

usr_05.txt Per Vim version 7.3. Ultima modifica: 2012 Nov 20

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Ivan Morgillo

Configurazioni personali

Vim può essere personalizzato affinché funzioni come volete. Questo capitolo vi mostra come far partire Vim con le opzioni impostate in modi differenti. Aggiungere plugin per estendere le capacità di Vim. O definire le vostre macro.

05.1	Il file vimrc
05.2	Spiegazione del file vimrc di esempio
05.3	Semplici mappature
05.4	Aggiungere un plugin
05.5	Aggiungere un file di Aiuto
05.6	La finestra delle opzioni
05.7	Le opzioni più usate

Capitolo seguente: |usr_06.txt| Usare l'evidenziazione della sintassi
Capitolo precedente: |usr_04.txt| Fare piccole modifiche
Indice: |usr_toc.txt|

=====

05.1 Il file vimrc

vimrc-intro

Probabilmente siete stanchi di scrivere i comandi che usate molto spesso. Per iniziare già con tutte le vostre opzioni preferite impostate e le vostre mappature, basta che le scriviate in un file chiamato vimrc. Vim esegue i comandi contenuti in questo file in fase di avvio.

Se già avete un file vimrc (ad es., quando il vostro amministratore di sistema ve ne ha preparato uno), potete modificarlo in questo modo: >

```
:edit $MYVIMRC
```

Se non avete già un file vimrc, guardate |vimrc| per sapere dove potete creare un file vimrc. Anche il comando ":version" mostra il nome del file utente vimrc che Vim cerca all'avvio.

Per i sistemi Unix viene usato sempre questo file: >

```
~/.vimrc
```

Per i sistemi MS-DOS e MS-Windows di solito si usa uno di questi: >

```
$HOME/_vimrc
$VIM/_vimrc
```

Il file vimrc può contenere tutti i comandi che voi scrivete dopo i due punti. I più semplici sono per le impostazioni delle opzioni. Per esempio, se volete che Vim parta sempre con l'opzione 'incsearch' attivata, aggiungete questa riga al vostro file vimrc: >

```
set incsearch
```

Affinché questa nuova riga abbia effetto dovete riavviare Vim. In seguito imparerete a fare questa operazione senza riavviare Vim.

Questo capitolo spiega solo la maggior parte degli elementi di base. Per ulteriori informazioni su come scrivere un file script per Vim: |usr_41.txt|.

=====

05.2 Spiegazione del file vimrc di esempio

vimrc_example.vim

Nel primo capitolo è stato spiegato come il file vimrc di esempio (incluso nella distribuzione di Vim) possa essere usato per lanciare Vim in modalità non-compatibile (vedere |not-compatible|). Il file può essere trovato qui:

```
$VIMRUNTIME/vimrc_example.vim ~
```

In questa sezione spiegheremo i vari comandi usati in questo file. Questo vi darà una mano su come impostare le vostre preferenze. Non sarà, però, spiegato tutto. Usate il comando ":help" per sapere di più.

>

```
set nocompatible
```

Come detto nel primo capitolo, questi manuali spiegano come Vim funziona in modo migliore, cioè non completamente compatibile con Vi. Disabilitando '**compatible**', l'opzione '**nocompatible**' si occupa di ciò.

>

```
set backspace=indent,eol,start
```

Questo specifica dove in modo Insert **<BS>** può cancellare il carattere che segue il cursore. I tre oggetti, separati dalle virgole, dicono a Vim di cancellare lo spazio bianco all'inizio della riga, l'interruzione di riga e il carattere prima del punto in cui è iniziato il modo Insert.

>

```
set autoindent
```

Questo consente a Vim di usare il rientro della riga precedente per la riga appena creata. Per esempio quando si preme **<Enter>** in modo Insert, e quando si usa il comando "o" per creare una nuova riga.

>

```
if has("vms")
  set nobackup
else
  set backup
endif
```

Questo dice a Vim di creare una copia di backup di un file quando lo si sovrascrive. Ma non entro il sistema VMS, poiché esso già conserva le vecchie versioni dei file. Il file di backup avrà lo stesso nome del file originale con aggiunto "~". Vedere |07.4|

>

```
set history=50
```

Conserva 50 comandi e 50 stringhe di ricerca nel file di history. Impiegate un altro numero se desiderate che vengano memorizzate più o meno linee.

>

```
set ruler
```

Mostra sempre la posizione corrente del cursore nell'angolo in basso a destra della finestra di Vim.

