Ciclo while

```
while [condition]
do
command
...
done
```

Versione C-style per condizione su numero:

```
while (( a <= LIMIT ))
  ...
```

Nei cicli for e while si possono usare i soliti `break` e `continue` (se strettamente necessario)

Un loop di inserimento da tastiera

```
while [ "$var" != "end" ]
do
   echo "Inserisci un nome (end per uscire) "
   read var
   echo "Nome inserito = $var"
done
```

- Tutti gli spazi indicati tra le quadre sono necessari al buon funzionamento

Manipolazioni elementari di flussi di testo strutturati

- L'output di molti comandi è costituito da flussi di testo strutturati: ogni riga rappresenta una parte del risultato ed è divisa in campi (fields) ognuno col proprio significato
- Esempio:

```
  PID TTY          TIME CMD
 4264 pts/5    00:00:00 bash
 4707 pts/5    00:00:00 ps
```

- Per questo sono disponibili molti strumenti per la manipolazione di flussi di testo
- Vediamo solo alcuni semplici esempi: tr, cut e l'utilizzo di anchor con grep

Sostituzioni di caratteri: tr

- Il comando tr effettua sostituzioni o eliminazioni di caratteri su un flusso in input inviandone il risultato sullo standard output
- Uso per sostituzioni:
  `tr setchar1 setchar2`

  ogni carattere di setchar1 viene sostituito dal carattere corrispondente per posizione all'interno di setchar2
- I due set devono avere la stessa cardinalità, si estende set2 col suo ultimo char se troppo corto o lo si tronca se troppo lungo
- Esempi:
  `tr "a-z" "A-Z"` converte le minuscole in maiuscole
  `tr "A-Z" "a-z"` converte le maiuscole in minuscole
  `tr "123" "ABC"` sostituisce 1 con A, 2 con B ...
  `tr "0-9" "#"` sostituisce qualunque numero con #

Altri usi di tr

`tr -d setchar1`

Elimina tutti i caratteri inclusi nel setchar1

Esempio:

`tr -d ",.;:!?"` per eliminare la punteggiatura

`tr -s setchar1`

Elimina tutte le ripetizioni dei caratteri inclusi nel setchar1 sostituendole con una loro istanza singola

Esempio:

`tr -s "  "` per eliminare gli spazi ripetuti

Comando cut

- Il comando `cut` serve a selezionare una parte di ogni riga di un flusso di testo e a inviarla sullo standard output
- Le righe si considerano suddivise in fields separati da tab o da altro separatore specificato con l'opzione `-d`
- L'opzione `-f` serve a specificare quali fields (uno o un range) devono essere riprodotti sullo standard output
- Esempi:

  ```
  who | cut -d " " -f1
  ```

  visualizza solo i nomi degli utenti attualmente collegati

  ```
  ps | tr -s " "|cut -d " " -f1
  ```

  visualizza solo la colonna dei PID correnti
  (se PID > 9999, altrimenti serve `-f2` per tener conto di spaziatura iniziale)

Anchor per ricerca stringhe

- Nell'utilizzo di grep (e anche di altri strumenti) i caratteri `^` e `$` sono anchor che specificano la posizione (iniziale o finale) all'interno della stringa

- `^` usato come primo carattere, indica l'inizio della stringa: `^From` indica tutte le stringhe che iniziano con `From`

- `$` usato come ultimo carattere, indica la fine della stringa: : `Ciao$` indica tutte le stringhe che finiscono con `Ciao`

- Esempio:
  `ps| tr -s " "| grep " bash$" | cut -d" " -f1`
  visualizza solo i PID dei processi corrispondenti al comando bash (se PID > 9999, come sopra)

Definire funzioni

E’ possibile definire una funzione in uno script

```
function function_name {
command...
command...
}
```

Sintassi alternativa

```
function_name () {
command...
command...
}
```

e altre varianti sintattiche ragionevoli ...

Invocare funzioni

- Si può invocare una funzione semplicemente scrivendo il suo nome nello script DOPO che è stata definita

```
#!/bin/bash

funky ()
{
  echo "This is a funky function."
  echo "Now exiting funky function."
} # Function declaration must precede call.

funky
```

Argomenti delle funzioni

- Non si dichiarano gli argomenti formali
- E' possibile invocare la funzione con un numero arbitrario di argomenti attuali che, nel corpo della funzione saranno riferibili come $1, $2, ecc: attenzione all'omonimia
- I parametri sono passati per valore
- Sta al programmatore gestire numero e correttezza dei parametri

```
#!/bin/bash
fun  ()
{
 echo $1 # SI STAMPA IL PRIMO PARAMETRO DELLA FUNZIONE
}
echo $1  # SI STAMPA IL PRIMO PARAMETRO DA COMMAND LINE
fun UNASCRITTA
```

Valore di ritorno

- Ogni funzione restituisce un valore, chiamato exit status.
- L'exit status può essere stabilito esplicitamente tramite l'istruzione return, altrimenti è l'exit status dell'ultimo comando nella funzione (0 se OK, diverso da zero altrimenti, come noto).
- L'exit status può essere riferito come $?

```
#!/bin/bash
fun  ()
{
 echo $1
 return 1
}
fun UNASCRITTA
echo $?
```

Scope delle variabili

- Eventuali variabili definite nel corpo di una funzione sono considerate globali, la loro inizializzazione avviene dopo la prima invocazione della funzione

```
#!/bin/bash
fun  ()
{
 nome=PIPPO
}
echo $nome # QUI NON HA VALORE
fun
echo $nome # QUI HA VALORE
```

Variabili locali

- E’ possibile definire variabili locali facendole precedere dalla parola chiave local
- Al solito, tali variabili hanno vita separata per ogni invocazione della funzione

```
#!/bin/bash
fun  ()
{
 local nome=PIPPO
 echo $nome # QUI HA VALORE
}
echo $nome # QUI NON HA VALORE
fun
echo $nome # NEMMENO QUI HA VALORE
```