usr_40.txt Per **Vim version 8.0.** Ultima modifica: 2013 Aug 05

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Fabio Teatini e Roberta Fedeli

Definire nuovi comandi

Vim è un elaboratore di testi assai estendibile. Potete prendere una sequenza di comandi che usate spesso e trasformarla in un nuovo comando. Oppure potete ridefinire un comando esistente. Gli "autocomandi" permettono di eseguire dei comandi automaticamente.

==========

40.1 Mappatura dei tasti

Una semplice mappatura è stata spiegata nella sezione |**05.3**|. Il principio è che una sola sequenza di tasti premuti viene tradotta in un'altra sequenza di tasti. È un meccanismo semplice, ma potente.

La forma più semplice è che ad un singolo tasto venga attribuito il significato di una sequenza di tasti. Poiché i tasti funzione, salvo <F1>, non hanno un significato predefinito in Vim, questi sono una buona scelta per definire delle mappature. Esempio: >

```
:map <F2> GoData: <Esc>:read !date<CR>kJ
```

Questo mostra come vengano usate tre modalità. Dopo avere raggiunto l'ultima linea con "G", il comando "o" aggiunge una nuova linea e avvia l'Insert mode. Il testo "Data: " viene inserito ed <Esc> vi porta fuori dall'Insert mode.

Notate che i tasti speciali sono indicati dentro i caratteri <>. Ciò viene chiamato notazione delle parentesi angolari. Scrivete ciò come caratteri separati, non premendo il tasto corrispondente. Ciò rende la mappatura più leggibile e potrete copiare ed incollare il testo senza problemi.

Il carattere ":" porta Vim sulla linea di comando. Il comando ":read !date" legge l'output emesso dal comando "date" e lo inserisce sotto la linea corrente. Il <CR> serve per eseguire il comando ":read".

A questo punto di esecuzione il testo apparirà così:

```
Date: ~
Fri Jun 15 12:54:34 CEST 2001 ~
```

Ora "kJ" sposta il cursore in su e unisce le due linee assieme.

Per decidere quale tasto o tasti usare per la mappatura, vedere |**map-which-keys**|.

MAPPATURA E MODALITA'

Il comando ":map" effettua la ridefinizione per i tasti in Normal mode. Potete anche definire mappature per altre modalità. Per esempio, ":imap" si applica all'Insert mode. Usatelo per inserire una data sotto il cursore: >

```
:imap <F2> <CR>Data: <Esc>:read !date<CR>kJ
```

Assomiglia molto alla mappatura per <F2> in Normal mode, soltanto l'inizio è diverso. La mappatura di <F2> in Normal mode c'è ancora. Così potete mappare lo stesso tasto diversamente per ciascuna modalità.

Attenzione che sebbene questa mappatura parta in Insert mode, essa finisce in Normal mode. Se voleste continuare in Insert mode, aggiungete una "a" alla mappatura.

Ecco un riassunto dei comandi di mappatura ed in quale modalità funzionano:

```
:map            Normal, Visual ed Operator-pending
:vmap           Visual
```

```
        :nmap           Normal
        :omap           Operator-pending
        :map!           Insert e Command-line
        :imap           Insert
        :cmap           Command-line
```

L'Operator-pending mode è quello in cui vi trovate dopo aver scritto un operatore come "d" o "y" e Vim attende che voi digitiate un comando di movimento od un oggetto di testo. Quindi, quando digitate "dw", il "w" viene inserito in Operator-pending mode.

Supponete di voler definire <F7> affinché il comando d<F7> cancelli un blocco di programma C (il testo racchiuso nelle parentesi graffe {}). Allo stesso modo, y<F7> potrebbe copiare il blocco di programma entro un registro anonimo. Quindi, ciò che dovete fare è di definire <F7> per selezionare il blocco del programma corrente. Potete farlo con il seguente comando: >

```
        :omap <F7> a{
```

Ciò fa sì che <F7> selezioni un blocco "a{" in Operator-pending mode, proprio come l'avete scritto. Questa mappatura è utile qualora sulla vostra tastiera risulti difficile la digitazione di un {.