>

```
set showcmd
```

Mostra un comando non completo nell'angolo in basso a destra della finestra di Vim, a sinistra del regolo. Ad es., se scriveste "2f", Vim attenderebbe che scriviate il carattere da trovare e verrebbe mostrato "2f". Se poi scriveste "w", verrebbe eseguito il comando e rimosso il "2f".

```
+-----+
| testo entro la finestra di Vim |
| ~                               |
| ~                               |
| -- VISUAL --                    | 2f      43,8    17% |
+-----+
| ^^^^^^^^^^ ^^^^^^^^^^ ^^^^^^^^^^ |
| 'showmode' 'showcmd' 'ruler'    |
```

>

```
set incsearch
```

Propone una possibile corrispondenza della stringa di ricerca mentre la state scrivendo.

>

map Q gq

Definisce una mappatura di tasti. Troverete di più su questo argomento nella sezione che segue. Ciò definisce il comando "Q", per formattare con l'operatore "gq". Questo è ciò che avveniva prima di Vim 5.0. Altrimenti il comando "Q" fa partire il modo Ex, ma ciò non vi sarà necessario.

```
>
    vnoremap _g y:exe "grep /" . escape(@, '\\/') . "/" *.c *.h"<CR>
```

Questa mappatura copia il testo selezionato in Visual mode e lo usa come argomento di ricerca in programmi sorgenti C. Questa è una mappatura complessa. Notate come le mappature possano venire impiegate per fare cose piuttosto complesse. Tuttavia, è soltanto una sequenza di comandi che vengono eseguiti come se li scriveste direttamente.

```
>
    if &t_Co > 2 || has("gui_running")
        syntax on
        set hlsearch
    endif
```

Rende attiva l'evidenziazione della sintassi, ma solo se i colori sono disponibili. Inoltre l'opzione '**hlsearch**' dice a Vim di evidenziare le occorrenze dell'ultima stringa di ricerca utilizzata. Il comando "if" è utilissimo per impostare delle opzioni soltanto se viene verificata una condizione. Di più sull'argomento in [|usr_41.txt|](#).

filetype plugin indent on

vimrc-filetype >

Questo comando avvia tre meccanismi molto intelligenti:

1. Riconoscimento del tipo di file.
Ogni volta che iniziate a lavorare su di un file, Vim tenta di capire di che tipo di file si tratti. Se lavorate su "main.c", Vim noterà l'estensione ".c" e concluderà che si tratta di un file del tipo "c". Se aprite un file che inizia con "#!/bin/sh", Vim riconoscerà un file di tipo "sh".
Il riconoscimento del tipo di file viene usato sia per l'evidenziazione della sintassi che per le altre due funzioni viste prima.
Vedere [|filetypes|](#).
2. Utilizzare i file di plugin per il tipo di file.
Tipi di file diversi vengono elaborati con opzioni diverse. Ad es., lavorando con un file "c", risulta utile per impostare l'opzione '**cindent**' per rientrare automaticamente le linee. Le impostazioni di queste utili opzioni vengono fornite insieme a Vim sotto forma di plugin relativi al tipo di file. Potete aggiungerne anche dei vostri. Vedere [|write-filetype-plugin|](#).
3. Utilizzo dei file di rientro
Scrivendo codice il rientro di una linea può essere spesso calcolato automaticamente. Vim viene fornito con queste regole di rientro per un certo numero di tipi di file. Vedere [|:filetype-indent-on|](#) e '**indentexpr**'.

```
>
    autocmd FileType text setlocal textwidth=78
```

Ciò fa sì che Vim interrompa il testo per evitare linee che superino la lunghezza di 78 caratteri.

Ma solo per file che vengano riconosciuti come file di puro testo. E' composto di due parti. "autocmd FileType text" è un autocomando. Dispone che venga eseguito automaticamente il comando che segue se il tipo di file viene riconosciuto come "text". "setlocal textwidth=78" imposta a 78 l'opzione '**textwidth**', soltanto entro questo file locale.

restore-cursor >

```
>
    autocmd BufReadPost *
        \ if line("'\"") > 1 && line("'\"") <= line("$") |
        \   exe "normal! g`\"" |
        \ endif
```

Un altro autocomando. Questa volta viene impiegato dopo l'apertura di qualunque file. Quella roba complicata che segue verifica se sia stato definito il segnaposto '"', e conseguentemente salta ad esso. La barra inversa all'inizio di una riga serve per continuare il comando che inizia nella riga precedente. Ciò permette di non avere linee eccessivamente lunghe. Vedere [|line-continuation|](#). Funziona soltanto entro uno script di Vim e non direttamente dalla linea di comando.

```
=====
*05.3* Semplici mappature
```

