

Uno script `bash` potrebbe necessitare di effettuare letture e scritture su file, anche se non vengono passate come argomento.

È possibile ottenere un **file descriptor** sulla base di uno specifico file che si trova sul disco, ed utilizzarlo per effettuare operazioni di input/output. È possibile **specificare uno specifico file descriptor** tramite il comando `exec`. È possibile specificare uno *specifico* file descriptor, oppure lasciare che il sistema operativo **scelga un file descriptor libero** (quest'ultima scelta è preferibile per evitare conflitti). Si usano i seguenti **metacaratteri di redirezione**:

- `<` : solo lettura
- `>` : solo scrittura
- `>>` : append mode
- `<>` : lettura e scrittura

Alcuni esempi:

```
exec {FD}< /home/vulpi/ciao.txt
# In questo caso la variabile FD contiene il file descriptor che rappresenta
# il file /home/vulpi/ciao.txt; viene considerata la sola lettura.
while read -u ${FD} STRINGA ; do
    echo "Ho letto: ${STRINGA}"
done
# Legge riga per riga!
#####
exec {FD}> /home/vulpi/out.txt
for name in CIA01 CIA02 CIA03 ; do
    echo "Inserisco ${name}" 1>&${FD}
done
# Vedremo dopo cosa fa quella cosa alla fine di echo.
```

I file descriptor aperti da una specifica `bash` si trovano nella directory speciale `/proc/$$`, dove `$$` è una variabile che restituisce il **Process ID della shell**. In ogni sottodirectory di `/proc` è presente la directory `fd`, che contiene i file descriptor utilizzati da tale programma in quel momento.

È possibile chiudere un qualsiasi stream i/o tramite la strana sintassi:

```
exec n>&-
```

con n il numero di file descriptor.

Non è possibile utilizzare uno stesso file descriptor una volta che viene chiuso.

## Ridirezionamento

Quando una shell genera un processo figlio, tale processo figlio ottiene **UNA COPIA** dei file descriptor in quel momento aperti dal padre. È possibile però modificare tali file descriptor, associandoli a stream i/o diversi. In tal caso i file descriptor del figlio avranno gli stessi valori di quelli del padre, ma che si riferiscono a stream differenti.

Una shell padre può aprire e chiudere file, in base alle operazioni che deve compiere. Può tuttavia anche **modificare lo stream di alcuni dei suoi file descriptor**; in questo modo, utilizzando i vecchi file descriptor, può accedere **ad altri stream di dati**. Anche qui si hanno alcuni **operatori**:

- < : permette di ricevere un input da un file specificato successivamente.
- > : lo *standard output* del programma che precede viene inserito nel file successivo a questo carattere, **eliminando il contenuto di quest'ultimo**.
- >> : lo *standard output* del programma che precede viene inserito nel file successivo a questo carattere, **inserendolo in coda**.
- | (detto *pipe*): lo *standard output* del programma che precede viene usato come input del programma successivo al carattere.

### Input da file

```
programma < input_file
```

programma vedrà il contenuto di `input_file` come se gli venisse digitato da tastiera.

Si noti che, a differenza dello `stdout`, il programma si accorge che le cose da leggere sono terminate quando il file stesso è terminato. Nel caso fossimo da tastiera, al fine di segnalare la fine dell'inserimento, usiamo la combo *CTRL+D*.

### Output su file

```
programma >> out_file # Append  
programma > out_file # Overwrite
```

Ipotizziamo che l'output di `programma` sia `cane gatto cane` e che `out_file` contenga `animali animalosi`.

Dopo la prima operazione, l'output di `programma` verrà **aggiunto in coda** a `out_file`; ne consegue che `out_file` contenga `animali animalosi cane gatto cane`.

Dopo la seconda operazione, l'output di `programma` **sovrascriverà** il contenuto di `out_file`; ne consegue che `out_file` contenga `cane gatto cane`.

I ridirezionamenti di input e di output possono essere fatti contemporaneamente:

```
program < file_input > file_output
```

È possibile ridirezionare *standard output* e *standard error* contemporaneamente tramite `&>` e separatamente specificando `>` per lo *standard output* e `> 2>` per lo *standard error*.

**NB:** I ridirezionamenti non cambiano il valore del file descriptor che viene rediretto e nemmeno il numero dei file descriptor aperti fino a quel momento; è in tutto e per tutto **una redirezione**.

## Generalizzazione operatori

Possiamo generalizzare gli operatori `>` e `<` per specificare da quale file descriptor considerare la redirezione.

- `N> Target` redireziona la stream con file descriptor `N` sul file `Target`. Omesso `N` si intende *standard output*.
- `<N Source` redireziona il file `Source` **sul** file descriptor `N` del programma specificato a sinistra dell'operatore. Omesso `N` si intende standard input.

## Redirezionamento tramite pipe |

È possibile eseguire due comandi contemporaneamente, inviando lo *standard output* del primo come input nel secondo, tramite la *pipe* `|`.

```
program1 | program2
```

Cosa **fondamentale** della pipe è che se a sinistra o a destra della pipe sono presenti **sequenze di comandi** o **cicli iterativi** tali istruzioni sono **eseguire in una shell figlia**.