Soluzioni Esercitazione di Venerdì 16 Marzo 2018

Nota bene: verrà usato all'inizio delle linee che rappresentano i vari comandi il simbolo $ per indicare il prompt dei comandi!

CAT

1) Usando la ridirezione dello standard output del filtro cat, creare un file di nome prova (inserire almeno 5-6 linee con più parole per linea).

```
$ cat > prova
```

Ricordarsi che la fine dei caratteri dallo standard input si ottiene con ^D (CTRL-D).

2) Usando la ridirezione dello standard output del comando pwd e del comando ls -l, aggiungere tale output al file prova.

```
$ pwd >> prova
$ ls -l >> prova
```

3) Usando il comando cat con le opportune ridirezioni, creare una copia del file prova e dargli nome p.txt.

```
$ cat < prova > p.txt
```

4) Usando più volte la ridirezione dello standard output in append aggiungere il contenuto del file prova al file p.txt almeno altre 5 volte in modo da avere un contenuto di molte linee.

```
$ cat prova >> p.txt
$ cat prova >> p.txt
$ cat prova >> p.txt
$ cat prova >> p.txt
$ cat prova >> p.txt
```

MORE

5) Usando la ridirezione dello standard input e il filtro more, visualizzare il contenuto del file p.txt.

```
$ more < p.txt
```

Ricordarsi che per andare avanti di una videata si deve usare la barra spaziatrice, per avanzare di una singola linea si deve usare il tasto enter (invio) e per uscire se non si vuole più proseguire si deve usare il tasto q (QUIT).

6) Verificare il comportamento del comando more, usando il comando more p*.

`$ more p*` ➔ visualizza contenuto dei due file (prova e p.txt) con l'intestazione del nome del file e in caso di p.txt una videata alla volta.

SORT

7) Usando la ridirezione dello standard input e il filtro sort, verificare se il file di nome prova è ordinato alfabeticamente.

`$ sort -c < prova` ➔ indica su quale linea l'ordinamento non è soddisfatto;
dato che probabilmente questa informazione riportata sullo standard ERROR non ci interessa vale la pena usare il comando in questo modo:

```
$ sort -c < prova 2> /dev/null
```

Sia nel caso precedente che in questo, il valore di ritorno del comando indica insuccesso

`$ echo $?` ➔ 1

Vediamo un altro modo per ottenere lo stesso risultato

`$ sort -C < prova` ➔ NON scrive nulla e quindi per sapere se il file è ordinato o meno si deve usare

`$ echo $?` ➔ 1

8) Usando la ridirezione dello standard input e il filtro sort, mostrare il contenuto del file prova ordinato secondo il normale ordinamento alfabetico.

```
$ sort < prova
```

ATTENZIONE: SE NON VIENE RIPORTATO L'ORDINAMENTO ALFABETICO BISOGNA ESEGUIRE QUESTI DUE COMANDI PRIMA DEL SORT

```
$ LC_ALL=C
$ export LC_ALL
```

9) Stessa cosa di 8 ma invertendo l'ordinamento.

```
$ sort -r < prova
```

10) Stessa cosa di 8, ma ignorando la differenza fra maiuscole e minuscole.
```
$ sort -f < prova
```
11) Stessa cosa di 8, ma ridirigendo lo standard output su un file di nome ordinato.
```
$ sort < prova > ordinato
```
12) Stessa cosa di 7, ma sul file di nome ordinato.
```
$ sort -c < ordinato 2> /dev/null
$ echo $?
```
oppure
```
$ sort -C < ordinato; echo $?
```
13) Editare il file prova andando a duplicare e/o triplicare alcune linee.
`$ vi (o vim) prova` ➔ copia: Y; incolla: p oppure P
14) Stessa cosa di 8, ma attuando l'ordinamento alfabetico senza replicazioni e scrivendo lo standard output su un file di nome ordinato-senza-doppi.
```
$ sort -u < prova > ordinato-senza-doppi
```

