

---

# Manuale sull'editor VI

---

A cura di Giuseppe Ciaburro

<http://www.ciaburro.it>

[info@ciaburro.it](mailto:info@ciaburro.it)

# Indice

<b>1</b>	<b>Per iniziare</b>	<b>3</b>
<b>2</b>	<b>Comandi operativi</b>	<b>5</b>
<b>3</b>	<b>Comandi per il controllo dei file</b>	<b>13</b>
<b>4</b>	<b>Opzioni</b>	<b>21</b>
<b>5</b>	<b>Macro</b>	<b>24</b>
<b>6</b>	<b>Suggerimenti</b>	<b>27</b>

# Capitolo 1

## Per iniziare

L'editor "vi" viene invocato con il seguente comando:

```
vi [-r] [+comando] [file1 [file2 ... fileN]]
```

dove:

- o -r è un'opzione da introdurre dopo un crash del sistema (per recuperare i cambiamenti eventuali non memorizzati nella sessione interrotta);
- o +comando apre la sessione dell'editor eseguendo il "comando" (sono contemplati solo i comandi di editor di linea).

Quando vengono specificati più nomi di file da editare, 'vi' inizia l'editing sul primo file suggerito, e su richiesta (comando ':n') passerà seguendo l'ordine ai successivi. L'editor vi visualizza una schermata di file alla volta, permettendo lo spostamento del cursore con appositi comandi al punto dove si vuole inserire o aggiornare il testo. Per un funzionamento ottimale è necessario conoscere l'esatto tipo di terminale a disposizione. Verrà quindi settata la variabile TERM con la sintassi propria del sistema su cui si lavora; per default, la sintassi standard è:

```
TERM=type ; export TERM
```

dove "type" corrisponde ad un nome di terminale (con tutte le funzionalità ad esso legate) descritto nella libreria "terminfo". Nella prima schermata di "vi", compariranno le prime 24 righe del file richiamato (con "~" come primo carattere sulle righe vuote), e ci si troverà in COMMAND MODE. Quest'ultimo è uno dei tre possibili stati dell'editor "vi".

### 1)COMMAND MODE

Il cursore è posizionato sul testo, la tastiera è utilizzabile solo per richiedere l'esecuzione di comandi, e non per introdurre testo. I caratteri digitati non vengono visualizzati.

### 2)INPUT MODE

Tutti i caratteri digitati vengono visualizzati ed inseriti nel testo. Si passa in input mode ad ogni richiesta di inserimento di testo.

### 3)DIRECTIVE MODE

Ci si trova (tramite ":") posizionati con il cursore nella linea direttive (l'ultima linea del video) e si possono richiedere a "vi" tutti i comandi per il controllo del file.

I passaggi di stato avvengono con i seguenti caratteri:

- 1) DIRECTIVE MODE -----> COMMAND MODE <RET>
- 2) COMMAND MODE -----> INPUT MODE oOiiAa
  - o apre una nuova linea al di sotto della linea corrente e permette l'inserimento del testo;
  - O apre una nuova linea al di sopra della linea corrente e permette l'inserimento del testo;
  - i inserisce testo prima del cursore;
  - I inserisce testo prima dell'inizio della linea corrente;
  - a aggiunge testo dopo il cursore;
  - A aggiunge testo dopo la fine della linea corrente;
- 3) INPUT MODE -----> COMMAND MODE <ESC>
- 4) COMMAND MODE -----> DIRECTIVE MODE :/?

E' inoltre possibile lanciare comandi appartenenti alla shell da cui è stato richiamato il "vi" stesso, tramite l'apposito escape alla shell (questo comando verrà trattato meglio più avanti):

:!comando

Per eliminare del testo, 'vi' ha un singolo buffer senza nome, nel quale viene salvata l'ultima porzione di testo cancellata, nove buffer numerati (1 2 ... 9) ed un insieme di buffer individuati dalle lettere dell'alfabeto (a b ... z) a disposizione dell'utente. I buffer, in caso di editing di più file in successione, non vengono cancellati nel passare da un file all'altro, permettendo così spostamenti di parte di testo tra file diversi. Naturalmente possono anche essere utilizzati per cancellazioni e ripristini o per spostamenti di parti di testo all'interno di uno stesso file.

## Capitolo 2

# Comandi operativi

Per utilizzare i comandi di spostamento ci si deve trovare in COMMAND MODE.

