usr_30.txt Per Vim version 7.1. Ultima modifica: 2007 Nov 10

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Alessandro Melillo

Editare programmi

Vim ha diversi comandi che sono d'aiuto nello scrivere programmi per computer. Compila un programma e salta direttamente agli errori riscontrati. Definisce automaticamente il rientro per molti linguaggi e formatta i commenti.

Capitolo seguente: |usr_31.txt| Sfruttare la GUI
Capitolo precedente: |usr_29.txt| Spostarsi attraverso i programmi
Indice: |usr_toc.txt|

==========

30.1 Compilazione

Vim ha una serie di comandi cosiddetti "quickfix". Questi permettono di compilare un programma all'interno di Vim e successivamente esaminare gli errori generati e correggerli (fortunatamente). Poi è possibile ricompilare e correggere ogni altro errore si trovasse, sino a che il nostro programma non compili senza errori.

Il comando seguente lancia "make" (con qualsiasi argomento gli si passi) e cattura il risultato: >

```
:make {argomenti}
```

Se vengono generati errori, vengono catturati e l'editor vi posiziona nel punto in cui si è verificato il primo errore.
   Date un'occhiata a una sessione ":make" d'esempio. (Le sessioni :make tipiche generano molti errori ed assai meno di quelli stupidi). Dopo aver scritto ":make" lo schermo appare così:

```
:!make | &tee /tmp/vim215953.err ~
gcc -g -Wall -o prog main.c sub.c ~
main.c: In function 'main': ~
main.c:6: too many arguments to function 'do_sub' ~
main.c: At top level: ~
main.c:10: parse error before '}' ~
make: *** [prog] Error 1 ~

2 returned ~
"main.c" 11L, 111C ~
(3 of 6): too many arguments to function 'do_sub' ~
Premi INVIO o un comando per proseguire ~
```

Da qui potete vedere che ci sono errori nel file "main.c". Premendo **<Enter>**, Vim mostra il file "main.c", con il cursore posizionato alla linea 6, la prima contenente un errore. Non avete avuto bisogno di specificare il file od il numero di riga, Vim sapeva dove andare per guardare i messaggi d'errore.

```
                +-------------------------------------------------+
                |int main()                                       |
                |{                                                |
                |       int i=3;                                  |
     cursore -> |       do_sub("foo");                            |
                |       ++i;                                      |
                |       return (0);                               |
                |}                                                |
                |}                                                |
                | ~                                               |
                |(3 of 12): too many arguments to function 'do_sub' |
                +-------------------------------------------------+
```

Il comando seguente va dove si trova il prossimo errore: >

```
:cnext
```

Vim salta alla linea 10, l'ultima del file, dove c'è una '}' di troppo.
Quando non c'è sufficiente spazio, Vim abbrevia il messaggio d'errore. Per vedere il messaggio intero usate: >

```
:cc
```

Potete ottenere un sommario di tutti gli errori con il comando ":clist".
L'output appare così: >

```
:clist
< 3 main.c: 6:too many arguments to function 'do_sub' ~
  5 main.c: 10:parse error before '}' ~
```

Solo le righe in cui Vim riconosce un nome di file ed un numero di linea vengono elencate. Assume che quelle siano le linee interessanti e che il resto sia costituito solo da messaggi noiosi. Comunque, a volte linee non riconosciute contengono qualcosa che volete vedere. Per esempio, l'output del linker per una funzione non definita.
Per vedere tutti i messaggi aggiungete un "!" al comando: >

```
:clist!
< 1 gcc -g -Wall -o prog main.c sub.c ~
  2 main.c: In function 'main': ~
  3 main.c:6: too many arguments to function 'do_sub' ~
  4 main.c: At top level: ~
  5 main.c:10: parse error before '}' ~
  6 make: *** [prog] Error 1 ~
```

Vim evidenzierà l'errore corrente. Per tornare al precedente, usate:
>

```
:cprevious
```

Altri comandi per spostarvi nella lista errori:

```
:cfirst         al primo errore
:clast          all'ultimo errore
:cc 3           all'errore numero 3
```

USARE UN ALTRO COMPILATORE

Il nome del programma da eseguire quando si impartisce il comando ":make" è definito dall'opzione **'makeprg'**. Solitamente è settata a "make", ma gli utenti di Visual C++ dovrebbero impostarlo a "nmake" eseguendo il seguente comando: >

```
:set makeprg=nmake
```