Una mappatura vi consente di raggruppare una sequenza di comandi sotto un solo tasto. Supponiamo per esempio, che dobbiate includere certe parole tra parentesi graffe. In altre parole dovete trasformare una parola come "amount" in "{amount}". Con il comando :map, potete dire a Vim che il tasto F5 svolge questo lavoro. Il comando risulterà come segue: >

```
:map <F5> i{<Esc>ea}<Esc>
```

<

Nota:

Per immettere questo comando dovrete scrivere <F5>, quattro caratteri. Analogamente, <Esc> non si inserisce schiacciando il tasto <Esc>, ma scrivendo cinque caratteri. Fate caso a questa differenza mentre leggete il manuale!

Scomponiamo quanto sotto:

```
<F5>      Il tasto funzione F5. È il segnale di avvio che fa eseguire
           il comando quando il tasto viene premuto.

i{<Esc>    Inserisce il carattere {. Il tasto <Esc> termina il modo
           Insert.

e          Sposta il cursore alla fine della parola.

a}<Esc>     Appone la } dopo la parola.
```

Dopo avere realizzato il comando ":map", tutto ciò che dovete fare per immettere {} attorno ad una parola è porre il cursore sul primo carattere e premere F5.

In questo esempio il segnale di avvio è un singolo tasto; potrebbe essere una stringa di caratteri. Ma se usaste un comando esistente di Vim il comando stesso non sarebbe più disponibile. Meglio evitarlo.

L'unico tasto che può essere usato per eseguire una mappatura è la barra rovesciata. Poiché certamente vorrete definire più di una sola mappatura, aggiungete un altro carattere. Potreste mappare "\p" per aggiungere parentesi tonde attorno ad una parola e "\c" per porvi parentesi graffe, ad es.: >

```
:map \p i(<Esc>ea)<Esc>
:map \c i{<Esc>ea}<Esc>
```

Dovrete digitare la \ e la p rapidamente in sequenza, così Vim saprà che lavorano insieme.

Il comando ":map" (senza argomenti) elenca le vostre mappature esistenti.

Almeno quelle per il modo Normal. Altro sulle mappature nella sezione [|40.1|](#).

```
=====
*05.4* Aggiungere un plugin                                *add-plugin* *plugin*
```

Le funzionalità di Vim possono essere estese aggiungendo plugin. Un plugin non è altro che uno script di Vim che viene caricato automaticamente all'avvio di Vim. Potete aggiungere facilmente un plugin inserendolo nella vostra directory dei plugin.

**{non disponibile se Vim è stato compilato senza la funzionalità
|+eval|}**

Ci sono due tipi di plugin:

```
plugin globali: Usati per ogni tipo di file
filetype plugin: Usati solo per un tipo di file specifico
```

Prima parleremo dei plugin globali, poi di quelli relativi al tipo di file `|add-filetype-plugin|`.

PLUGIN GLOBALI

standard-plugin

Avviando Vim, questi caricherà automaticamente un certo numero di plugin globali.

Non siete obbligati a fare nulla per ottenere ciò. Aggiungono funzionalità che potrebbero servire a molti, ma che sono state implementate come script di Vim anziché venir compilate entro di esso. Le potete trovare elencate nell'indice di help `|standard-plugin-list|`. Vedere anche `|load-plugins|`.

add-global-plugin

Potete creare plugin globali per aggiungere funzionalità che pensate di dover usare frequentemente durante l'utilizzo di Vim. Servono due soli passaggi per aggiungere un plugin globale:

1. Ottenere una copia del plugin.
2. Metterlo nella directory giusta.

COME OTTENERE UN PLUGIN GLOBALE

Dove potete trovare i plugin?

- Qualcuno è compreso insieme con Vim. Lo potete trovare nella directory `$VIMRUNTIME/macros` e nelle sue sub-directory.
- Scaricatelo dalla rete. Ce n'è un'ampia collezione in <http://www.vim.org>.
- Ne vengono inviati molti tramite la `|maillist|` di Vim.
- Potreste scrivervelo anche da soli, vedere `|write-plugin|`.

USARE UN PLUGIN GLOBALE

Prima leggete il testo entro il plugin stesso per verificare l'esistenza di qualsiasi condizione speciale.