### MOVIMENTO DEL CURSORE

```
h      1 spazio a sinistra (come backspace)
l      1 spazio a destra (come space)
k      1 linea sopra (stessa colonna)
j      1 linea sotto (stessa colonna)
```

(in sostituzione di questi quattro caratteri, si possono anche utilizzare le frecce per spostarsi nelle diverse direzioni)

```
G      posizionamento sull'ultima linea del testo
#G     posizionamento sulla linea identificata da "#"
       (dove #=numero)
```

Esempi:      3G    5G    175G

```
:#     stesso funzionamento di #G
#move; ripete "move" "#" volte (dove move=qualsiasi comando di
       spostamento)
```

### RISULTATI A VIDEO

```
^G     mostra lo stato del file corrente (informazioni varie)
```

:= mostra il numero di linee del file  
:.= mostra il numero della linea corrente

^L refresh del video  
^D si posiziona mezza pagina avanti  
(in INSERT MODE shifta indietro di un TAB)  
^U si posiziona mezza pagina indietro  
^F si posiziona sulla pagina successiva  
^B si posiziona sulla pagina precedente

(i tasti di controllo delle pagine, ^D-^U-^F-^B, possono anche essere utilizzati con un opzionale moltiplicatore, come ad esempio:

5^F 5 pagine avanti  
^E sposta la schermata avanti senza spostare il cursore

^Y sposta la schermata indietro senza spostare il cursore  
H si posiziona all'inizio della schermata

M si posiziona al centro della schermata  
L si posiziona al fondo della schermata

<< shifta la linea corrente a sinistra  
<L shifta tutte le linee, da quella corrente alla fine dello schermo, verso sinistra  
>> shifta la linea corrente a destra  
(come TAB in INSERT MODE)

>L shifta tutte le linee, da quella corrente alla fine dello schermo, verso destra

## PARAGRAFO

{ sposta il cursore all'inizio del paragrafo  
corrente (o della sezione, se si è in  
modalità C)  
} sposta il cursore alla fine del paragrafo  
corrente (o della sezione, se si è in  
modalità C)

( si posiziona all'inizio della frase corrente  
) si posiziona alla fine della frase corrente

[[ si posiziona all'inizio di una sezione  
]] si posiziona alla fine di una sezione (è possibile anche associare  
comandi ai precedenti spostamenti)

## LINEA

\$ si posiziona alla fine della linea  
^ si posiziona al primo carattere non-blank della  
linea  
0 si posiziona alla colonna zero (inizio linea)  
  
#| si posiziona ad un'esatta colonna sulla linea  
(colonna "#")  
Esempi: 5| 1| 9|  
w si posiziona all'inizio della parola  
successiva (una parola è delimitata da un carattere non  
alfanumerico)  
e si posiziona alla fine della parola successiva  
  
b torna all'inizio della parola precedente  
  
W,E,B stesso funzionamento di w,e,b (parola delimitata  
da spazio)  
+ posizionamento al primo carattere non-blank  
della linea successiva (come RETURN)  
- posizionamento al primo carattere non-blank  
della linea precedente

## EDIT

ESC ESCAPE: fa uscire dall'INPUT MODE, per terminare  
quindi l'inserimento e tutte le modifiche  
in generale  
(se c'è qualcosa che non va, funziona anche ^C)  
. ripete l'ultimo cambiamento effettuato (o meglio,  
l'ultimo comando dato)  
o OPEN: inserisce una linea sotto quella corrente  
  
O OPEN: inserisce una linea sopra quella corrente  
  
i INSERT: inserisce caratteri a sinistra del  
cursore  
I INSERT: inserisce caratteri all'inizio della  
linea  
a APPEND: aggiunge caratteri a destra del cursore  
  
A APPEND: aggiunge caratteri alla fine della linea  
  
J JOIN: concatena la linea successiva alla  
corrente

#s       SUBSTITUTE: sostituisce per "#" caratteri, di  
          default 1 carattere  
#S       SUBSTITUTE: sostituisce per "#" intere linee,  
          di default 1 linea  
#r       REPLACE: sostituisce "#" caratteri, di default 1  
          carattere (non è necessario premere ESC); con  
          "rcarattere" (dove 'carattere' è un qualsiasi  
          carattere), il carattere di posizionamento del  
          cursore viene sostituito con quello specificato  
          dopo 'r'  
R        REPLACE: entra in modalità di sovrascrittura  
  
c[x]     CHANGE: cambia fino a dove specificato con "x"

Esempio:

cw       sostituisce la parola corrente  
cc       CHANGE: cambia l'intera linea  
  
C        CHANGE: cambia fino alla fine della linea

Nota: in tutte le modalità di inserzione di testo elencate sopra  
(i,a,o,c,r), è possibile anteporre un numero a queste; ciò  
comporterà una duplicazione della funzione desiderata  
tante volte quante specificate con quel numero.