Potete anche includere argomenti in questa opzione. Caratteri speciali devono essere protetti con un backslash. Esempio: >

```
:set makeprg=nmake\ -f\ project.mak
```

Potete includere parole chiave speciali di Vim nella specifica del comando. Il carattere % si espande al nome del file corrente. Così, se eseguite il comando: >

```
:set makeprg=make\ %
```

Mentre editate main.c, allora ":make" esegue il seguente comando: >

```
make main.c
```

Questo non è troppo utile, quindi potete ridefinire un po' il comando ed usare il modificatore :r (root): >

```
:set makeprg=make\ %:r.o
```

Ora il comando eseguito è questo: >

```
make main.o
```

Maggiori informazioni sui modificatori si trovano qui: |**filename-modifiers**|.

VECCHIE LISTE D'ERRORE

Supponiamo di eseguire ":make" su un programma. C'è un messaggio di avviso in un file e uno di errore in un altro. Correggete l'errore ed usate ancora ":make" per controllare se sia stato realmente corretto. Adesso volete guardare il messaggio di avvertimento. Non compare nell'ultima lista di errori, poiché il file con l'avviso non è stato ricompilato. Si può tornare alla lista precedente con: >

```
:colder
```

E poi usare ":clist" e ":cc" **{nr}** per saltare al punto con l'avviso.
Per andare avanti alla successiva lista di errori: >

```
:cnewer
```

Vim ricorda dieci liste di errori.

ALTERNARE I COMPILATORI

Bisogna dire a Vim in quale formato il nostro compilatore produca i messaggi d'errore. Ciò viene fatto con l'opzione **'errorformat'**. La sintassi di questa opzione è piuttosto complicata e si può adattare a qualsiasi compilatore. Potete trovare la spiegazione è qui: |**errorformat**|.

Potreste usare diversi compilatori. Settare l'opzione **'makeprg'**, e specialmente **'errorformat'** ogni volta non è semplice. Vim offre un metodo semplice per farlo. Per esempio, per passare ad usare il compilatore Microsoft Visual C++: >

```
:compiler msvc
```

Questo comando troverà lo script di Vim per il compilatore "msvc" ed imposterà le opzioni appropriate.
Potete anche scrivere i vostri file di opzioni. Vedere |**write-compiler-plugin**|.

RE-INDIRIZZAMENTO DELL'OUTPUT

Il comando ":make" redirige l'output del programma eseguito ad un file d'errore. Come ciò funzioni dipende da diversi fattori quali la **'shell'**. Se il vostro comando ":make" non cattura l'output verificate le pozioni **'makeef'** e **'shellpipe'**. Anche le opzioni **'shellquote'** e **'shellxquote'** potrebbero essere utili.

Nel caso in cui non riusciate ad ottenere che ":make" rediriga l'output per voi, un'alternativa è compilare il programma in un'altra finestra e reindirizzare l'output in un file. Quindi farlo leggere a Vim con: >

```
:cfile {nomefile}
```

Saltando agli errori funzionerà come con il comando ":make".

==========

30.2 Rientro dei file in C

Un programma è molto più facile da capire quando le linee siano state rientrate appropriatamente. Vim offre numerosi modi per farlo con meno lavoro.
Per i programmi in C occorre settare l'opzione **'cindent'**. Vim sa molto dei programmi in C e cercherà in ogni modo di impostare il rientro per voi. Impostate l'opzione **'shiftwidth'** al numero di spazi che volete per un livello più profondo. Quattro spazi vanno bene. Basta un solo comando ":set": >

```
:set cindent shiftwidth=4
```

Con questa opzione abilitata, quanto scrivete qualcosa tipo "if (x)", la linea successiva verrà automaticamente rientrata ad un livello aggiuntivo.

```
                                if (flag)
        Rientro automatico  --->        do_the_work();
        Ritorno automatico   <--    if (other_flag) {
        Rientro automatico  --->        do_file();
        Mantenimento  rientro           do_some_more();
        Ritorno automatico <--      }
```

Quando scrivete qualcosa tra parentesi graffe ({}), il testo verrà rientrato all'inizio ma non alla fine. Il ritorno indietro verrà fatto dopo che avete battuto '}', poiché Vim non può indovinare cosa state per scrivere.

