La sintassi di un comando DOS è tipicamente:

```
COMANDO [PARAMETRI] [OPZIONI]
```

inizia sempre con un comando e può essere seguito da altre informazioni

- COMANDO è un qualunque programma appartenente al set di comandi MS-DOS, che li distingue tra comandi interni (fanno parte del cuore - kernel - del sistema operativo) e comandi esterni (si trovano in una directory riservata); il comando si trova sempre all'inizio
- [PARAMETRI] specifica su "cosa" (elemento/oggetto) agisce il comando; per esempio un file, una directory, …
- [OPZIONI] servono a specificare "come" il comando agisce; ci sono molti modi in cui un comando restituisce le informazioni richieste, l'utente può selezionare quella che gli interessa
- Le informazioni racchiuse tra parentesi quadre [ ] sono facoltative: alcuni comandi le richiedono, altri no; dipende dal comando e dalle intenzioni dell'utente

Esempio:

```
DIR ..\SUBDIR /P
```

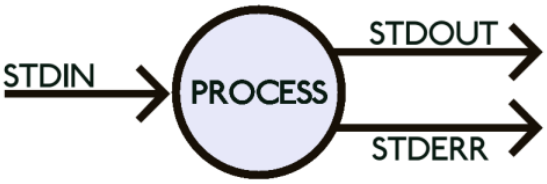
Il comando DIR elenca il contenuto della directory, ma... quale? è specificato nel parametro ..\SUBDIR; in che modo? Una pagina alla volta, opzione /P. Se manca il parametro, DIR elenca i file della directory corrente, senza interruzioni; quasi tutti i comandi hanno un comportamento predefinito ("di default") che viene attivato in assenza di parametri e/o opzioni.

Un comando dell'ambiente CLI (Command Line Interface, detto anche SHELL) può prevedere input e/o output, ovvero informazioni attese – input – che riceve dall'utente, e informazioni restituite - output - inviate all'utente.

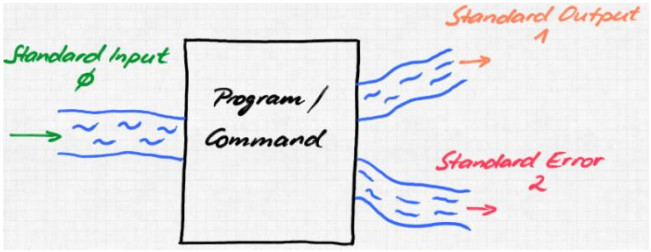
Input/Output & standard streams

In informatica e in tutti i moderni sistemi operativi gli **standard streams** rappresentano i canali logici di input e di output che collegano un programma/task/processo con l'ambiente (periferiche, I/O in generale, utenti) in cui esso viene eseguito; gli standard streams sono associati al programma al suo avvio.

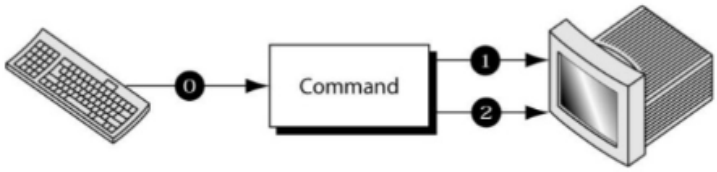
I tre canali di input/output predefiniti sono detti **standard input**, **standard output** e **standard error**, tecnicamente definiti come: **stdin**, **stdout** e **stderr**; hanno un numero (**descrittore**) che li identifica: 0 per stdin, 1 per stdout, 2 per stderr.



Un programma/task/processo (quindi anche un comando MSDOS) prevede input e output; prende gli input un byte dopo l'altro, invia gli output un byte dopo l'altro; questo "uno dopo l'altro" suggerisce l'idea del flusso (stream), e possiamo veramente pensare a un flusso di byte che arrivano al programma/task/comando e a un flusso di byte che vengono emessi dal programma/task/comando, come riportato in figura:

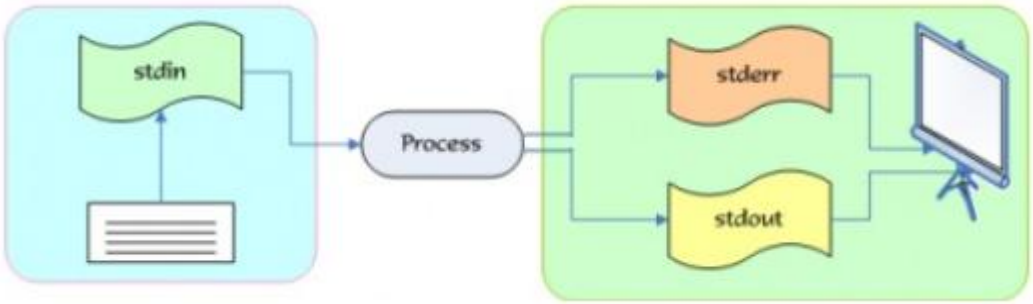


Questi flussi o canali sono agganciati a qualcosa, normalmente a una periferica; il flusso di input tipicamente proviene dalla tastiera, e il flusso di output è inviato allo schermo, come pure lo standard error:



L'accoppiata tastiera+video viene chiamata **Terminale** (o **console**, pronuncia: cònsol); in passato il terminale era solo testuale: tastiera e video inviavano e ricevevano caratteri (byte) codificati col codice ASCII.

La figura seguente evidenzia gli standard streams e l' "aggancio" classico (predefinito) alle periferiche



questa è figura importante: evidenzia che tipicamente gli stream vengono agganciati a periferiche.

Per gestire i flussi ci sono memorie (buffer) che accumulano i byte in transito per poi dirigerli alla destinazione. In un contesto multiutente o anche solo multitasking emergono problemi delicati, come l'accesso concorrente (=da parte di più processi) alla stessa periferica: come disciplinare e separare i vari flussi? Il tema è squisitamente sistemistico, e viene trattato nei corsi di informatica.