Nei successivi comandi richiesti da INPUT MODE, invece, non è possibile  
specificare il numero delle operazioni da compiere:

^D       elimina un rientro (TAB)  
  
^^D      elimina tutti i rientri per la linea corrente  
          (digitare il carattere '^' e poi un CTRL-D)  
^W      cancella l'ultima parola introdotta  
          (nota: può essere settato con il comando 'ssty')  
^U      cancella l'ultima linea introdotta  
          (nota: può essere settato con il comando 'ssty')  
^H      corrisponde a backspace, cioè torna indietro di  
          un carattere  
          (nota: può essere settato con il comando 'ssty')

Esiste un altro comando richiesto da INPUT MODE:

^V      permette di inserire nel testo il carattere ESC

EXIT

ZZ salva il file ed esce (da COMMAND MODE) :[#,#]w[!] [filename]  
WRITE: scrive senza uscire (nel file "filename",  
se viene specificato, altrimenti nel file  
corrente); vengono utilizzati "#,#" per  
specificare gli indirizzi 'from' e 'to' (da/a),  
e "!" (bang) per forzare la scrittura (cioè  
per riscrivere)

Esempi:

:1,5 w abcd scrive le linee da 1 a 5 nel file abcd

:/from/,/to/ w efg scrive da 'from' a 'to' nel file efg

:/from/,+5 w hij scrive da 'from' a 5 linee dopo nel file hij

:x WRITE and QUIT: aggiorna, se necessario, ed esce

:wq WRITE and QUIT: aggiorna ed esce

:w !UNIX-CMD WRITE: scrive l'output e lo passa attraverso una pipe ad un  
comando UNIX

Esempio: :1,10 w !lpr  
manda le prime 10 linee alla stampante

:q[!] QUIT: esce senza salvare le modifiche (! 'bang' forza  
l'uscita)

UNDO

u UNDO: annulla l'ultima modifica

U UNDO: annulla tutte le modifiche fatte alla  
linea corrente fino a quando il cursore non  
è stato spostato dalla linea

"[1-9]PASTE utilizza un comando PASTE per incollare uno dei  
buffer numerati; i buffer numerati immagazzinano  
le ultime 9 cancellazioni effettuate, con il  
metodo "first in last out".  
Solitamente si utilizzano per riportare sul file  
porzioni testo precedentemente cancellate

Esempi: "1P "2p

## CUT

#dd DELETE: cancella "#" linee, di default una linea  
(quella corrente)  
d<movesrch> DELETE: cancella fino a dove viene specificato  
da "move" o "search"

### Esempi:

dfe cancella fino a quando trova 'e'  
dta cancella fino a quando trova 'a'  
d20| cancella fino alla 20esima colonna

d#move DELETE: cancella fino a dove viene specificato da "#move"

### Esempio:

d3k cancella 3 linee al di sopra della linea corrente e questa  
inclusa  
dw cancella la parola corrente rispetto alla posizione del cursore

d/<patrn> DELETE: cancella fino a quando viene trovato  
"pattern" (in avanti)  
d?<patrn> DELETE: cancella fino a quando viene trovato  
"pattern" (in dietro)  
d'<chr> DELETE: cancella fino al mark 'chr' (vedere la  
miscellanea per imparare come creare dei mark)  
D DELETE: cancella dalla posizione corrente fino  
alla fine della riga  
#x X-OUT: cancella un carattere sotto al cursore  
(oppure "#" caratteri)  
#X X-OUT: cancella un carattere prima del cursore  
(oppure "#" caratteri)

## YANK

y<movesrch> esegue uno YANK fino a dove specificato da  
"move" o "search"

### Esempi:

yw yank di una parola  
yfg yank fino a quando trova 'g'  
y#move esegue uno YANK fino a dove specificato da "#move"

### Esempio:

y3j esegue lo yank di 3 linee verso il basso oltre alla linea  
 corrente di partenza  
 y/pattern/ esegue uno YANK fino a trovare 'pattern'  
 (in avanti)  
 y?pattern? esegue uno YANK fino a trovare 'pattern'  
 (in dietro) y'<chr> esegue uno YANK fino al mark 'chr' (vedere la  
 miscellanea per imparare come creare dei mark)  
 Y, yy esegue uno YANK di una linea (quella corrente)  
 #Y, #yy esegue uno YANK di "#" linee sottostanti "[char]YANK copia lo YANK  
 in 'char'

## PASTE

P PASTE: incolla il contenuto del buffer prima del  
 cursore  
 p PASTE: incolla il contenuto del buffer dopo il  
 cursore  
 "<char>PASTE PASTE: incolla il buffer richiamato (char=a-z),  
 utilizzando uno dei comandi sopraelencati  
 "#PASTE PASTE: incolla il buffer numerato (#=1-9)  
 utilizzando uno dei comandi sopraelencati

## RICERCHE

f char FIND: cerca la successiva occorrenza di 'char'  
 sulla linea  
 t char 'TIL: si sposta fino alla successiva occorrenza  
 di 'char' sulla linea  
 F char FIND: cerca la precedente occorrenza di 'char'  
 sulla linea  
 T char 'TIL: si sposta fino alla precedente occorrenza  
 di 'char' sulla linea  
 ; ripete l'ultimo f, t, F o T eseguito , capovolge l'ordine  
 dell'ultimo f, t, F o T eseguito  
 % mostra il confronto con (), [] o {}  
 n NEXT: ripete l'ultimo comando / o ?  
 N NEXT: capovolge l'ordine dell'ultimo comando  
 / o ?  
 /string trova 'string' cercando in avanti; in aggiunta,  
 /string/ /string;/str2/

sposterà il cursore sulla prima 'str2' dopo  
'string'  
?string trova 'string' cercando a ritroso;

l'opzione ";" ?string? ha lo stesso funzionamento come sopra /, ?  
utilizza la stringa usata per la ricerca  
precedente

NOTA : pattern utilizzati per le ricerche con / e ?

