help.txt   Per Vim version 6.2. Ultima modifica: 2004 Gen 08

                VIM - file di aiuto principale
              Traduzione di questo capitolo: Antonio Colombo

                                                                    k
        Movimenti: Usate le frecce, oppure "h" per sinistra,   h   l
                    "j" per giù, "k" per su, "l" per destra.   j
Chiudere finestra: Usate ":q<Invio>".
    Uscire da Vim: Usate ":qa!<Invio>" (attenti, le modifiche saranno perse!)

 Vedere argomento: Mettere il cursore su una tag tra |bars| e battere CTRL-].
       Col mouse: ":set mouse=a" per abilitare il mouse (in xterm o GUI).
                   Doppio-click col bottone di sinistra del mouse su una tag
                   posta tra |bars|.
 Tornare indietro: Battere CTRL-T o CTRL-O (più volte per andare ancora più
                   indietro).

  Aiuto specifico: Si può andare direttamente a qualsiasi argomento su cui
                   si voglia aiuto, fornendo un argomento al comando :help
                   |:help|.
                   E' possibile specificare ulteriormente il contesto:
                                                        help-context
                        COSA   PREFISSO   ESEMPIO   ~
                     Comandi modalità Normale   (nulla)   :help x
                     Comandi modalità Visuale   v_   :help v_u
                     Comandi modalità Inserim.   i_   :help i_<Esc>
                     Comandi su linea comando   :   :help :quit
                     Modifiche linea comando   c_   :help c_<Del>
                     Argomenti comando Vim   -   :help -r
                     Opzioni   '   :help 'textwidth'
 Ricerca di aiuto: Battere ":help word", e poi CTRL-D per vedere gli aiuti
                   disponibili per "word".

VIM sta per VI Migliorato. Vim è principalmente opera di Bram Moolenaar, ma
non sarebbe stato possibile senza l'aiuto di molti altri. Vedere |credits|.
----
                                                doc-file-list *Q_ct*
GUIDA BASE:

|quickref| Panoramica sui comandi di uso più frequente
|tutor| Corso di 30 minuti per cominciare
|copying| Copie e copyright
|iccf| Aiuta i bambini poveri dell'Uganda
|sponsor| Sponsorizza lo sviluppo di Vim, registrati come utente Vim
|www| Vim sul World Wide Web
|bugs| Dove spedire notifiche di errore

MANUALE UTENTE: Questi file spiegano come effettuare una sessione di edit.

|usr_toc.txt| Indice

Per iniziare ~
|usr_01.txt| Sui manuali
|usr_02.txt| I primi passi con Vim
|usr_03.txt| Muoversi nel file
|usr_04.txt| Fare piccole modifiche
|usr_05.txt| Configurazioni personali
|usr_06.txt| Usare l'evidenziazione della sintassi
|usr_07.txt| Elaborare più di un file
|usr_08.txt| Dividere le finestre
|usr_09.txt| Usare la GUI
|usr_10.txt| Fare grandi modifiche
|usr_11.txt| Recupero dopo un blocco
|usr_12.txt| Trucchi ingegnosi

Editare efficacemente ~
|usr_20.txt| Immissione rapida dei comandi sulla linea di comando
|usr_21.txt| Andarsene e ritornare
|usr_22.txt| Trovare il file da aprire
|usr_23.txt| Modifica di altri file
|usr_24.txt| Inserzione rapida
|usr_25.txt| Lavorare con testo formattato
|usr_26.txt| Ripetizione
|usr_27.txt| Comandi di ricerca e modelli
|usr_28.txt| La piegatura
|usr_29.txt| Spostarsi attraverso i programmi
|usr_30.txt| Editare programmi

|usr_31.txt| Sfruttare la GUI

Messa a punto di Vim ~
|usr_40.txt| Definire nuovi comandi
|usr_41.txt| Preparare uno script Vim
|usr_42.txt| Aggiungere nuovi menù
|usr_43.txt| Utilizzo dei tipi di file
|usr_44.txt| Evidenziazione della vostra sintassi
|usr_45.txt| Selezionate la vostra lingua

Far girare Vim ~
|usr_90.txt| Installare Vim

MANUALE DI RIFERIMENTO: Questi file spiegano Vim in maniera dettagliata.

Argomenti generali ~
|intro.txt| introduzione generale a Vim; notazione nei files di help
|help.txt| panoramica e referenza veloce (questo file)
|index.txt| indice alfabetico di tutti i comandi
|help-tags| tutte le tag alle quali potete saltare (indice tag)
|howto.txt| come fare i più comuni tipi di modifica
|tips.txt| vari suggerimenti sull'uso di Vim
|message.txt| messaggi (di errore) e spiegazioni relative
|quotes.txt| commenti degli utenti di Vim
|todo.txt| problemi conosciuti e future estensioni
|develop.txt| sviluppo di Vim
|uganda.txt| condizioni di distribuzione di Vim e come contribuire

Modifiche base a un testo ~
|starting.txt| eseguire Vim, argomenti di Vim, inizializzazione
|editing.txt| modificare e scrivere file
|motion.txt| comandi per muoversi nel file
|scroll.txt| far scorrere il testo nella finestra (scrolling)
|insert.txt| modalità Inserimento e modalità Sostituzione
|change.txt| cancellare e sostituire del testo
|indent.txt| indentazione automatica per il C e altri linguaggi
|undo.txt| Undo e Redo
|repeat.txt| comandi ripetuti, script Vim e debug di Vim
|visual.txt| usare la modalità Visuale (selezione area di testo)
|various.txt| vari comandi ulteriori
|recover.txt| ripartenza dopo una sessione finita male

Modifiche avanzate a un testo ~
|cmdline.txt| modificare usando comandi dalla linea di comando
|options.txt| descrizione di tutte le opzioni
|pattern.txt| modelli di espressione regolari e comandi di ricerca
|map.txt| mappatura tasti e abbreviazione
|tagsrch.txt| tag e ricerche speciali
|quickfix.txt| comandi per un ciclo veloce modifica-compila-correggi
|windows.txt| comandi per avere più finestre e più buffer (file)
|syntax.txt| evidenziazione sintattica
|diff.txt| lavorare con due o tre versioni dello stesso file
|autocmd.txt| eseguire comandi automaticamente a fronte di eventi
|filetype.txt| impostazioni specifiche per un tipo di file
|eval.txt| valutazione di espressioni, comandi condizionali
|fold.txt| nascondere (piegare) linee nel testo

Modifiche speciali ~
|remote.txt| uso di Vim come "server" o "client"
|term.txt| usare terminali e mouse differenti
|digraph.txt| lista di digrammi inseribili (caratteri non in tastiera)
|mbyte.txt| supporto testi che richiedono supporto multi-byte
|mlang.txt| supporto linguaggi diversi dall'inglese
|arabic.txt| supporto modifiche in lingua araba
|farsi.txt| supporto modifiche in lingua farsi (persiano)
|hebrew.txt| supporto modifiche in lingua ebraica
|hangulin.txt| modalità d'immissione in hangul (coreano)
|rileft.txt| modalità modifica lingue scritte da destra a sinistra

GUI ~
|gui.txt| Graphical User Interface (GUI)
|gui_w16.txt| GUI Windows 3.1
|gui_w32.txt| GUI Win32
|gui_x11.txt| GUI X11

Interfacce ~
|if_cscop.txt| uso di cscope con Vim

|if_perl.txt| interfaccia Perl
|if_pyth.txt| interfaccia Python
|if_sniff.txt| interfaccia SNiFF+
|if_tcl.txt| interfaccia Tcl
|if_ole.txt| interfaccia per Win32 di OLE automation
|if_ruby.txt| interfaccia Ruby
|debugger.txt| interfaccia con un programma di debug
|workshop.txt| interfaccia con Sun Visual Workshop
|netbeans.txt| interfaccia con NetBeans External Editor
|sign.txt| segni di aiuto per un programma di debug

Versioni ~
|vi_diff.txt| differenze principali tra Vim e Vi
|version4.txt| differenze fra Vim version 3.0 e 4.x
|version5.txt| differenze fra Vim version 4.6 e 5.x
|version6.txt| differenze fra Vim version 5.7 e 6.x
                                                sys-file-list
Osservazioni su vari sistemi operativi ~
|os_390.txt| OS/390 Unix
|os_amiga.txt| Amiga
|os_beos.txt| BeOS e BeBox
|os_dos.txt| MS-DOS e MS-Windows NT/95 - caratteristiche comuni
|os_mac.txt| Macintosh
|os_mint.txt| Atari MiNT
|os_msdos.txt| MS-DOS (modalità DOS e finestra DOS sotto Windows)
|os_os2.txt| OS/2
|os_qnx.txt| QNX
|os_risc.txt| RISC-OS
|os_unix.txt| Unix
|os_vms.txt| VMS
|os_win32.txt| MS-Windows 95/98/NT

Plugin Standard   *standard-plugin-list* ~
|pi_netrw.txt| leggere e scrivere file in rete
|pi_gzip.txt| leggere e scrivere file compressi
|pi_expl.txt| esplora file

AGGIUNTE LOCALI: *local-additions*
|latexhelp.txt| For Vim version 6.0. Last change: 2001 Dec 20
|matchit.txt| Extended "%" matching
|vimspell.txt| On the fly spell checker with ispell/aspell.   v1.84

----
bars   Esempio di testo fra barre

Ora che siete arrivati qui con CTRL-] o un doppio click col mouse, potete
tornare indetro dove eravate prima con CTRL-T, CTRL-O, g<TastoDestro>,
oppure <C-TastoDestro>.
----
 vim:tw=78:fo=tcq2:isk=!-~,^*,^\|,^\":ts=8:ft=help:norl:

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

usr_toc.txt   Per Vim version 6.2. Ultima modifica: 2003 Ago 18

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Bartolomeo Ravera

Indice *user-manual*

==
Panoramica ~

Il manuale utente inglese è disponibile in un unico file HTML e PDF,
pronto per la stampa, a questo indirizzo:
        http://vimdoc.sf.net

==
Per iniziare ~

Leggetelo dall'inizio alla fine per apprendere i comandi essenziali.

==
Editare efficacemente ~

Capitoli che possono essere letti indipendentemente.

==
Messa a punto di Vim ~

Fate funzionare Vim come volete voi.

==
Far girare Vim ~

Prima che possiate usare Vim.

==

La versione italiana del "Vim User Manual" è stata tradotta
da un gruppo di volontari coordinati da Bartolomeo Ravera.

Grazie agli amici di Zena, Pluto e SannioLug.

