La redirezione dell I/O su Linux

Tutti i sistemi operativi basati su Unix forniscono almeno tre diversi canali input e output – chiamati rispettivamente `stdin`, `stdout` e `stderr` – che permettono la comunicazione fra un programma e l'ambiente in cui esso viene eseguito. In Bash ciascuno di questi canali viene numerato da 0 a 2, e prende il nome di *file descriptor* poiché fa riferimento ad un particolare file: così come avviene con un qualunque altro file memorizzato nel sistema, è possibile manipolarlo, copiarlo, leggerlo o scrivere su di esso. Bash offre anche la possibilità di aprirne di nuovi se le esigenze lo richiedono (utilizzando i numeri successivi al 2).

Quando un ambiente Bash viene avviato, tutti e tre i file descriptor di default puntano al terminale nel quale è stata inizializzata la sessione: l'input (`stdin` – 0) corrisponde a ciò che viene digitato nel terminale, ed entrambi gli output – `stdout` (1) per i messaggi tradizionali e `stderr` (2) per quelli di errore – vengono inviati al terminale. Infatti, un terminale aperto in un sistema operativo basato su Unix, è in genere esso stesso un file, comunemente memorizzato in `/dev/tty0`; quando viene aperta una nuova sessione in parallelo ad una esistente, il nuovo terminale sarà `/dev/tty1` e così via. Perciò, inizialmente i tre file descriptor puntano tutti al file che rappresenta il terminale in cui vengono eseguiti.

L'operatore `>` si utilizza per effettuare il **redirect dell'output**. Eseguendo:

```
comando >file
```
Copy

Bash tenta prima di aprire il file con i permessi di scrittura e – nel caso questa operazione abbia successo – invia lo stream `stdout` al file appena aperto (sovrascrivendone il contenuto esistente). Lo stesso effetto si ottiene attraverso il comando:

```
comando 1>file
```
Copy

Il motivo è che `stdout` è il file descriptor numero 1. In generale è possibile scrivere `comando n>file` per redirezionare il file descriptor `n` verso `file`. Nel caso in cui si desideri preservare il contenuto del file e far sì che l'output stream gli venga **concatenato**, l'operatore da utilizzare diventa `>>`.

È anche possibile redirezionare sia `stdout` che `stderr` verso uno stesso file utilizzando l'operatore `&>` (o equivalentemente `>&`), cioè scrivendo:

```
comando &> file
```
Copy

Lo stesso effetto si ottiene redirezionando i due stream in sequenza, come ad esempio:

```
comando >file 2>&1
```
Copy

L'output stream viene redirezionato verso `file` e successivamente `stderr` viene redirezionato verso l'output stream, cioè `file`. Dato che i comandi vengono valutati sequenzialmente l'ordine in cui si specificano i redirect è significativo, perciò scrivere:

```
comando >file 2>&1
```
Copy

Non è equivalente a scrivere `comando 2>&1 >file`: dopo aver eseguito quest'ultimo, solo `stdout` sarà redirezionato verso `file`, poichè `stderr` è stato redirezionato verso la destinazione di default dell'output stream (`/dev/tty0`) prima che questa venisse cambiata.

Quando un comando è troppo verboso, non si è interessati al suo output o lo si vuole nascondere all'utente, è possibile redirezionare `stdout` o `stderr` (o entrambi) verso lo speciale nodo `/dev/null`, il cui contenuto viene sempre scartato dal sistema.

Come è facile immaginare, la **redirezione di un canale di input** (come quello predefinito `stdin`) avviene utilizzando l'operatore <: volendo dare un file in input ad un comando o ad uno script, si utilizza quindi la notazione seguente:

```
comando <file
```
Copy

Ad esempio, se vogliamo leggere il contenuto della prima riga di un file e assegnarlo alla variabile `riga`, è possibile semplicemente eseguire:

```
read -r riga < file
```
Copy

Infine, nel caso dei canali di input, l'operatore << seguito da un marker impone alla shell di leggere l'input da `stdin` fintantoché non incontra il marker; a quel punto, Bash trasmette l'intero contenuto dell'input letto fino a quel momento al canale `stdin` del comando che lo riceve.

Tutte le redirezioni dei canali viste finora sono state effettuate per un singolo comando; ma come si fa a stabilire una **redirezione permanente dei canali I/O** da un certo momento in avanti? Tramite il comando `exec`:

```
exec 2>file
comando1
comando2
...
```
Copy

Nell'esempio appena riportato, lo stream `stderr` viene redirezionato verso `file` dal comando `exec 2>file`; da quel momento in avanti lo stream `stderr` relativo ai comandi `comando1`, `comando2` e tutti i successivi verrà scritto nel file specificato.

Un'alternativa per effettuare il redirect soltanto temporaneo di comandi multipli verso uno stesso file si ottiene sequenzializzandoli mediante l'operatore `;`:

```
(comando1 ; comando2) >file
```
Copy

Il codice precedente redireziona lo standard output di ciascun comando all'interno delle parentesi verso il file specificato.

Un altro operatore molto utile per la manipolazione degli stream I/O è quello di **pipelining**: `|`. Tramite questo simbolo, una sequenza di comandi separata da esso fa sì che ciascun comando venga eseguito allo stesso momento e che l'output di ognuno venga passato come input a quello successivo, da sinistra verso destra. Dobbiamo però fare attenzione: ogni comando viene eseguito in una subshell, perciò ogni variabile modificata da ciascuno non sarà poi letta dagli altri comandi della sequenza o nell'ambiente in cui viene eseguita l'intera pipeline, come si nota eseguendo il codice seguente:

```
echo "Prova" | (read var ; echo $var)
```
Copy

Per concludere, un comando molto utile – non fornito da Bash di default ma presente in pressoché tutti i sistemi basati su Unix – è **`tee`**: esso reindirizza lo stream input sia verso `stdout` sia verso un file specificato. Eseguire:

```
echo messaggio | tee file
```
Copy

stamperà `messaggio` a schermo e nel file specificato come argomento.