Poi copiate il file nella vostra directory dei plugin:

system	directory dei plugin	~
Unix	~/.vim/plugin/	
PC e OS/2	\$HOME/vimfiles/plugin o \$VIM/vimfiles/plugin	
Amiga	s:vimfiles/plugin	
Macintosh	\$VIM:vimfiles:plugin	
Mac OS X	~/.vim/plugin/	
RISC-OS	Choices:vimfiles.plugin	

Esempio per Unix (nel caso non ci sia ancora le directory dei plugin): >

```
mkdir ~/.vim
mkdir ~/.vim/plugin
cp /usr/local/share/vim/vim60/macros/justify.vim ~/.vim/plugin
```

Tutto qui! Ora potete impiegare i comandi definiti in questo plugin per giustificare il testo.

FILETYPE PLUGIN

add-filetype-plugin* *ftplugins

La distribuzione di Vim prevede un certo numero di plugin per tipi di file diversi che potete avviare con il seguente comando: >

```
:filetype plugin on
```

Tutto qui! Vedere `|vimrc-filetype|`.

Invece di mettere plugin direttamente nella directory plugin/, potete organizzarli meglio mettendoli in sotto-directory sotto plugin/. Ad es., potreste usare `"~/.vim/plugin/perl/*.vim"` per tutti i vostri plugin Perl.

Se aveste perso uno dei plugin per un tipo di file che state usando, o ne aveste trovato uno migliore, potete aggiungerlo. Ci sono due passaggi per aggiungere un filetype plugin:

1. Trovare una copia del plugin.
2. Copiarlo nella directory giusta.

COME TROVARE UN FILETYPE PLUGIN

Potete trovarlo negli stessi posti dei plugin globali. Guardate se si menziona il tipo del file, così potrete sapere se il plugin sia globale o riferito al tipo del file. Gli script in \$VIMRUNTIME/macros sono tutti globali, i filetype plugin sono in \$VIMRUNTIME/ftplugin.

COME USARE UN FILETYPE PLUGIN

ftplugin-name

Potete aggiungere un filetype plugin copiandolo nella directory giusta. Il nome di questa directory è nella stessa directory citata prima per i plugin globali, ma l'ultima parte è "ftplugin". Supponiamo che abbiate trovato un plugin per il tipo di file "stuff", e stiate usando un sistema Unix. Potete spostare questo file nella directory ftplugin: >

```
mv thefile ~/.vim/ftplugin/stuff.vim
```

Se tale file esistesse già vorrebbe dire che avete già un plugin per "stuff". Potreste verificare che il plugin esistente non confligga con quello che state aggiungendo. Se risultasse OK, potreste dargli un altro nome: >

```
mv thefile ~/.vim/ftplugin/stuff_too.vim
```

L'underscore viene usato per separare il nome del tipo di file dal resto, che può essere a piacere. Se usaste "otherstuff.vim" non funzionerebbe, sarebbe caricato per il filetype "otherstuff".

Su MS-DOS non potete usare nomi lunghi. Vi trovereste nei guai aggiungendo un secondo plugin il cui tipo di file avesse più di sei caratteri. Potete adoperare un'altra directory per aggirare ciò: >

```
mkdir $VIM/vimfiles/ftplugin/fortran
copy thefile $VIM/vimfiles/ftplugin/fortran/too.vim
```

I nomi generici per i filetype plugin sono: >

```
ftplugin/<filetype>.vim
ftplugin/<filetype>_<name>.vim
ftplugin/<filetype>/<name>.vim
```

Qui "<name>" può essere qualsiasi nome preferiate. Esempi per il tipo di file "stuff" su Unix: >

```
~/.vim/ftplugin/stuff.vim
~/.vim/ftplugin/stuff_def.vim
~/.vim/ftplugin/stuff/header.vim
```

La parte **<filetype>** è il nome del tipo di file per cui il plugin deve essere usato.

Solo file di questo tipo utilizzeranno le impostazioni del plugin. La parte **<name>** del file plugin non è un problema, potete usarla in molti plugin per lo stesso tipo di file. Nota Il nome deve terminare in ".vim".

Ulteriori letture:

filetype-plugins	Documentazione per i filetype plugin ed informazioni su come evitare che la mappatura causi problemi.
load-plugins	Quando i plugin globali vengono caricati all'avvio.
ftplugin-override	Come forzare le impostazioni di un plugin globale.
write-plugin	Come scrivere uno script di plugin.
plugin-details	Per ulteriori informazioni su come usare i plugin o se un plugin non vi funzionasse.
new-filetype	Come riconoscere un nuovo filetype.

```
=====
*05.5* Aggiungere un file di Aiuto      *add-local-help* *matchit-install*
```

Se siete fortunati, il plugin che avete installato avrà con sé un file di help.