Un ringraziamento particolare va ad Antonio Colombo e
all'infaticabile Giuliano Bordonaro.

Se avete commenti, osservazioni o errori da segnalare,
mandate un messaggio email a: barrav@libero.it

usr_01.txt   Per Vim version 6.2. Ultima modifica: 2003 Gen 12

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Giuliano Bordonaro

Sui manuali

Questo capitolo introduce i manuali disponibili con Vim. Leggetelo per sapere come i comandi verranno spiegati.

==

01.1 Due manuali

La documentazione di Vim consiste in due parti:

1. Il manuale utente
   Spiegazioni orientate alle operazioni, da semplici a complesse. Da leggere dall'inizio alla fine come un libro.

2. Il manuale di riferimento
   Descrizione accurata di come funzioni ogni cosa in Vim.

La notazione usata in questi manuali viene spiegata qui: |notation|

SALTANDO QUA E LA'

Il testo contiene iperlinks tra le due parti, che vi consentono di saltare velocemente tra la descrizione di un'operazione di modifica di un file ed una precisa spiegazione dei comandi ed opzioni usate per compierla. Usate questi due comandi:

        Premete CTRL-] per saltare ad un oggetto sotto il cursore.
        Premete CTRL-O per tornare indietro (ripetuta va ancora indietro).

Molti links sono racchiusi tra barre verticali, come questo: |bars|. Un nome di opzione, come 'number', un comando racchiuso entro virgolette doppie come ":write" ed ogni altra parola può venire usata come link. Provate: Portate il cursore su CTRL-] e premete CTRL-] su di esso.

Potete trovare altri argomenti con il comando ":help", vedere |help.txt|.

==

01.2 A Vim installato

Ogni manuale da per scontato che Vim sia stato installato correttamente. Se il vostro non lo fosse fatelo adesso, o se Vim non girasse correttamente (ad es., non si trovassero i file o non venisse mostrato il menù nella GUI) prima leggete il capitolo sull'installazione: |usr_90.txt|.

not-compatible

I manuali spesso presumono che stiate utilizzando Vim con la Vi-compatibility disattivata. Per molti comandi ciò non è un problema, ma talvolta è importante, ad es., per l'undo multilivello. Un modo semplice per accertarsi che si stia usando una corretta messa a punto, è quello di copiare il file di esempio di vimrc. Per fare ciò entro Vim non avrete bisogno di cercare ove esso sia collocato.
   Come farlo dipende dal sistema che state usando:

Unix: >
        :!cp -i $VIMRUNTIME/vimrc_example.vim ~/.vimrc
MS-DOS, MS-Windows, OS/2: >
        :!copy $VIMRUNTIME/vimrc_example.vim $VIM/_vimrc
Amiga: >
        :!copy $VIMRUNTIME/vimrc_example.vim $VIM/.vimrc

Se il file esiste probabilmente dovreste trovarlo.

Adesso avviando Vim, l'opzione 'compatible' dovrebbe essere disattivata.
Potete verificarlo con questo comando: >

```
        :set compatible?
```

Se viene risposto con "nocompatible" state andando bene. Se la risposta è "compatible" avete un problema. Dovrete riuscire a capire perchè l'opzione sia ancora attiva. Vim potrebbe non trovare il file che avete scritto. Usate questo comando per trovarlo: **>**

```
        :scriptnames
```

Se il vostro file non viene elencato, andate a vedere la sua collocazione ed il suo nome. Se fosse nella lista, dovrebbe esistere un altro posto ove l'opzione 'compatible' viene nuovamente attivata.

Per ulteriori informazioni vedere |vimrc| e |compatible-default|.

        Note:
        Questo manuale descrive l'uso normale di Vim. Esiste un'alternativa
        chiamata "evim" (easy Vim). E' ancora Vim, ma si usa in modo da farlo
        rassomigliare ad un editor clicca-e-scrivi come Notepad. Rimane
        sempre in Insert mode, così appare molto differente. Non viene
        spiegato nel manuale utente, poichè dovrebbe essere autoesplicante.
        Vedere |evim-keys| per i particolari.

==
01.3 Usare il tutor di Vim   *tutor* *vimtutor*

Invece di leggere il testo potete usare il vimtutor per apprendere i vostri primi comandi di Vim. E' un corso di trenta minuti che insegna praticamente le funzionalità del Vim di base.

Su Unix e MS-Windows, se Vim è stato installato correttamente, potete avviarlo dalla shell:
>

```
        vimtutor
```

Questo farà una copia del file tutor, in modo che lo possiate modificare senza correre il rischio di danneggiare l'originale.
   Ci sono poche versioni tradotte del tutor. Per scoprire se la vostra esiste, usate il codice a due lettere dei linguaggi. Per il francese:

```
        vimtutor fr
```

Per OpenVMS, se Vim è stato correttamente installato, potete far partire vimtutor dal prompt di VMS con:

```
        @VIM:vimtutor
```

Volendo si può aggiungere il codice a due lettere dei linguaggi, come visto sopra.

Su altri sistemi dovrete fare ancora un lavoretto:

1. Copiare il file del tutor. Potete farlo con Vim (sa come farlo):
>

```
        vim -u NONE -c 'e $VIMRUNTIME/tutor/tutor' -c 'w! TUTORCOPY' -c 'q'
```

<
   Ciò scriverà il file "TUTORCOPY" nella directory corrente. Per usare una versione tradotta del tutor, aggiungete al nome del file il codice a due lettere dei linguaggi. Per il francese:
>

```
        vim -u NONE -c 'e $VIMRUNTIME/tutor/tutor.fr' -c 'w! TUTORCOPY' -c 'q'
```

<
2. Modificate il file copiato con Vim:
>

```
        vim -u NONE -c "set nocp" TUTORCOPY
```

<
   Gli argomenti extra garantiscono che Vim parta di buon umore.

3. Cancellate il file copiato dopo avere finito con esso.
>

```
        del TUTORCOPY
```

<
==
01.4 Copyright   *manual-copyright*

Il manuale utente di Vim ed il manuale di riferimento sono protetti da

frombook

Parti del manuale utente provengono dal libro "Vi IMproved - Vim" di Steve Oualline (pubblicato a cura di New Riders Publishing, ISBN: 0735710015). Questo libro è soggetto alla Open Publication License. Solo determinate parti sono state incluse e queste sono state modificate (ad. es., rimuovendo le figure, aggiornando il testo per Vim 6.0 e correggendo gli errori). L'omissione del tag |frombook| non significa che il testo non provenga dal libro.

Grazie a Steve Oualline ed ai New Riders per avere creato e pubblicato questo libro sotto la OPL! E' stato un grande aiuto per scrivere il manuale utente.
   Non soltanto per avere fornito il testo, ma anche per avere impostato tono e stile.

Se guadagnerete vendendo i manuali, siete fortemente invitati a donare parte del profitto per aiutare le vittime dell'AIDS in Uganda. Vedere |iccf|.

==

Capitolo seguente: |usr_02.txt| I primi passi con Vim

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

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

usr_02.txt   Per Vim version 6.2. Ultima modifica: 2003 Mag 04

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Giuliano Bordonaro

I primi passi con Vim

Questo capitolo fornisce qualche sommaria informazione per scrivere un file con Vim. Non bene od alla svelta, ma potete scrivere. Dedicate un po' di tempo ad impratichirvi con questi comandi, essi sono la base per quanto segue.

  Capitolo seguente: |usr_03.txt| Muoversi nel file
Capitolo precedente: |usr_01.txt| Sui manuali
             Indice: |usr_toc.txt|

==

02.1 Avviare Vim la prima volta

Per avviare Vim, usate questo comando: **>**

```
gvim file.txt
```

In UNIX potete scriverlo in ogni prompt di comando. Se usaste Microsoft Windows, aprite una finestra MS-DOS e digitate il comando.
   In ogni caso, Vim viene avviato aprendo un file chiamato file.txt. Se questo fosse un file nuovo otterreste una finestra vuota. La vostra schermata apparirebbe così:

```
+----------------------------------------+
|#                                       |
|~                                       |
|~                                       |
|~                                       |
|~                                       |
|"file.txt" [New file]                   |
+----------------------------------------+
        ('#" è la posizione del cursore.)
```

Le linee che iniziano con una tilde (~) indicano di non far parte del file.
   In altre parole, quando Vim va oltre la fine del file mostra le linee con la tilde. In fondo allo schermo una linea di messaggio informa che il file si chiama file.txt ed indica che voi state creando un file nuovo. Il messaggio informativo è temporaneo ed informazioni successive lo sovrascrivono.

IL COMANDO VIM

Il comando gvim fa sì che l'editor apra una nuova finestra in cui scrivere.

Se invece usate questo comando: **>**

```
vim file.txt
```

lavorerete entro la vostra finestra di comando. In altre parole, se lavorate entro un xterm, l'editor impiegherà la vostra finestra di xterm. Se state usando una finestra di comando MS-DOS sotto Microsoft Windows, lavorerete entro questa finestra. Il testo entro la finestra sarà identico in entrambe le versioni, ma con gvim vi sono altre e maggiori possibilità, come una barra di menu. Maggiori informazioni più avanti.

==

02.2 Inserire del testo

L'editor Vim è un editor modale. Ciò significa che si comporterà diversamente a seconda del modo in cui vi trovate. I due modi principali si chiamano Normal mode ed Insert mode. In Normal mode i caratteri che scrivete sono comandi. In Insert mode gli stessi caratteri vengono inseriti come testo.
   Vim viene avviato in Normal mode. Per passare all'Insert mode inserite il

comando "i" (i sta per Insert). Poi potrete scrivere del testo. Esso verrà inserito entro il file. Non preoccupatevi di aver commesso degli errori; potrete correggerli dopo. Per scrivere la seguente canzoncina del programmatore, dovete digitare quanto segue: **>**

```
iA very intelligent turtle
Found programming UNIX a hurdle
```

Dopo aver scritto "turtle" premete il tasto <Enter> per iniziare una nuova linea. In ultimo premete il tasto <Esc> per uscire dall'Insert mode e tornare al Normal mode. Ora ci saranno due linee di testo nella vostra finestra di Vim:

```
+----------------------------------------+
|A very intelligent turtle               |
|Found programming UNIX a hurdle         |
|~                                       |
|~                                       |
|                                        |
+----------------------------------------+
```

QUAL'E' IL MODO?

