usr_10.txt   Per Vim version 6.3. Ultima modifica: 2004 Mar 12

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Gian Piero Carzino

Fare grandi modifiche

Nel capitolo 4 sono stati mostrati molti modi per fare piccole modifiche.
Questo capitolo affronta cambiamenti ripetuti o che possono modificare grandi
quantità di testo. Il Visual mode permette di fare varie cose con blocchi di
testo. Usate un programma esterno per fare cose veramente complesse.

==
10.1 Registrare e rieseguire comandi

Il comando "." ripete la precedente modifica. Ma come fare se volete ottenere
qualcosa di più di una singola modifica? Ecco dove entra in gioco la
registrazione dei comandi. Si compone di tre passi:

1. Il comando "q{registro}" fa partire la registrazione della sequenza di
   tasti premuti nel registro di nome {registro}. Il nome del registro deve
   essere un carattere fra a e z.
2. Scrivete i vostri comandi.
3. Per terminare la registrazione, premete q (da solo).

Ora potete eseguire la macro scrivendo il comando "@{registro}".

Date un'occhiata a come si possono usare questi comandi in pratica. Avete una
lista di nomi di file come questa:

```
stdio.h ~
fcntl.h ~
unistd.h ~
stdlib.h ~
```

E quello che volete ottenere è:

```
#include "stdio.h" ~
#include "fcntl.h" ~
#include "unistd.h" ~
#include "stdlib.h" ~
```

Cominciate portandovi sul primo carattere della prima riga. Poi eseguite i
seguenti comandi:

```
qa                      Inizia a registrare una macro nel registro a.
^                       Vai all'inizio della riga.
i#include "<Esc>        Inserisci la stringa #include " all'inizio
                        della riga.
$                       Vai alla fine della riga.
a"<Esc>                 Aggiungi il carattere doppie virgolette (")
                        alla fine della riga.
j                       Vai alla riga successiva.
q                       Termina la registrazione della macro.
```

Ora che avete fatto il lavoro una volta, potete ripetere la modifica
scrivendo il comando "@a" tre volte.
   Il comando "@a" può essere preceduto da un numero, che farà eseguire la
macro altrettante volte. In questo caso scrivereste: **>**

```
3@a
```

SPOSTARSI ED ESEGUIRE

Le righe che volete cambiare potrebbero essere in vari posti. Semplicemente spostate il cursore in ognuno dei posti e usate il comando "@a". Se lo avete già fatto una volta, lo potete ripetere con "@@". É ancora più facile da scrivere. Se ora eseguite il registro b con "@b", il prossimo "@@" userà il registro b.
   Se confrontate l'esecuzione di macro con l'uso di ".", troverete molte differenze. Per prima cosa, "." può solo ripetere una modifica. Come invece abbiamo visto nell'esempio qui sopra, "@a" può fare molte modifiche, e anche spostamenti. Secondariamente, "." può ricordare solo l'ultima modifica. Eseguire un registro vi permette di fare tutte le modifiche che volete, e poter sempre usare "@a" per ripetere quei comandi che sono stati registrati. Infine potete usare 26 diversi registri. Quindi potete ricordare 26 diverse sequenze di comandi da eseguire.

USARE I REGISTRI

I registri usati per registrare le macro sono gli stessi che si usano per i comandi yank e delete. Questo permette di mescolare la registrazione con altri comandi di manipolazione dei registri.
   Supponiamo che abbiate registrato alcuni comandi nel registro n. Quando lo eseguite con "@n" vi accorgete che avete sbagliato qualcosa. Potreste riprovare la registrazione, ma c'è il rischio di fare un'altro errore. Usate invece questo trucco:

G	Vai alla fine del file.
o<Esc>	Crea una riga vuota.
"np	Riversare il testo contenuto nel registro n. Potete vedere i comandi che avete registrato come semplice testo nel file.
{modifiche}	Modificate i comandi sbagliati. Non è altro che normale editing di un testo.
0	Vai all'inizio della riga.
"ny$	Copiare i comandi corretti nel registro n.
dd	Cancellare la riga dal file.