ELENCO DELLE MAPPATURE

Per vedere le mappature attualmente definite, usate ":map" senza argomenti. Oppure una delle varianti che comprendono la modalità in cui funzionano. Il risultato potrebbe apparire così:

```
           _g            :call MioGrep(1)<CR> ~
        v  <F2>          :s/^/> /<CR>:noh<CR>`` ~
        n  <F2>          :.,$s/^/> /<CR>:noh<CR>`` ~
           <xHome>       <Home>
           <xEnd>        <End>
```

La prima colonna dell'elenco mostra in quale modalità la mappatura funzioni. Questo valore è "n" per il Normal mode, "i" per l'Insert mode, ecc.. Uno spazio viene utilizzato per una mappatura definita con ":map", tanto che funzioni sia in Normal che in Visual mode.
   Un utile uso dell'elencare le mappature è di provare se i tasti speciali entro parentesi angolari <> siano stati riconosciuti (ma funziona solo quando il colore sia supportato). Per esempio, quando <Esc> risulta colorato, sta per il carattere di escape. Quando ha lo stesso colore dell'altro testo, sono solo cinque caratteri.

RE-MAPPING

Il risultato di una mappatura viene esaminato per le altre mappature in essa. Per esempio, la mappatura per <F2> illustrata di seguito potrebbe essere abbreviata in: >

```
        :map <F2> G<F3>
        :imap <F2> <Esc><F3>
        :map <F3>  oData: <Esc>:read !date<CR>kJ
```

In Normal mode <F2> è associato allo spostamento sull'ultima linea ed allora appare come se <F3> fosse stato premuto. In Insert mode <F2> arresta l'Insert mode con <Esc> ed allora usa anche <F3>. Allora <F3> viene mappato per fare questo lavoro.

Immaginate di dover utilizzare sempre intensamente l'Ex mode, e di voler usare il comando "Q" per formattare del testo (era così nelle vecchie versioni di Vim). Questa mappatura lo farà: >

```
        :map Q gq
```

Ma, in casi rari dovrete usare il modo Ex comunque. Mappiamo "gQ" a Q, così da poter andare in Ex mode: >

```
        :map gQ Q
```

Ciò che accade ora è che quando scrivete "gQ" viene mappato in "Q". Presto e bene. Ma allora "Q" viene mappato in "gq", così scrivendo "gQ" risulta in "gq", e comunque non potete andare in Ex mode.
Per evitare che dei tasti vengano mappati più volte, usate il comando "noremap": >

```
:noremap gQ Q
```

Ora Vim sa che "Q" non deve essere impiegato per mappature che gli vengano applicate. Esiste un comando analogo per qualunque modo:

```
:noremap        Normal, Visual e Operator-pending
:vnoremap       Visual
:nnoremap       Normal
:onoremap       Operator-pending
:noremap!       Insert e Command-line
:inoremap       Insert
:cnoremap       Command-line
```

MAPPING RICORSIVO

Se una mappatura puntasse a se stessa potrebbe andare avanti all'infinito. Ciò può essere usato per ripetere un'azione un numero illimitato di volte.
Ad esempio, avete un elenco di file che contengano nella prima linea un numero di versione. Potete aprire questi file con "vim *.txt". Adesso state elaborando il primo file. Definite questa mappatura: >

```
:map ,, :s/5.1/5.2/<CR>:wnext<CR>,,
```

Adesso scrivete ",,". Ciò avvia la mappatura. Cambia "5.1" con "5.2" nella prima linea. Allora compie un ":wnext" per salvare il file ed aprire il prossimo. La mappatura finisce con ",,". Ciò avvia un'altra volta la stessa mappatura, così facendo la sostituzione, etc.
Ciò continua sino a quando vi sia un errore. In questo caso potrebbe essere un file dove il comando substitute non trovasse una corrispondenza per "5.1". Potete allora effettuare un cambio per inserire "5.1" e continuare scrivendo ancora una volta ",,". Ovvero se fallisse ":wnext", perché vi trovate già nell'ultimo file della lista.
Quando una mappatura trova un errore a mezza strada il resto della mappatura viene saltato. CTRL-C interrompe la mappatura (CTRL-Break su MS-Windows).