Adesso spiegheremo come installarlo, così potrete trovare facilmente aiuto per i vostri nuovi plugin.

Usiamo il plugin "matchit.vim" come esempio (viene fornito con Vim). Questo plugin fa sì che il comando "%" salti ai tag HTML, if/else/endif negli scripts di Vim, etc. Utilissimo, anche se non compatibile con precedenti versioni di vim (e per questo motivo non viene abilitato di default).

Questo plugin viene fornito corredato della documentazione: "matchit.txt". Mettiamo la prima copia del plugin nella directory giusta. Questa volta lo faremo entro Vim, per poter usare \$VIMRUNTIME. (Potete saltare qualche comando "mkdir" se avete già la directory.) >

```

:mkdir ~/.vim
:mkdir ~/.vim/plugin
:cp $VIMRUNTIME/macros/matchit.vim ~/.vim/plugin

```

Il comando "cp" vale in ambiente Unix, in MS-DOS si può usare "copy".

Ora create una directory "doc" entro una delle directory entro il 'runtimepath'.

```

:mkdir ~/.vim/doc

```

Copiate il file di help entro la directory "doc". >

```

:cp $VIMRUNTIME/macros/matchit.txt ~/.vim/doc

```

Eccovi il trucco che vi consente di saltare sugli oggetti del nuovo file di help: generate il file locale dei tag con il comando |:helptags|. >

```

:helptags ~/.vim/doc

```

Ora potete usare il comando >

```

:help g%

```

per trovare aiuto per "g%" nel file di help che avete appena aggiunto. Potete vedere un accesso per il file di help locale facendo: >

```

:help local-additions

```

Le linee del titolo dai file di help locali verranno automaticamente aggiunte a questa sezione. Lì potrete vedere quali file locali di help siano stati aggiunti e saltare ad essi attraverso il loro tag.

Per scrivere un file locale di help vedere |write-local-help|.

=====

05.6 La finestra delle opzioni

Se state cercando un'opzione che faccia ciò che vi serve, la potrete trovare qui nei file di help: |options|. Un altro modo è quello di usare questo comando: >

```

:options

```

Ciò aprirà una nuova finestra con una lista di opzioni ed una linea di commento.

Le opzioni sono raggruppate per argomento. Portate il cursore sull'argomento e premete <Enter> per andare là. Premete <Enter> un'altra volta per tornare indietro. Oppure usate CTRL-O.

Potete cambiare il valore di un'opzione. Ad es., spostatevi sull'argomento "displaying text". Poi muovete il cursore più in basso, su questa linea:

```

set wrap          nowrap ~

```

Premendo <Enter>, la linea cambierà in :

```

set nowrap        wrap ~

```

L'opzione verrà disattivata.

Immediatamente sopra questa linea c'è una breve descrizione dell'opzione **'wrap'**. Spostate il cursore in alto di una linea per porlo entro questa riga. Adesso premete **<Enter>** e salterete all'help complessivo sull'opzione **'wrap'**.

Per opzioni che prevedono un argomento numerico o di stringa, potete mettere un nuovo valore.

Poi premete **<Enter>** per applicare il nuovo valore. Ad es., per spostare il cursore qualche linea più sopra:

```
set so=0 ~
```

Ponete il cursore sotto lo zero con "\$". Cambiatelo con cinque attraverso "r5". Ora premete **<Enter>** per assegnare il nuovo valore. Ora muovendo il cursore attorno noterete che il testo inizia a scorrere prima che abbiate trovato il margine. Ciò è quanto fa l'opzione **'scrolloff'**, che specifica un offset rispetto al bordo della finestra dove inizia lo scorrimento.

```
=====
*05.7* Le opzioni più usate
```

C'è un numero enorme di opzioni. Molte di esse non le userete quasi mai. Alcune delle più utili le citeremo qui. Non dimenticate che potete avere maggiore aiuto su queste opzioni tramite il comando **":help"**, racchiudendo il nome dell'opzione tra due virgolette singole. Ad es.: >

```
:help 'wrap'
```

Nel caso aveste smarrito il valore di un'opzione, potete riportarlo al valore di default scrivendo un ampersand (&) dopo il nome dell'opzione. Esempio: >

```
:set iskeyword&
```

LINEE NON SPEZZATE

Vim normalmente spezza le linee lunghe, affinché possiate vedere tutto del testo. Talvolta è meglio lasciare che il testo continui oltre il bordo destro della finestra. Vi toccherà scorrere il testo da sinistra a destra per vedere tutta la lunga linea. Disattivate il wrapping con questo comando: >

```
:set nowrap
```

Vim vi consentirà di spostarvi lungo il testo e raggiungere anche quello che non viene mostrato. Per visualizzare dieci caratteri oltre il bordo della finestra fate così: >

```
:set sidescroll=10
```

Ciò non altera il testo entro il file, solo il modo come esso viene mostrato.