Ora potete eseguire i comandi corretti con "@n". (Se i comandi che avevate registrato comprendevano degli <a capo>, adattate di conseguenza le due ultime righe nell'esempio per comprendere tutte le linee della macro.)

AGGIUNGERE AD UN REGISTRO

Finora abbiamo usato le lettere minuscole per i nomi di registro. Per aggiungere in fondo ad un registro, usate le lettere maiuscole.
   Supponiamo che abbiate registrato nel registro c un comando per modificare una parola. Funziona, ma volete aggiungere la ricerca della successiva parola da modificare. Lo si può fare con: **>**

```
qC/word<Enter>q
```

Cominciate con "qC", che registra nel registro c aggiungendo in fondo. Quindi scrivere in un registro con nome maiuscolo significa aggiungere in fondo al registro con la stessa lettera, ma minuscola.

Questo vale sia per la registrazione che per i comandi yank e delete. Supponiamo per esempio che vogliate raccogliere una sequenza di righe in un registro. Copiate la prima riga con: **>**

```
"aY
```

Ora spostatevi sulla seconda riga, e scrivete: **>**

```
"AY
```

Ripetete questo comando per tutte le righe da aggiungere. Alla fine il registro a conterrà tutte le righe, nell'ordine in cui le avete copiate.

==========

10.2 Sostituzione *find-replace*

Il comando ":substitute" vi permette di fare sostituzioni di stringhe su un insieme di righe. La forma generale di questo comando è la seguente: **>**

```
:[range]substitute/from/to/[flags]
```

Questo comando modifica la stringa "from" nella stringa "to" nelle righe specificate da [range]. Per esempio, potete cambiare "il Professore" in "l'Insegnante" in tutte le righe con il seguente comando: >

```
:%susbstitute/il Professore/l'Insegnante/
```

<

Note: Il comando ":substitute" non viene quasi mai scritto per intero. La maggior parte delle volte si usa la forma abbreviata ":s". Da qui in poi useremo l'abbreviazione.

Il "%" prima del comando indica che il comando va applicato a tutte le righe. Senza una indicazione di ambito di applicazione, ":s" opera solo sulla riga corrente. Il prossimo capitolo affronta più approfonditamente il tema degli ambiti di applicazione dei comandi.

Per default, il comando ":substitute" modifica solo la prima stringa trovata di ogni riga. Per esempio, il precedente comando modifica la riga:

```
Oggi il Professore Smith ha criticato il Professore Johnson. ~
```

in:

```
Oggi l'Insegnante Smith ha criticato il Professore Johnson. ~
```

Per cambiare ogni stringa corrispondente nella riga, dovete aggiungere il flag (globale) g. Il comando: >

```
:%s/il Professore/l'Insegnante/g
```

produce (a partire dalla riga originale):

```
Oggi l'Insegnante Smith ha criticato l'Insegnante Johnson. ~
```

Altri flag sono p (stampa), che fa stampare al comando ":substitute" ogni linea che modifica. Il flag c (conferma) fa chiedere al comando ":substitute" conferma prima di eseguire una modifica. Scrivete il seguente comando: >

```
:%s/il Professore/l'Insegnante/c
```

Vim trova la prima volta che ricorre "il Professore" e visualizza il testo che sta per modificare. Vi chiede: >

```
replace with l'Insegnante (y/n/a/q/l/^E/^Y)?
```

A questo punto potete rispondere con una delle seguenti lettere:

y	Yes; fai questa modifica.
n	No; salta questa ricorrenza.
a	All; fai questa modifica e tutte le rimanenti senza ulteriore conferma.
q	Quit; fine delle modifiche.
l	Last; fai ancora questa modifica e poi termina.
CTRL-E	Fai scorrere il testo una riga in su.
CTRL-Y	Fai scorrere il testo una riga in giù.

