tips.txt Per **Vim version 8.1.** Ultima modifica: 2009 Nov 07

VIM Manuale di Riferimento di Bram Moolenaar
Traduzione di questo testo: Antonio Colombo

Suggerimenti e idee per usare Vim *tips*

Sono qui descritti solo pochi esempi che riteniamo utile per molti utenti.
Molti di più possono essere trovati sulla wiki. L'URL relativo può essere
trovato nel sito http://www.vim.org

Non dimenticatevi di consultare il manuale utente, che contiene anche una
grande quantità di suggerimenti utili: |**usr_toc.txt**|.

==========

Modificare programmi C *C-editing*

Ci sono alcune funzionalità in Vim che possono esservi utili per editare file
di programmi C. Ecco qualche indicazione con i tag a cui saltare:

|**usr_29.txt**| Capitolo "Spostarsi nei programmi" nel manuale utente.
|**usr_30.txt**| Capitolo "Modifica programmi" nel manuale utente.
|**C-indenting**| Impostare automaticamente l'indentazione di una riga mentre di immette del testo.
|**=**| Re-indentare qualche riga.
|**format-comments**| Formattazione dei commenti inseriti nel programma.

|**:checkpath**| Lista tutti i file inclusi in maniera ricorsiva.
|**[i**| Ricerca l'identificativo sotto il cursore nel file corrente e in quelli inclusi.
|**[_CTRL-I**| Salta alla corrispondenza per "[i"
|**[I**| Lista tutte le righe nel file corrente e in quelli inclusi contenenti l'identificatore sotto il cursore.
|**[d**| Ricerca la define sotto il cursore nel file corrente e in quelli inclusi.

|**CTRL-]**| Salta alla tag sotto il cursore (per esempio, alla definizione di una funzione).
|**CTRL-T**| Salta indietro a dove si era prima del comando CTRL-].
|**:tselect**| Sceglie una tag in una lista di tag che costituisce il risultato di una ricerca.

|**gD**| Va alla dichiarazione di variabile globale sotto il cursore.

|**gf**| Vai in edit sul nome-file sotto il cursore.

|**%**| Vai alla (), {}, [], /* */, #if, #else, #endif corrispondente.
|**[/**| Vai al precedente inizio di commento.
|**]/**| Vai alla precedente fine di commento.
|**[#**| Torna indietro fino a un #if, #ifdef, o #else non chiuso.

|]#| Vai avanti fino a un #else o #endif non chiuso.
|[(| Torna indietro fino a una '(' non chiusa.
|])| Vai avanti fino a una ')' non chiusa.
|[{| Torna indietro fino a una '{' non chiusa.
|]}| Vai avanti fino a una '}' non chiusa.

|v_ab| Seleziona un blocco "a block" da "[(" a "])", parentesi comprese.
|v_ib| Seleziona un blocco interno "inner block" da "[(" a "])" (parentesi escluse).
|v_aB| Seleziona un blocco "a block" da "[{" a "]}", parentesi comprese.
|v_iB| Seleziona un blocco interno "inner block" da "[{" a "]}" (parentesi escluse).

==
Trovare dove sono usati identificatori *ident-search*

È probabile che già sappiate che le tag |tags| possono essere usate per saltare alla posizione in cui è stata definita una funzione o una variabile. Ma può capitare che vogliate saltare a tutti i posti in cui una funzione o variabile è utilizzata. Ciò è possibile in due modi:
1. Usando il comando |:grep|. Questo metodo dovrebbe funzionare in quasi tutti i sistemi Unix, ma può essere lento (legge tutti i file) e la ricerca è limitata a una sola directory.

2. Usando i programmi di utilità ID. Questo metodo è veloce e può lavorare con più di una directory. Un database è usato per memorizzare le posizioni. Saranno necessari alcuni programmi in più perché la cosa funzioni. E il database dovrà essere tenuto aggiornato.

Usare i programmi di utilità GNU id-tools:

Cosa vi serve:
- Il pacchetto GNU id-tools disponibile (mkid serve per creare ID e lid serve per usare le macro).
- Un file contenete il database degli identificativi, di nome "ID", nella directory corrente. Lo potete creare con il comando della shell "mkid file1 file2 ..".