Per sapere in quale modo vi trovate, scrivete questo comando: **>**

```
:set showmode
```

Potete notare che scrivendo il carattere due punti Vim sposta il cursore nell'ultima linea della finestra. In questa linea potete digitare i comandi due punti (comandi che iniziano con il carattere due punti). Il comando viene concluso premendo il tasto <Enter> (tutti i comandi iniziati con i due punti vengono conclusi così).
Adesso, se scrivete il comando "i", Vim farà apparire la scritta --INSERT-- alla base della finestra. Ciò indicherà che vi trovate in Insert mode.

```
+----------------------------------------+
|A very intelligent turtle               |
|Found programming UNIX a hurdle         |
|~                                       |
|~                                       |
|-- INSERT --                            |
+----------------------------------------+
```

Premendo <Esc> per tornare al Normal mode l'ultima linea tornerà vuota.

SUPERARE I PROBLEMI

Uno dei problemi per il principiante di Vim è la confusione dei modi, che può avvenire dimenticando in quale modo ci si trovi o scrivendo accidentalmente un comando che cambia modo. Per tornare nel Normal mode non c'è problema, qualunque sia il modo in cui vi troviate premete il tasto <Esc>. Se lo premete due volte Vim vi avvertirà con un suono che siete già nel Normal mode.

==========

02.3 Spostarsi attraverso il file

Dopo il vostro ritorno nel Normal mode, vi potete spostare usando questi tasti:

```
h   sinistra                                              *hjkl*
j   giù
k   su
l   destra
```

A prima vista potrebbe apparire che questi comandi siano stati scelti a casaccio. Dopo tutto, chi mai ha usato l per dire destra? Ma in realtà c'è una ragione molto valida alla base di queste scelte: lo spostamento del cursore è una delle cose più frequenti in un editor, e questi tasti si trovano in basso a destra nella tastiera. In altre parole questi comandi si trovano dove potete scriverli più velocemente (specialmente se scrivete con dieci dita).

Note:
Potete anche spostare il cursore usando i tasti freccia. Se lo fate, comunque, dovrete rallentare la vostra velocità di battitura per

premerli, dovendo muovere la mano dai tasti testuali a quelli freccia.
Considerando che dovrete farlo centinaia di volte all'ora, ciò
significa sprecare una considerevole quantità di tempo.
   Inoltre ci sono tastiere che non hanno i tasti freccia, o che li
hanno in posizioni insolite; così, conoscere l'uso dei tasti hjkl,
aiuta in queste situazioni.

Un modo per ricordarsi di questi comandi è che h si trova a sinistra, l è a
destra e j punta all'ingiù. Visualmente: **>**

```
                k
          h     l
            j
```

Il modo migliore di imparare questi comandi è di usarli. Con il comando "i"
inserite alcune linee di testo. Poi provate a spostarvi con i tasti hjkl ed
ad inserire qualche parola qua e là. Non scordate di premere <Esc> per
tornare al Normal mode. Il |vimtutor| è un altro modo piacevole per
imparare con la pratica.

Per utenti giapponesi, Hiroshi Iwatani suggerisce di fare così:

```
                    Komsomolsk
                        ^
                        |
   Huan Ho          <--- --->  Los Angeles
(Fiume giallo)          |
                        v
                  Java (l'isola, non il linguaggio)
```

==
02.4 La cancellazione di caratteri

Per cancellare un carattere, spostate il cursore su di esso e premete "x".
(Questo è un retaggio dei vecchi giorni della macchina per scrivere, quando si
cancellavano cose scrivendovi sopra xxxx.) Spostando il cursore all'inizio
della prima riga della canzoncina e scrivendo xxxxxxx (sette x) si cancellerà
"A very ". Il risultato dovrebbe apparire così:

```
+----------------------------------------+
|intelligent turtle                      |
|Found programming UNIX a hurdle         |
|~                                       |
|~                                       |
|                                        |
+----------------------------------------+
```

Adesso si può inserire del testo nuovo, ad esempio scrivendo: **>**

```
iA young <Esc>
```

Ciò inizia un'inserzione (la i), scrive le parole "A young", ed esce
dall'Insert mode (l'<Esc> finale). Il risultato:

```
+----------------------------------------+
|A young intelligent turtle              |
|Found programming UNIX a hurdle         |
|~                                       |
|~                                       |
|                                        |
+----------------------------------------+
```

CANCELLAZIONE DI UN'INTERA LINEA

Per cancellare una linea usate il comando "dd". La linea che segue si
sposterà verso l'alto a riempire il vuoto:

```
+----------------------------------------+
|Found programming UNIX a hurdle         |
|~                                       |
|~                                       |
|~                                       |
|                                        |
+----------------------------------------+
```

CANCELLAZIONE DI UN "A CAPO"

In Vim potete unire assieme due linee per farle diventare una sola, ciò significa che l'interruzione di linea tra di esse è stata cancellata. Il comando "J" fa ciò.
   Prendiamo queste due linee:

```
        A young intelligent ~
        turtle ~
```

Portate il cursore sulla prima linea e premete "J":

```
        A young intelligent turtle ~
```

==========

02.5 Undo e Redo

Supponiamo che abbiate cancellato più del dovuto. Bene, potete riscriverlo da capo, ma c'è un modo più semplice. Il comando "u" elimina l'ultima modifica. Osservate questa azione: dopo aver usato "dd" per cancellare la prima linea , "u" la riporta a come era originariamente.
   Ancora una: Portate il cursore sulla A nella prima linea:

```
        A young intelligent turtle ~
```

Ora scrivete xxxxxxx per cancellare "A young". Rimarrà quanto segue:

```
         intelligent turtle ~
```

Scrivete "u" per eliminare l'ultima cancellazione. Poichè delete aveva rimosso la g, undo ripristina il carattere.

```
        g intelligent turtle ~
```

Un ulteriore comando u ripristina il precedente carattere cancellato:

```
        ng intelligent turtle ~
```

Il prossimo comando u darà la u, e così via:

```
        ung intelligent turtle ~
        oung intelligent turtle ~
        young intelligent turtle ~
         young intelligent turtle ~
        A young intelligent turtle ~
```

        Note:
        Se premete "u" due volte, ed il risultato è che ottenete lo stesso testo, avete Vim configurato per lavorare in modo compatibile Vi. Andate a vedere qui per correggerlo: |not-compatible|.
           Questo manuale presume che stiate lavorando in "Modo Vim". Potreste preferire l'utilizzo del vecchio modo Vi, ma dovrete aspettarvi alcune piccole differenze nel testo in quel caso.

REDO

Se avete impiegato il comando u troppe volte, potete premere CTRL-R (redo) per invertire il comando precedente. In altre parole, ciò cancella la cancellazione. Per vederlo in azione premete CTRL-R due volte. Il carattere A e lo spazio dopo di esso spariranno:

```
        young intelligent turtle ~
```

C'è una versione speciale del comando undo, il comando "U" (undo line). Il comando undo line ripristina tutte le modifiche effettuate sull'ultima linea su cui avevate lavorato. Scrivendo questo comando due volte si cancellerà il precedente "U".

```
        A very intelligent turtle ~
          xxxx                          Cancella very

        A intelligent turtle ~
                      xxxxxx            Cancella turtle

        A intelligent ~
                                        Ripristina la linea con "U"
        A very intelligent turtle ~
                                        Cancella "U" usando "u"
```

        A intelligent ~

Il comando "U" modifica da solo, quello che il comando "u" cancella e CTRL-R rifà. Ciò potrebbe essere un tantino confuso. Non preoccupatevi, con un "u" e CTRL-R potrete ripristinare qualunque situazione aveste.

==
02.6 Altri comandi

Vim possiede un gran numero di comandi per modificare il testo. Guardate |Q_in| ed oltre.
Ve ne sono alcuni usati di rado.

L'APPENDING

Il comando "i" inserisce un carattere prima del carattere sotto il cursore.
   Lavora bene; ma cosa succede se volete inserirlo alla fine della linea?
Per fare ciò dovete inserire il testo dopo il cursore. Ciò si ottiene con il comando "a" (append).
   Ad esempio, per cambiare la linea

        and that's not saying much for the turtle. ~
in
        and that's not saying much for the turtle!!! ~

spostate il cursore sul punto alla fine della linea. Poi scrivete "x" per cancellare il punto. Il cursore è ora posizionato alla fine della linea sulla e di turtle. Adesso scrivete

        a!!!<Esc>

per aggiungere i tre punti esclamativi dopo la e in turtle:

        and that's not saying much for the turtle!!! ~

INSERIRE UNA NUOVA LINEA

Il comando "o" crea una nuova linea vuota sotto il cursore e pone Vim nell'Insert mode. Ora potete scrivere il testo per la nuova linea.
   Supponiamo che il cursore sia da qualche parte nella prima di queste due linee:

        A very intelligent turtle ~
        Found programming UNIX a hurdle ~

Se adesso usate il comando "o" e scrivete il testo:

        oThat liked using Vim<Esc>

Il risultato sarà:

        A very intelligent turtle ~
        That liked using Vim ~
        Found programming UNIX a hurdle ~

Il comando "O" (maiuscolo) apre una linea sopra il cursore.

USARE IL NUMERO DI RIPETIZIONI

Immaginiamo di voler salire di nove linee. Potete scrivere "kkkkkkkkk" od usare il comando "9k". Difatti, si possono far precedere molti comandi con un numero.
Prima in questo capitolo, ad esempio avete aggiunto tre punti esclamativi alla fine di una linea scrivendo "a!!!<Esc>". Un altro modo per farlo è di usare il comando "3a!<Esc>". Il numero 3 dice al comando che segue di venire eseguito tre volte. Analogamente per cancellare tre caratteri potete usare il comando "3x". Il numero deve venire prima del comando che deve essere eseguito.

==
02.7 Come uscire

Per uscire usate il comando "ZZ". Questo comando scrive il file ed esce.

        Note:

Diversamente da molti altri editor, Vim non farà automaticamente un file di backup. Se scrivete "ZZ", le vostre modifiche verranno sovrascritte e non potrete più tornare indietro. Potete configurare Vim per generare un file di backup, vedete |07.4|.

ABBANDONARE LE MODIFICHE

Sovente farete un sacco di modifiche per poi capire improvvisamente di aver fatto qualcosa di diverso da quanto volevate. Nulla di preoccupante; Vim ha un comando esci-e-getta-via-tutto. Si tratta di:

```
:q!
```

Non scordatevi di premere <Enter> per ultimare il comando.