CANCELLAZIONE DI UNA MAPPATURA

Per rimuovere una mappatura usate ":unmap ". Il modo nel quale viene applicata la rimozione delle mappatura dipende dal comando usato:

```
:unmap          Normal, Visual e Operator-pending
:vunmap         Visual
:nunmap         Normal
:ounmap         Operator-pending
:unmap!         Insert e Command-line
:iunmap         Insert
:cunmap         Command-line
```

Ecco un trucco per definire una mappatura che lavora in Normal ed Operator-pending mode, ma non in Visual mode. Prima definitelo per tutti e tre i modi, poi cancellatelo per il Visual mode: >

```
:map <C-A> /---><CR>
:vunmap <C-A>
```

Notare che i cinque caratteri "<C-A>" stanno per pressione contemporanea dei tasti CTRL-A.

Per eliminare tutte le mappature usate il comando |**:mapclear**|. Potete così osservare la differenza a seconda dei modi diversi. Attenzione che questo comando non consente di utilizzare undo.

CARATTERI SPECIALI

Il comando ":map" può essere seguito da un altro comando. Un carattere | separa i due comandi. Ciò significa anche che il carattere | non può essere usato entro un comando di mappatura. Per includerne uno usate <Bar> (cinque caratteri). Esempio:
>
```
:map <F8> :write <Bar> !checkin %:S<CR>
```

Lo stesso problema c'è con il comando ":unmap", con l'aggiunta che dovete prestare attenzione a non lasciare uno spazio vuoto. Questi due comandi sono diversi:
>
```
:unmap a | unmap b
:unmap a| unmap b
```

Il primo comando prova a cancellare la mappatura di "a ", seguito da uno spazio.

Per usare uno spazio entro una mappatura scrivete <Space> (sette caratteri): >

```
:map <Space> W
```

Ciò fa sì che la barra spaziatrice sposti in avanti di una parola separata da uno spazio bianco.

Non si può mettere un commento dopo una mappatura perché il carattere " verrebbe considerato parte della mappatura.

MAPPATURE ED ABBREVIAZIONI

Le abbreviazioni assomigliano molto a delle mappature in Insert mode. Gli argomenti vengono gestiti nello stesso modo. La differenza principale è il modo in cui vengono avviate. Un'abbreviazione si avvia scrivendo un carattere non parola dopo la parola. Una mappatura parte dopo aver scritto l'ultimo carattere.

Un'altra differenza è che i caratteri che scrivete per un'abbreviazione vengono inseriti nel testo mentre lo state scrivendo. Quando l'abbreviazione parte questi caratteri vengono cancellati e sostituiti da ciò che l'abbreviazione produce. Scrivendo i caratteri per una mappatura non viene inserito nulla sino a quando non scriverete l'ultimo carattere che la fa partire. Se l'opzione 'showcmd' è impostata, i caratteri che sono stati scritti vengono mostrati nell'ultima riga della finestra di Vim.

Un'eccezione si verifica quando una mappatura è ambigua. Supponiamo che abbiate fatto due mappature: >

```
:imap aa foo
:imap aaa bar
```

Ora, mentre scrivete "aa", Vim non può sapere se deve applicare la prima o la seconda mappatura. Attende che venga digitato un altro carattere. Se questo fosse una "a", verrebbe applicata la seconda mappatura e risulterebbe "bar". Se fosse invece uno spazio, ad esempio, verrebbe applicata la prima mappatura ed il risultato sarebbe "foo", ed allora lo spazio viene inserito.

INOLTRE...

La parola chiave <script> può essere usata per effettuare una mappatura locale ad uno script. Vedere |**:map-<script>**|.