Attenzione: i flussi di I/O possono essere **ridiretti**, cioè agganciati, non al terminale ma ad altre periferiche (stampante, linea seriale ecc.) o anche a file! Questo perché quasi tutti i sistemi

operativi – a un certo livello - trattano in modo equivalente file e periferiche[1]; è come se una periferica, per il fatto di ricevere/inviare byte in input/output, possa essere trattata come un file[2] sul quale si vada a leggere/scrivere; c'è una evidente analogia con le operazioni sui file, e questa analogia è stata implementata dai sistemi operativi; come conseguenza, con un linguaggio di programmazione come il C si può leggere e scrivere su una periferica allo stesso modo (cioè con le stesse funzioni) attraverso cui si agisce su un file.

Ri-direzione (redirect) dell'input e dell'output

E' possibile modificare l'input/output standard del comando tramite gli **operatori di ri-direzione**:

< **>** **>>**

In altri termini, si può specificare un input o un output diversi da quelli di default propri del comando, che sarebbe il terminale; tipicamente i nuovi input/output sono file di testo (ma potrebbero essere anche altre periferiche device files[3]).

A seconda dei comandi possiamo incontrare espressioni del tipo:

```
COMANDO ...  ...   < file_testo1    > file_testo2    >> file_testo3
```

L'operatore **<** preleva l'input dal file indicato. L'operatore **>** crea il file indicato e lo riempie con l'output del comando; se il file esiste già viene sovrascritto e il contenuto precedente viene perso. L'operatore **>>** appende l'output al file indicato, cioè lo inserisce in fondo. Non si possono inserire insieme i due operatori **>** e **>>**, sarebbe ambiguo; uno stream può essere collegato a una sola periferica/file; per dirla tutta si possono anche scrivere insieme, ma vince l'ultimo.

Facciamo qualche esempio; il comando:

```
C:\>DIR C:\WINDOWS
```

produce la visualizzazione sull'output standard (lo schermo) del comando DIR, mentre

```
C:\>DIR C:\WINDOWS > C:\DOCUMENTI\list1.dat
```

mostra la lista dei file non sullo schermo bensì crea un file "list1.dat" nella cartella "DOCUMENTI" del disco C: contenente l'output del comando "DIR". In questo caso si ha una ridirezione dell'output.

---

[1] i testi parlano di "dispositivi di I/O"; qui parliamo di "periferiche" per non creare confusione con gli I/O device (buffer & latch) trattati con questo termine in altri corsi

[2] ovviamente sono oggetti diversi che alla fine verranno gestiti secondo le proprie caratteristiche, ma sino a un certo livello vengono trattati allo stesso modo

[3] MS-DOS prevede i **device files** (file di periferica), entità trattate a tutti gli effetti come file ma corrispondenti in realtà a dispositivi fisici (periferiche).
I più comuni device files sotto DOS sono:

- `CON:` console (terminale) = periferica standard di input (tastiera) + periferica standard di output (schermo)
- `COM1:` (anche `AUX:`) = porta seriale, numerata: `COM1:, COM2:, ... COMn:`
- `LPT1:` (anche `PRN:`) = porta parallela normalmente utilizzata dalla stampante; ulteriori porte parallele vengono indicate con `LPT2:, LPT3:, ...`
- `NUL:` dispositivo virtuale solo di output, usato per sopprimere l'output di programmi o i messaggi di sistema

Invece il comando:

```
C:\>DIR C:\WINDOWS >> C:\DOCUMENTI\list2.dat
```

appende l'output del comando DIR in fondo a list2.dat; se il file non esiste già, allora viene creato.

L'input standard del comando "MORE" è la tastiera, l'output standard lo schermo; il comando:

```
C:\>MORE > test1.txt
```

preleva l'input dalla tastiera (sino all'introduzione del carattere convenzionale di "fine file": CTRL+Z) e lo invia al file "test1.txt"; è questo un modo comune di scrivere del testo creando un file, in assenza di un comando di editing appropriato (è il caso della shell di Windows).

Vediamo un comando che prevede sia input che output. Il comando:

```
C:\>SORT  <test1.txt  >test2.txt
```

legge il file di testo test1.txt, lo ordina in senso crescente (riga per riga), e lo riversa su test2.txt; test2.txt viene o creato o – se già esistente – sovrascritto.

PIPE

E' possibile passare l'output di un comando all'input di un altro comando tramite un collegamento virtuale chiamato **"pipe"** (=tubo), indicato col simbolo **|** . Tramite questo operatore è possibile concatenare una serie di comandi, sfruttandone gli effetti in cascata.

In generale, nell'espressione:

```
COMANDO1 | COMANDO2 | ... | COMANDOn
```

l'output del comando1 viene ridiretto all'input del comando2, l'output del comando2 viene ridiretto all'input del comando3, e così via. Ad esempio:

```
C:\>DIR C:\WINDOWS | SORT | MORE
```

visualizza il contenuto della cartella "Windows", in ordine alfabetico (SORT), una pagina per volta (MORE), sullo schermo.

ASPETTI AVANZATI

La sintassi

```
COMANDO ...  ...  <file1    >file2
```

può essere riscritta così:

```
COMANDO   0<file1   1>file2  2>file3
```

evidenziando i numeri dei descrittori degli standard stream. (1>… e 2>… non possono coesistere se ridiretti sullo stesso file)

È possibile ridirezionare stdout e stderr verso lo stesso file:

```
COMANDO >file 2>&1  (=stderr ridirezionato allo stdout)
```

EX: utilizzare il comando dir (che emette messaggi anche sullo stderr) e provare le sintassi