

Quando scriviamo il nome di un comando o di un eseguibile sulla *shell* è possibile invocare una delle varie tipologie di eseguibili presenti al suo interno:

- Alcuni comandi sono disponibili semplicemente **digitandone il nome**, come ad esempio `cd` o `ls`. Queste tipologie di comandi sono detti **comandi built-in** della *shell*.
- I file eseguibili (chiamati **binari o script**) devono essere invocati specificandone il **percorso assoluto** a partire da `/` oppure il **percorso relativo** a partire dalla directory corrente. Unica eccezione sono gli eseguibili presenti **all'interno delle directory specificate nella variabile d'ambiente PATH**, i quali possono essere eseguiti semplicemente digitandone il nome.

Per costruire il **percorso relativo** a partire dalla directory attuale è necessario utilizzare i metacaratteri `.` e `..`: il primo verrà interpretato come **directory attuale**, mentre il secondo verrà interpretato come **la directory superiore a quella attuale**.

```
# Supponiamo di essere in /home/vulpi/Desktop, e nella cartella ./ciao è
presente un eseguibile ciao.sh
cd .
# Risultato: non succede niente, in quanto ci stiamo spostando nella
directory attuale, di fatto non spostandoci
cd ../..
# Risultato: la directory attuale diventa /home. Eseguiamo ora
/home/vulpi/Desktop/ciao/ciao.sh da qui usando un percorso relativo.
./vulpi/Desktop/ciao/ciao.sh
# Risultato: il file verrà eseguito avendo specificato il percorso relativo
in modo corretto (. è interpretato come /home).
```

Le subshell

Una **subshell** (chiamata *shell figlia*), è una *shell* che è stata creata da un'altra *shell* (detta *shell padre*). Essa può essere creata in diversi modi:

- Esecuzione di **più comandi raggruppati**;
- Esecuzione di **script**;
- Esecuzione di processi **in background**

I comandi built-in **non** creano una *shell* figlia, venendo eseguiti direttamente su quella padre.

Shell figlia e padre hanno delle **importanti differenze tra loro**:

- La *shell* figlia ha una propria directory corrente, che viene **inizialmente ereditata dal padre**.

- Ogni *shell* ha **delle proprie variabili**; alla creazione di una *shell* figlia **essa riceve una copia delle variabili d'ambiente della shell padre***; le variabili locali **non vengono copiate**.

La creazione di una *subshell* è determinata da come una qualsiasi *shell* deve eseguire un dato script:

- All'esecuzione, la *shell* legge la prima riga dello script e capisce quale interprete di comandi deve utilizzare per eseguire lo script. Nel caso di `bash` sarà presente, come prima riga, `#!/bin/bash`.
- Viene a questo punto creata una *shell figlia*, la quale esegue lo script; le viene passato come argomento anche il nome dello script, se specificato tramite il *tag* `-c`.
- Una volta portato a compimento l'esecuzione, la *shell figlia* restituisce il controllo al padre, restituendo un valore intero che riporta l'esito dell'operazione.

Dunque la sostanziale differenza tra **variabili d'ambiente e variabili locali** è che le **locali** non vengono trasmesse alle *shell figlie* nel momento della loro creazione, mentre di quelle **d'ambiente** viene passata una copia; in particolare, essendo una copia, se il loro valore viene modificato da una *shell figlia*, tale valore non verrà modificato anche nelle variabili della *shell* padre.

E' possibile rendere una variabile locale già dichiarata come variabile d'ambiente tramite il comando `export`; inoltre è possibile ottenere una lista di tutte le variabili d'ambiente tramite il comando `env`:

```
miaVar=val # Dicho la variabile miaVar, locale.

export miaVar # miaVar è diventata una variabile d'ambiente.

export altraVar=ciao # Posso anche dichiarare e rendere direttamente
                     d'ambiente una variabile, assegnandole un valore.

env
# Output: Una lista di variabili d'ambiente, in cui compariranno anche
          miaVar e altraVar
```

Ogni eseguibile ha una copia del proprio ambiente in memoria: quando la sua esecuzione termina, tale ambiente viene **eliminato dalla memoria** se non salvato.

È possibile eseguire uno script **senza che venga creata una shell figlia**, eseguendolo quindi direttamente sulla *shell* padre; in questo modo le variabili locali e d'ambiente della *shell* padre vengono modificate. Ciò è reso possibile tramite i comandi `built-in source` e `..`

```
./mioscript.sh # Questo script modifica la variabile PATH aggiungendo qualcosa.  
# Risultato: viene creata una shell figlia in cui viene eseguito mioscript.sh  
source ./mioscript.sh  
# Risultato: NON viene creata la subshell, mioscript.sh viene eseguito sulla shell padre direttamente.
```

Il comando `source` possiede dei rischi di sicurezza per quanto riguarda la modifica di variabili d'ambiente importanti, quali `PATH` e `HOME` - è necessario eseguire uno script con `source` solo se si è sicuri di ciò che si sta facendo. Inoltre, quando eseguiamo uno script con `source`, non verrà considerato l'interprete indicato nella prima riga dello script.

E' possibile dichiarare delle variabili d'ambiente prima di eseguire uno script o un comando semplicemente assegnando le variabili prima dell'esecuzione del comando.

```
miaVar=ciao ./mioscript.sh  
# La subshell dove mioscript.sh viene eseguito vede e può usare miaVar  
echo miaVar  
# Output: nessuno! miaVar non esiste più.
```

C'è una grande differenza tra una **variabile vuota** e una **variabile non esistente**:

- Una variabile vuota è una variabile **dichiarata** a cui è stato assegnata la stringa vuota `""`.
- Una variabile non esistente è una variabile che **non è mai stata dichiarata**. E' possibile eliminare una variabile (renderla dunque *non esistente*) tramite il comando `unset`. Bisogna prestare attenzione - l'uso di variabili vuote o non esistenti può causare errori di sintassi in esecuzione.

history expansion

Il comando *built-in* `history` memorizza i comandi lanciati in precedenza dalla *shell*, permettendone la visualizzazione in una lista completa dove ogni comando è preceduto dall'indice di esecuzione.

E' possibile rilanciare un comando lanciato in precedenza tramite la **history expansion** `!`, la quale ha due diversi modi d'uso:

- `!<indice>` esegue il comando di indice `indice` in `history`
- `!<str>` esegue il comando più recentemente eseguito nella `history` che inizia per `str`.

```
cd /home/vulpi
cat ./ciao.txt

history
# Gli ultimi due comandi in lista saranno quelli appena scritti sopra
seguiti da history.

!c
# Esegue cat ./ciao.txt
```

Comando `set`

Il comando *built-in* `set` ha diverse funzionalità in base agli argomenti che gli vengono passati:

- **Senza argomenti:** stampa a video le variabili e le funzioni impostate nella shell, d'ambiente e non.
- **Alcuni esempi con argomenti:**
 - `set +o history` disabilita la memorizzazione dei comandi successivi in `history`. E' possibile riabilitare la memorizzazione tramite `set -o history`.
 - `set -a` renderà ogni variabile dichiarata successivamente a questo comando **automaticamente d'ambiente**. E' possibile disabilitare questa cosa tramite `set +a`.
 - Altre funzioni di `set` sono visualizzabili tramite il comando `man set`. NB: una volta impostato `set -a`, per dichiarare una variabile d'ambiente è necessario utilizzare il comando `export` con il tag `-n`.