Mettete quanto segue nel vostro .vimrc: >
```
        map _u :call ID_search()<Bar>execute "/\\<" . g:word . "\\>"<CR>
        map _n :n<Bar>execute "/\\<" . g:word . "\\>"<CR>

        function! ID_search()
          let g:word = expand("<cword>")
          let x = system("lid --key=none ". g:word)
          let x = substitute(x, "\n", " ", "g")
          execute "next " . x
        endfun
```

Per usarlo, mettete il cursore sopra una parola, battete "_u" e vim caricherà il file che contiene la parola. Potete trovare la successiva occorrenza della parola nello stesso file con "n". Passate al prossimo file con "_n".

Quanto sopra è stato testato usando id-utils-3.2 (è questo il nome del file che contiene il software id-tools sul sito mirror di gnu-ftp-mirror più vicino a voi).

[l'idea per quanto sopra è di Andreas Kutschera]

==
Cambiare schermo in un xterm *xterm-screens* *xterm-save-screen*

(scritto da Juergen Weigert in comp.editors, rispondendo a una domanda)

```
:> Un'altra domanda è che dopo essere usciti da vim, lo schermo è lasciato
:> com'era, cioè i contenuti del file che stavo vedendo (editando) sono
:> ancora sullo schermo. L'output del mio precedente "ls" si è perduto,
:> cioè, non è più disponibile nel buffer scorrevole. So che c'è un modo per
:> ripristinare lo schermo dopo essere usciti da vim o da un altro editor
:> simile a vi, ma non so come fare. Ogni aiuto è apprezzato. Grazie.
:
```

:Immagino che qualcun altro possa rispondere a questa domanda. Suppongo che
:vim e vi si comportino entrambi allo stesso modo a fronte di una data
:configurazione di xterm.

Il comportamento non è necessariamente lo stesso, potrebbe esserci un problema di utilizzo di termcap invece che terminfo. Dovresti tener presente che ci sono due database che descrivono gli attributi di un particolare tipo di terminale: termcap e terminfo. Questo può implicare delle differenze quando le descrizioni sono differenti E quando uno dei programmi in questione usa terminfo, mentre l'altro usa termcap (vedere anche |**+terminfo**|).

Nel tuo caso particolare, stai cercando le sequenze di caratteri di controllo ^[[?47h e ^[[?47l. Questo fanno passare dallo schermo alternato a quello principale del buffer. Come soluzione temporanea veloce, una sequenza di comandi come >

```
        echo -n "^[[?47h"; vim ... ; echo -n "^[[?47l"
```

può produrre l'effetto da te desiderato. (La mia notazione ^[indica il tasto <ESC>, vedrai più sotto che i database usano invece \E).

A inizio programma, vim invia il valore della variabile termcap ti (terminfo smcup) al terminale. In uscita, invia te (terminfo: rmcup). Per cui queste due variabili sono il posto giusto in cui dovrebbero essere inserite le sequenze di controllo menzionate sopra.

Confrontate le righe termcap che si riferiscono a xterm (le trovate in /etc/termcap) con la descrizione terminfo dello stesso (che si può avere battendo "infocmp -C xterm"). Entrambe dovrebbero contenere righe simili a: >

```
        :te=\E[2J\E[?47l\E8:ti=\E7\E[?47h:
```

PS: Se rilevate delle differenze, qualcuno (il vostro amministratore di sistema?) farebbe bene a controllare gli interi database termcap e terminfo per assicurarsi che siano consistenti fra loro.

NOTA 1: Se ricompilate Vim con FEAT_XTERM_SAVE definita in feature.h, l'xterm interno di Vim comprenderà le righe di descrizione menzionate, "te" e "ti".