La parte "from" del comando di sostituzione è in effetti uno schema. Lo stesso tipo di schema che si usa per il comando di ricerca. Per esempio il comando che segue sostituisce "the" solo se compare all'inizio di una riga: >

```
:s/^the/these/
```

Se nella parte "from" o nella parte "to" c'è una barra (/), dovete mettere un backslash (\) prima di essa. Un modo più semplice è usare un altro carattere come separatore invece della barra. Ad esempio un segno più: >

```
:s+uno/due+uno o due+
```

==========

10.3 Ambito di applicazione dei comandi

Il comando ":substitute", e molti altri comandi che iniziano per ":", possono essere applicati ad un insieme di righe. Questo viene chiamato intervallo.
La forma più semplice di intervallo è {numero},{numero}. Per esempio: >

```
:1,5s/questo/quello/g
```

Esegue il comando di sostituzione nelle righe dalla 1 alla 5. Anche la riga

5 viene inclusa. L'intervallo è sempre posto prima del comando.

Un singolo numero può essere usato per indicare una specifica riga: **>**

```
:54s/President/Fool/
```

Alcuni comandi operano sull'intero file quando non si specifica un intervallo. Per farli operare sulla riga corrente si può usare l'indirizzo ".". Il comando write funziona in questo modo. Senza un intervallo, salva l'intero file. Per fargli salvare in un file solo la riga corrente: **>**

```
:.write altrofile
```

La prima riga ha sempre il numero uno. E l'ultima? Per indicare questa si usa il carattere "$". Per esempio, per fare una sostituzione dalla riga del cursore alla fine del file: **>**

```
:.,$s/yes/no/
```

L'intervallo "%" che abbiamo usato prima altro non è che una abbreviazione di "1,$", cioè dalla prima all'ultima riga.

USARE UNO SCHEMA DI RICERCA IN UN INTERVALLO

Supponiamo che stiate modificando un capitolo di un libro, e vogliate sostituire tutti i giallo con grigio. Ma solo nel capitolo corrente, non nel prossimo. Voi sapete che la parola "Capitolo" ad inizio riga si trova solo all'inizio dei capitoli. Questo comando ottiene il risultato desiderato: **>**

```
:?^Capitolo?,/^Capitolo/s=giallo=grigio=g
```

Come potete vedere si usa due volte uno schema di ricerca. Il primo "?^Capitolo?" trova la riga precedente che corrisponde a questo schema. Quindi l'intervallo ?schema? serve a cercare all'indietro. Analogamente, "/^Capitolo/" serve a cercare in avanti l'inizio del prossimo capitolo.
   Per rendere più chiaro il comando, abbiamo usato il carattere "=" invece della barra nel comando di sostituzione. Si poteva anche usare la barra o un altro carattere.

AGGIUNTE E SOTTRAZIONI

C'è un piccolo errore nel comando che abbiamo appena descritto: se nel titolo del capitolo successivo ci fosse la parola "giallo" sarebbe stata sotituita anch'essa. Forse era ciò che volevate, ma se invece non volevate modificarla? In tal caso potevate indicare uno spostamento.
   Per cercare uno schema e poi indicare la riga precedente: **>**

```
/^Capitolo/-1
```

Potete usare qualunque numero al posto di 1. Per indicare la seconda riga dopo quella individuata da uno schema: **>**

```
/^Capitolo/+2
```

Gli spostamenti si possono usare anche con altri tipi di voci negli intervalli. Guardate questo: **>**

```
:.+3,$-5
```

Questo indica un intervallo che inizia tre righe sotto il cursore e finisce 5 righe prima dell'ultima riga del file.

USARE I SEGNALIBRI

Invece di trovare il numero di riga di certe posizioni, ricordarseli e poi scriverli in un intervallo, potete usare i segnalibri.
   Piazzate i segnalibri come indicato nel capitolo 3. Per esempio, usate "mt" per segnare l'inizio di una zona e "mb" per segnare la fine. Potete poi usare il seguente intervallo per indicare le righe fra i due segnalibri (comprese le righe segnate): **>**

```
:'t,'b
```

VISUAL MODE E INTERVALLI

Potete selezionare del testo in Visual mode. Se poi premete ":" per scrivere un comando di questo tipo, vedrete: **>**

```
:'<,'>
```

Ora potete scrivere il comando e sarà applicato all'insieme di righe che era stato selezionato visualmente.

> Note: Quando si seleziona col Visual mode parte di una riga, o usando CTRL-V un blocco di testo, i comandi ":" si applicano ugualmente a righe intere. Questo comportamento potrebbe cambiare in future versioni di Vim.

