Lezione 3

- *Ridefinire il comando `rm` in modo tale che non sia chiesta conferma prima della cancellazione dei file.*
  `alias rm='rm -f'`

- *Definire il comando `rmi` (`rm` interattivo) che chiede conferma prima di rimuovere un file.*
  `alias rmi='rm -i'`

- *Sapendo che il comando `ps` serve ad elencare i processi del sistema, scrivere una pipeline che fornisca in output il numero di tutti i processi in esecuzione.*
  `ps -e --no-headers | wc -l`

- *Salvare in un file di testo l'output dell'ultimo evento contenente il comando `ls`.*
  `!?ls? > output_ls.txt`

- *Scrivere un comando che fornisce il numero dei comandi contenuti nella history list.*
  Digitare
  `wc -l ~/.bash_history`
  oppure
  `history | wc -l`
  Nel primo caso il conteggio è relativo al contenuto del file `.bash_history` contenuto nella propria home directory (si noti l'uso del metacarattere `~` per far riferimento al percorso della propria home). Siccome tale file viene salvato (aggiornato) al momento del logout, la lista di comandi in esso contenuta non conterrà gli ultimi comandi digitati a partire dall'ultimo login.
  Nel secondo caso invece il conteggio comprende tutti i comandi dell'history list, compreso il comando `history | wc -l` stesso.

- *Scrivere un comando che fornisce i primi 15 comandi della history list.*
  `history | head -15`

- *Quali sono i comandi Unix disponibili nel sistema che iniziano con `lo`?*

  ```
  > lo<Tab><Tab>
  load-gnomecard-addressbook  lockfile
  load-pine-addressbook       log_server_status
  loadkeys                    logger
  loadshlib                   login
  loadunimap                  logname
  local                       logout
  locale                      look
  localedef                   lookbib
  ```

  (ovviamente l'output può variare a seconda del sistema).

- *Fornire almeno due modi diversi per ottenere la lista dei file della vostra home directory il cui nome inizia con `al`.*

- `ls -d ~/al*`
- `ls ~/al<Tab><Tab>`

- *Qual è l'effetto dei seguenti comandi?*
  - `ls -R || (echo file non accessibili > tmp)`
  - `(who | grep rossi) && cd ~rossi`
  - `(cd / ; pwd ;  ls | wc -l )`

  - `ls -R || (echo file non accessibili > tmp)` produce una lista ricorsiva dei file e delle directory contenuti nella directory corrente stampando il messaggio `file non accessibili` nel file `tmp` nella directory corrente nel caso in cui `ls -R` fallisca, i.e., nel caso in cui si incontrino file su cui non si hanno i permessi di lettura.
  - `(who | grep rossi) && cd ~rossi` cambia la directory corrente in quella home dell'utente `rossi` nel caso in cui quest'ultimo sia collegato al sistema (i.e., se il comando composto `(who | grep rossi)` ha successo).
  - `(cd / ; pwd ;  ls | wc -l )` cambia la directory corrente in `/` (comando `cd /`), la stampa su standard output (comando `pwd`), stampa su standard output il numero di elementi contenuti nella directory corrente +1, a causa dell'intestazione (comando `ls | wc -l`).

- *Qual è la differenza tra **programma** e **processo**?*
  Un programma è un'entità **statica** che risiede in un file su disco, mentre un processo è la sua controparte **dinamica** generata da un'esecuzione del programma stesso. In particolare quindi un singolo programma può dar luogo all'esecuzione di diversi processi (ognuno con una propria area di memoria ed un proprio stato).

- *Qual è la differenza tra **processo** e **job**?*
  L'esecuzione di un programma genera un processo; in particolare l'esecuzione di una pipeline comporta la generazione di diversi processi corrispondenti ai vari comandi che occorrono nella pipeline. Quest'ultima genera invece **un** singolo job: quindi tutti i comandi utilizzati nella pipeline in realtà contribuiscono alla creazione di un unico job.

- *Scrivere una pipeline che fornisca in output il numero di processi appartenenti all'utente `root`.*
  Digitare la pipeline `ps -u root --no-headers | wc -l`

- *Il comando*

  `> emacs &`

  *provoca l'avvio di un processo in **background**. Invece il comando*

  `> emacs`

  *provoca l'avvio di un processo in **foreground**. Come si può mandare tale processo in esecuzione in background in modo da rendere il terminale nuovamente disponibile per l'invio di ulteriori comandi?*
  Premere `Ctrl-z` e immettere il comando `bg`.