Per coloro che fossero interessati ai particolari, le tre parti di questo comando sono i due punti (:), che fa entrare in modo Command-line; il comando q che effettua la chiusura dell'editor; ed il modificatore di comando ignora (!).
Il comando ignora è necessario poichè Vim non vorrebbe ignorare le modifiche. Se scriveste soltanto ":q", Vim mostrerebbe un messaggio di errore e rifiuterebbe di uscire:

```
E37: No write since last change (use ! to override) ~
```

Specificando di ignorare, state effettivamente dicendo a Vim, "Lo so che ciò che sto facendo sembra stupido, ma io sono adulto e voglio davvero fare questo."

Se voleste continuare a lavorare sul file con Vim: il comando ":e!" ricaricherebbe la versione originale del file.

==
02.8 Trovare un aiuto

Tutto ciò che vorreste sapere può essere trovato nei files di help di Vim. Non esitate a chiedere!
Per avere un aiuto generico usate questo comando:

```
:help
```

Si può usare il primo tasto di funzione <F1>. Se la vostra tastiera avesse un tasto <Help> questo potrebbe funzionare altrettanto bene.
Se non gli fornite un argomento, ":help" mostra la finestra generica di aiuto. I creatori di Vim hanno fatto qualcosa di molto astuto (o pigro) con il sistema di help: hanno fatto la finestra di help come una normale finestra di editing. Potete usare tutti i comandi normali di Vim per navigare attraverso le informazioni di help. Comunque h, j, k, e l operano uno spostamento del cursore verso sinistra, giù, su e destra.
Per uscire dalla finestra di help, usate lo stesso comando che usereste per uscire dall'editor: "ZZ". Si chiuderà solo la finestra di help, non Vim.

Leggendo il testo di aiuto, noterete del testo racchiuso entro barre verticali (ad esempio, |help|). Ciò indica un iperlink. Se posizionate il cursore tra le barre e premete CTRL-] (salta al tag), il sistema di help vi darà l'oggetto indicato. (Per ragioni non discusse qui, la terminologia di Vim per un iperlink è tag. Così CTRL-] salta alla posizione del tag indicato dalla parola sotto il cursore.)
Dopo pochi passi, potreste voler tornare indietro. CTRL-T (pop tag) ritorna alla posizione precedente. Lavora bene anche CTRL-O (salta alla posizione più vecchia).
Alla sommità dello schermo di help, c'è la nota *help.txt*. Questo nome tra caratteri "*" viene usata dal sistema di help per definire un tag (destinazione dell'iperlink).
Vedere |29.1| circa i particolari per l'uso dei tag.

Per avere aiuto su un oggetto dato, usate il comando seguente:

```
:help {subject}
```

Per ottenere aiuto sul comando "x", ad esempio, scrivete quanto segue:

```
:help x
```

Per scoprire come cancellare del testo, usate questo comando:

```
:help deleting
```

Per avere un completo indice dei comandi di Vim, usate il comando seguente:

```
:help index
```

Se vi servisse aiuto per un comando a carattere di controllo (ad esempio, CTRL-A), dovete specificarlo con il prefisso "CTRL-".

```
:help CTRL-A
```

L'editor Vim ha molte modalità diverse. Di default il sistema di help mostra i comandi in Normal-mode. Ad esempio, il comando che segue mostra un aiuto per il comando CTRL-H in Normal mode: >

```
:help CTRL-H
```

Per identificare gli altri modi, utilizzate un prefisso di modo. Se volete un aiuto per la versione Insert-mode di un comando, usate "i_". Per CTRL-H ciò vi fornisce il comando che segue:

```
:help i_CTRL-H
```

Avviando l'editor Vim, potete usare molti argomenti a linea di comando.
Tutti questi iniziano con un trattino (-). Per trovare cosa faccia l'argomento -t, per esempio, usate il comando:

```
:help -t
```

L'editor Vim possiede un certo numero di opzioni che vi consentono di configurare e personalizzare l'editor. Se voleste un aiuto per qualche opzione, dovete racchiuderla tra due virgolette singole. Per ottenere informazioni su cosa faccia l'opzione 'number', ad esempio, usate il seguente comando:

```
:help 'number'
```

La tabella con i prefissi per tutti i modi si può trovare qui: |help-context|.

I tasti speciali sono racchiusi tra parentesi angolate. Per ottenere aiuto sul tasto freccia in su nell'Insert mode, per esempio, usate questo comando: >

```
:help i_<Up>
```

Se vedeste un messaggio di errore che non capite, ad esempio:

```
E37: No write since last change (use ! to override) ~
```

Potete usare l'ID di errore per iniziare a trovare informazioni circa esso: >

```
:help E37
```

==

Capitolo seguente: |usr_03.txt| Muoversi nel file

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

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

usr_03.txt   Per Vim version 6.2. Ultima modifica: 2004 Gen 17

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Bartolomeo Ravera

Muoversi nel file

Prima di inserire o cancellare del testo, il cursore deve essere portato sul punto giusto. Vim ha un grande numero di comandi per posizionare il cursore.
   Questo capitolo vi mostrerà come usare quelli principali. Potete trovare una lista di questi comandi in |Q_lr|.

  Capitolo seguente: |usr_04.txt| Fare piccole modifiche
Capitolo precedente: |usr_02.txt| I primi passi con Vim
              Indice: |usr_toc.txt|

==
03.1 Movimenti di parola

Per muovere il cursore avanti di una parola, usate il comando "w". Come molti comandi di Vim, potete usare un prefisso numerico per muovervi di più parole.
   Per esempio, "3w" muove avanti di tre parole. Questa figura mostra come questo accada:

```
        Questa è una linea di esempio~
          ----->->------------>
             w   w     3w
```

Notate che se il cursore è già all'inizio di una parola, "w" muove all'inizio della parola seguente.
   Il comando "b" muove indietro all'inizio della parola precedente:

```
        Questa è una linea di esempio ~
        <------<-<---------<--<---
            b   b     2b     b  b
```

C'è anche il comando "e" che muove alla fine della parola seguente, e "ge" che muove alla fine della parola precedente:

```
        Questa è una linea di esempio ~
             <-    <---- --->   ---->
             ge      ge   e      e
```

Se siete posizionati sull'ultima parola di una linea, il comando "w" vi porterà alla prima parola della riga seguente. Potete usare "w" per spostarvi all'interno di un paragrafo molto più rapidamente che usando "l". "b" si comporta allo stesso modo, ma in direzione opposta.

Una parola termina al primo carattere che non ricorre in una parola, ad esempio ".", "-" oppure ")". Per modificare la lista dei caratteri che fanno parte di una parola, si veda l'opzione 'iskeyword'.
   E' anche possibile muoversi fra PAROLE (WORDS) separate da spazi bianchi. Queste non sono parole nel senso di cui prima. In questo caso si usano le lettere maiuscole. Guardate il seguente esempio:

```
             ge       b        w                                  e
           <---      <-       --->                             ----->
    Questa è-una linea, con parole/separate/stranamente (e altro ancora). ~
         <------ <------       ------------------------->         ------>
            gE       B                   W                         E
```

Con questi vari comandi minuscoli e maiuscoli, potete muovervi rapidamente avanti e indietro all'interno di un paragrafo.

==
03.2 Spostarsi all'inizio o alla fine di una riga

Il comando "$" sposta il cursore alla fine della linea. Se la vostra tastiera ha un tasto <Fine>, premendolo otterrete lo stesso effetto.

Il comando "^" muove il cursore al primo carattere non-blank della linea. Il comando "0" (zero) muove esattamente al primo carattere della linea. Il tasto <Home> fa la stessa cosa. In una immagine:

```
                ^
           <---------
     .....Questa è una linea di esempio~
     <--------------      ------------>
            0                $
```

(i puntini "....." in questo esempio indicano degli spazi)

   Il comando "$" può essere preceduto da un numero, come molti comandi di movimento. Poiché muoversi più volte alla fine della linea non ha senso, l'editor muoverà il cursore alla fine di un'altra linea. Per esempio, "1$" vi sposterà alla fine della prima linea, (quella dove siete), "2$" alla fine della linea seguente, e così via.
   Il comando "0" non può essere preceduto da alcun numero, perché lo "0" farebbe parte del numero. Contrariamente alle aspettative, l'uso di un numero davanti a "^" non ha alcun effetto.

==
03.3 Spostarsi verso un carattere

Uno dei più utili comandi di movimento è il comando di ricerca di un singolo carattere. Il comando "fx" cerca avanti nella linea la prima occorrenza del carattere x. Suggerimento: "f" sta per "Find", ovvero "Cerca" in inglese.
   Supponiamo che nel seguente esempio vogliate andare verso la u della parola umano. Eseguendo il comando "fu" il cursore sarà posizionato sopra la u:

```
     Errare è umano. Per fare un vero disastro ci vuole un computer. ~
     --------->-------------------------------------------------->
        fu                           fp
```

Nello stesso esempio, il comando "fp" vi sposta al centro della parola computer.
   Potete specificare un numero; in questo modo, potete ad esempio andare alla "o" di "disastro" con "3fo":

```
     Errare è umano. Per fare un vero disastro ci vuole un computer. ~
               ------------------------------>
                           3fo
```

Il comando "F" cerca verso sinistra:

```
     Errare è umano. Per fare un vero disastro ci vuole un computer. ~
              <----------------------
                        Fm
```

Il comando "tx" lavora allo stesso modo di "fx", ad eccezione del fatto che si ferma un carattere prima del carattere ricercato. Suggerimento: "t" sta per "To", ovvero "Verso" in inglese. Per andare a ritroso usare "Tx".

```
     Errare è umano. Per fare un vero disastro ci vuole un computer. ~
               <-----------   ---------------------->
                    Tm                 tl
```

Questi quattro comandi possono essere ripetuti con ";". "," ripete nella direzione opposta. Il cursore non è mai spostato su un'altra linea. Neppure quando la frase continua.

Può capitare di iniziare una ricerca, e di accorgersi di aver usato il comando sbagliato. Per esempio, digitate "f" per cercare in avanti, mentre volevate usare "F". Per annullare una ricerca, premete <Esc>. Così "f<Esc>" terminerà la ricerca e non farà nient'altro. Note: <Esc> annulla molte operazioni, non solo le ricerche.

==
03.4 Spostarsi sulla parentesi corrispondente

Quando si scrive un programma, spesso si utilizzano dei costrutti con parentesi () annidate. Il comando "%" può allora tornare utile: sposta il cursore sulla parentesi corrispondente. Se il cursore è su una "(", si porterà sulla corrispondente ")". Se è su una ")", si porterà sulla

corrispondente "(".

```
                        %
                     <----->
               if (a == (b * c) / d) ~
                  <---------------->
                        %
```

Questo funziona anche con le parentesi [] e {}. (Questa lista si può modificare con l'opzione 'matchpairs').