I '< e '> sono in effetti dei segnalibri, posti all'inizio e alla fine della selezione Visuale. I segnalibri rimangono fissi finché non viene fatta un'altra selezione Visuale. Quindi potete usare il comando "'<" per saltare all'inizio dell'area selezionata Visualmente. E potete mescolare i segnalibri con altri elementi: **>**

```
:'>,$
```

Questo individua le righe dalla fine dell'area Visuale alla fine del file.

UN CERTO NUMERO DI RIGHE

Quando sapete quante righe volete modificare, potete scrivere il numero e poi ":". Per esempio, scrivendo "5:", otterrete: **>**

```
:.,.+4
```

Ora potete scrivere il comando che desiderate. Userà l'ntervallo da "." (riga corrente) a ".+4" (quattro righe sotto). Appunto cinque righe in totale.

==========

10.4 Il comando global

Il comando ":global" è una delle caratteristiche più potenti di Vim. Vi permette di trovare una corrispondenza per uno schema ed eseguire lì un certo comando. la forma generale è: **>**

```
:[range]global/{schema}/{comando}
```

È simile al comando ":substitute". ma invece di sostituire il testo individuato con l'altro testo, esegue il comando {comando}.

> Note: Il comando eseguito da ":global" deve essere uno che inizia con ":".
> I comandi del Normal mode non possono essere eseguiti direttamente.
> Il comando |:normal| può eseguirli per voi.

Supponiamo che vogliate cambiare "foobar" in "barfoo", ma solo all'interno dei commenti in stile C++. Questi commenti iniziano con "//". Usate il comando: **>**

```
:g+//+s/foobar/barfoo/g
```

Questo comando inizia per ":g". È la forma abbreviata di ":global", così come ":s" è la forma abbreviata di ":substitute". Segue poi lo schema, fra due segni più. Siccome lo schema che stiamo cercando contiene delle barre, usiamo i segni più come separatori. Infine il comando di sostituzione che cambia "foobar" in "barfoo".
   L'intervallo di default per il comando global è l'intero file. Questo spiega perché non è stato specificato un intervallo in questo esempio. Notare la differenza col comando ":substitute", che viene eseguito solo su una riga, se non è specificato un intervallo.
   Il comando non è perfetto, poiché agisce anche sulle righe in cui "//" appare a metà, e la sostituzione viene applicata anche prima del "//".

Proprio come il comando ":substitute", si può usare ogni schema. Quando avrete imparato schemi più complicati, li potrete usare anche qui.

==========

10.5 Visual block mode

Con CTRL-V potete iniziare la selezione di un'area rettangolare di testo. Ci

sono alcuni comandi che fanno operazioni speciali sui blocchi di testo.

Speciale è ad esempio l'uso del comando "$" mentre si è in Visual block mode. Quando l'ultimo comando di spostamento usato è "$", tutte le righe selezionate in Visual block mode si estendono fino a fine riga, anche se la riga col cursore è più corta. Questo comportamento rimane finché non si usa un comando di spostamento orizzontale. Quindi l'uso "j" lo mantiene, "h" lo interrompe.

INSERIMENTO DI TESTO

Il comando "I{stringa}<Esc>" inserisce il testo {stringa} in ogni riga, alla sinistra del blocco Visuale. Iniziate premendo CTRL-V per entrare nel Visual block mode. Ora spostate il cursore per definire il blocco che desiderate. Poi scrivete I per entrare nel modo inserimento, e di seguito il testo da inserire. Mentre scrivete, il testo compare solo sulla prima riga.
Dopo aver premuto <Esc> per terminare l'inserimento, il testo sarà automaticamente inserito in tutte le altre righe della selezione Visuale. Esempio:

```
include one ~
include two ~
include three ~
include four ~
```

Spostate il cursore alla "o" di "one" e premete CTRL-V. Spostatelo in giù con "3j" fino a "four". Ora avete un blocco selezionato che si estende per quattro righe. Ora scrivete: **>**

```
Imain.<Esc>
```

Il risultato è:

```
include main.one ~
include main.two ~
include main.three ~
include main.four ~
```

Se il blocco include righe corte che non arrivano dentro il blocco, il testo non viene inserito in queste righe. Per esempio, fate una selezione Visual block che comprende le parole "long" della prima e dell'ultima riga di questo testo (quindi la seconda riga non ha testo selezionato):

```
This is a long line ~
short ~
Any other long line ~

          ^^^^ blocco selezionato
```

Ora usate il comando "Ivery <Esc>". Il risultato è:

```
This is a very long line ~
short ~
Any other very long line ~
```

Nella riga corta non è stato inserito alcun testo.