NOTA 2: Se volete disabilitare il passaggio di schermo e non volete cambiare le vostre termcap, potete aggiungere queste righe al vostro .vimrc: >

```
        :set t_ti= t_te=
```

==========

Scorrere in Modo Insert *scroll-insert*

Se siete in Modo Insert e volete vedere qualcosa che si trova vicino a quel che vedete sull schermo, potete usare CTRL-X CTRL-E e CTRL-X CTRL-Y per far scorrere lo schermo.
|**i_CTRL-X_CTRL-E**|

Per facilitare la cosa, potreste usare queste mappature: >

```
        :inoremap <C-E> <C-X><C-E>
        :inoremap <C-Y> <C-X><C-Y>
```

(Battete questo letteralmente, assicuratevi che il flag '<' non sia specificato in 'cpoptions').
In questo modo perdete la possibilità di copiare del testo sopra/sotto la riga del cursore |**i_CTRL-E**|.

Considerate anche se impostare 'scrolloff' a un valore maggiore, in modo da poter sempre vedere qualche riga di contesto attorno al cursore.
Se 'scrolloff' è più alto della metà della finestra, il cursore starà sempre nel mezzo, e il testo viene fatto scorrere quando si sposta il cursore in su o in giù.

==========

Scorrere dolcemente *scroll-smooth*

Se volete uno scorrimento un po' più dolce, potete usare queste mappature: >

```
        :map <C-U> <C-Y><C-Y><C-Y><C-Y><C-Y><C-Y><C-Y><C-Y><C-Y><C-Y><C-Y><C-Y>
<C-Y><C-Y><C-Y>
        :map <C-D> <C-E><C-E><C-E><C-E><C-E><C-E><C-E><C-E><C-E><C-E><C-E><C-E>
<C-E><C-E><C-E>
```

(Battete questo letteralmente, assicuratevi che il flag '<' non sia

specificato in 'cpoptions').

==========
Correggere refusi frequenti *type-mistakes*

Se ci sono alcune parole che spesso battete in maniera scorretta, create
delle abbreviazioni per correggerle. Per esempio: >
```
:ab gil gli
:ab pre per
```

==========
Contare, parole, righe, etc. *count-items*

Per contare le occorrenze di un'espressione regolare nel buffer corrente, usate
il comando substitute, aggiungendo il flag 'n' per evitare di fare realmente
la sostituzione. Il numero di sostituzioni riportato è il numero delle
occorrenze. Esempi: >

```
:%s/./&/gn              caratteri
:%s/\i\+/&/gn           parole
:%s/^//n                righe
:%s/il/&/gn             "il" da qualsiasi parte (anche dentro parole)
:%s/\<il\>/&/gn         "il" inteso come parola
```

Potreste annullare 'hlsearch' o impostare ":nohlsearch".
Aggiungere il flag 'e' se non voleta una segnalazione di errore quando non
viene trovata nessuna corrispondenza.

Una alternativa è usare |**v_g_CTRL-G**| in Modo Visual.

Per trovare corrispondenze in più di un file, potete usare |**:vimgrep**|.

count-bytes
Se volete contare byte, potete fare così:

Selezionate i caratteri in Modo Visual (o anche nel Modo
Block Visual
Usate "y" per copiare i caratteri
Usate la funzione strlen(): >
```
:echo strlen(@")
```
Ogni carattere di fine linea è contato come un byte.

==========
Ripristinare la posizione del cursore *restore-position*

Qualche volta potreste voler definire una mappatura che fa qualche tipo di
modifica da qualche parte nel file, e poi ripristina la posizione del cursore,
senza far scorrere il testo. Per esempio, per cambiare l'indicatore della
data in un file: >
```
:map <F2> msHmtgg/Last [cC]hange:\s*/e+1<CR>"_D"=strftime("%Y %b %d")<CR>p'tzt`s
```

Dettaglio salvataggio posizione:
```
ms      metti posizione cursore nel marcatore 's'
H       vai alla prima linea della finestra
mt      metti questa posizione nel marcatore 't'
```

Dettaglio ripristino posizione:
```
't      vai alla linea che era prima in cima alla finestra
zt      scorri per muovere questa linea in cima alla finestra
`s      salta alla posizione originale del cursore
```

Per qualcosa di più sofisticato vedere |**winsaveview()**| e |**winrestview()**|.