La parola chiave <buffer> può essere usata per effettuare una mappatura locale ad un buffer specifico. Vedere |**:map-<buffer>**|

La parola chiave <unique> può essere usata per ottenere che una mappatura fallisca quando essa già esistesse. Altrimenti una nuova mappatura sovrascriverebbe la vecchia. Vedere |**:map-<unique>**|.

Per far sì che un tasto venga disattivato, mappatelo con <Nop> (cinque caratteri). Ciò otterrà che il tasto <F7> venga del tutto disattivato: >

```
:map <F7> <Nop>| map! <F7> <Nop>
```

Non ci deve essere alcuno spazio dopo <Nop>.

==
40.2 Definizione di comandi da linea di comando

L'editor Vim vi consente di definire i vostri comandi. Eseguirete questi comandi proprio come ogni altro comando nel modo Command-line.
Per definire un comando usate l'istruzione ":command", come segue: >

```
:command DeleteFirst 1delete
```

Ora quando eseguite il comando ":DeleteFirst" Vim esegue ":1delete", che cancella la prima linea.

Nota:
I comandi definiti dall'utente debbono iniziare con una lettera maiuscola. Non potete usare ":X", ":Next" e ":Print". L'underscore ("_") non può essere usata! Potete usare i numeri, ma ciò viene sconsigliato.

Per elencare i comandi definiti dall'utente eseguite il comando che segue: >

```
:command
```

Come i comandi originali anche quelli definiti dall'utente possono essere abbreviati. Dovrete solo digitare quanto basta per distinguere un comando dall'altro. Il completamento da linea di comandi può essere usato per ottenere l'intero comando.

NUMERO DI ARGOMENTI

I comandi definiti dall'utente supportano diversi argomenti. Il numero di tali argomenti deve venir specificato nell'opzione -nargs. Supponiamo che il comando d'esempio :DeleteFirst non preveda argomenti, potreste definirlo come segue: >

```
:command -nargs=0 DeleteFirst 1delete
```

Comunque poiché il default è zero argomenti, non è necessario che aggiungiate "-nargs=0". Gli altri valori di -nargs sono come segue:

```
-nargs=0        Nessun argomento
-nargs=1        Un solo argomento
-nargs=*        Qualsiasi numero di argomenti
-nargs=?        Zero od uno argomenti
-nargs=+        Uno o più argomenti
```

USARE GLI ARGOMENTI

Nella definizione di un comando gli argomenti vengono rappresentati dalla parola chiave <args>. Ad esempio: >

```
:command -nargs=+ Say :echo "<args>"
```

Ora scrivendo >

```
:Say Hello World
```

Vim scriverà a schermo "Hello World". Comunque aggiungendo virgolette doppie non funzionerebbe. Ad esempio: >

```
:Say he said "hello"
```

Per inserire entro una stringa caratteri speciali, opportunamente protetti per usarli come un'espressione usate "<q-args>": >

```
:command -nargs=+ Say :echo <q-args>
```

Adesso il precedente comando ":Say" verrà correttamente eseguito: >

```
:echo "he said \"hello\""
```

La parola chiave <f-args> contiene la stessa informazione di <args>, fatta eccezione per un formato da usare come funzione per la chiamata di argomenti. Ad esempio: >

```
:command -nargs=* DoIt :call AFunction(<f-args>)
:DoIt a b c
```

Esegue il comando seguente: >

```
:call AFunction("a", "b", "c")
```

INTERVALLO DI LINEE

Qualche comando prevede un intervallo tra i propri argomenti. Per dire a Vim che state definendo un comando dovrete specificare l'opzione -range. I valori per questa opzione sono i seguenti:

-range	È ammesso un intervallo; il default è la linea attuale.
-range=%	È ammesso un intervallo; il default è l'intero file.
-range={count}	È ammesso un intervallo; l'ultimo numero in esso viene usato come un unico numero il cui default è {count}.