^ inizio della linea

\$ fine della linea

. qualsiasi char

\* qualsiasi num. di chars

\< inizio di parola

\> fine di parola [str] qualsiasi char in str

[^str] chars non in str [x-y] chars compresi tra x e y

## Capitolo 3

# Comandi per il controllo dei file

Si tratta di un approfondimento dei comandi dati da DIRECTIVE MODE (":").

Nota: se in fondo al video compare un prompt ":" e non si riesce a lavorare con vi, può significare che si è entrati nell'editor di linea "ed". Digitare 'vi' al prompt per rientrare in editor "vi" (è possibile invece confermare l'entrata in "ed" digitando ^\$).

```
:#,#<command>[!][filenm]
    formato generale; gli '#'
    rappresentano comandi di movimento,
    oppure numeri di linea per
    indicare le posizioni da/a.
    Uno dei '#' può essere:
    +# #sotto          -# #sopra
    . linea corrente  $ ultima linea
```

```
:s/<search>/<replace>/[#gc]
    SUBSTITUTE: sostituisce <search>
    con <replace> una volta in una
    linea, se è specificato 'g'
    allora sostituisce tutto
    all'interno della linea.
    'c' serve per la conferma, '#'
    serve per sostituire in "#" linee.
    Vedere <SEARCH> e <REPLACE> per
    maggiori informazioni su
    questi comandi
```

```
<SEARCH>          <SEARCH> può avere sub-espressioni,
\<exp1\>[junk]\<exp2\>
    definite con una coppia di "\" e
    "\" (fino a 9 espressioni)
```

```
<REPLACE>        <REPLACE> può avere descrittori
\# junk \# \# junk
```

delle espressioni, definite da un  
"\#", dove '#' è un numero 1 - 9

Esempio:

```
s/\(.*\)=(.*)/\2 = \1/  
scambia LHS con RHS di un  
segno 'uguale a'  
in una linea con il formato:  
[something] = [something]
```

:#1,#2g/<search>/<ex command>

questo comando cercherà ogni  
occorrenza di <search> compresa  
tra le linee #1 e #2 ed eseguirà  
il comando <ex command>.  
L' "ex command":

s//<replace>/[gc]

cercherà <search> (// significa  
l'ultima ricerca) e lo sostituirà con  
<replace>.  
Utilizzare altri comandi come  
elencato sotto, quali "p" o "d".

#1,#2 c <text><.>

CHANGE: cambia le linee da #1 a #2  
con <text>; per uscire da questa  
modalità, digitare un '.' (period)  
da solo all'inizio di una linea.

:#1,#2 co #3 COPY: copia linee nello stesso  
modo descritto sopra

:#1,#2 t #3 comando corrispondente a 'co'

:#1,#2 d DELETE: cancella le linee da  
#1 a #2

Esempio:

```
: 'a, 'bd
```

cancella le linee dal mark 'a' fino a 'b'

:#1,#2 y "<alpha>

esegue lo YANK di linee nel buffer  
con nome <alpha>; se viene utilizzato  
l'alfabeto maiuscolo, allora yank  
aggiungerà al buffer

:e[!] [filename]

esegue l'EDIT del file corrente("!"  
indica una forzatura, per editare  
'filename' se viene specificato);  
nel caso in cui non sia presente la  
forzatura, se si vuole editare un

nuovo file e quello corrente non è stato salvato, viene rilasciato un avvertimento ed il nuovo edit non viene eseguito (sarà quindi necessario salvare, oppure editare con la forzatura; in quest'ultimo caso vengono persi i cambiamenti fatti nel file corrente)

:e# esegue l'EDIT alternativamente di due file (è necessario editare un altro file, e poi usare l'"e#" per editare nuovamente il primo file)

:e # esegue l'EDIT del file alternato e riprende all'ultima posizione del cursore

:#1,#2 m #3 MOVE: sposta le linee tra #1 e #2 e le posiziona a #3  
Esempi:

:10,15m25 sposta 5 linee(comprese tra la 10 e la 15) alla linea 25

:1,.m \$ sposta alla fine del file le linee comprese tra la linea 1 e quella corrente

:#1,#2 p PRINT: stampa le linee indicate (senza fare alcun cambiamento al file)

