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 " " -f2
  ```

  visualizza solo la colonna dei PID correnti

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" " -f2`
  visualizza solo i PID dei processi corrispondenti al comando bash

Il case

```
case "$variable" in

stringa1 ) #stringa1 può essere il valore di una variabile p.e. "$varcaso1"
command...
;;

stringa2 ) #stringa2 può essere il valore di una variabile p.e. "$varcaso2"
command...
;;
*)
default command
;;
esac
```

Esempi articolati su: http://tldp.org/LDP/abs/html/testbranch.html

Esempio di case

```
case $( arch ) in #"arch" returns machine architecture.
                  # Equivalent to 'uname -m' ...
i386 ) echo "80386-based machine";;
i486 ) echo "80486-based machine";;
i586 ) echo "Pentium-based machine";;
i686 ) echo "Pentium2+-based machine";;
*    ) echo "Other type of machine";;
esac
```

Altro esempio

```
case "$1" in
  "E" | "e" )
  # Esegui qualcosa corrispondente alla lettera e/E
  ;;
  "J" | "j" )
  # Esegui qualcosa corrispondente alla lettera j/J
 ;;
 * )
  # Tratta il caso di default
;;

esac
```

Ciclo for

```
for variabile in lista
do
  comando1
  comando2
  ....
done
```

- Come solito, il corpo del ciclo viene ripetuto facendo assumere successivamente a `variabile` tutti i possibili valori presenti in `lista`
- `do` e `done` sono comandi: devono stare su linee separate

Ciclo for

- La lista può essere semplicemente specificata come una sequenza di valori separati da spazi (o altri caratteri spaziatori):
  `for nomedocente in Andrea Nicola PietroB PietroM`
- La lista può anche essere ottenuta per filename expansion
  `for file in *.txt`
- La lista può anche essere ottenuta come risultato di un comando

```
for nome in $(cat elenco.txt)
   ...
for file in $(find . -name *.tex)
   ...
```

- Esiste (ma dovrebbe essere di uso raro) la versione C-style

```
for ((a=1; a <= 10 ; a++))
  ...
```

  Equivalente per shell-puristi: `for i in $(seq 1 10)`

Ciclo sugli argomenti della linea di comando

- Se non si specifica alcuna lista (omettendo anche la keyword `in`) in un ciclo `for` è sottintesa quella degli argomenti passati da linea di comando
- In alcuni casi può essere necessario eseguire un ciclo solo sugli argomenti da una certa posizione in poi (p.e. se nelle prime posizioni c'e' la specifica di un'opzione)
- A tale scopo si può usare il comando `shift` che “elimina” uno o più argomenti nelle posizioni iniziali dalla linea di comando: in questo modo un eventuale ciclo opera solo sugli argomenti rimanenti
- Ovviamente `shift` viene applicato dopo aver opportunamente usato il valore dei primi argomenti (che altrimenti verrebbero persi e ignorati)

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