==========
Rinominare file *rename-files*

Supponiamo di avere una directory che contiene i seguenti file (directory
scelta a caso :-):

```
buffer.c
charset.c
digraph.c
...
```

e di voler rinominare *.c in *.bla. Farei così: >

```
$ vim
:r !ls *.c
:%s/\(.*\).c/mv & \1.bla
:w !sh
:q!
```

==========

Cambiare un nome in più di un file *change-name*

Esempio di uso di uno script file per cambiare nome a più file:

Create un file "subs.vim" che contiene dei comandi substitute e un comando :update :

```
:%s/Rossi/Bianchi/g
:%s/Mario/Pietro/g
:update
```

<

Invocate Vim per i file che volete modificare, ed eseguite lo script per ogni file specificato nella lista argumenti: >

```
vim *.let
argdo source subs.vim
```

Vedere |**:argdo**|.

==========

Velocizzare comandi esterni *speed-up*

In alcune situazioni, l'esecuzione di un comando esterno può essere molto lenta. Trovate qui alcuni suggerimenti per poter procedere più rapidamente.

Se il vostro file .cshrc (o il file corrispondente, a seconda della shell che usate) è molto lungo, dovreste dividerlo in una parte dedicata all'utente che interagisce, e in una parte per uso non-interattivo (spesso chiamata "shell secondaria"). Quando eseguite un comando da Vim come per esempio ":!ls", non avete alcun bisogno della parte "interattiva" (per esempio, quella che imposta il prompt). Mettete la parte che non vi serve dopo le linee seguenti: >

```
if ($?prompt == 0) then
        exit 0
endif
```

un altro modo è si specificare alla shell il flag "-f", per esempio: >

```
:set shell=csh\ -f
```

(il "\" serve per inserire lo spazio seguente nell'opzione).
Questo parametro farà sì che lo shell non utilizzi affatto il file .cshrc.
Ciò può a sua volta far sì che alcune cose siano non funzionanti.

==========

Mappature utili *useful-mappings*

Ecco alcune mappature che piacciono a qualcuno.

map-backtick >

```
:map ' `
```

Fa sì che l'apice semplice di comporti come l'apice retroverso. Mette il cursore sulla colonna di una marcatura, invece che andare al primo carattere non-bianco della linea.

emacs-keys

Per modifiche su riga-comando in stile Emacs: >

```
" vai a inizio linea
:cnoremap <C-A>         <Home>
" indietro di un carattere
:cnoremap <C-B>         <Left>
" cancella carattere sotto il cursore
:cnoremap <C-D>         <Del>
" vai a fine linea
```

```
        :cnoremap <C-E>         <End>
        " avanti di un carattere
        :cnoremap <C-F>         <Right>
        " richiama la linea-comando più recente
        :cnoremap <C-N>         <Down>
        " richiama la linea-comando precedente (più vecchia)
        :cnoremap <C-P>         <Up>
        " indietro di una parola
        :cnoremap <Esc><C-B>    <S-Left>
        " avanti di una parola
        :cnoremap <Esc><C-F>    <S-Right>
```

NOTA: Questo richiede che il flag '<' non sia specificato in 'cpoptions'.

format-bullet-list

Questa mappatura formatterà ogni lista a punti. Richiede che ci sia una linea vuota prima e dopo ogni elemento della lista. La serie di comandi è stata scritta in modo da poter aggiungere commenti a ogni singola parte della mappatura. >

```
        :let m =     ":map _f  :set ai<CR>"   " 'autoindent' va impostato
        :let m = m . "{O<Esc>"                " linea vuota sopra elemento
        :let m = m . "}{)^W"                  " al testo dopo il punto
        :let m = m . "i     <CR>     <Esc>"  " aggiungi spazio x indentatura
        :let m = m . "gq}"                    " formatta testo dopo il punto
        :let m = m . "{dd"                    " togli linea vuota
        :let m = m . "5lDJ"                   " metti testo dopo il punto
        :execute m                           |" definisci mappatura
```

(<> Notazione |**<>**|. Notare che i caratteri vanni immessi come. sono scritti. ^W è "^" "W", non CTRL-W. Potete copiare/incollare questo testo in Vim se il flag '<' non è incluso in 'cpoptions'.)