Quando il cursore non è su un carattere adatto, "%" cercherà in avanti per trovarne uno. Così se il cursore si trova all'inizio della linea del precedente esempio, "%" cercherà avanti e troverà il primo "(". Poi muoverà il cursore alla parentesi corrispondente:

```
               if (a == (b * c) / d) ~
               ---+---------------->
                  %
```

==
03.5 Spostarsi sulla linea desiderata

Se siete un programmatore C o C++, avrete già visto un messaggio di errore simile al seguente:

```
        prog.c:33: j undeclared (first use in this function) ~
```

Questo vi dice che dovete correggere qualcosa alla riga 33. Come trovare la linea 33? Un metodo è quello di digitare "9999k" per andare all'inizio del file, e poi "32j" per muovervi verso il basso di 32 linee. Non è il massimo, ma funziona. E' comunque meglio usare il comando "G".
   Associato a un numero, questo comando vi posiziona sulla linea specificata dal numero. Per esempio, "33G" vi sposta sulla linea numero 33. (Per un metodo migliore per scandire la lista degli errori di compilazione, si veda |usr_30.txt|, che contiene informazioni sul comando :make).
   Se usato senza argomenti, il comando "G" vi posiziona alla fine del file. Un metodo veloce per andare all'inizio del file è usare "gg". "1G" si comporta allo stesso modo, ma bisogna premere qualche tasto in più.

```
            |   prima linea di un file  ^
            |   testo testo testo testo |
            |   testo testo testo testo | gg
        7G  |   testo testo testo testo |
            |   testo testo testo testo
            |   testo testo testo testo
            V   testo testo testo testo |
                testo testo testo testo | G
                testo testo testo testo |
                ultima linea di un file V
```

Un altro modo per muoversi verso una linea è usare il comando % con un numero.
   Per esempio "50%" muove alla metà (50 %) del file. "90%" muove verso la fine del file.

I comandi precedenti presumono che vogliate muovervi verso una linea del file, e non importa se la linea è presente sulla schermata o no. Cosa fare se volete muovervi su una delle linee che vedete? Questa figura vi mostra tre comandi che potete utilizzare:

```
                      +---------------------------+
               H -->  | testo esempio testo       |
                      | esempio testo             |
                      | testo esempio testo       |
                      | esempio testo             |
               M -->  | testo esempio testo       |
                      | esempio testo             |
                      | testo esempio testo       |
                      | esempio testo             |
               L -->  | testo esempio testo       |
                      +---------------------------+
```

Suggerimento: "H" sta per "Home" (in questo caso "in Alto"), "M" per "Middle" ("in Mezzo") e "L" per "Last" (in questo caso "in Basso").

==
03.6 Sapere dove siete

Per vedere dove siete in un file, ci sono tre metodi:

1. Usare il comando CTRL-G. Ottenete un messaggio come questo (assumendo che l'opzione 'ruler' non sia attiva):

```
"usr_03.txt" line 233 of 650 --35%-- col 45-52 ~
```

   Questo messaggio mostra il nome del file che state editando, il numero di linea dove si trova il cursore, il numero totale delle linee, in che posizione percentuale siete rispetto a tutto il file e la colonna su cui è posizionato il cursore.
      A volte potreste vedere un doppio numero di colonna. Per esempio, "col 2-9". Questo indica che il cursore è posizionato sul secondo carattere, ma poiché il primo carattere è una tabulazione, che occupa otto spazi, la colonna sullo schermo è la 9.

2. Attivare l'opzione 'number'. Sarà visualizzato il numero della riga davanti a ogni riga: **>**

```
:set number
```
<
   Per disattivare questa opzione: **>**

```
:set nonumber
```
<
   Poiché 'number' è una opzione binaria, premettendo un "no" all'opzione si ottiene la sua disattivazione. Una opzione binaria ha solo due valori: on e off.
   Vim ha molte opzioni. Oltre a quelle binarie, ci sono opzioni con valore numerico e stringhe. Si vedranno altri esempi di opzione dove tornerà utile usarle.

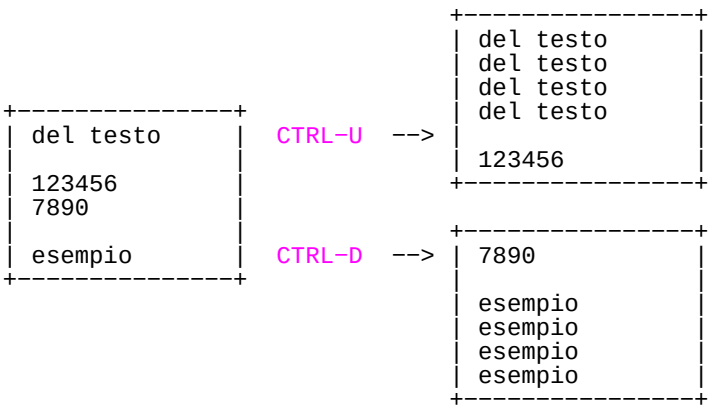
3. Attivare l'opzione 'ruler'. Sarà visualizzata la posizione del cursore nell'angolo in basso a destra della finestra di Vim: **>**

```
:set ruler
```

L'uso dell'opzione 'ruler' ha il vantaggio di non occupare molto spazio, cosicché c'è più spazio per il vostro testo.

==
03.7 Paginazione

Il comando CTRL-U fa "scendere" una mezza schermata di testo. Pensate di guardare il testo attraverso una finestra, e di spostare questa finestra verso l'alto, per una altezza pari a metà della altezza della finestra. In questo modo la finestra di sposta all'insù, verso il testo che si trova più indietro nel file. Non preoccupatevi se fate fatica a ricordare quale parte è più in alto. Succede così alla maggior parte degli utilizzatori.
   Il comando CTRL-D sposta la finestra di visualizzazione verso il basso, e quindi sposta il vostro testo verso l'alto di una mezza schermata.

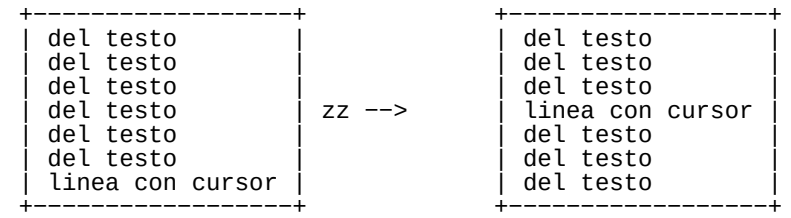


Per scendere di una linea per volta usate CTRL-E (pagina all'insù) e CTRL-Y (pagina all'ingiù). Pensate a CTRL-E come il modo per vedere una linea Extra. (Se utilizzate una mappatura dei tasti simile a MS-Windows, CTRL-Y serve per rifare una modifica (REDO), invece che per paginare).

Per andare in avanti di una intera schermata (meno un paio di linee), usate CTRL-F. Nella direzione opposta, il comando da usare è CTRL-B. Fortunatamente [...per gli inglesi] CTRL-F va "Forward" (in avanti), e CTRL-B

va "Backward" (all'indietro).

Una situazione comune è che dopo esservi mossi all'ingiù di parecchie linee con "j", il vostro cursore è in fondo allo schermo. Se volete vedere nel suo contesto la linea dove si trova il cursore, basta usare il comando "zz".



Il comando "zt" porta la linea su cui si trova il cursore in cima ("top") allo schermo, "zb" la posta a fondo schermata ("bottom"). Ci sono alcuni altri comandi per paginare, si veda |Q_sc|. Per lasciare sempre alcune linee di contesto visibili attorno alla linea del cursore, usate l'opzione 'scrolloff'.

==
03.8 Ricerche semplici

Per cercare una stringa, si usa il comando "/stringa". Per trovare la parola include, per esempio, usate il comando: >

```
/include
```

Potete notare che quando digitate "/" il cursore salta all'ultima linea della finestra di Vim, come quando usate il comando due punti. Su questa riga digiterete la parola da ricercare. Potete premere il tasto backspace ("freccia all'indietro" o <BS>) per fare delle correzioni. Usate i tasti cursore <Left> ("freccia sinistra") e <Right> ("freccia destra") se necessario.
Il comando viene eseguito quando premete <Invio>.

```
Note:
I caratteri .*[]^%/\?~$ hanno un significato speciale. Se volete
usarli come caratteri in una ricerca, dovete premettere una \
davanti ad essi. Si veda più oltre.
```

Per trovare l'occorrenza successiva della stessa stringa si usa il comando "n". Per trovare il successivo #include dopo il cursore usare: >

```
/#include
```

E poi digitare "n" diverse volte. Raggiungerete ogni #include nel testo. Potete anche usare un numero se sapete verso quale occorrenza spostarvi. Così "3n" cerca la terza occorrenza. L'uso di un contatore con "/" non funziona.

Il comando "?" ha la stessa funzione di "/", ma ricerca all'indietro:

```
?parola
```

Il comando "N" ripete l'ultima ricerca nell'opposta direzione. Così "N" dopo un comando "/" cerca all'indietro, "N" dopo "?" cerca in avanti.

IGNORARE IL MINUSCOLO/MAIUSCOLO

Normalmente dovete digitare esattamente quello che volete cercare. Se non vi interessa distinguere le maiuscole dalle minuscole in una parola, impostate l'opzione 'ignorecase': >

```
:set ignorecase
```

Se ora cercate "parola", troverete sia "Parola" che "PAROLA". Per tornare all'impostazione precedente: >

```
:set noignorecase
```

CRONOLOGIA

Supponiamo che abbiate fatto queste tre ricerche: >

```
/uno
/due
/tre
```

Ora iniziate una ricerca con un semplice "/", senza premere <Invio>. Se premete <Up> (tasto "freccia in sù"), Vim mette "/tre" sulla linea di comando. A questo punto, premendo <Invio> cerca tre. Se non premete <Invio>, ma nuovamente <Up>, Vim cambia il prompt in "/due". Se premete ancora <Up> ottenete "/uno".
Potete ovviamente anche usare il tasto cursore <Down> ("freccia in giù") per muovervi attraverso la cronologia dei comandi di ricerca nell'altra direzione.

Se sapete come inizia una precedente espressione di ricerca che avete usato, e volete utilizzarlo nuovamente, potete inserire tale lettera prima di premere <Up>. Nel precedente esempio, potete digitare "/u<Up>" e Vim metterà "/uno" sulla linea di comando.

I comandi che iniziano con ":" hanno anch'essi una cronologia. Ciò vi permette di richiamare un precedente comando e di eseguirlo nuovamente. Queste due cronologie sono separate.