Se la stringa che inserite contiene un <a capo>, il comando "I" si comporta come un comando di inserimento Normale, e agisce solo sulla prima riga del blocco.

Il comando "A" si comporta alla stessa maniera, salvo che aggiunge il testo dopo il lato destro del blocco.
C'è un caso speciale per "A": selezionate un blocco Visuale e poi usate "$" per farlo estendere fino alla fine di ogni riga. Ora usare "A" farà aggiungere il testo alla fine di ogni riga.
Usando lo stesso esempio di prima, e scrivendo "$A XXX<Esc>", otterrete questo risultato:

```
This is a long line XXX ~
short XXX ~
Any other long line XXX ~
```

Per far questo è proprio necessario usare il comando "$". Vim si ricorda che è stato usato. Creare la stessa selezione spostando il cursore alla fine della riga più lunga con altri comandi di spostamento non ottiene lo stesso risultato.

MODIFICARE IL TESTO

Il comando "c" in Visual block mode cancella il blocco e vi pone in Insert mode per permettervi di scrivere una stringa. La stringa sarà inserita in ogni riga del blocco.
   Iniziando con la stessa selezione delle parole "long" come sopra, e poi scrivendo "c_LONG_<Esc>", otterrete questo:

```
This is a _LONG_ line ~
short ~
Any other _LONG_ line ~
```

Come nel caso di "I", la riga più corta non viene modificata. Inoltre non ci può essere un <a capo> nel testo inserito.

Il comando "C" cancella il testo dal lato sinistro del blocco fino alla fine della riga. Vi immette quindi in modo Inserimento, così che possiate scrivere una stringa, che verrà aggiunta alla fine di ogni riga.
   Partendo sempre dallo stesso testo, e scrivendo "Cnew text<Esc>" otterrete:

```
This is a new text ~
short ~
Any other new text ~
```

Notate che, anche se era selezionata solo la parola "long", è stato cancellato anche il testo che la segue. Così l'unica cosa che conta davvero è la posizione del lato sinistro del blocco Visuale.
   Anche in questo caso le righe corte che non raggiungono il blocco sono escluse.

Altri comandi che cambiano i caratteri nel blocco sono:

```
~       inverti maiuscole/minuscole     (a -> A e A -> a)
U       rendi maiuscole                 (a -> A e A -> A)
u       rendi minuscole                 (a -> a e A -> a)
```

RIEMPIRE CON UN CARATTERE

Per riempire l'intero blocco con un carattere, usate il comando "r". Ripartendo sempre dallo stesso esempio di prima, e scrivendo "rx":

```
This is a xxxx line ~
short ~
Any other xxxx line ~
```

Note: Se volete includere nel blocco anche i caratteri che sono oltre la fine della riga, guardate la caratteristica 'virtualedit' nel capitolo 25.

SPOSTAMENTO

Il comando ">" sposta il testo selezionato a destra di una determinata quantità inserendo degli spazi. Il punto di partenza per questo spostamento è il lato sinistro del blocco Visuale.
   Sempre con lo stesso esempio, ">" dà questo risultato:

```
This is a         long line ~
short ~
Any other         long line ~
```

La quantità di spostamento è specificata con l'opzione 'shiftwidth'. Per cambiarla a 4 spazi: **>**

```
:set shiftwidth=4
```

Il comando "<" rimuove la stessa quantità di spazio bianco dal lato sinistro del blocco. Questo comando è limitato dalla quantità di testo che c'è effettivamente; quindi se c'è meno spazio bianco della quantità richiesta, rimuove quello che può.