Notare che l'ultimo commento inizia con |", perché il comando ":execute" non accetta un commento normale (lo tratterebbe come un parametro).

Dovete anche impostare 'textwidth' a un valore diverso da zero, per esempio; >

```
        :set tw=70
```

Segue una mappatura che fa quasi lo stesso, ma prende l'indentatura per la lista dalla prima linea (Notare: questa mappatura è una singola linea molto lunga contenente parecchi spazi): >

```
        :map _f :set ai<CR>}{a
    <Esc>WWmmkD`mi<CR><Esc>kkddpJgq}'mJO<Esc>j
<
```

collapse

Le mappature seguenti riducono una sequenza di linee vuote (;b) o contenenti solo spazi (;n) a una sola linea >

```
    :map ;b   GoZ<Esc>:g/^$/.,/./-j<CR>Gdd
    :map ;n   GoZ<Esc>:g/^[ <Tab>]*$/.,/[^ <Tab>]/-j<CR>Gdd
```

==========

Comprimere i file di help *gzip-helpfile*

Per chi fra voi abbia veramente poco spazio disco, è possibile comprimere i file di help, continuando a poterli visualizzare all'interno di Vim. Questo rallenta un po' il tempo di accesso ai file, e richiede che sia disponibile il programma gzip.

(1) Comprimete tutti i file di help: "gzip doc/*.txt".

(2) Andate in edit su "doc/tags" e cambiate ".txt" in ".txt.gz": >

```
        :%s=\(\t.*\.txt\)\t=\1.gz\t=
```

(3) Aggiungete questa linea al vostro file vimrc: >

```
        set helpfile={nome-directory}/help.txt.gz
```

Dove {nome-directory} è la directory in cui risiedono i file di help. Il plugin per |**gzip**| si occuperà della decompressione dei file.
Dovete assicurarvi che $VIMRUNTIME sia impostato a dove si trovano gli altri file di Vim, quando non siano nella stessa posizione della directory "doc" compressa. Vedere |**$VIMRUNTIME**|.

==========
Eseguire comandi della shell in una finestra *shell-window*

Vedere |**terminal**|.

Un'altra soluzione è di dividere in due il vostro terminale o la vostra finestra con il programma "splitvt". Potete probabilmente procurarvelo da qualche server ftp. La persona più esperta riguardo a questo programma è Sam Lantinga <slouken@cs.ucdavis.edu>.

Un'ulteriore alternativa è il comando "window", disponibile nei sistemi BSD Unix, che supporta molte finestre, una sopra l'altra. O il programma "screen", disponibile nel sito www.uni-erlangen.de, che supporta una pila di finestre.

==========
Modificare in esadecimale *hex-editing* *using-xxd*

Vedere la sezione |**23.4**| del manuale utente.

Se i vostri file binari usano una particolare estensione (tipo exe, bin, etc.) potete trovar utile automatizzare il procedimento con il seguente pezzettino di autocomando nel vostro file <.vimrc>. Cambiate "*.bin", usato nell'esempio, con una qualsiasi lista di estensioni, separate da virgola, che indichi i file che volete modificare in maniera binaria: >

```
" vim -b : edit in forma binaria usando il formato using xxd!
augroup Binary
  au!
  au BufReadPre  *.bin let &bin=1
  au BufReadPost *.bin if &bin | %!xxd
  au BufReadPost *.bin set ft=xxd | endif
  au BufWritePre *.bin if &bin | %!xxd -r
  au BufWritePre *.bin endif
  au BufWritePost *.bin if &bin | %!xxd
  au BufWritePost *.bin set nomod | endif
augroup END
```