RICERCA DI UNA PAROLA NEL TESTO

Supponiamo di vedere la parola "FunzioneConNomeLungo" nel testo e di voler cercare la prossima occorrenza di tale parola. Potete digitare "/FunzioneConNomeLungo", ma dovete scrivere molto.
C'è un metodo più semplice: posizionare il cursore sulla parola e usare il comando "*". Vim cattura la parola sotto il cursore e la usa come stringa di ricerca.
Il comando "#" fa la stessa cosa nell'altra direzione. Potete premettere un numero: "3*" cerca la terza occorrenza della parola sotto il cursore.

RICERCA DI PAROLE INTERE

Se digitate "/sono" trovate anche "sonoro". Per trovare solamente le parole che finiscono con "sono" digitate: **>**

```
/sono\>
```

La voce "\>" è una speciale marcatura che indica la fine di una parola. Similmente "\<" indica l'inizio di una parola. Così per cercare esattamente la parola "sono" si usa: **>**

```
/\<sono\>
```

Questo non trova "sonoro" o "consono". Notate che i comandi "*" e "#" usano questi marcatori di fine-parola e inizio-parola per cercare unicamente le parole complete (si possono usare "g*" e "g#" per trovare parole parziali).

EVIDENZIARE I RISULTATI DELLE RICERCHE

Immaginiamo di editare un programma e di vedere una variabile chiamata "nr", e di voler controllare dove è usata. Potete posizionare il cursore su "nr" e usare il comando "*" e poi premere "n" per visualizzare tutte le corrispondenze.
C'è un altro metodo. Digitate questo comando: **>**

```
:set hlsearch
```

Se ora ricercate "nr", Vim evidenzierà tutte le corrispondenze. Questo è un ottimo metodo per vedere dove è usata una variabile, senza dover digitare altri comandi.
Per annullare questa impostazione: **>**

```
:set nohlsearch
```

Ora dovrete riattivare l'impostazione se volete usarla per il prossimo comando di ricerca. Se volete solo rimuovere l'evidenziazione, usate questo comando: **>**

```
:nohlsearch
```

Questo non disattiva l'opzione, ma disabilita l'evidenziazione. Subito dopo che avrete eseguito la ricerca, l'evidenziazione sarà usata di nuovo. Questo

vale anche per i comandi "n" e "N".

AFFINAMENTO DELLE RICERCHE

Ci sono alcune opzioni che modificano il comportamento delle ricerche. Queste sono quelle essenziali:
>

```
:set incsearch
```

Questo fa sì che Vim visualizzi i risultati della ricerca mentre state ancora digitando. Usate questa opzione per controllare se verrà trovata la corrispondenza che cercate. Poi premete <Invio> per spostarvi realmente nel posto evidenziato. Oppure digitate altre lettere per modificare la stringa di ricerca.
>

```
:set nowrapscan
```

Questa opzione interrompe la ricerca alla fine del file. Oppure, se state cercando all'indietro, la interrompe all'inizio del file. L'opzione 'wrapscan' è attivata per default, e quindi le ricerche proseguono ad anello, passando dalla fine all'inizio del file (o viceversa).

INTERMEZZO

Se gradite una delle opzioni appena menzionate, e volete attivarla ogni volta che usate Vim, potete inserire il comando nel file di configurazione di Vim.
Editate il file, come menzionato in |not-compatible|. Oppure usate questo comando per trovare dove sia tale file: >

```
:scriptnames
```

Editate il file, per esempio con: >

```
:edit ~/.vimrc
```

Poi aggiungete una linea con il comando per impostare l'opzione, esattamente come avreste fatto in Vim. Esempio: >

```
Go:set hlsearch<Esc>
```

"G" vi posiziona alla fine del file. "o" inizia una nuova riga, dove digitate il comando ":set". Infine uscite dalla modalità di inserimento con <Esc>. Ora salvate il file: >

```
ZZ
```

Se avviate nuovamente Vim, l'opzione 'hlsearch' sarà impostata.

==
03.9 Modelli semplici di ricerca

L'editor Vim usa delle espressioni regolari per specificare cosa si vuole cercare. Le espressioni regolari sono un mezzo estremamente compatto e potente per specificare una espressione da cercare. Sfortunatamente, questa potenza ha un prezzo, perché le espressioni regolari devono essere specificate con molta pignoleria...
In questa sezione menzioneremo solo le più essenziali. Potete trovare maggiori informazioni sulle espressioni e sui comandi di ricerca nel capitolo 27 |usr_27.txt|. Potete trovare una spiegazione esauriente qui: |pattern|.

INIZIO E FINE DI LINEA

Il carattere ^ indica l'inizio di una linea. Ad esempio, l'espressione "include" trova la parola include ovunque sulla linea.
L'espressione "^include" invece trova la parola include solo se questa è all'inizio di una linea.
Il carattere $ indica la fine di una linea. Così, "was$" trova la parola was solo se questa si trova alla fine di una linea.

In questa linea di esempio, sono indicate con delle "x" le posizioni dove è stata trovata la stringa "the":

```
the solder holding one of the chips melted and the ~
xxx                     xxx                 xxx
```

Usando "/the$" si trova soltanto:

```
        the solder holding one of the chips melted and the ~
                                                        xxx
```

E con "/^the" si trova soltanto:

```
        the solder holding one of the chips melted and the ~
        xxx
```

Se si prova a cercare con "/^the$", si troveranno solo le linee che consistono unicamente della parola "the". Gli spazi bianchi in questo caso hanno importanza, quindi se una linea contiene uno spazio dopo la parola, come "the ", l'espressione non sarà trovato.

TROVARE OGNI SINGOLO CARATTERE

Al carattere . (punto) corrisponde ogni possibile carattere. Per esempio, l'espressione "c.m" trova una stringa in cui il primo carattere è una c, il secondo carattere è un carattere qualunque, e il terzo carattere è una m. Esempio:

```
        We use a computer that became the cummin winter. ~
                 xxx             xxx      xxx
```

TROVARE CARATTERI SPECIALI

Se volete davvero trovare il carattere "." (punto), dovete "avvertire" Vim, mettendo un backslash (\) prima del punto stesso.
   Se cercate "ter.", troverete questi risultati:

```
        We use a computer that became the cummin winter. ~
                      xxxx                          xxxx
```

Cercando "ter\." si trova invece solo il secondo risultato.

==
03.10 Marcare il testo

Quando saltate in una posizione con il comando "G", Vim ricorda la posizione occupata prima di questo salto. Questa posizione è chiamata marcatore. Per tornare dove eravate partiti, usate questo comando: **>**

```
        ``
```

Il carattere ` è un backtick, cioè una virgoletta singola (un accento grave).
   (Nota del traduttore: sulle tastiere italiane si ottiene con "<AltGr>'" nei sistemi Linux, e con "<Alt>96" nei sistemi Windows).
   Se usate lo stesso comando una seconda volta, tornerete dove eravate. Questo perché il comando ` è un salto a se stesso, e la posizione prima del salto viene memorizzata.

Generalmente, ogni comando che muove il cursore in una linea che non sia la stessa linea di partenza, è considerato un salto. Questo include i comandi di ricerca "/" e "n" (non importa quanto distante sia la corrispondenza), ma non le ricerche effettuate con "fx" e "tx" o i movimenti di parola "w" e "e".
   Ugualmente, "j" e "k" non sono considerati un salto. Neppure quando si usa un contatore per muovere il cursore in una posizione molto lontana.

Il comando `` salta avanti e indietro, fra due punti. Il comando CTRL-O salta verso la precedente posizione (Suggerimento: O sta per "Older", ossia "più vecchio" in inglese). CTRL-I salta alla posizione più recente (Suggerimento: I è immediatamente vicino a O sulla tastiera). Considerate questa sequenza di comandi: **>**

```
        33G
        /^Qui
        CTRL-O
```

Prima saltate alla linea 33, poi cercate una linea che inizia con "Qui". Poi CTRL-O vi porta indietro alla linea 33. Un altro CTRL-O vi riporta dove avete iniziato. Se ora usate CTRL-I, tornerete nuovamente alla linea 33. E usando un altro CTRL-I salterete alla parola "Qui" precedentemente trovata.

```
        |   esempio testo     ^           |
    33G |   esempio testo     | CTRL-O    | CTRL-I
        |   esempio testo     |           |
        V   linea 33 testo    ^           V
        |   esempio testo     |           |
  /^Qui |   esempio testo     | CTRL-O    | CTRL-I
        V   Qui voi siete     |           V
            esempio testo
```

Note:

CTRL-I funziona allo stesso modo di <Tab>.

Il comando ":jumps" fornisce una lista delle posizioni verso le quali siete saltati. La posizione che avete usato per ultima è segnata con ">".

MARCATORI CON NOME

Vim vi permette di posizionare i vostri personali marcatori nel testo. Il comando "ma" marca la posizione sotto il cursore come il marcatore a. Potete posizionare 26 marcatori (usando le lettere dalla a alla z) nel vostro testo. Non potete vederli, sono solo posizioni che Vim memorizza.

Per andare su un marcatore, usate il comando `{marcatore}, dove con "{marcatore}" si intende la lettera prescelta. Così per muoversi sul marcatore a si usa:

>

```
`a
```

Il comando 'marcatore (virgoletta semplice, o apostrofo) vi posiziona invece all'inizio della linea che contiene il marcatore. Questa è la differenza fondamentale rispetto al comando `marcatore, il quale muove sulla colonna marcata.

I marcatori possono essere veramente utili quando si lavora su due parti collegate di un file. Supponete di avere del testo che dovete avere sott'occhio vicino all'inizio del file, mentre state lavorando su del testo vicino alla fine del file.

Muovetevi all'inizio del testo e posizionate qui il marcatore i (inizio): **>**

```
mi
```

Poi muovetevi sul testo dove volete lavorare e posizionate qui il marcatore f (fine): **>**

```
mf
```

Ora potete muovervi avanti e indietro, e quando volete vedere l'inizio del file, usate questo comando per saltare lì: **>**

```
'i
```

Quindi potete usare '' per saltare indietro dove eravate, oppure 'f per saltare sul testo dove state lavorando alla fine.

La scelta di i per inizio ed f per fine poteva essere diversa, queste lettere sono solo più facili da ricordare.