UNIRE LE RIGHE

Il comando "J" unisce tutte le righe selezionate in una sola riga. Rimuove quindi le interruzioni di riga. In effetti l'interruzione di riga, lo spazio

bianco ad inizio riga e quello a fine riga sono sostituiti da un singolo spazio. Dopo un punto di fine riga, vengono usati due spazi (ciò può essere modificato con l'opzione 'joinspaces').
   Usiamo l'esempio con cui siamo ormai familiari. Il risultato dell'uso del comando "J" è:

```
        This is a long line short Any other long line ~
```

Il comando "J" non richiede che sia stata fatta una selezione di tipo blocco. Funziona allo stesso identico modo con le selezioni di tipo "v" e "V".

Se non volete che gli spazi vengano alterati, usate il comando "gJ".

==
10.6 Leggere e scrivere parte di un file

Quando state scrivendo un messaggio e-mail, potreste desiderare di includere un altro file. Questo si può fare con il comando ":read {nomefile}". Il testo dell'altro file viene inserito dopo la riga del cursore.
   Se partiamo da questo testo:

```
        Ciao John, ~
        Ecco il diff che corregge l'errore: ~
        Saluti, Pierre. ~
```

Spostate il cursore nella seconda riga e scrivete: **>**

```
        :read patch
```

Il file di nome "patch" verrà inserito, ottenendo:

```
        Ciao John, ~
        Ecco il diff che corregge l'errore: ~
        2c2 ~
        <       for (i = 0; i <= length; ++i) ~
        --- ~
        >       for (i = 0; i < length; ++i) ~
        Saluti, Pierre. ~
```

Il comando ":read" accetta un intervallo. Il file verrà inserito dopo l'ultima riga di questo intervallo. Così ":$r patch" aggiunge il file "patch" alla fine del file.
   E se voleste inserire il file prima della prima riga? Questo si può fare con il numero di riga zero. Questa riga non esiste in realtà, e sareste avvisati con un messaggio d'errore usandola nella maggior parte dei comandi. Ma è permesso scrivere: **>**

```
        :0read patch
```

Il file "patch" sarà posto prima della prima riga del file.

SCRIVERE UN GRUPPO DI RIGHE

Per scrivere un gruppo di righe in un file si può usare il comando ":write". Senza specificare un intervallo salva l'intero file. Se si specifica un intervallo, solo le righe indicate vengono scritte: **>**

```
        :.,$write tempo
```

Questo scrive le righe dal cursore fino alla fine del file nel file "tempo". Se quest'ultimo file esiste già si otterrà un messaggio di errore. Vim impedisce di sovrascrivere accidentalmente un file già esistente. Se proprio volete sovrascriverlo, aggiungete un !: **>**

```
        :.,$write! tempo
```

ATTENZIONE: il ! deve seguire immediatamente il comando ":write", senza spazi. Altrimenti diventa un comando filtro, che è spiegato più avanti in questo capitolo.

AGGIUNGERE AD UN FILE

Nella prima sezione di questo capitolo viene spiegato come raccogliere un certo numero di righe in un registro. Lo stesso si può fare per raccogliere righe in un file. Scrivete la prima riga con questo comando: **>**

```
:.write raccolta
```

Ora spostate il cursore sulla seconda riga che volete raccogliere, e scrivete: **>**

```
:.write >>raccolta
```

Il ">>" dice a Vim che il file "raccolta" non deve essere scritto da capo, ma che la riga deve essere aggiunta alla fine. Potete ripetere quest'ultimo comando tante volte quante volete.

==========

10.7 Formattare un testo

Quando state scrivendo del semplice testo, è bello che la lunghezza di ogni riga sia automaticamente limitata per stare nella finestra. Perché questo succeda mentre state inserendo il testo, assegnate un valore all'opzione 'textwidth': **>**

```
:set textwidth=72
```

Se ricordate, nel file di esempio di vimrc si usava questo comando in ogni file di testo. Così, se state usando quel file vimrc, state già usando quel comando. Per verificare il valore corrente di 'textwidth': **>**

```
:set textwidth
```

Ora le righe verranno spezzate per contenere al massimo 72 caratteri. Ma se voi inserite del testo a metà di una riga, o se cancellate qualche parola, le righe diverranno troppo lunghe o troppo corte. Vim non le rimette in forma automaticamente.