GREP

15) Usando la ridirezione dello standard input e il filtro grep, cercare le linee che contengono una certa stringa (o anche un semplice carattere) nel file ordinato-senza-doppi.
```
$ grep hello < ordinato-senza-doppi
```
16) Stessa cosa di 15, ma mostrando i numeri di linea.
```
$ grep -n hello < ordinato-senza-doppi
```
17) Assicurandosi di avere nel file ordinato-senza-doppi la stessa stringa scritta in maiuscola e in maiuscolo almeno 2-3 volte, stessa cosa di 15, ma cercando la stringa ignorando maiuscole/minuscole.
```
$ grep -i hello < ordinato-senza-doppi
```
18) Stessa cosa di 15, ma cercando le linee che NON contengono una certa stringa (o anche un semplice carattere).
```
$ grep -v hello < ordinato-senza-doppi
```
19) Stessa cosa di 15, ma cercando solo le linee che INIZIANO per una certa stringa (o anche un semplice carattere).
```
$ grep  '^hello ' < ordinato-senza-doppi
```
20) Stessa cosa di 15, ma cercando solo le linee che TERMINANO per una certa stringa (o anche un semplice carattere).
```
$ grep 'hello$ ' < ordinato-senza-doppi
```
21) Stessa cosa di 15, ma cercando solo le linee che TERMINANO per il carattere '.' (PUNTO).
```
$ grep  '\.$ ' < ordinato-senza-doppi
```
22) Utilizzando la soluzione di uno degli esercizi fra 15 e 21, ridirigere lo standard output in un file di nome prova-grep.
```
$ grep  '\.$ ' < ordinato-senza-doppi > prova-grep
```

REV

23) Usando la ridirezione dello standard input e il filtro rev, verificare lo standard output utilizzando il file prova-grep.
```
$ rev < prova-grep
```

HEAD e TAIL

24) Usando la ridirezione dello standard input e il filtro head, selezionare le prime 10 linee del file p.txt (vedi esercizio 4).
`$ head < p.txt` ➔ head senza opzioni, di default riporta le prime 10 linee
25) Stessa cosa di 24, ma selezionando la prima linea.
```
$ head -1 < p.txt
```
Nota bene: è possibile anche usare la forma seguente
```
$ head -n 1 < p.txt
```
26) Stessa cosa di 24, ma selezionando le prime 3 linee.
```
$ head -3 < p.txt
```
27) Usando la ridirezione dello standard input e il filtro tail, selezionare le ultime 10 linee del file p.txt.
`$ tail < p.txt` ➔ anche tail senza opzioni, di default riporta le prime 10 linee

28) Stessa cosa di 27, ma selezionando l'ultima linea.

```
$ tail -1 < p.txt
```

Nota bene: è possibile anche usare la forma seguente

```
$ tail -n 1 < p.txt
```

29) Stessa cosa di 27, ma selezionando le ultime 3 linee.

```
$ tail -3 < p.txt
```

30) Utilizzando la soluzione di uno degli esercizi fra 24 e 29, ridirigere lo standard output in un file di nome prova-head o prova-tail a seconda dei casi.

```
$ tail -3 < p.txt > prova-tail
```

31) Utilizzando il piping dei comandi, isolare in un file di nome p.txt.terza la terza linea a partire dall'inizio del file p.txt.

```
$ head -3 < p.txt | tail -1 > p.txt.terza
```

Nota bene: è possibile anche usare la forma seguente

```
$ head -3 p.txt | tail –n 1 > p.txt.terza
```

➔ Nota bene: head come comando

32) Utilizzando il piping dei comandi, isolare in un file di nome p.txt.terzultima la terza linea a partire dalla fine del file p.txt.

```
$ tail -3 < p.txt | head -1 > p.txt.terzultima
```

Nota bene: è possibile anche usare la forma seguente

```
$ tail -3 p.txt | head –n 1 > p.txt.terzultima
```

➔ Nota bene: tail come comando

WC

33) Usando la ridirezione dello standard input e il filtro wc, contare le linee del file p.txt (vedi esercizio 4)

```
$ wc -l < p.txt
```

34) Stessa cosa di 33, ma contando i caratteri.

```
$ wc -c < p.txt
```

35) Stessa cosa di 35, ma contando le parole.

```
$ wc -w < p.txt
```

36) Usando la soluzione dell'esercizio 33, calcolare la somma del numero di linee dei file p.txt e prova.

```
$ expr `wc -l < p.txt` + `wc -l < prova`
```

➔ Nota bene: APICI ROVESCI!