Un effetto collaterale del rientro automatico è che vi aiuta a trovare rapidamente gli errori nel codice. Quando battete una } per concludere una funzione, il solo vedere che il rientro automatico non si collochi dove previsto vi aiuta a capire che manca una }. Usate il comando "%" per trovare quale { corrisponda alla } che avete appena battuto.
Anche una) o un ; mancanti causano un ulteriore rientro. Così, se vedete più spazio bianco di quello che vi aspettavate, controllate le linee precedenti.

Quando avete del codice che sia mal formattato, oppure avete inserito e cancellato delle linee, dovrete rientrarlo di nuovo. L'operatore "=" lo fa. La forma più semplice è: >

```
==
```

Questo rientra la linea corrente. Come con tutti gli operatori, ci sono tre modi per usarlo. In Visual_Mode "=" rientra le linee selezionate. Un utile oggetto testuale è "a{". Questo seleziona il blocco {} corrente. Così, per rientrare di nuovo il blocco di codice in cui si trova il cursore: >

```
=a{
```

Se avete del codice veramente mal rientrato, potete rientrare di nuovo l'intero file con: >

```
gg=G
```

Comunque, non fatelo con dei file che avete rientrato con cura manualmente. Il rientro automatico fa un buon lavoro, ma in certe situazioni potreste volerlo evitare.

IMPOSTARE LO STILE DI RIENTRO

Persone diverse hanno differenti stili di rientro. Per default, Vim fa un gradevole buon lavoro di rientro, nel modo in cui lo fa il 90% dei programmatori. Comunque ci sono diversi stili; quindi, se volete, potete personalizzare lo stile di rientro con l'opzione **'cinoptions'**.
Per default **'cinoptions'** è vuota e Vim impiega il proprio stile predefinito. Potete aggiungere istanze quando volete qualcosa di diverso. Per esempio, per far sì che le parentesi graffe vengano posizionate così:

```
if (flag) ~
  { ~
    i = 8; ~
    j = 0; ~
  } ~
```

Usate questo comando: >

```
:set cinoptions+={2
```

Ci sono molti oggetti del genere. Vedere |**cinoptions-values**|.

==========

30.3 Rientro automatico

Se non volete attivare l'opzione 'cindent' manualmente ogni volta che editate un file in C? Ecco come richiederla in automatico: >

```
:filetype indent on
```

In realtà, questo fa molto di più che attivare 'cindent' per i sorgenti C. Prima di tutto, abilita l'individuazione del tipo di file. La stessa usata per l'evidenziazione della sintassi.
Una volta conosciuto il tipo di file, Vim cercherà un tipo di rientro per questo tipo di file. La distribuzione di Vim include un buon numero di questi tipi per vari linguaggi di programmazione. Il file di rientro poi si preoccuperà di predisporre il giusto rientro per il file corrente.

Se non vi piace il rientro automatico, potete disattivarlo: >

```
:filetype indent off
```

Se non vi piace il rientro per un determinato tipo di file, ecco come evitarlo. Create un file con questa linea: >

```
:let b:did_indent = 1
```

Adesso dovete salvarlo con un nome specifico:

```
{directory}/indent/{filetype}.vim
```

Dove {filetype} è il nome del tipo di file, come "cpp" o "java". Potete vedere l'esatto nome che Vim ha rilevato con questo comando: >

```
:set filetype
```

In questo file l'output è:

```
filetype=help ~
```

E quindi usereste "help" come {filetype}.
Per la parte {directory} dovete utilizzare la vostra directory di runtime. Guardate l'output di questo comando: >

```
set runtimepath
```

Adesso utilizzate il primo elemento, il nome che precede la prima virgola. Quindi, se l'output appare così:

```
runtimepath=~/.vim,/usr/local/share/vim/vim60/runtime,~/.vim/after ~
```

Usate "~/.vim" come {directory}. Quindi il nome di file risultante è:

```
~/.vim/indent/help.vim ~
```

Invece di disattivare il rientro, potreste scrivere un vostro file di rientro. Come farlo è spiegato qui: |indent-expression|.