Quando un intervallo viene specificato le parole chiave <line1> e <line2> danno il numero della prima e dell'ultima linea dell'intervallo. Ad esempio, il comando seguente definisce il comando SaveIt, che salva l'intervallo di linee specificato entro il file "save_file": >

```
:command -range=% SaveIt :<line1>,<line2>write! save_file
```

ALTRE POSSIBILITA'

Alcune delle altre opzioni sono come segue:

-count={numero}	Il comando inizia a contare dal valore di default {numero}. Il valore risultante può venire usato per mezzo della parola chiave <count>.
-bang	Potete usare un !. Se presente, usando <bang> si otterrà un !.
-register	Potete specificare un registro. (Il default è il registro senza nome.) La specifica del registro è disponibile come <reg> (a.k.a. <register>).
-complete={tipo}	Tipo di completamento usato dalla linea di comando. Vedere |**:command-completion**| per l'elenco dei valori possibili.
-bar	Il comando può essere seguito da | ed un altro comando, oppure " ed un commento.
-buffer	Il comando è disponibile soltanto per il buffer corrente.

In ultimo c'è la parola chiave <lt>. Sta per il carattere <. Usatelo per evitare il significato speciale degli elementi menzionati <>.

RIDEFINIZIONE E CANCELLAZIONE

Per ridefinire lo stesso comando usate l'argomento !: >

```
:command -nargs=+ Say :echo "<args>"
:command! -nargs=+ Say :echo <q-args>
```

Per cancellare un comando utente impiegate ":delcommand". Supporta un solo argomento che è il nome del comando. Esempio: >

```
:delcommand SaveIt
```

Per cancellare tutti i comandi utente: >

```
:comclear
```

Attenzione, non si torna indietro!

Più particolari di ciò nel manuale di riferimento: |**user-commands**|.

==========

40.3 Autocomandi

Un autocomando è un comando che viene eseguito automaticamente in risposta ad un dato evento, come un file che venga letto o scritto od una modifica del buffer. Tramite l'impiego degli autocomandi potete allenare Vim a modificare dei file compressi, ad esempio. Ciò avviene nel plugin |**gzip**|.
Gli autocomandi sono molto potenti. Usateli con cura e vi aiuteranno, evitandovi di scrivere molti comandi. Usateli con noncuranza e vi daranno un sacco di grattacapi.

Immaginate di voler modificare la data alla fine di un file ogni volta che esso venga scritto. Prima definite una funzione: >

```
:function DateInsert()
:  $delete
:  read !date
:endfunction
```

Volete che questa funzione venga chiamata sempre, appena prima dopo che un buffer venga salvato su un file. Ciò farà sì che avvenga: >

```
:autocmd BufWritePre *  call DateInsert()
```

"BufWritePre" è l'evento tramite il quale viene fatto agire questo comando: Proprio prima ("Pre") di scrivere un buffer su un file. Il simbolo "*" è una espressione regolare che specifica il nome del file. In questo esempio, si applica a tutti i file.
Avendo abilitato questo comando, quando viene richiesto un ":write", Vim cerca qualsiasi autocomando che corrisponda con BufWritePre e lo esegue, e solo in seguito viene eseguito ":write".
La forma generale del comando :autocmd comando è quella che segue: >

```
:autocmd [gruppo] {eventi} {file_pattern} [nested] {comando}
```

Il nome [gruppo] è facoltativo. Viene usato per gestire e chiamare i comandi (maggiori particolari su di ciò più avanti). Il parametro {eventi} è un elenco di eventi (separati da una virgola) che innescano il comando.
{file_pattern} è il nome di un file, di solito con "wildcard" (metacaratteri). Ad esempio, usare "*.txt" fa sì che l'autocomando venga utilizzato con tutti i file il cui nome termina in ".txt". Il flag facoltativo [nested] consente l'annidamento degli autocomandi (vedere più avanti) e infine {comando} è il comando che dovrà essere eseguito.