Per dire a Vim di formattare il paragrafo corrente: **>**

```
gqap
```

Questo inizia con il comando "gq", che è un operatore. Segue "ap", l'oggetto di testo su cui deve operare, che significa "un paragrafo". I paragrafi sono separati gli uni dagli altri da una riga vuota.

```
Note:
Una riga bianca, che contiene spazio bianco, NOT separa i paragrafi.
Questo è molto difficile da notare!
```

Invece di "ap" potreste usare ogni spostamento o oggetto di testo. Se i vostri paragrafi sono separati correttamente, potete usare il seguente comando per formattare l'intero file: **>**

```
gggqG
```

"gg" vi porta alla prima riga, "gq" è l'operatore di formattazione e "G" è lo spostamento che salta all'ultima riga.

Se invece i vostri paragrafi non sono definiti in modo chiaro, potete formattare le righe che selezionate manualmente. Muovete il cursore alla prima riga che volete formattare. Iniziate con il comando "gqj". Questo formatta la riga corrente e la successiva. Se la prima riga era corta, saranno aggiunte parole prese dalla seconda. Se era troppo lunga, alcune parole saranno spostate nella successiva. Il cursore si sposta alla seconda riga. Ora potete usare "." per ripetere il comando. Continuate a farlo finché raggiungete la fine del testo che volevate formattare.

==========

10.8 Cambiare Maiuscole/minuscole

Supponiamo che abbiate del testo con i titoli di sezione in lettere minuscole. Volete cambiare la parola "sezione" in tutte maiuscole. Fatelo con l'operatore "gU". Iniziate con il cursore nella prima colonna: **>**

```
                              gUw
<       sezione titolo      ---->      SEZIONE titolo
```

L'operatore "gu" fa esattamente l'opposto: **>**

```
                              guw
<       SEZIONE titolo      ---->      sezione titolo
```

Potete anche usare "g~" per cambiare tutte le maiuscole in minuscole e viceversa. Tutti questi sono operatori, quindi lavorano con ogni comando di

spostamento, con oggetti di testo e in Visual mode.
   Per far lavorare un operatore sulle righe lo si raddoppia. L'operatore di cancellazione è "d", quindi per cancellare una riga si scrive "dd". Analogamente "gugu" mette in minuscolo un'intera riga. Questo può essere abbreviato "guu". "gUgU" si abbrevia "gUU" e "g~g~" diventa "g~~".
Esempio: **>**

```
                                  g~~
<       Some GIRLS have Fun      ---->   sOME girls HAVE fUN ~
```

==========

10.9 Usare un programma esterno

Vim ha un insieme di comandi molto potente, può fare qualunque cosa. Ma ci sono cose che un comando esterno può fare meglio o più velocemente.
   Il comando "!{spostamento}{programma}" prende un blocco di testo e lo filtra attraveso un programma esterno. In altre parole, fa partire il programma di sistema di nome {programma}, fornendogli il blocco di testo rappresentato da {spostamento} come input. L'output di questo comando poi sostituisce il blocco selezionato.
   Siccome questo è un po' troppo sintetico se non si conoscono i filtri UNIX, guardate un esempio. Il comando sort ordina un file. Se eseguite il seguente comando, il file non ordinato input.txt sarà messo in ordine alfabetico e scritto nel file output.txt. (Questo funziona sia in UNIX che in Microsoft Windows). **>**

```
        sort <input.txt >output.txt
```

Ora facciamo la stessa cosa in Vim. Volete ordinare le righe da 1 a 5 di un file. Iniziate ponendo il cursore sulla riga 1. Poi eseguite il seguente comando: **>**

```
        !5G
```

Il "!" dice a Vim che state eseguendo una operazione di filtro. L'editor Vim si aspetta ora uno spostamento, che gli indichi quale parte del file deve filtrare. Il comando "5G" dice a Vim di andare alla riga 5, così sa che la parte da filtrare va dalla riga 1 (la riga corrente) alla 5.
   In previsione del filtraggio, il cursore si sposta in fondo allo schermo e si presenta un !. Ora potete scrivere il nome del programma filtro, in questo caso "sort". Quindi il vostro comando completo è: **>**

```
        !5Gsort<Enter>
```