==========

30.4 Altri rientri

La forma più semplice di rientro automatico è quella dell'opzione 'autoindent'. Usa il rientro della riga precedente. Un po' più furba è l'opzione 'smartindent'. E' utile per i file che non hanno un file di rientro. 'smartindent' non è intelligente come 'cindent' ma lo è sempre più di 'autoindent'.
Con 'smartindent' impostata viene aggiunto un livello extra di rientro dopo ogni { e tolto un livello dopo ogni }. Viene aggiunto un livello extra anche per ognuna delle parole contenute nell'opzione 'cinwords'. Le righe che iniziano con # vengono trattate in maniera speciale: viene tolto ogni rientro. Il motivo è che le direttive di preprocessore in questo modo inizieranno tutte a colonna 1. Il rientro viene ripristinato alla linea successiva.

CORREZIONE DEI RIENTRI

Quando state usando 'autoindent' o 'smartindent' per ottenere il rientro della

prima linea, molte volte dovrete aggiungere o rimuovere un valore di 'shiftwidth' del rientro. Un modo rapido per farlo è usare i comandi CTRL-D e CTRL-T in Insert mode.
Ad esempio, state scrivendo uno script di shell che si suppone appaia come questo:

```
if test -n a; then ~
   echo a ~
   echo "-------" ~
fi ~
```

Iniziate impostando questa opzione: >

```
:set autoindent shiftwidth=3
```

Iniziate scrivendo la prima linea, <Enter> e l'inizio della seconda:

```
if test -n a; then ~
echo ~
```

Ora vi accorgete di aver bisogno di un rientro. Scrivete CTRL-T. Il risultato:

```
if test -n a; then ~
   echo ~
```

Il comando CTRL-T, in Insert mode, aggiunge uno 'shiftwidth' di rientro a prescindere dalla posizione in cui vi trovate nella linea.
Continuate a scrivere la seconda linea, <Enter> e la terza linea. Questa volta il rientro è OK. Ancora <Enter> e la prossima linea. Adesso avrete questo:

```
if test -n a; then ~
   echo a ~
   echo "-------" ~
   fi ~
```

Per rimuovere il rientro superfluo nell'ultima linea premete CTRL-D. Ciò toglierà uno 'shiftwidth' di rientro, a prescindere dalla posizione in cui vi trovate nella linea.
Quando siete in Normal mode, potete usare i comandi ">>" e "<<" per cambiare linea. ">" e "<" sono operatori, così avete i soliti tre modi per specificare le linee che volete rientrare. Una combinazione utile è: >

```
>i{
```

Ciò aggiungerà un rientro all'attuale blocco di linee, entro {}. Le linee comprese tra { e } verranno lasciate non modificate. ">a{" le include. In questo esempio il cursore è su "printf":

```
original text           after ">i{"             after ">a{"

if (flag)               if (flag)               if (flag) ~
{                       {                           { ~
printf("yes");              printf("yes");          printf("yes"); ~
flag = 0;               flag = 0;                   flag = 0;  ~
}                       }                           } ~
```

==========

30.5 Tabulazioni e spazi

'tabstop' è impostato ad otto di default. Sebbene lo possiate cambiare, facilmente vi ritroverete nei problemi dopo. Altri programmi potrebbero non sapere quale valore di tabstop abbiate usato. Essi probabilmente usano il valore di default di otto spazi ed il vostro testo sembrerà improvvisamente assai diverso. Parimenti molte stampanti usano un valore fisso di tabstop di otto spazi. Così è meglio lasciar perdere 'tabstop'. (Se state lavorando con un file che sia stato scritto con una diversa impostazione di tabstop, vedete |25.3| per correggere ciò).
Rientrare le linee di un programma usando un multiplo di otto spazi vi farà rapidamente andare verso il bordo destro della finestra. Usare uno spazio solo non fornirà abbastanza differenza alla vista. Molti preferiscono usare quattro spazi, un buon compromesso.

Sino a quando un **<Tab>** è di otto spazi e voi invece volete rientrare di quattro, non potete usare un carattere di **<Tab>** per fare il vostro rientro. Ci sono due modi per gestire la cosa:

1. Usare un mix di **<Tab>** e di spazi. Poiché un **<Tab>** prende il posto di otto spazi, nel vostro file ci staranno meno caratteri. Inserire un **<Tab>** è più rapido che inserire otto spazi. Il backspace lavora più velocemente e bene.

2. Usare solo degli spazi. Evita fastidi con programmi che usano un valore diverso di tabstop.

Fortunatamente, Vim supporta entrambi i metodi altrettanto bene.

SPAZI E TABULATORI

Usando una combinazione di tabulatori e di spazi lavorate correttamente. I default di Vim consentono di utilizzare agevolmente ciò.