37) Usando il comando wc su un file di nome pippo (che non esiste) ridirigendo lo standard error su /dev/null , verificare il valore di ritorno del comando/.

```
$ wc -l pippo 2> /dev/null; echo $?
```

➔ Nota bene: si chiede di usare il comando e NON il filtro!

38) Utilizzando il comando ps in piping con tee t e in piping con wc –l verificare: a) il numero visualizzato; b) il contenuto del file di nome t creato dal filtro tee.

```
$ ps | tee t | wc –l ➔ 5
$ cat t
  PID TTY          TIME CMD
28497 pts/1    00:00:00 bash
28516 pts/1    00:00:00 ps
28517 pts/1    00:00:00 tee
28518 pts/1    00:00:00 wc
```

VARIABILI

39) Memorizzare in una variabile di nome x, il numero di linee del file p.txt.

```
$ x=`wc -l < p.txt`
```

40) Memorizzare in una variabile di nome y, il numero di linee del file prova.

```
$ y=`wc -l < prova`
```

41) Memorizzare in una variabile di nome z, la somma della variabile x e della variabile y e visualizzarne il valore.

```
$ z=`expr $x + $y`
```

42) Con un editor, scrivere un file comandi prova.sh che visualizzi il valore delle variabili x, y e z inserendo anche dei commenti significativi. Rendere eseguibile tale file comandi (verificare che sia eseguibile con ls –l prova.sh) e mandarlo in esecuzione. Quale è il risultato?

File prova-inibizioni.sh:

```
#!/bin/sh
echo x = $x #stampo il valore della variabile x
echo y = $y #stampo il valore della variabile y
echo z = $z #stampo il valore della variabile z
```

```
$ chmod +x prova.sh
$ prova.sh ➔ oppure ./prova.sh se nella var. di ambiente PATH non c'è il .
  x =
  y =
  z =
```

43) Rendere la variabile z una variabile di ambiente e riprovare ad eseguire prova.sh. Quale è il risultato?

```
$ export z
$ prova.sh ➔ oppure ./prova.sh
  x =
  y =
  z = 32
```

44) Verificare con il comando env, la presenza di z nell'ambiente.

```
$ env
```

45) Editare il file prova.sh, aggiungendo un comando che modifica il valore della variabile z e lo visualizza nuovamente, sempre inserendo commenti significativi. Riprovare ad eseguire prova.sh. Quale è il risultato? Quale valore ha la variabile z nella shell interattiva?

File prova-inibizioni.sh:

```
#!/bin/sh
echo x = $x #stampo il valore della variabile x
echo y = $y #stampo il valore della variabile y
echo z = $z #stampo il valore della variabile z
z=0     #aggiorno z
echo z = $z #stampo il valore aggiornato della variabile z
```

```
$ prova.sh ➔ oppure ./prova.sh
  x =
  y =
  z = 32
  z = 0
$ echo $z
  32
```

METACARATTERI

46) Utilizzando la ridirezione a vuoto, creare diversi file con nomi che iniziano e terminano con varie lettere dell'alfabeto maiuscole e minuscole e con numeri; creare anche un paio di directory. Verificare quindi il comportamento dei comandi:
   a) echo [a-z]*
   b) echo [A-Z]*
   c) echo [0-9]*
   d) echo *[a-z]
   e) echo *[A-Z]
   f) echo *[0-9]