Il risultato è che il programma sort viene eseguito sulle prime cinque righe. L'uscita del programma sostituisce queste righe.

```
        line 55                         line 11
        line 33                         line 22
        line 11          -->            line 33
        line 22                         line 44
        line 44                         line 55
        last line                       last line
```

Il comando "!!" filtra la riga corrente attraverso un filtro. In UNIX il comando "date" stampa la data e ora corrente. "!!date<Enter>" sostituisce la riga corrente con l'output di "date". Questo è utile per aggiungere data e ora ad un file.

QUANDO NON FUNZIONA

Far partire una shell, inviarle testo e catturare il risultato richiede che Vim conosca esattamente come funziona la shell. Quando avete problemi per filtrare, controllate i valori di queste opzioni:

```
        'shell'          specifica il programma che Vim usa per eseguire i
                         programmi esterni.
        'shellcmdflag'   argomento per passare un comando alla shell
        'shellquote'     virgolette da usare intorno al comando
        'shellxquote'    virgolette da usare intorno al comando e alla
                         ridirezione
        'shelltype'      tipo di shell (solo per l'Amiga)
        'shellslash'     usare le barre normali nel comando (solo per
                         MS-Windows e simili)
```

In ambiente Unix questo è raramente un problema, perché ci sono due tipi di shell: quelle tipo "sh" e quelle tipo "csh". Vim controlla l'opzione 'shell' e imposta le corrispondenti opzioni automaticamente, a seconda che trovi "csh"

da qualche parte in 'shell'.
   In MS-Windows, invece, ci sono molte diverse shell e potrebbe essere necessario calibrare le opzioni per far funzionare il filtraggio. Per ulteriori informazioni, vedere l'aiuto delle opzioni.

LEGGERE L'OUTPUT DEI COMANDI

Per leggere il contenuto della directory corrente e metterlo nel file, usare:

in Unix: **>**
```
:read !ls
```
in MS-Windows: **>**
```
:read !dir
```

Il risultato del comando "ls" o "dir" è catturato e inserito nel testo, sotto il cursore. Questo è analogo a leggere un file, eccetto che si usa il "!" per dire a Vim che segue un comando.
   Il comando può avere degli argomenti. E si può inserire un intervallo per dire a Vim dove mettere le righe: **>**

```
:0read !date -u
```

Questo inserisce l'ora e data corrente in formato UTC all'inizio del file (Se avete un comando "date" che accetta l'argomento "-u"). Notate la differenza rispetto a usare "!!date": quello sostituiva una riga, mentre ":read !date" la inserisce.

INVIARE DEL TESTO AD UN COMANDO

Il comando Unix "wc" conta le parole. Per contare le parole nel corrente file: **>**

```
:write !wc
```

Questo è lo stesso comando che abbiamo già incontrato, ma invece del nome di un file viene posto il carattere "!" e il nome di un comando esterno. Il testo che verrebbe scritto viene invece inviato al comando specificato come suo standard input. Il risultato potrebbe assomigliare a:

```
       4      47     249 ~
```

Il comando "wc" è molto sintetico. Questo risultato vuol dire che avete 4 righe, 47 parole e 249 caratteri.

Attenti a questo errore: **>**

```
:write! wc
```

Questo scriverà il file "wc" nella directory corrente, forzando la scrittura nel caso ce ne sia già uno. Gli spazi sono importanti in questo caso!

RIDISEGNARE LO SCHERMO

Se il comando esterno ha prodotto un messaggio di errore, lo schermo può essere stato scombinato. Vim è molto efficiente e riscrive solo quelle parti dello schermo che sa che devono essere riscritte. Ma non può sapere cosa ha combinato un programma esterno. Per dire a Vim di ridisegnare tutto lo schermo: **>**

```
CTRL-L
```

==========

Capitolo seguente: |usr_11.txt| Recupero dopo un blocco

Segnalare refusi a Bartolomeo Ravera - E-mail: barrav at libero.it