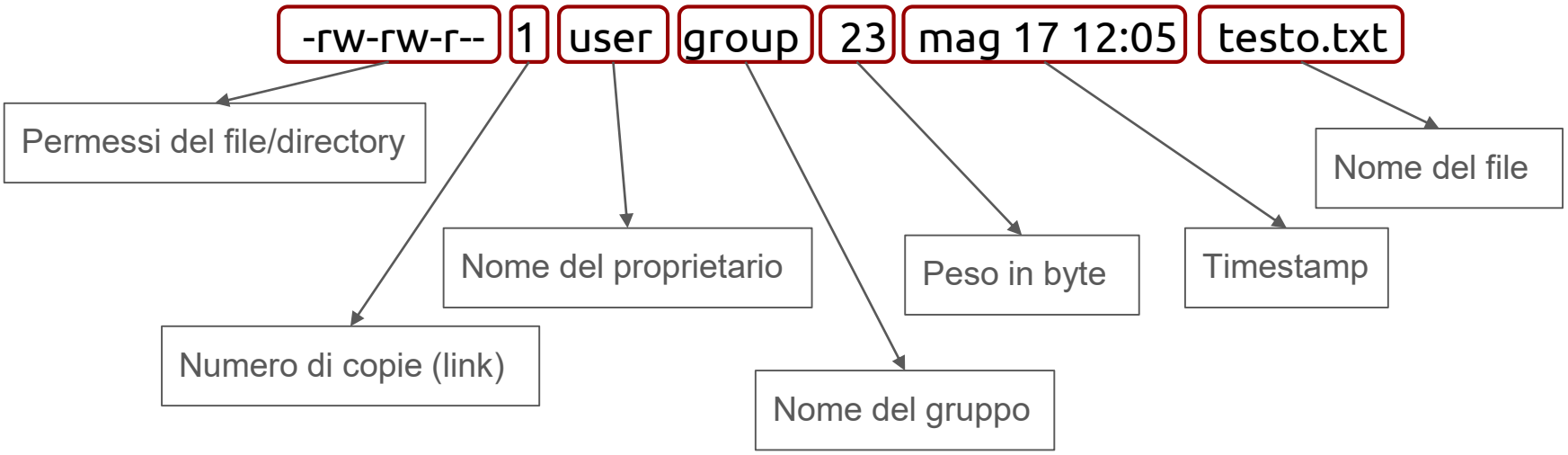
Esercizio: Creare un file "testo.txt" all'interno della nuova cartella, usando **nano**. Inserire del testo e poi salvare (CTRL + X). Verificare il contenuto usando il comando **cat**.

Attributi di un file

Gli attributi di un file sono le caratteristiche relative a permessi, dimensione, data di creazione, utenti, etc.

Per vedere gli attributi dei file presenti in una directory bisogna eseguire il comando **ls -l**. Il risultato, per ogni file, è una linea di testo con tutte le informazioni relative al file.

```
$ ls -l
-rw-rw-r-- 1 user user 23 mag 17 12:05 testo.txt
```



I permessi dei file/directory - 1

-rw-rw-r-- 1 user user 23 mag 17 12:05 testo.txt

I permessi sono organizzati in gruppi di blocchi da (4-3-3) caratteri ciascuno.

I 3 blocchi sono relativi, rispettivamente, ai permessi per

- Utente (user, u)
- Gruppo (g)
- Tutti gli altri utenti (o)

Ogni blocco presenta i permessi di

- Lettura (read, **r**) | int 4
- Scrittura (write, **w**) | int 2
- Esecuzione (execute, **x**) | int 1
- Mancanza di permesso (**-**) | int 0

Nel nostro file **testo.txt**, abbiamo il permesso di leggere, modificare o eliminare il file, essendo noi l'utente in questione.

I permessi dei file/directory - 2

Proviamo a creare una cartella **prova_cartella** ed eseguiamo nuovamente il comando **ls -l**.

```
$ ls -l
drwxrwxr-x  2 daniele daniele 4096 mag 17 12:29 prova_cartella
-rw-rw-r--  1 daniele daniele   23 mag 17 12:05 testo.txt
```

Il primo spazio del blocco dei permessi indica se il file in questione è una directory (**d**), se è un link simbolico (**l**) o un file (**-**).

Nel caso di una directory, il permesso **x** (eXecute) indica che all'interno di quella cartella l'utente può eseguire i comandi e attraversarla.

Cambiare i permessi

Per cambiare i permessi di lettura, scrittura ed esecuzione si usa il comando **chmod**. Esistono due modi per assegnare i permessi:

- Utilizzando i nomi simbolici degli assegnatari **a** (all), **u** (user), **g** (group), **o** (others) e identificando i permessi con le lettere **r** (read), **w** (write) e **x** (execute).

