Scrivere testi - ed

Un programma che consente di creare file testo si dice ***editor***.
L'editor standard e più semplice di UNIX è `ed`.

Se è invocato con `ed file`, `ed` ha inizialmente:

- ***file di lavoro*** `file`, e
- ***buffer*** o ***area di lavoro*** contenente il testo di `file`

Se `ed` è invocato senza argomento non ha file di lavoro e il suo buffer è vuoto.

Il comando a (*appendi*) prepara all'inserimento di testo alla fine del buffer

```
~ $ ed
a
Ne mai piu tocchero le sacre sponde
ove il mio corpo fanciulletto giacque
```

per smettere di inserire testo, battete un `.` ad inizio di riga

Il comando **`w file`** scrive il contenuto del buffer su `file`:
se `file` non esiste viene creato, se esiste viene sovrascritto

`w` da solo scrive il buffer sul file di lavoro: se questo non è definito, `ed` risponde con ?

```
.
w
?
w poesia
74
```

Come si vede, `ed` risponde a `w` con il numero di caratteri che sono stati scritti.

ed: uscita

Dopo aver scritto con **w**, potete uscire con il comando **q**

```
q
~ $ cat poesia
Ne mai piu tocchero le sacre sponde
ove il mio corpo fanciulletto giacque
```

Se ed viene invocato su un file che non esiste, avverte:

```
~ $ ed nuovo
?nuovo
q
```

Se invece il file esiste, ed ne mostra la dimensione:

```
~ $ ed poesia
74
```

Aggiungiamo una riga al testo di poesia:

```
a
Zacinto mia che te specchi nell'onde
.
```

Adesso cerchiamo di uscire con **q**, senza aver salvato prima con **w**:
ed risponde con un ?: se insistiamo con **q** ci fa uscire, ma perdiamo il lavoro:

```
q
?
q
~ $ wc poesia
      2     13      74   poesia
```

Come si vede, poesia ha ancora solo 2 righe.

Il comando **Q** fa uscire subito (ed non dà nemmeno il ?):

```
~ $ ed poesia
97
a
non volevo aggiungere niente!
.
Q
~ $
```

ed, cont.

Sotto ed (ma non durante l'inserimento di testo), si può eseguire un comando UNIX senza bisogno di uscire: basta far precedere il comando da `!`.

```
~ $ ed poesia
74
!wc poesia
      2     13      74   poesia
!
```

Per vedere la riga 1,2,.... del buffer basta il comando `1` o `2`.... o `$` per l'ultima riga,

```
1
Ne mai piu tocchero le sacre sponde
$
ove il mio corpo fanciulletto giacque
```

Fatto questo, le righe successive si possono vedere premendo il tasto `Return`.
Per vedere le righe da *m* a *n* (*n* può essere $), si usa il comando *m*`,`*n*p.

```
1,$p
Ne mai piu tocchero le sacre sponde
ove il mio corpo fanciulletto giacque
```

Per andare indietro di una riga si usa `-`, di *n* righe `-`*n*.
I numeri di riga si possono combinare con + e -:

$-2,$p	*stampa le ultime 3 righe*
1,2+3p	*stampa le prime 5 righe*
$,1p	*è illegale*

Aggiungiamo una riga:

```
a
In che senso sacre?
.
```

Pattern

Su un grosso file, cercare una riga visualizzandole una per una è lento.
Si può visualizzare una riga che contiene i caratteri `sacre` con il comando **`/sacre/`**

```
/sacre/
Ne mai piu tocchero le sacre sponde
```

`//` ricerca ulteriori righe che contengano `sacre`

```
//
In che senso sacre?
```

`?corpo?` cerca *all'indietro* una riga contenente `corpo`; **`??`** ripete la ricerca indietro.

```
?corpo?
ove il mio corpo fanciulletto giacque
```

Più in generale, si possono usare ***pattern***, come in `grep`.