:#1r[!] [filename]  
READ: legge il file corrente (applicando le opzioni sopra descritte); se sono specificati il "filename" e "#1", allora inserisce il file alla posizione "#1" (l'inizio del file è la linea 0, e la fine

#### Esempio

:r pippo carica il file 'pippo' nella posizione in cui si trova attualmente il cursore

:r !UNIX-CMD READ: legge l'output del comando UNIX 'UNIX-CMD' come input al file, caricandolo nel buffer

`:n[filenames] [!]`  
NEXT: edita il file successivo nella lista degli argomenti; se viene messa la forzatura '!', non aggiorna il file corrente. Nel caso in cui sia stato specificato un elenco di file da editare, vi edita i file uno alla volta

Esempio:

`:n a.c b.c c.c d.c` edita 4 file, cominciando con a.c e successivamente  
`:n e.c f.c` aggiunge i file e.c ed f.c alla lista, andando in editing di e.c

Nota: con vi è possibile editare molteplici file già con il comando di lancio dell'editor.

Esempio:

`vi *.c`

`:args` mostra i file che 'vi' stà attualmente editando; dopo il comando `":n"` sopra riportato nell'esempio, digitando `":args"` comparirà  
[a.c] b.c c.c d.c  
dove le parentesi [] includono il file corrente

`:rew [!]` REWIND: ricomincia ad editare i file a partire dal primo file della lista; vedere il comando `":n"`

`:st[op]` STOP: blocca il 'vi' ed esce alla shell; per rientrare in 'vi', digitare `"fg"` al prompt di UNIX

`:sh` lancia una shell (solitamente una bourne shell); si ritorna nell'editor con `^d`

`!:<command>` esegue un comando UNIX, indicato con 'command', all'esterno del 'vi'

Esempio:

`:1,10 w !lpr` invia le prime 10 linee alla stampante senza che sia necessario creare un altro

file (cioè utilizza il file  
senza causarne variazioni)

!!! ripete l'ultimo comando  
'!<command>'

#!<command> esegue il comando shell indicato  
con 'command' e sostituisce il  
numero di linee specificato  
con '#' con l'output del comando;  
se non viene specificato '#', il  
default e' 1. Se il comando UNIX  
si aspetta uno standard input,  
allora le linee specificate  
verranno usate come input

Esempio:

10!!sort sorta le 10 linee successive

!<lines><command> funziona come il precedente  
comando '#!<command>', tranne che  
per il fatto che 'lines' è un  
indirizzo di linea, o delimitatore,  
necessario per poter eseguire  
il comando.

Esempio

!Gsort sorta le rimanenti linee del file  
Il testo compreso tra la posizione  
corrente del cursore ed il  
delimitatore 'lines' è dato  
in input a 'command' (ad esempio,  
sort) e viene poi sostituito con  
l'output del processo 'command'  
lanciato. Delimitatori validi  
possono essere, ad esempio, ')' o  
'})' per dare in input a  
'command' una frase o un  
paragrafo.  
Si noti che solo quando viene  
digitato il delimitatore (che  
può essere anche un  
secondo punto esclamativo)  
il cursore si posiziona sulla  
linea delle direttive,  
visualizzando '!'

& esegue solamente l'ultima ricerca-  
sostituzione

% corrisponde a "1,\$", cioè:

`:%s/a/b/`  
sostituirà la prima occorrenza di  
'a' con 'b' attraverso tutto il file

Alcuni simboli speciali sono:

`&` sostituisce la parola chiave  
ricercata

Esempio:

`:s/tr/&ace` sostituisce 'tr' con 'trace'

`\U` trasforma in lettere maiuscole ciò  
che lo segue

Esempio:

`:s/case/\U&/` cambia 'case' in 'CASE'

`\u` trasforma in maiuscolo solamente  
il primo carattere

`\L` converte in lettere minuscole

`\l` converte in minuscolo solamente  
il primo carattere

#### ESEMPI DI RICERCHE GLOBALI

Con i comandi EX abbiamo già visto come sia possibile effettuare delle ricerche globali, che agiscono sull'intero file editato.

`:g/bad/d` cancella tutte le linee che  
contengono la stringa 'bad'

`:g/bad/p` stampa tutte le linee che  
contengono la stringa 'bad'

`:g/bad/co $` copia tutte le linee che  
contengono la stringa 'bad'  
alla fine del file (per  
invertire il comando, e  
quindi aggiungerle all'inizio  
del file, lanciare il comando  
con '1' invece di '\$')

`:g/hello/ y"A` esegue uno YANK di tutte le  
occorrenze di 'hello' nel  
buffer con nome 'a'.  
È importante notare  
la lettera maiuscola nel  
nome del buffer; se viene

utilizzata la lettera  
minuscola, dopo aver  
eseguito il comando nel  
buffer esisterà  
solamente l'ultima  
occorrenza della parola

#### Varie

.           come caso speciale, quando  
          l'ultimo comando si riferisce  
          ad un buffer di testo numerato,  
          il comando '.' incrementa il  
          numero del buffer prima di  
          ripetere il comando