AMPLIARE IL CAMPO D'AZIONE DEI COMANDI DI MOVIMENTO

Molti comandi per spostarsi attraverso il testo non vanno oltre l'inizio o la fine della linea. Potete cambiare ciò con l'opzione **'whichwrap'**. Quanto segue la imposta al valore di default: >

```
:set whichwrap=b,s
```

Ciò permette al tasto **<BS>**, quando usato all'inizio di una linea, di muovere il cursore alla fine della linea precedente. Ed il tasto **<Space>** sposterà il cursore dalla fine della linea all'inizio della successiva.

Per consentire ai tasti cursore **<Left>** e **<Right>** di avere il medesimo comportamento, usate questo comando: >

```
:set whichwrap=b,s,<,>
```

Ciò tuttavia soltanto nel modo Normal. Per permettere a **<Left>** e **<Right>** di fare ciò in modo Insert fate così: >

```
:set whichwrap=b,s,<,>[,]
```


Ci sono pochi altri flag che si possono aggiungere, vedere `'whichwrap'`.

VEDERE LE TABULAZIONI

Quando ci sono delle tabulazioni `[Tab]` entro un file non potete vedere dove siano. Per renderle visibili: >

```
:set list
```

Adesso ogni Tab verrà mostrato come `^I`. Ed un carattere `$` verrà mostrato alla fine di ogni linea, così potrete vedere eventuali spazi inutili alla fine della linea che altrimenti non sarebbero visibili.

Uno svantaggio è che ciò diventa noioso se ci sono molti Tab entro un file. Se avete un terminale a colori o state usando la GUI, Vim può mostrare spazi e Tab come caratteri evidenziati. Usate l'opzione `'listchars'`: >

```
:set listchars=tab:>-,trail:-
```

Adesso ogni Tab verrà mostrato come `>---` (con un numero variabile di `"-"`) e gli spazi inutili come `"-"`. Va molto meglio, non è vero?

PAROLE CHIAVE

L'opzione `'iskeyword'` specifica quali caratteri possano apparire entro una parola: >

```
< :set iskeyword
iskeyword=@,48-57,_,192-255 ~
```

La `"@"` sta per tutte le lettere dell'alfabeto. `"48-57"` sta per i caratteri ASCII da 48 a 57, che sono i numeri da 0 a 9. `"192-255"` sono i caratteri stampabili latini.

Talvolta vorrete includere una linea nella parole chiave, per fare sì che comandi come `"w"` considerino `"upper-case"` come una sola parola. Potete farlo così: >

```
:set iskeyword+==
:set iskeyword
< iskeyword=@,48-57,_,192-255,- ~
```

Se osservate il nuovo valore, vedrete che Vim ha aggiunto una virgola al vostro posto.

Per eliminare un carattere usate `"-=`". Ad es., per rimuovere l'underscore: >

```
:set iskeyword-=_
:set iskeyword
< iskeyword=@,48-57,192-255,- ~
```

Questa volta una virgola verrà cancellata automaticamente.

SPAZIO PER LE COMUNICAZIONI

Quando avviate Vim c'è una linea in basso che viene usata per i messaggi. Se un messaggio fosse lungo, verrebbe troncato, così potreste vederne solo una parte, oppure il testo scorrerebbe e voi dovrete premere `<Enter>` per continuare.

Potete impostare l'opzione `'cmdheight'` per il numero di linee da usare per i messaggi. Esempio: >

```
:set cmdheight=3
```

Significa che ci sarà meno spazio per scrivere del testo, si tratta di un compromesso.

```
=====
```

Capitolo seguente: |usr_06.txt| Usare l'evidenziazione della sintassi

Copyright: vedere |manual-copyright| vim:tw=78:ts=8:ft=help:norl:

Per segnalazioni scrivere a vimdoc.it at gmail dot com
oppure ad Antonio Colombo azcl00 at gmail dot com