  Esempi:

  - **chmod u=rw file.txt** (attribuisce i permessi di lettura e scrittura all'utente proprietario)
  - **chmod o+w file.txt** (aggiunge il permesso di scrittura al file a tutti gli altri utenti del sistema)
  - **chmod o-w file.txt** (rimuove il permesso di scrittura a tutti gli utenti)

Cambiare i permessi: sintassi ottale

L'alternativa è utilizzare la sintassi ottale.

I permessi, in questo modo, possono essere assegnati a tutti i livelli simultaneamente. Al posto delle lettere rwx, si utilizzano 3 numeri, ognuno dei quali può assumere un valore da 0 a 7. I tre numeri definiscono i permessi rispettivamente per l'utente proprietario, il gruppo e gli altri utenti.

Ogni numero, se convertito in binario, fornisce un gruppo di 3 cifre binarie relative ai tre permessi rwx. Se la cifra è 1 il relativo permesso è accordato, se è 0 no.

Esempio: **chmod 760 file**:

- 7 in binario risulta 111 -> permesso di lettura, scrittura ed esecuzione all'utente proprietario
- 6 in binario risulta 110 -> solo permesso di lettura e scrittura al gruppo
- 0 in binario risulta 000 -> nessun permesso a tutti gli altri utenti

Esercizio: togliere i permessi di scrittura al file **testo.txt** e vedere cosa succede quando si tenta di modificarlo e salvarlo con **nano**.

Copiare e spostare un file

Per copiare un file si utilizza il comando **cp**.

```
$ cp <path/file_sorgente> <path/file_destinazione>
```

Esempio:
cp testo.txt copia_testo.txt (copia il file orginale, creando un altro file chiamato copia_testo.txt)

Per spostare un file nel file system si usa il comando **mv**.

```
$ mv <path/file_sorgente> <path/file_destinazione>
```

Il comando sposta il file da una cartella all'altra rinominando (se necessario) il file. Per rinominare un file, tipicamente si usa il comando **mv**.

Esercizio: creare due file "file1" e "file2" e una cartella "contenitore". Spostare i due file all'interno della cartella "contenitore".

*Consiglio: è possibile usare * per spostare entrambi i file con un unico comando.*

Rimozione di un file o directory

Per rimuovere un file nella directory corrente si utilizza il comando **rm**.

```
$ rm <nome_file>
```

Per rimuovere tutti i file all'interno di una directory si usa * dopo il comando.

Per rimuovere una directory si utilizza il comando **rmdir**.

```
$ rmdir <nome_directory>
```

Cosa succede se la directory non è vuota?

Il Linking

Quando si gestiscono molte copie di un file, la cosa può diventare onerosa e rischiosa: quando un file viene modificato, occorre aggiornare anche tutte le altre copie.

Il linking viene utilizzato in situazioni in cui viene richiesta la duplicazione di un file in altre directory. Il linking di un file si può effettuare con il comando **cp -l** oppure con **ln**.

```
$ cp -l <nome_file> <nome_link>
```

```
$ ln <nome_file> <nome_link>
```

Per rimuovere un link basta eseguire il comando **rm** sul link stesso.

Esercizio: Creare un link dal file **testo.txt**. Modificare il contenuto del file link appena creato e vedere cosa succede al file originale **testo.txt**.

Reindirizzamento della shell

Ogni processo in Unix è caratterizzato da

- Una linea di comando
- Un valore di ritorno
- Variabili d'ambiente
- Uno stream standard (stdin, stdout, stderr)

Il reindirizzamento di uno stream serve per indirizzare l'output di un comando o un programma su un file o altro output. Il reindirizzamento si esegue con **>**.

```
$ ls -l > output.txt
```

In questo esempio, se il file **output.txt** esiste già, viene rimpiazzato con uno nuovo, inserendo l'output del comando eseguito.

Se si usa **>>**, il file non viene sovrascritto, ma l'output viene concatenato al contenuto del file (se esiste).

Esercizio: concatenate l'output di 2 comandi a un file esistente.

Creare uno script eseguibile dalla shell

Creare uno script significa racchiudere diversi comandi in un singolo file che rappresenta il nostro programma.

Proviamo a scrivere il nostro primo script (Hello Shell) utilizzando **nano**.

```
echo "My first bash script"
echo "Hello Shell"
exit 0
```

Per eseguirlo

- **Modifica permessi:** `chmod u+x <nome_script>`
- **Esecuzione:** `./nome_script`

Esercizio: Scrivere uno script "script.sh" che, quando eseguito, crea un file e salva sul file il contenuto della cartella corrente e della cartella padre. Provare a lavorare direttamente dal terminale utilizzando il comando **nano** per scrivere sul file script.

Per chi vuole approfondire...

- www.html.it
- http://codex.altervista.org/guidabash/guidabash_1_11.pdf
- https://www.gnu.org/savannah-checkouts/gnu/bash/manual/bash.html
- https://tiswww.case.edu/php/chet/bash/bashref.html