EVENTI

Uno degli eventi più utili è BufReadPost. Viene fatto partire dopo che si sia creato un file nuovo. Viene comunemente usato per impostare valori di opzione. Ad esempio, sapete che i file "*.gsm" sono in linguaggio assembler GNU. Per ottenere il giusto file di sintassi, definite questo autocomando: >

```
:autocmd BufReadPost *.gsm  set filetype=asm
```

Se Vim riesce ad identificare il tipo di file, imposterà l'opzione 'filetype' al vostro posto. Ciò avvia l'evento Filetype. Impiegatelo per fare qualcosa quando aprite un certo tipo di file. Ad esempio, per caricare un elenco di abbreviazioni per file di testo: >

```
:autocmd Filetype text  source ~/.vim/abbrevs.vim
```

Aprendo un nuovo file potete fare inserire a Vim uno scheletro: >

```
:autocmd BufNewFile *.[ch]  0read ~/skeletons/skel.c
```

Vedere |**autocmd-events**| per un elenco completo degli eventi.

ESPRESSIONI

L'argomento {file_pattern} può essere proprio un elenco di tipi di file separati da una virgola. Ad esempio: "*.c,*.h" seleziona i file che terminano con ".c" ed ".h".
Potete usare i caratteri Jolly (wildcard) usate di solito per i file. Ecco un esempio di quelle usate più spesso:

*	Seleziona qualsiasi carattere per qualunque numero di volte esso appaia
?	Seleziona qualsiasi carattere una sola volta
[abc]	Seleziona il carattere a, b o c
.	Seleziona un punto
a{b,c}	Seleziona "ab" ed "ac"

Se la stringa di ricerca includesse una barra (/) Vim confronterebbe solo nomi di directory. Senza la barra soltanto l'ultima parte di un nome di file viene usata. Ad esempio, "*.txt" trova "/home/biep/readme.txt". Anche la stringa "/home/biep/*" lo selezionerebbe. Ma "home/foo/*.txt" non lo farebbe.
Includendo una barra, Vim ricerca al stringa sia entro il percorso completo del file ("/home/biep/readme.txt") che entro quello relativo (e.g., "biep/readme.txt").

Nota:
Se si lavora con un file system che impiega la barra rovesciata per separare i file, come MS-Windows, potrete impiegare barre diritte negli autocomandi. Ciò rende più facile scrivere la stringa di ricerca, poiché la barra rovesciata ha un significato speciale. Anche ciò rende portatili gli autocomandi.

CANCELLAZIONE

Per cancellare un autocomando usate lo stesso comando di quando lo avete definito, ma tralasciate il {comando} alla fine ed usate un !. Esempio: >

```
:autocmd! FileWritePre *
```

Ciò cancellerà tutti gli autocomandi per l'evento "FileWritePre" che impieghino l'elemento "*".

ELENCO

Per elencare tutti gli autocomandi attualmente definiti, usate questo: >

```
:autocmd
```

La lista può essere lunghissima, specialmente se si usa la determinazione del tipo di file. Per elencare solo parte dei comandi, specificate il gruppo, l'evento e/o l'elemento. Ad esempio, per elencare tutti gli autocomandi BufNewFile: >

```
:autocmd BufNewFile
```

Per elencare tutti gli autocomandi corrispondenti alla stringa di ricerca "*.c": >

```
:autocmd * *.c
```

Usando "*" per gli eventi elencherà tutti gli eventi. Per elencare tutti gli autocomandi per il gruppo dei programmi c: >

```
:autocmd cprograms
```

GRUPPI

L'elemento {gruppo}, che usate per definire un autocomando, raggruppa assieme i relativi autocomandi. Ciò può venir usato per cancellare tutti gli

autocomandi appartenenti ad un certo gruppo, ad esempio.
   Definendo molti autocomandi per un certo gruppo, usate il comando ":augroup". Ad esempio, definendo degli autocomandi per dei programmi C: >

```
:augroup cprograms
:  autocmd BufReadPost *.c,*.h :set sw=4 sts=4
:  autocmd BufReadPost *.cpp   :set sw=3 sts=3
:augroup END
```