Potete usare questo comando per ottenere una lista dei marcatori: **>**

```
:marks
```

Potete notare alcuni marcatori speciali. Fra questi:

```
'    Posizione del cursore prima di effettuare un salto
"    Posizione del cursore quando avete editato il file l'ultima volta
[    Inizio dell'ultimo cambiamento
]    Fine dell'ultimo cambiamento
```

==========

Capitolo seguente: |usr_04.txt| Fare piccole modifiche

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

usr_04.txt   Per Vim version 6.2. Ultima modifica: 2004 Gen 17

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Bartolomeo Ravera

Fare piccole modifiche

Questo capitolo mostra diversi modi di effettuare correzioni e spostare il testo. Vi insegnerà i tre metodi di base per modificare il testo: operatore-movimento, Visual_mode e oggetti di testo.

  Capitolo seguente: |usr_05.txt| Configurazioni personali
Capitolo precedente: |usr_03.txt| Muoversi nel file
            Indice: |usr_toc.txt|

==

04.1 Operatori e spostamenti

Nel capitolo 2 avete imparato il comando "x" per cancellare un singolo carattere. E usando un contatore: "4x" cancella quattro caratteri.
   Il comando "dw" cancella una parola. Potete ricordare il comando "w" come il comando di movimento di una parola. In effetti, il comando "d" può essere seguito da ogni comando di movimento, e cancella dalla posizione attuale fino a quella in cui il cursore viene spostato.
   Il comando "4w", per esempio, muove il cursore di 4 parole. Il comando d4w cancella quattro parole.

```
        To err is human. To really foul up you need a computer. ~
                         ------------------>
                                 d4w

        To err is human. you need a computer. ~
```

Vim cancella solamente dalla posizione successiva a quella da cui si trova il cursore. Questo perchè Vim sa che probabilmente non volete cancellare il primo carattere di una parola. Se usate il comando "e" per muovervi alla fine di una parola, Vim penserà che vogliate includere l'ultimo carattere:

Vim cancella soltanto sopra la posizione su cui il movimento porta il cursore. Ciò avviene perchè Vim sa che voi probabilmente non intendete cancellare il primo carattere di una parola. Usando il comando "e" per muovere il cursore alla fine di una parola, Vim immagina che vogliate includere l'ultimo carattere:

```
        To err is human. you need a computer. ~
                         -------->
                            d2e

        To err is human. a computer. ~
```

Se il carattere che sta sotto il cursore viene incluso o no, dipende dal comando usato per muovervi verso tale carattere. Il manuale di riferimento lo chiama comando "esclusivo" quando il carattere non è incluso, e "inclusivo" quando lo è.

Il comando "$" sposta il cursore alla fine della linea. Il comando "d$" cancella dalla posizione del cursore sino alla fine della linea. Questo è un movimento inclusivo, quindi l'ultimo carattere della linea è incluso nell'operazione di cancellazione:

```
        To err is human. a computer. ~
                         ------------>
                              d$

        To err is human ~
```

C'è uno schema qui: operatore-movimento. Voi prima scrivete un comando operatore. Per esempio, "d" è l'operatore di cancellazione. Allora voi scrivete un comando di movimento, come "4l" o "w". Così potete operare su qualsiasi testo su cui possiate spostarvi.

==
04.2 Cambiare il testo

Un altro operatore è "c", cambio. Questo agisce come l'operatore "d", ad eccezione del fatto che vi lascia nella Insert_mode.
  Per esempio, "cw" cambia una parola. O più precisamente, cancella una parola e vi pone in Insert_mode.

        To err is human ~
           ------->
             c2wbe<Esc>

        To be human ~

Questo "c2wbe<Esc>" è composto da queste parti:

        c   l'operatore di cambio
        2w   sposta avanti di due parole (vengono cancellate e parte il
                Insert_mode)
        be   inserisce questo testo
        <Esc>   torna in Normal_mode

Se avete fatto attenzione, avrete notato qualcosa di strano: lo spazio prima di "human" non è stato cancellato. C'è un proverbio che dice che per ogni problema esiste una risposta semplice, chiara e sbagliata. Come in questo caso, per questo esempio relativo al comando "cw". Questo in realtà lavora esattamente come "ce", cambia fino alla fine di una parola. Quindi lo spazio dopo la parola non è incluso. Questa è una eccezione che risale al vecchio Vi. Poichè molte persone ci si sono ormai abituate questa incoerenza è rimasta in Vim.

ULTERIORI MODIFICHE

Come "dd" cancella un'intera linea, "cc" cambia un'intera linea. Questo comando conserva l'indentazione esistente (aggiungendo spazi bianchi).

Esattamente come "d$" cancella fino alla fine della linea, "c$" sostituisce fino alla fine della linea. E' come scrivere "d$" per cancellare il testo e poi usare "a" per dare inizio alla Insert_mode ed aggiungere nuovo testo.

SCORCIATOIE

Alcuni comandi operatore-movimento sono di così frequente uso che sono stati attribuiti loro comandi di una sola lettera:

        x equivale a   dl (cancella il carattere sotto il cursore)
        X equivale a   dh (cancella il carattere a sinistra del cursore)
        D equivale a   d$ (cancella i caratteri sino alla fine della linea)
        C equivale a   c$ (sostituisce sino alla fine della linea)
        s equivale a   cl (sostituisce un solo carattere)
        S equivale a   cc (sostituisce un'intera linea)

DOVE POSIZIONARE IL CONTATORE

I comandi "3dw" e "d3w" cancellano tre parole. Cercando il pelo nell'uovo, il primo comando, "3dw" cancella una parola tre volte; il comando "d3w" cancella tre parole una volta sola. In pratica, non vi è differenza. Potete addirittura utilizzare due contatori. Per esempio, "3w2d" cancella due parole, ripetendo l'azione tre volte, per un totale di sei parole.

SOSTITUIRE UN CARATTERE

Il comando "r" non è un operatore. Questo attende che digitiate un carattere, per rimpiazzare con questo il carattere sotto il cursore. Potete ottenere lo stesso risultato con "cl" o con il comando "s", ma con "r" non dovete premere <Esc>

        there is somerhing grong here ~

```
        rT          rt    rw

        There is something wrong here ~
```

Usando un contatore con "r", molti caratteri saranno rimpiazzati con lo stesso carattere. Ad esempio:

```
        There is something wrong here ~
                            5rx

        There is something xxxxx here ~
```

Per sostituire un carattere con un'interruzione di linea, usate "r<Enter>". Ciò cancella un solo carattere ed inserisce una interruzione di linea. Usando un conto qui solo relativo al numero di caratteri cancellati: "4r<Enter>" sostituisce quattro caratteri con una interruzione di linea.

==

04.3 Ripetere una modifica

Il comando "." è uno dei più semplici ma potenti comandi in Vim. Tale comando ripete l'ultima modifica. Ad esempio, supponete di star lavorando su un file HTML e di voler cancellare tutte le etichette <B>. Posizionate il cursore sul primo < e cancellate <B> con il comando "df>". Poi spostatevi su < del prossimo </B> e cancellatelo usando il comando ".". Il comando "." esegue l'ultimo comando di modifica (in questo caso, "df>"). Per cancellare un'altra etichetta, posizionate il cursore su < e usate il comando ".".

```
                                To <B>generate</B> a table of <B>contents ~
        f<   find first <       --->
        df>  delete to >           -->
        f<   find next <             --------->
        .    repeat df>                    --->
        f<   find next <                      ------------->
        .    repeat df>                                  -->
```

Il comando "." funziona su ogni comando, ad eccezione di "u" (undo), CTRL-R (redo) e dei comandi che iniziano con un due punti (:).

Un altro esempio: si vuole sostituire la parola "quattro" con "cinque". "quattro" appare diverse volte nel vostro testo. Potete fare ciò più velocemente con questa sequenza di comandi:

```
        /quattro<Enter> trova la prima stringa "quattro"
        cwcinque<Esc>   cambia la parola in "cinque"
        n               trova il prossimo "quattro"
        .               ripete il cambiamento in "cinque"
        n               trova il prossimo "quattro"
        .               ripete il cambiamento
                        etc.
```

==

04.4 Visual_mode

Per cancellare voci semplici le sostituzioni degli operatori di movimento funzionano abbastanza bene. Ma spesso non è così semplice decidere quale comando vi sposti sul testo che volete cambiare. In questo caso potete usare il Visual_mode.

Accedete al Visual_mode premendo "v". Muovete il cursore sul testo su cui volete lavorare. Mentre fate ciò, il testo viene evidenziato. Infine digitate il comando operatore.
   Per esempio, per cancellare dalla metà di una parola alla metà di un'altra parola:

```
                This is an examination sample of visual_mode ~
                           ---------->
                             velllld

                This is an example of visual_mode ~
```

Facendo questo, non dovrete conoscere esattamente quante volte digitare "l" per portarvi sulla giusta posizione. Potete immediatamente vedere la porzione di testo che sarà cancellata quando premerete "d".

Se ad un certo punto doveste decidere di non modificare il testo evidenziato, basterà premere <Esc> e il Visual_mode sarà concluso senza nessun cambiamento.

SELEZIONE DI LINEE

Se volete lavorare su un'intera linea, usate "V" per attivare il Visual_mode. Potete vedere come l'intera linea venga evidenziata, senza che il cursore sia mosso. Muovendovi verso destra o sinistra nulla cambia. Quando vi spostate nella riga in alto o in basso, la selezione si estende alle linee interessate, nella loro interezza.
   Per esempio, selezionate tre linee con "Vjj":

```
                        +------------------------+
                        | text more text         |
                     >> | more text more text    | |
     linee selezionate >> | text text text         | | Vjj
                     >> | text more              | V
                        | more text more         |
                        +------------------------+
```

SELEZIONE DI BLOCCHI

Se volete lavorare su un blocco rettangolare di caratteri, usate CTRL-V per attivare il Visual_mode. Ciò è veramente utile quando si lavora con delle tabelle.

```
               name            Q1      Q2      Q3
               pierre          123     455     234
               john            0       90      39
               steve           392     63      334
```

Per cancellare la colonna "Q2" posta in mezzo, spostate il cursore sulla "Q" di "Q2". Premete CTRL-V per attivare Visual_mode con modalità a blocco. Ora spostate il cursore tre linee in basso con "3j" e poi sulla prossima parola con "w". Potete vedere che viene incluso anche il primo carattere dell'ultima colonna. Per escluderlo, usate "h". Ora premete "d" e la colonna in mezzo è a posto.

ANDARE DALL'ALTRO LATO

Se avete già selezionato del testo in Visual_mode, e vi accorgete di dover fare un cambiamento dall'altra parte della selezione, usate il comando "o" (Suggerimento: "o" sta per "other", in inglese "altra" parte). Il cursore si posizionerà dall'altra parte, e potrete muovere il cursore per cambiare dove la selezione inizia. Premendo "o" nuovamente, sarete portati dall'altra parte.