Potete far vita migliore impostando l'opzione **'softtabstop'**. Questa opzione dice a Vim di far sembrare che il tasto **<Tab>** sia stato impostato al valore di **'softtabstop'**, mentre si sta usando una combinazione di spazi e tabulazioni.

Dopo aver eseguito il comando che segue, ogni volta che premerete il tasto **<Tab>** il cursore si sposterà al limite delle prossime quattro colonne: >

```
:set softtabstop=4
```

Iniziando dalla prima colonna e premendo **<Tab>**, avrete quattro spazi inseriti nel vostro testo. La seconda volta Vim assumerà i quattro spazi e li porrà entro un **<Tab>** (portandovi così all'ottava colonna). Così Vim usa il massimo numero possibile di **<Tab>** e riempie il resto con degli spazi.

L'uso del backspace opera analogamente in direzione opposta. Un **<BS>** cancellerà sempre il numero di spazi specificato in **'softtabstop'**. Quindi si adoperano quanti **<Tab>** è possibile e spazi per riempire i vuoti.

Quanto segue mostra cosa accada premendo **<Tab>** alcune volte, e poi usando **<BS>**. Un "." sta per uno spazio e "------->" per un **<Tab>**.

```
type                    result ~
<Tab>                   ....
<Tab><Tab>              ------->
<Tab><Tab><Tab>         ------->....
<Tab><Tab><Tab><BS>     ------->
<Tab><Tab><Tab><BS><BS> ....
```

Un'alternativa consiste nell'usare l'opzione **'smarttab'**. Quando questa viene impostata, Vim usa **'shiftwidth'** ad ogni **<Tab>** usato per rientrare una linea ed un vero **<Tab>** quando si batte dopo il primo carattere non-bianco. Comunque **<BS>** non opererà come con **'softtabstop'**.

SOLTANTO SPAZI

Se non volete affatto tabulatori nel vostro file potete impostare l'opzione **'expandtab'**: >

```
:set expandtab
```

Quando questa opzione è impostata il tasto **<Tab>** inserisce una serie di spazi. Così otterrete la stessa quantità di spazi come se fosse stato inserito un carattere **<Tab>** ma non esisterà un vero carattere **<Tab>** entro il vostro file.

Il tasto backspace cancellerà uno spazio alla volta. Così dopo aver scritto un solo **<Tab>** dovrete premere **<BS>** otto volte per cancellarlo. Se vi trovate entro un rientro, premendo **CTRL-D** andrete molto più svelti.

CAMBIARE I TABULATORI IN SPAZI (E VICEVERSA)

Impostando **'expandtab'** non si modificano i tabulatori esistenti. In altri termini ogni tabulatore nel documento resta un tabulatore. Se desiderate convertire i tabulatori in spazi, usate il comando ":retab". Usate questi comandi: >

```
:set expandtab
:%retab
```

Adesso Vim avrà cambiato il proprio rientro per usare spazi invece dei <Tab>. Comunque tutti i <Tab> che vengono dopo un carattere diverso dallo spazio vengono conservati. Se volete convertire anche questi ultimi aggiungete un !: >

```
:%retab!
```

Ciò è un po' più pericoloso perché può cambiare i tabulatori entro una stringa. Per vedere se ciò possa avvenire potreste usare questo: >

```
/"[^"\t]*\t[^"]*"
```

Si raccomanda di non usare tabulazioni dure entro una stringa. Sostituitele con "\t" per evitare problemi.

Un altro modo altrettanto valido è: >

```
:set noexpandtab
:%retab!
```

==

30.6 Formattazione dei commenti

Una delle grandi cose di Vim è che capisce i commenti. Potete chiedere a Vim di formattare un commento e lui farà la cosa giusta.
Supponiamo, ad esempio, che si abbia il seguente commento:

```
/* ~
 * This is a test ~
 * of the text formatting. ~
 */ ~
```

Potete chiedere a Vim di formattarlo ponendo il cursore all'inizio del commento e scrivendo: >

```
gq]/
```

"gq" è l'operatore per formattare del testo. "]/" è il movimento che vi porterà alla fine del commento. Il risultato sarà:

```
/* ~
 * This is a test of the text formatting. ~
 */ ~
```

Osservate come Vim abbia gestito correttamente l'inizio di ogni linea.
Un'alternativa consiste nel selezionare il testo da formattare in Visual mode e scrivere "gq".