Ciò farà lo stesso di: >

```
:autocmd cprograms BufReadPost *.c,*.h :set sw=4 sts=4
:autocmd cprograms BufReadPost *.cpp   :set sw=3 sts=3
```

Per cancellare tutti gli autocomandi del gruppo "cprograms": >

```
:autocmd! cprograms
```

ANNIDAMENTO

Di solito i comandi eseguiti come risultato di un evento di autocomando non generano a loro volta nuovi eventi. Se leggeste un file in conseguenza dell'evento FileChangedShell, esso non farà partire gli autocomandi che potessero impostare la sintassi, ad esempio. Per far sì che gli eventi li facciano partire aggiungete l'argomento "nested": >

```
:autocmd FileChangedShell * nested  edit
```

L'ESECUZIONE DEGLI AUTOCOMANDI

È possibile avviare un autocomando fingendo che un evento sia avvenuto. Ciò è utile per ottenere che un autocomando ne avvii un altro. Esempio: >

```
:autocmd BufReadPost *.new  execute "doautocmd BufReadPost " . expand("<afil
e>:r")
```

Ciò definisce un autocomando che si avvia quando viene creato un nuovo file. Il nome del file deve terminare in ".new". Il comando ":execute" utilizza la valutazione di espressione per creare un nuovo comando ed eseguirlo. Scrivendo il file "tryout.c.new" il comando eseguito sarà: >

```
:doautocmd BufReadPost tryout.c
```

La funzione expand() prende l'argomento "<afile>", che sta per il nome del file per cui era stato eseguito l'autocomando, e prende la radice del nome del file con ":r".

":doautocmd" viene eseguito sul buffer corrente. Il comando ":doautoall" lavora come "doautocmd" eccetto per il fatto che viene eseguito per tutti i buffers.

USARE COMANDI IN MODO NORMALE

I comandi eseguiti da un autocomando sono comandi a linea di comando. Se volete usare dei comandi in modalità Normal, potete usare il comando ":normal". Ad esempio: >

```
:autocmd BufReadPost *.log normal G
```

Ciò farà saltare il cursore sull'ultima linea dei file *.log quando iniziate a modificarli.
   Usare il comando ":normal" è un po' ingannevole. Prima di tutto accertatevi che il suo argomento sia un comando completo, includente tutti gli argomenti. Usando "i" per andare nell'Insert mode, ci deve essere anche un <Esc> per uscire ancora dall'Insert mode. Se usate una "/" per avviare la ricerca di una stringa, ci deve essere un <CR> per eseguirla.
   Il comando ":normal" utilizza tutto il testo che lo segue come comandi. Così non ci può essere alcun | ed un altro comando a seguirlo. Per lavorare con questo comando metterlo entro un comando ":execute". Ciò rende possibile anche di passare caratteri non stampabili in modo conveniente. Esempio: >

```
:autocmd BufReadPost *.chg execute "normal ONew entry:\<Esc>" |
        \ 1read !date
```

Ciò mostra anche l'utilizzo della barra rovescia per frazionare un lungo comando su più linee. Si può usare negli script di Vim (non dalla linea di comando).

Volendo far eseguire agli autocomandi qualcosa di complicato, che comporti di saltare in giro per il file e di tornare poi nella posizione originale, potreste voler ripristinare la vista sul file. Vedere |**restore-position**| per un esempio.

IGNORARE GLI EVENTI

A volte non vorrete far partire un autocomando. L'opzione 'eventignore' contiene un elenco di eventi che saranno totalmente ignorati. Ad esempio, quanto segue fa sì che vengano ignorati eventi di ingresso e di uscita da una finestra: >

```
:set eventignore=WinEnter,WinLeave
```

Per ignorare tutti gli eventi usate il comando che segue: >

```
:set eventignore=all
```

Per ripristinare l'aspetto normale lasciate vuoto 'eventignore': >

```
:set eventignore=
```

==

Capitolo seguente: |**usr_41.txt**| Preparare uno script Vim

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