   Rifare i comandi precedenti usando la negazione: ad esempio per a) echo [!a-z]*

   Utilizzare nuovamente i pattern precedenti utilizzando però il comando ls –l: che cosa cambia?

```
$ > test1 ➔ creazione file utilizzando la ridirezione a vuoto
$ > test2
$ > test3
$ > 1234
$ > test_only_char
$ > testUPPERCASE
$ mkdir testDir1 ➔ creazione directory
```

```
$ mkdir TESTDIR2
$ mkdir simple_dir
$ > simple_dir/test_in_directory ➔ creazione file nella dir. simple_dir
$ mkdir 12345
$ echo [a-z]* ➔ simple_dir test1 test2 test3 testDir1 testUPPERCASE
test_only_char
$ echo [A-Z]* ➔ TESTDIR2
$ echo [0-9]* ➔  1234 12345
$ echo *[a-z] ➔  simple_dir test_only_char
$ echo *[A-Z] ➔  testUPPERCASE
$ echo *[0-9] ➔  1234 12345 TESTDIR2 test1 test2 test3 testDir1
$ echo [!a-z]* ➔ 1234 12345 TESTDIR2
$ ls -l [a-z]*
    -rw-r--r--  1 utente  staff  0 16 Mar 15:36 test1
    -rw-r--r--  1 utente  staff  0 16 Mar 15:36 test2
    -rw-r--r--  1 utente  staff  0 16 Mar 15:36 test3
    -rw-r--r--  1 utente  staff  0 16 Mar 15:53 testUPPERCASE
    -rw-r--r--  1 utente  staff  0 16 Mar 15:36 test_only_char

    simple_dir:
    total 0
    -rw-r--r--  1 utente  staff  0 16 Mar 16:20 test_in_directory

    testDir1:
```

INIBIZIONI

47) Con un editor, scrivere un file comandi prova-inibizioni.sh che assegna ad una variabile di shell di nome a la stringa ciao; quindi visualizzare con il comando echo il valore di a, della directory corrente e della espansione del metacarattere *. Inserire dei commenti significativi. Rendere eseguibile il file comandi e mandarlo in esecuzione. Quale è il risultato?

```
File prova-inibizioni.sh:
  #!/bin/sh
  a=ciao #assegno alla variabile a la parola ciao
  echo $a `pwd` * # stampo il contenuto di a l'output del comando pwd e
                    l'espansione di *
$ chmod +x prova-inibizione.sh
$ ./prova-inibizione.sh ➔ verificare l'output
```

48) Modificare con un editor il file comandi prova-inibizioni.sh e ricopiare la linea con il comando echo andando a inibire TUTTE le sostituzioni. Mandarlo nuovamente in esecuzione. Quale è il risultato?

```
File prova-inibizioni.sh:
  #!/bin/sh
  a=ciao #assegno alla variabile a la parola ciao
  echo $a `pwd` * # stampo il contenuto di a l'output del comando pwd e
                    l'espansione
  echo '$a pwd *' # inibisco le espansioni
$./prova-inibizione.sh ➔  verificare l'output
```

49) Modificare con un editor il file comandi prova-inibizioni.sh e ricopiare la linea con il comando echo andando a inibire SOLO l'ultima sostituzione. Mandarlo nuovamente in esecuzione. Quale è il risultato?

```
File prova-inibizioni.sh:
  #!/bin/sh
  a=ciao #assegno alla variabile a la parola ciao
  echo $a `pwd` * # stampo il contenuto di a l'output del comando pwd e
                    l'espansione
  echo '$a pwd *' # inibisco le espansioni
  echo "$a `pwd` *" # inibisco solo l'ultima espansione
```

`$ ./prova-inibizione.sh` ➔ *verificare l'output*

50) Assegnare ad una variabile di shell di nome l la stringa ls –l $z. Fare eseguire il comando con $l. Quale è il risultato? Assegnare alla variabile di shell z il valore prova-inibizioni.sh. Fare eseguire di nuovo il comando con $l. Il risultato è cambiato rispetto a prima? Provare ora il comando eval $l: cosa cambia?

`$ l='ls -l $z'` ➔ assegno alla variabile l la stringa

`$ $l` ➔ esegue ls -l $z ➔ ls: $z: No such file or directory

`$ z=prova-inibizioni.sh`

`$ $l` ➔ rieseguo ls -l $z con z avente valore ➔ ls: $z: No such file or directory ➔ ANCORA ERRORE!

`$ eval $l` ➔ funziona!!!

```
-rwxr-xr-x  1  utente  staff  231 16 Mar 16:28 prova-inibizioni.sh
```