#### Esempio:

"1pu.u.	ripristina il buffer 1, annulla la modifica, ripristina il buffer 2, annulla la modifica
m char	MARK: crea un segno, o etichettatura, in questa posizione, e lo chiama 'char'
' char	(quote character) ritorna alla linea con nome 'char'
` char	(quota character) ritorna al posto con nome 'char'
'' o ``	(quote quote) ritorna al posizionamento del cursore precedente all'ultimo movimento effettuato
"[a-z][1-9]DEL	esegue DEL, YANK o PASTE di testo
"[a-z][1-9]YANK	dei buffer dalla 'a' alla 'z' o
"[a-z][1-9]PASTE	da '1' a '9' (solo uno alla volta)

#### Esempi:

"ad}	"5dw,	"by3y
"2Y,	"dp	"5P
~ (tilde)	converte il carattere corrente in maiuscolo/minuscolo (all'opposto dello stato attuale del carattere)	

z<RETURN>        posiziona la linea corrente in  
                  alto sullo schermo  
z.                posiziona la linea corrente alla  
                  metà dello schermo  
z-                posiziona la linea corrente in  
                  basso sullo schermo  
  
>movement        shifta a destra fino a dove viene  
                  specificato dal comando "movement"  
<movement        shifta a sinistra fino a dove  
                  viene specificato dal comando  
                  "movement"

!<movesrch><UNIX> esegue un comando UNIX sulle linee  
                  fino a quando viene specificato  
                  da "movesrch" (questo è  
                  diverso da ':!', in cui l'output  
                  del comando sostituisce l'input;  
                  in "movesrch" l'input è  
                  composto da tutte le linee)

Esempio:

!Gsort

                  sorta le linee di un file  
                  dalla posizione corrente  
                  fino alla fine del file.

## Capitolo 4

# Opzioni

Le opzioni influenzano l'ambiente dell'editor VI con lo scopo di adattarlo alle esigenze dell'utente.

L'editor 'vi' ha 3 tipi di opzioni: a valori numerici, a valori stringa e a valori booleani (vero, falso).

- Si attribuisce un valore alle opzioni numeriche o a valori stringa con un comando del tipo:

```
:set opzione=valore
```

- Si attribuisce il valore "vero" alle opzioni booleane (cioè le si attiva) con un comando del tipo:

```
:set opzione
```

- Si attribuisce un valore "falso" alle opzioni booleane (cioè le si disattiva) con un comando del tipo:

```
:set noopzione
```

- Per avere una lista di tutte le opzioni:

```
:set all
```

- Per avere una lista di tutte le opzioni modificate:

```
:set
```

· Per avere il valore di una opzione di tipo numerico o stringa:

```
:set opzione
```

Oltre all'utilizzo del comando 'set', che permette quindi disettare le opzioni all'interno dell'editor stesso, è anche possibile farlo prima di entrarvi, definendo la variabile di ambiente EXINIT oppure utilizzando il file .exrc (posizionato nella home directory dell'utente, contiene tutte le opzioni da questo personalizzate). Sia la variabile EXINIT che il file .exrc possono quindi contenere i valori personalizzati delle opzioni vi. Con questi due altri metodi le personalizzazioni vengono mantenute quando si esce dall'editor, mentre con il comando ":set" le opzioni valgono solamente all'interno di quella sessione di editor.

Di seguito riportiamo alcune delle numerose opzioni che possono essere settate all'interno del 'vi' (i valori di default sono riportati tra ( ) ):

OPZIONE	VALORE	DESCRIZIONE
autoindent	ai (noai)	Indentazione automatica del testo
autowrite	aw (noaw)	Scrivere automaticamente il buffer di editing nel file (nel caso in cui risulti modificato dopo l'ultima scrittura) prima dei comandi come ':n' o prima dell'esecuzione di sub-comandi con '!'
ignorecase	ic (noic)	In fase di ricerca, tratta le lettere maiuscole come minuscole
list	list (nolist)	Visualizza i TAB con '^L' e newline con '\$'
number	nu (nonu)	Prefissa le linee col numero di linea
shiftwidth	sw=8	Shifta la distanza per il TAB stop utilizzato da "autoindent", per i comandi shift (> e <) e per i comandi di input di testo (^D e ^T)
showmatch	sm (nosm)	Evidenzia ) e } corrispondenti a ( e { digitato
showmode	smd (nosmd)	Mostra la modalità di inserimento corrente nella linea dei messaggi Ad esempio, con il comando 'a' viene visualizzato 'APPEND MODE'
tabstop	ts=8	Spazi corrispondenti ad un TAB (è consigliabile farlo coincidere con l'opzione shiftwidth)
wrapmargin	wm=0	'vi' inserisce automaticamente un newline quando trova naturale spezzare il testo all'interno di

spazi dichiarati come 'wm'  
(opzione utile per l'editing di  
testo); se wm=0, va a capo dopo  
80-N chars

La sintassi per settare un'opzione è la seguente:

```
:set <opt>
```

Esempio:

```
:set number  
inserisce il numero davanti a ciascuna linea del file.
```

Con il comando:

```
:set no<opt>
```

si toglie l'opzione.