Un pattern è a tutti gli effetti un numero di riga →si può premettere a un comando:

`1,/sacre/p`	stampa da 1 al primo `sacre`
`?sacre?+1,$p`	stampa dal `sacre` precedente + 1 riga alla fine

Un comando *command* si può rendere ***globale*** così:

m,*n*g/*pattern*/*command*

questo significa: tra le righe *m* e *n* applica *command* a quelle selezionate da *pattern*; per default il range *m*,*n* è 1,$; p.es.:

g/*casa*/*p*

Con v al posto di g, si lavora sulle righe che ***non*** sono selezionate.

ed - modifiche

Ad ogni istante, una riga del buffer si dice ***corrente***.
Inizialmente è l'ultima; quando si esce dall'inserimento è l'ultima inserita;
quando si visualizza una riga, questa diventa corrente.

Per controllare qual è la riga corrente si usa il comando `.` o `.n` (dà anche il numero)
(**NB** il comando `.` si usa anche, *in un altro contesto*, per uscire dall'inserimento).

```
/corpo/
ove il mio corpo fanciulletto giacque
.n
2:  ove il mio corpo fanciulletto giacque
```

Modificare testo nel buffer:

*n*a	entra in modalità di inserimento, il testo viene aggiunto dopo la riga *n*
*n*i	inserisce testo prima della riga *n*
m,*n*d	cancella le righe dalla *m* alla *n*
m,*n*c	cambia righe da *m* a a *n* (scrivendo sopra)

se non vengono specificati *n* o *m* e *n*, i comandi hanno effetto sulla riga corrente.

Unire righe*: m,n*j unisce su una sola riga le righe da *m* a *n*

Dividere righe: si sostituisce un "a capo" quotato da \:

s/*parte1parte2*/*parte1*\ *parte2*/	*questo "a capo" verrà inserito tra parte1 e parte2*
s/ /\ /g	*sostituisce ogni spazio (anche quelli tra parole) con "a capo"* *quindi mette una parola per riga*

Uso dei file

f	mostra il nome del file di lavoro;
f *file*	cambia il nome del file di lavoro in *file*; non cambia il buffer di lavoro;
e	copia il file di lavoro nel buffer di lavoro
e *file*	rende *file* il file di lavoro e lo copia nel buffer di lavoro (come se ed ripartisse con *file*)
w *file*	scrive il buffer di lavoro su *file* (per default, il file di lavoro)
*n*r *file*	legge *file* inserendolo dopo la riga *n*

ed - sostituzione

Avrete notato gli errori di ortografia nel file `poesia`:

```
~ $ ed poesia
97
1
Ne mai piu tocchero le sacre sponde
```

Sostituire “Ne” con “Ne'” (le vocali accentate potrebbero non essere usabili):

```
s/Ne/Ne'/
p
Ne' mai piu tocchero le sacre sponde
```

`s/o/o'/g` sostituisce *tutte* le o sulla riga corrente con `o'`:

```
s/o/o'/g
p
Ne' mai piu to'cchero' le sacre spo'nde
```

Adesso eliminare le `o'` scorrette (la p alla fine della sostituzione ne mostra l’effetto):

```
s/to'/to/p
Ne' mai piu tocchero' le sacre spo'nde
s/o'n/on/p
Ne' mai piu tocchero' le sacre sponde
```

Per applicare la sostituzione non alla riga corrente ma alle righe da *m* a *n*: *m*`,`*n*`s/`*pattern1*`/`*pattern2*`/g`:

```
1,$s/o/0/g
1,$p
Ne' mai piu t0ccher0' le sacre sp0nde
0ve il mi0 c0rp0 fanciullett0 giacque
In che sens0 sacre?
```

Per *annullare* una sostituzione indesiderata si usa il comando **u**:

```
u
1,$p
Ne' mai piu tocchero' le sacre sponde
ove il mio corpo fanciulletto giacque
In che senso sacre?
```

Il carattere `&` nella stringa che fa da rimpiazzamento sta per la stringa rimpiazzata:

`s/big/very &/`	*sostituisce* `big` *con* `very big`
`s/and/\&/`	*sostituisce* `and` *con il carattere* `&`

Inviare testo in uscita: `cat`

Finora, per visualizzare sul terminale il contenuto di un file `f`, si è usato il comando `cat f`.

