Comando `set`

Se lanciato senza nessun parametro, mostra a video un elenco di variabili di sistema, sia locali, sia d'ambiente.
A seconda dei flag modifica alcuni comportamenti:

- `set +o history` disattiva la memorizzazione di comandi nella `history`; il flag `-o` la abilita nuovamente.
- `set -a` causa che tutte le variabili che creerò successivamente al comando siano dichiarate **d'ambiente** senza specificare `export`; il flag `+a` ripristina il comportamento normale.
  - Se abbiamo eseguito `set -a`, per dichiarare variabili locali, si usa `export -n <nomevar>`.

Un comando può accettare, come abbiamo visto, degli **argomenti**. Usiamo come esempio

```
chmod u+x /home/ciao.sh
```

- `chmod` è l'effettivo comando (chiamato anche argomento di indice **zero**)
- `u+x` è l'argomento di indice 1
- `/home/ciao.sh` è l'argomento di indice 2
- si dice che questo comando ha **2** argomenti.

Flag per avvio di `bash`

- Usare il flag `-c` quando si avvia `bash` al fine di avviare script rende la `bash` figlia **non interattiva**, ossia adibita alla sola esecuzione di script.
- Non usare flag quando si avvia `bash` avvia una shell interattiva **non di login** - questa `bash` è quella disponibile di default quando si apre il terminale su lUbuntu
- Usare il flag `-l` o `--login` quando si avvia `bash` verrà avviata una shell interattiva **di login** - verranno chieste credenziali al fine di utilizzare il terminale.
  La shell interattiva di login, prima di essere effettivamente eseguita, cerca di eseguire i seguenti file:
- `/etc/profile`
- Uno tra `.bash_profile, .bash_login, .profile` nella `/home` dell'utente
- `.bashrc`
  che sono file di setup di variabili fondamentali (come la `PATH`) al fine di una corretta esecuzione della shell.

La shell interattiva non di login eseguirà invece solo i comandi presenti all'interno di `.bashrc`.
La shell non interattiva non di login è la shell che verrà utilizzata per l'esecuzione di script: non è infatti "interattiva" con l'utente, ma solo con lo script che esegue. Questa tipologia di shell non esegue file di configurazione.

Riga di comando, separatore di comando e comandi multipli

Consideriamo il seguente comando:

```
echo ciao!
```

Verrà stampato a schermo "ciao!".

Si possono eseguire più comandi separandoli da punti e virgola:

```
echo ciao! ; PIPPO=pero; pwd
```

eseguirà prima `echo ciao!`, poi `PIPPO=pero` e infine `pwd`.

NB: ogni carattere speciale può avere la sua funzione disabilitata aggiungendo un backslash `\` davanti; **ogni `\` disabiliterà SOLO il carattere che ha dopo!**

E' possibile disabilitare il comportamento di multipli caratteri speciali includendoli tra doppi apici e apici singoli; la differenza è che:

- Tra doppi apici `" "` alcuni comportamenti di alcuni caratteri speciali sono comunque permessi
- Tra apici singoli `' '` anche questi comportamenti non sono attivati.

Brace Expansion: questa formattazione permette di creare più stringhe sulla base di una serie di stringhe presenti tra parentesi graffe. La formattazione è la seguente:

```
echo <preambolo>{(parola1),(parola2),...}<postscritto>
```

- Il preambolo è la stringa di "apertura", il postscritto quella di "chiusura".
- Le parole incluse tra graffe sono separate da virgole
  Esempio:

```
echo ci{siam, a, cl}o
```

Avrà come output `cisiamo ciao ciclo`.

Tilde Expansion: il carattere tilde: `~` sostituisce il percorso assoluto della home directory dell'utente che sta eseguendo il comando. Se l'utente `vulpi` sta eseguendo il comando, `~` equivale a `/home/vulpi`.

Se `~` è seguita da un nome utente esistente, allora equivarrà alla home directory relativa a quel nome utente. Se ad esempio l'utente `vulpi` sta eseguendo il comando ed esiste un user `winzows` nel sistema operativo, allora `~winzows` equivale a `/home/winzows`.
Se il nome utente non esiste, allora `~` non ha effetto, l'espansione non viene eseguita e quindi il tutto viene trattato come una stringa.

Pathname Substitution: lo scopo delle espansioni di pathname hanno come obiettivo quello di sostituire caratteri o stringhe interi nella ricerca di file all'interno del filesystem.
Si utilizzano le wildcards `*`, `?` e `[...]`:

- `*` sostituisce una intera stringa all'interno di una ricerca di uno o più file
- `?` sostituisce un singolo carattere all'interno di una ricerca di uno o più file
- `[...]` contengono una lista di caratteri da ricercare
  In qualsiasi caso, se un file con le specifiche caratteristiche non esiste, la expansion non avviene.

`mkdir`, `rmdir`, `rm`, `touch`

Questi sono 4 comandi atti a creare e rimuovere directory e files;

- Con `mkdir <name>` si crea una directory di nome `name`
- Con `touch <name>` si crea un file di nome `name`.
- Con `rm` rimuovo un file o una directory VUOTA
  - La flag `-r` permette l'eliminazione delle directory dentro le directory in modo ricorsivo
  - La flag `-f` rende il processo forzato
- Con `mv <file> <dir>` è possibile spostare `file` nella directory `dir`
- Con `rmdir` si elimina una directory vuota

Variabili di argomento e istruzione `for`

Ci sono delle variabili locali che indicano l'indice e il numero degli argomenti passati in un comando/script.

- $# riporta il numero di argomenti passati nel comando/script
- $1, $2, è l'argomento di indice 1, 2, ecc...; $0 è il nome del comando/script stesso

- $* contiene la concatenazione di argomenti escluso il primo. Attenzione, stringhe già concatenate (ossia contenenti spazi) non saranno più considerate concatenate in $*.
- $@ contiene, similmente a un vettore, i singoli argomenti. Quotare con doppi apici non causerà lo stesso problema di $*, pertanto non si avranno problemi a gestire stringhe già concatenate.

E' possibile dichiarare dei cicli all'interno della bash, tramite la seguente forma:

```
for <dummyname> in <list> ; do
    <commands> ;
done
```

do e done equivalgono a delle parentesi graffe, quindi al loro interno si possono