## Capitolo 5

# Macro

Se viene settato correttamente, 'vi' può eseguire una serie di comandi con poche parole chiave. Questo può essere raggiunto in due modi:

1. Il corpo della macro può essere messo in un registro, ad esempio 'a', e poi è possibile eseguire la macro creata.

Esempio:

digitare la seguente macro come testo su una linea nuova in 'vi', utilizzando un INSERT MODE:

```
ameow^V^[
```

(il tasto ESC deve essere digitato dopo un ^V, che evita per una sola volta che questo tasto faccia uscire da INSERT MODE).

Ora, con il cursore posizionato su 'a', eseguire uno yank del testo in un buffer, digitando:

```
"aY$ oppure "ayy$
```

che corrisponde al comando di memorizzazione nel buffer dalla linea corrente fino alla fine.

Successivamente, da COMMAND MODE sarà sufficiente digitare il comando:

```
@a
```

(dove 'a', nell'esempio sopra riportato, è il nome del buffer utilizzato) per eseguire la macro contenuta nel buffer 'a'; in questo modo viene eseguito un append della parola 'meow' al testo ove si è posizionati.

2. È possibile definire una macro utilizzando il comando 'map', per richiedere a 'vi' di considerare un certo tasto (carattere o tasto funzione) come una sequenza di tasti.

L'operazione di map risulta essere tipicamente come segue:

```
:map carattere sequenza
```

Si noti che se la "sequenza" comprende un newline, questo deve essere prefissato da ^V.

Esempio:

```
:map[!] lhs rhs<CR>
```

dove:

- 'lhs': deve essere una parola chiave, può essere un carattere oppure una function key.

Esempi:

```
caratteri      : a, b, c, A, B, ^, >  
control keys  : ^A, ^B (digitare ^V per l'escape prima  
                di digitare ^A, ecc.)  
function keys : sono disponibili #[0-9] (corrispondono  
                alle key F0, F1, ecc. su tutte le  
                tastiere)
```

- 'rhs': deve essere una sequenza di caratteri non più lunga di 100 caratteri.

Esempio:

```
I/*^[${a*/^[^M  
  inserisce un '/*' all'inizio della linea ed un '*/'  
  alla fine della stessa (trasforma un qualsiasi codice  
  C in un commento).
```

- '!': questa opzione rende attiva la key in INSERT MODE.

Esempi:

```
:map q :wq^VRETURN  
  se da COMMAND MODE si richiede 'q', avremo  
  l'aggiornamento ed il rilascio di 'vi'; poichè  
  'map' è una direttiva, va chiusa da un RETURN  
  in aggiunta a quello eventuale specificato in "sequenza"
```

```
:map p asono io ^VESC  
  se da COMMAND MODE si richiede 'p', avremo l'inserimento  
  delle parole "sono io"
```

```
:map V k/\/*/  
  questo comando definirà "V" in maniera che ricerchi  
  '/*' e '*/' e li cancelli (tolga i commenti da una linea
```

di sorgente scritto in linguaggio C)

```
:map #2 :!budvisor^M
  questo comando, che verrà scritto nel file '.exrc',
  permette di richiamare il programma "budvisor" premendo
  la key F2
```

Le macro vengono dismesse con la direttiva "unmap".

Esempio:

```
:unmap lhs
```

Per avere un elenco di tutte le macro definite per il COMMAND MODE,  
digitare:

```
:map
```

Per avere un elenco di tutte le macro definite per l'INSERT MODE,  
digitare:

```
:map!
```

## Capitolo 6

# Suggerimenti

Le abbreviazioni valgono per l'INPUT MODE e sono una generalizzazione delle macro; permettono di digitare una breve parola ottenendone l'espansione (alla pressione della barra spaziatrice) in una sequenza qualsiasi di caratteri. Per tale scopo esiste la direttiva 'ab'.

Esempio:

```
:ab US UNIVERSITA' DEGLI STUDI
```

La direttiva 'unab' rimuove le abbreviazioni.

Esempio:

```
:unab US
```

Come già accennato, le opzioni, le macro e le abbreviazioni possono essere attivate non solo dall'interno dell'editor, ma anche da shell; questo avviene assegnando alla variabile EXINIT la sequenza di comandi necessari per la attivazione, separati unicamente dal simbolo di pipe ('|'). In tal caso non si dimentichi di esportare la variabile EXINIT.