Ma `cat` ha una forma più generale:

`cat` *f1 f2 ...*

Questa *concatena* i file *f1 f2 ...* e li invia sulla standard output:

```
~ $ cat agenda
oggi studio
domani studio
~ $ cat telefoni
245676
~ $ cat agenda telefoni >agenda.compl
~ $ cat agenda.compl
oggi studio
domani studio
245676
```

NB: `cat` *f1 f2* > *f1* causa la perdita dei dati originari di *f1*

Ecco come si può creare un file lungo da uno di tre righe:

```
~ $ wc poesia
      3     17      94   poesia
~ $ cat poesia poesia poesia >poesia3
~ $ cat poesia3 poesia3 > poesia6
~ $ cat poesia6 poesia6> long
```

Visualizzare file: `more`

Per visualizzare il file `long` una schermata alla volta, si usa

`more long`

comparirà la prima schermata di `long`.

I principali comandi accettati da `more` a questo punto sono:

barra spaziatrice	passa alla schermata successiva
b	torna indietro di una schermata
q	esce
h	(help) per avere informazioni su altri comandi

```
~ $ more long
Ne mai piu tocchero le sacre sponde
ove il mio corpo fanciulletto giacque
...
...
--More--
```

Come quasi tutti i comandi, `more` senza argomenti legge dalla standard input.

`more` si può dunque usare in pipe con i comandi che generano più di una schermata di testo. P.es.

```
~ $ ls -R /          (genera molte righe!!!)
...
...
~ $ ls -R / | more
...
...                              (prima schermata)
--More--
```

Stampa di un file

Su alcune versioni di UNIX, il comando per stampare è `lp`,
ma sulla maggior parte è:

`lpr` [*opzioni*] [*f1 f2* ...]

Alcune caratteristiche di `lpr`:

- per default, `lpr` stampa la standard input.
- `lpr` inserisce il testo da stampare in una *coda di stampa*:
- sia *t* l'istante in cui viene eseguito `lpr`: la stampa viene effettivamente effettuata da un processo detto *print daemon*, a un istante *t'*>*t*, quando la stampante diventa disponibile
- `lpr` *f* stampa *f* come era al tempo *t* (perché `lpr` per default copia subito *f* nella coda di stampa)
- `lpr -s` *f* stampa *f* come è al tempo *t'* (`lpr -s` copia un link nella coda di stampa).
- `lpr -r` *f* cancella *f* dopo averlo messo in coda;
- `lpr -r -s` *f* cancella *f* dopo averlo stampato (cioè al tempo *t'*)

Il comando per visualizzare la coda di stampa dei job con i loro identificatori è:

`lpq`

Il comando per rimuovere un file con identificatore *job-id* dalla coda di stampa è:

`lprm` *job-id*

Formattare la stampa: `pr`

Output di `lpr` *f* non formattato (p.es. con intestazioni o salti pagina)

Il comando `pr` formatta il suo input (file o stdin) e lo invia su stdout.

`pr` ha numerose opzioni, tra cui:

`pr +`*p*	comincia a stampare da pagina *p*
`pr –l`*n*	imposta la lunghezza della pagina a *n* righe, questo è sfruttato nell'esempio su questo lucido
`pr –n`*d*	numera le righe di testo con *d* cifre (default 4)
`pr –t`	elimina intestazione e piè di pagina
`pr –o`*m*	rientra ogni riga di *m* spazi (ampiezza tot riga w*c*+o*m*)

```
~ $ pr -l15 -n f > g          f contiene 7 righe
~ $ cat g


May 03 19:43 1999  f Page 1


    1 riga di f
    2 riga di f
    3 riga di f
    4 rigo di f
    5 riga di f




                        ultima riga di Page 1


May 03 19:43 1999  f Page 2


    6 riga di f
    7 riga di f
```