Per aggiungere una nuova linea al commento ponete il cursore sulla linea di mezzo e premete "o". Il risultato sarà il seguente:

```
/* ~
 * This is a test of the text formatting. ~
 * ~
 */ ~
```

Vim ha inserito automaticamente un asterisco ed uno spazio al vostro posto. Adesso potete scrivere il testo del commento. Se esso venisse più lungo di 'textwidth', Vim andrà a capo. Anche questa volta l'asterisco verrà inserito automaticamente:

```
/* ~
 * This is a test of the text formatting. ~
 * Typing a lot of text here will make Vim ~
 * break ~
 */ ~
```

Affinché ciò funzioni debbono esserci alcuni flag presenti in 'formatoptions':

```
        r       inserisce l'asterisco battendo <Enter> nell'Insert_mode
        o       inserisce l'asterisco usando "o" od "O" nel Normal_mode
        c       spezza il testo del commento secondo 'textwidth'
```

Vedere |fo-table| per ulteriori flag.

DEFINIZIONE DI UN COMMENTO

L'opzione **'comments'** definisce a cosa debba assomigliare un commento. Vim distingue tra commenti di una sola linea e commenti che abbiano diverso inizio, fine e parte di mezzo.
Molti commenti su una sola linea iniziano con un carattere specifico. In C++ viene impiegato //, nei Makefile #, negli script di Vim ". Ad esempio per far capire a Vim i commenti di C++: >

```
        :set comments=://
```

I due punti separano i flag di un oggetto dal testo da cui si riconosce il commento. La forma di un oggetto in **'comments'** è:

```
        {flags}:{text}
```

La parte **{flags}** può essere anche vuota, come nel caso attuale.
Molti di questi oggetti possono essere concatenati, separati da virgole. Ciò consente di riconoscere diversi tipi di commento nello stesso tempo. Ad esempio, modifichiamo un messaggio di e-mail. Rispondendo, il testo scritto da altri verrà preceduto dai caratteri ">" e "!". Il comando funzionerebbe così: >

```
        :set comments=n:>,n:!
```

Ci sono due cose, una per i commenti che iniziano con ">" ed un'altra per quelli che lo fanno con "!". Entrambi usano il flag "n". Ciò significa che questi commenti sono annidati l'uno nell'altro. Così una linea che cominci con ">" può avere un altro commento dopo il ">".
Ciò consente di formattare messaggi come questo:

```
        > ! Did you see that site? ~
        > ! It looks really great. ~
        > I don't like it.  The ~
        > colors are terrible. ~
        What is the URL of that ~
        site? ~
```

Provate impostando **'textwidth'** ad un valore diverso, e.g., 80, e formattate il testo selezionandolo in Visual_mode e scrivendo "gq". Il risultato sarà:

```
        > ! Did you see that site?  It looks really great. ~
        > I don't like it.  The colors are terrible. ~
        What is the URL of that site? ~
```

Noterete che Vim non sposta il testo da un tipo di commento ad un altro. La lettera "I" nella seconda linea potrebbe venire posta alla fine della prima linea, ma poiché questa linea inizia con "> !" e la seconda linea con ">", Vim sa che si tratta di un diverso tipo di commento.

COMMENTI IN TRE PARTI

Un commento C inizia con "/*", ha "*" nel mezzo e "*/" alla fine. Si dovrà porre in **'comments'** affinché appaia così: >

```
        :set comments=s1:/*,mb:*,ex:*/
```

L'inizio è definito con "s1:/*". La "s" indica l'inizio di un commento in tre parti. I due punti separano i flags dal testo da essi il commento viene riconosciuto: "/*". C'è un solo flag: "1". Ciò dice a Vim che la parte di mezzo è scostata di uno spazio.
La parte di mezzo "mb:*" comincia con "m", che indica che si tratta di una parte di mezzo. Il flag "b" significa che uno spazio vuoto deve seguire il testo. Altrimenti Vim potrebbe considerare testo come "*pointer" come il mezzo di un commento.

La parte finale "ex:*/" viene identificata da una "e". Il flag "x" ha un significato speciale. Significa che Vim inserirà dopo un asterisco automaticamente, scrivendo / eliminerà gli spazi di troppo.

Per ulteriori dettagli vedere |**format-comments**|.

==

Capitolo seguente: |**usr_31.txt**| Sfruttare la GUI

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