==========
Uso della notazione <> negli autocomandi *autocmd-<>*

La notazione <> non è riconosciuta come argomento in un :autocmd. Per evitare di dover usare dei caratteri speciali, potreste usare una mappatura che si cancella dopo essere stata usata, per poter adoperare la notazione <>, e quindi richiamare la mappatura all'interno di un autocomando. Esempio:

map-self-destroy >

```
 " Per aggiungere automaticamente il nome del file alla lista menù.
 " Usa una mappatura auto-distruttiva!
 " 1. usare una linea nel buffer per convertire i '.' nel nome-file a \.
 " 2. memorizzare il risultato nel registro '"'
 " 3. aggiungere quel nome alla lista menù dei buffer
 " AVVISO: ci sono alcuni effetti collaterali, come la sovrascrittura del
 " contenuto corrente del registro, e la rimozione di mappature eventualmente
 " presenti per il comando "i".
 "
 autocmd BufNewFile,BufReadPre * nmap i :nunmap i<CR>O<C-R>%<Esc>:.g/\./s/\./\\./g<C
R>0"9y$u:menu Buffers.<C-R>9 :buffer <C-R>%<C-V><CR><CR>
 autocmd BufNewFile,BufReadPre * normal i
```

Un altro metodo, forse migliore, è di usare il comando ":execute". Nella stringa potete usare la notazione <> premettendo un backslash ('\').
Non dimenticatevi di raddoppiare il numero dei backslash già esistenti, e di mettere un backslash prima di '"'.
>

```
  autocmd BufNewFile,BufReadPre * exe "normal O\<C-R>%\<Esc>:.g/\\./s/\\./\\\\./g\<C
R>0\"9y$u:menu Buffers.\<C-R>9 :buffer \<C-R>%\<C-V>\<CR>\<CR>"
```

Per un vero menù dei buffer, sarebbe meglio usare delle funzioni (vedere |**:function**|), ma in quel caso la notazione <> non è usata, il che impedisce di usarla qui come esempio.

==========

Evidenziare la parentesi corrispondente *match-parens*

Questo esempio mostra l'uso di alcuni funzionalità avanzate:
- usare l'evento di autocomando |**CursorMoved**|
- usare |**searchpairpos()**| per trovare una parentesi corrispondente
- usare |**synID()**| per capire se il cursore è in una stringa o in un commento
- usare |**:match**| per evidenziare qualcosa
- usare un'espressione regolare |**pattern**| per trovare una posizione specifica
  in un file

Questo andrebbe messo in un file script di Vim, perché usa variabili locali allo script. Salta le corrispondenze nelle stringhe o nei commenti, a meno che il cursore sia inizialmente in una stringa o commento. È necessario che sia attiva l'evidenziazione sintattica.

Una versione lievemente più complessa è usata nel plugin |**matchparen**|.
>
```
        let s:evidenzia_paren = 0
        function s:Evidenzia_Paren_Corrisp()
          if s:evidenzia_paren
            match none
            let s:evidenzia_paren = 0
          endif

          let c_num_riga = line('.')
          let c_colonna = col('.')

          let c = getline(c_num_riga)[c_colonna - 1]
          let plist = split(&coppia_corrisp, ':\|,')
          let i = index(plist, c)
          if i < 0
            return
          endif
          if i % 2 == 0
            let s_flags = 'nW'
            let c2 = plist[i + 1]
          else
            let s_flags = 'nbW'
            let c2 = c
            let c = plist[i - 1]
          endif
          if c == '['
            let c = '\['
            let c2 = '\]'
          endif
          let s_skip ='synIDattr(synID(line("."), col("."), 0), "name") ' .
                \ '=~?  "string\\|comment"'
          execute 'if' s_skip '| let s_skip = 0 | endif'

          let [m_num_riga, m_colonna] = searchpairpos(c, '', c2, s_flags, s_skip)

          if m_num_riga > 0 && m_num_riga >= line('w0') && m_num_riga <= line('w$')
            exe 'match Search /\(\%' . c_num_riga . 'l\%' . c_colonna .
                  \ 'c\)\|\(\%' . m_num_riga . 'l\%' . m_colonna . 'c\)/'
            let s:evidenzia_paren = 1
          endif
        endfunction

        autocmd CursorMoved,CursorMovedI * call s:Evidenzia_Paren_Corrisp()
        autocmd InsertEnter * match none
<
```

vim:tw=78:ts=8:noet:ft=help:norl:

Per segnalazioni scrivere a vimdoc.it at gmail dot com
oppure ad Antonio Colombo azc100 at gmail dot com