Esempio:

```
EXINIT='set ai nu | ab US UNIVERSITA' DEGLI STUDI'  
export EXINIT
```

Volendo automatizzare questa assegnazione, in modo che l'utente non sia costretto a digitarlo ogni volta che accede al sistema, va inserito nel file di inizializzazione per l'utente (.profile per sh, .login/.cshrc per csh e tcsh).

Qui di seguito troverete elencati alcuni tra i molti suggerimenti possibili per risolvere problemi frequenti:

· Per cancellare una parola: dw

- Per cancellare più parole: `d#w`  
(dove '#' è il numero delle parole da cancellare)
- Per cancellare dalla posizione corrente fino a 'parola':  
`d/parola`
- Per cancellare 5 linee e metterle in un buffer 'a':  
`"a5dd`
- Per cancellare la frase corrente fino al punto di posizionamento del cursore:  
`d(`
- Per cancellare la parte restante di un paragrafo dove si è posizionati:  
`d}`
- Per cambiare una parola: `cw`
- Per invertire l'ordine dei caratteri: `xp`
- Per trasformare una parola inglese in plurale (aggiunta della 's' finale):  
`eas`
- Per cancellare fino all'inizio di una linea: `d0`
- Per memorizzare 3 parole nel buffer 'x': `"xy3w`
- Per memorizzare nel buffer 's' 5 linee, a partire dalla corrente, e stamparle sopra di essa:  
`"s5YP`
- Per ricercare e visualizzare tutte le linee che contengono la stringa 'pattern':  
`:g/pattern/p`
- Per ricercare e visualizzare attraverso il contenuto dei buffer numerati:  
`"1pu.u.u ('u' e '.' alternativamente)`
- Per sostituire completamente 'str' con 'replace':  
`:g/str/s//replace/[g][c]`
- Per sostituire una parola ('old') con un'altra ('new') in tutto il file:  
`:%s/old/new/g`  
mentre per avere un prompt di conferma ad ogni sostituzione:  
`:%s/old/new/gc`
- Utilizzo di 'word erase' o 'line erase' in fase di inserimento per sostituire quello che si è appena scritto.  
Esempi:  
`hello tim^W` riposiziona il cursore sulla 't' di "tim"  
`hello tim^U` riposiziona il cursore all'inizio della linea

Poichè queste key possono essere ridefinite, controllare il loro stato attraverso il comando Unix:

```
stty -a
controllando 'werase' e 'kill'
```

· Per copiare un gruppo di linee da un file all'altro:

1. salvare il file corrente utilizzando ':w'
2. editare il file contenente i dati richiesti, utilizzando ':e <filename>'
3. eseguire ora un yank delle linee desiderate in un buffer; ad esempio, '"a3Y' eseguirà uno yank di 3 linee nel buffer 'a'
4. editare il file originale utilizzando ':e! <filename>'
5. posizionare il cursore nel posto desiderato; utilizzando il comando '"ap' o '"aP' il testo contenuto nel buffer 'a' verrà copiato

· Per ritornare alla posizione del file precedentemente editato, utilizzare:

```
CTRL^ (tenere premuto il tasto 'control' mentre si preme il tasto '^')
```

questo è un modo abbreviato per sostituire il comando ':e #'

· Per commentare linee in linguaggio C (/\*...\*/) è possibile definire delle map sui caratteri come 'v' e 'V' nella seguente maniera:

```
map v I/*^[${a*}/^[^M
map V /\/*/^MNxx/\*\/^Mxx''
```

il carattere 'v' commenterà una singola linea, mettendo '/' e '/' rispettivamente all'inizio e alla fine della linea, mentre 'V' rimuoverà questi da una linea o da un paragrafo commentati (è necessario essere all'inizio oppure all'interno dei commenti)

· Per stampare un certo range di linee direttamente sulla stampante, senza doverle salvare attraverso un file:

```
:#1,#2 w !lpr
```

invierà le linee da #1 a #2 alla stampante di default

· Le linee cancellate possono essere recuperate passando attraverso i buffer numerati; tali buffer, numerati da 1 a 9,

contengono le ultime 9 cancellazioni effettuate. Per passare attraverso i buffer:

1. digitare: "1p visualizzerà il buffer #1
2. digitare: u eliminerà la copia dei dati
3. digitare: . eseguirà l'ultimo comando '"1p'  
ma solo in questo unico caso  
aggiunge 1 ed esegue il comando  
'.', che corrisponde a '"2p';  
all'esecuzione successiva, '.'  
corrisponde a '"3p'
4. quando si trova il buffer corretto è sufficiente tenere  
il ripristino effettuato.

· Per marcare una parte di testo, si utilizza 'm' seguito da una lettera.

Esempio:

m a all'inizio del testo da marcare

m b alla fine del testo da marcare

È possibile copiare successivamente tale parte nel punto del testo ove è posizionato il cursore:

: 'a, 'b co .

oppure è possibile spostarla:

: 'a, 'b mo .

o ancora cancellarla:

: 'a, 'b del .