Quando usate la selezione con modalità a blocchi, avete quattro angoli. "o" minuscolo vi porta sull'angolo opposto, in diagonale. Per spostarvi sull'altro angolo sulla stessa linea, si usa "O" maiuscolo.

Note: "o" e "O" in Visual_mode si comportano in modo differente dal Normal_mode, nel quale aprono una nuova linea sotto o sopra il cursore.

==
04.5 Muovere il testo

Quando cancellate qualcosa con "d", "x", o un altro comando, il testo viene memorizzato. Potete incollarlo nuovamente usando il comando "p". (Vim chiama ciò "put").
   Osservate come questo funziona. Prima potete cancellare un'intera linea, posizionando il cursore sulla linea che volete cancellare e digitando "dd", Ora spostate il cursore nella posizione dove volete incollare la linea e usate il comando "p" (put). La linea verrà inserita sulla linea sotto il cursore.

```
        una linea       una linea       una linea
        linea 2    dd   linea 3     p   linea 3
        linea 3                        linea 2
```

Poichè avete cancellato un'intera linea, il comando "p" ha posizionato la linea di testo sotto il cursore. Se cancellate parte di una linea (una parola, per esempio), il comando "p" la incollerà esattamente dopo il cursore.

```
        Some more boring try text to out commands. ~
                        ---->
                         dw

        Some more boring text to out commands. ~
```

```
                        ------->
                          welp

        Some more boring text to try out commands. ~
```

ALTRO SULL'INCOLLAGGIO

Il comando "P" incolla il testo come "p", ma prima del cursore. Quando avrete cancellato un'intera linea con "dd", "P" la incollerà prima del cursore. Quando avrete cancellato una parola con "dw", "P" la incollerà proprio prima del cursore.

Potete ripetere l'incollatura quante volte volete. Verrà usato sempre lo stesso testo.

Potete usare un contatore con "p" e "P". Il testo sarà ripetuto tante volte quante ne sono state specificate con il contatore. Così, "dd" e poi "3p" incolla tre copie della stessa linea cancellata.

SCAMBIARE DUE CARATTERI

Capita spesso che mentre si digita, le dita corrano più velocemente del cervello (o viceversa...). Il risultato è una cosa simile a questa: "teh" invece di "the". Vim rende più facile correggere questo tipo di problema. Posizionate il cursore sulla "e" di "teh" ed eseguite il comando "xp". Questo lavora in questo modo: "x" cancella il carattere e lo pone in un registro. "p" incolla il testo dopo il cursore, che è dopo la "h".

```
        teh     th     the ~
         x       p
```

==

04.6 Copiare il testo

Per copiare del testo da un posto ad un altro, potete cancellarlo, usare "u" per annullare la cancellazione e poi usare "p" per incollarlo dove si vuole. C'è un metodo più semplice: usare lo "yank" (in inglese: "strappare"). L'operatore "y" copia il testo in un registro. Poi, il comando "p" può essere usato per incollarlo.
   "Strappare" è solo un sinonimo che Vim usa al posto di "copiare". Questo perchè la lettera "c" era già stata usata per l'operatore di cambiamento, mentre la lettera "y" era ancora disponibile. Chiamando questo operatore "yank", è più facile ricordare l'uso del tasto "y".

Poichè "y" è un operatore, si usa "yw" per copiare una parola. Come al solito, è possibile usare un contatore. Per copiare due parole, usate "y2w". Per esempio:

```
        let sqr = LongVariable * ~
                  -------------->
                       y2w

        let sqr = LongVariable * ~
                               p

        let sqr = LongVariable * LongVariable ~
```

Notate che "yw" include lo spazio bianco dopo una parola. Se non desiderate ciò, usate "ye".

Il comando "yy" copia un'intera linea, come "dd" cancella un'intera linea. Contrariamente alle aspettative, mentre "D" cancella dal cursore fino alla fine della linea, "Y" funziona come "yy", cioè copia l'intera linea.

```
        a text line   yy       a text line           a text line
        line 2                 line 2         p      line 2
        last line              last line             a text line
                                                     last line
```

==

04.7 Usare la clipboard

Se state usando la versione GUI di Vim (gvim), potete trovare la voce "Copia" nel menu "Edit". Prima selezionate del testo con il Visual_mode, poi usate il menu Modifica/Copia. Il testo selezionato è ora stato copiato nella clipboard (NdT: clipboard=parte della memoria in cui è temporaneamente

memorizzato un testo). Potete incollare il testo in altri programmi.
Ovviamente, anche nello stesso Vim.

Se avete copiato, da un'altra applicazione, del testo nella clipboard, potete
incollarlo in Vim con il menu Modifica/Incolla. Questo funziona in
Normal_mode e in Insert_mode. In Visual_mode il testo selezionato è
sostituito con il testo incollato.

La voce "Taglia" del menù cancella il testo prima di porlo nella clipboard.
Le voci "Copia", "Taglia" e "Incolla" sono anche disponibili nel menù
a discesa.

Se il vostro Vim ha una barra degli strumenti, potete trovare queste voci
anche lì.

Se non state usando la GUI, oppure se non vi piace usare il menù, dovete
utilizzare un altro metodo. Usate i soliti comandi "y" (yank) e "p" (put), ma
premettete loro "* (virgolette asterisco). Per copiare una linea nella
clipboard: **>**

```
"*yy
```

Per trasferire del testo dalla clipboard di nuovo nel testo: **>**

```
"*p
```

Ciò funziona solo per le versioni di Vim che includono il supporto per la
clipboard. Potete trovare maggiori informazioni sulla clipboard nella sezione
|09.3| e qui: |clipboard|.

==
04.8 Oggetti di testo

Se il cursore è a metà di una parola che volete cancellare, dovete tornare
indietro all'inizio di questa parola prima di usare il comando "dw".
Esiste un modo più rapido: "daw".

```
this is some example text. ~
                daw

this is some text. ~
```

La "d" di "daw" è l'operatore di cancellazione. "aw" è un oggetto di testo.
Suggerimento: "aw" ricorda "A Word", cioè "Una Parola" in inglese. Per la
precisione, viene cancellato anche lo spazio bianco che segue la parola (e lo
spazio bianco prima della parola se ci si trova alla fine della linea).

L'uso degli oggetti di testo rappresenta un terzo modo per effettuare dei
cambiamenti in Vim. C'erano già operatore-movimento e Visual_mode; ora si
aggiunge alla nostra lista anche l'operatore-oggetti di testo.
   E' molto simile all'operatore-movimento, ma invece di operare sul testo fra
la posizione del cursore prima e dopo il comando di spostamento, l'oggetto di
testo è usato come un blocco. Non importa dove si trova il cursore
all'interno dell'oggetto.

Per modificare un'intera frase, usate "cis". Considerate questo testo:

```
Hello there.  This ~
is an example.  Just ~
some text. ~
```

Spostatevi all'inizio della seconda linea, su "is an". Ora usate "cis":

```
Hello there.    Just ~
some text. ~
```

Il cursore è posizionato fra gli spazi bianchi nella prima linea. Ora
digitate la nuova frase "Another line.":

```
Hello there.  Another line.  Just ~
some text. ~
```

"cis" è composto dall'operatore "c" (change) e dall'oggetto di testo "is".
Questo sta per "Inner Sentence", ovvero "Frase interna", "Frase in cui è il
cursore". Esiste anche l'oggetto "as" ("a sentence", "una frase"). La
differenza consiste nel fatto che "as" include lo spazio bianco dopo la frase,
mentre "is" non lo fa. Se volete cancellare una frase, contemporaneamente
desiderate cancellare lo spazio bianco, quindi usate "das". Se volete

digitare del nuovo testo, lo spazio bianco può rimanere, quindi usate "cis".

Potete usare oggetti di testo anche in Visual_mode. L'oggetto di testo sarà incluso nella selezione Visuale. Il Visual_mode continuerà a permanere, cosicchè potete fare questa operazione diverse volte. Per esempio, date inizio al Visual_mode con "v" e selezionate una frase con "as". Ora potete ripetere "as" per includere altre frasi. Infine, usate un operatore per fare ciò che desiderate con le frasi selezionate.

Potete trovare una lunga lista di oggetti di testo qui: |text-objects|.

==
04.9 Replace_mode

Il comando "R" fa entrare Vim in Replace_mode. In questa modalità, ogni carattere digitato sostituisce quello sotto il cursore. Questo comportamento permane fino a che non si digita <Esc>.
In questo esempio, date inizio al Replace_mode sulla prima "t" di "testo":

```
This is text. ~

        Rinteresting.<Esc>

This is interesting. ~
```

Avrete notato come questo comando abbia sostituito 5 caratteri nella linea con altri dodici. Il comando "R" estende automaticamente la lunghezza della linea se questa è troppo corta per contenere i caratteri da sostituire.
Non prosegue sulla linea successiva.

Potete passare alternativamente fra Insert_mode e Replace_mode con il tasto <Insert>.

Quando usate <BS> (backspace) per fare delle correzioni, potete notare come venga nuovamente immesso sulla riga il vecchio testo. Ciò funziona come un comando "undo" ("annulla") per l'ultimo carattere digitato.

==
04.10 Conclusioni

Gli operatori, i comandi di movimento e gli oggetti di testo vi offrono la possibilità di creare molte combinazioni. Ora che conoscete il loro funzionamento, potete usare N operatori combinati con M comandi di movimento per ottenere N * M comandi!

Potete trovare una lista di operatori qui: |operator|

Per esempio, ci sono molti altri modi per cancellare porzioni di testo. La lista seguente ne illustra alcuni fra i più usati:

```
x       cancella il carattere sotto il cursore (abbreviazione per "dl")
X       cancella il carattere prima del cursore (abbreviazione per "dh")
D       cancella dal cursore alla fine della linea (abbreviazione per "d$")
dw      cancella dal cursore all'inizio della prossima parola
db      cancella dal cursore all'inizio della parola successiva.
diw     cancella la parola sotto il cursore (esclusi gli spazi bianchi)
daw     cancella la parola sotto il cursore (inclusi gli spazi bianchi)
dG      cancella fino alla fine del file
dgg     cancella fino all'inizio del file
```

Usando "c" invece di "d", i comandi precedenti diventeranno comandi di cambiamento. E così via.

Ci sono poi alcuni altri comandi usati frequentemente, che non è possibile classificare facilmente:

```
        ~       cambia da maiuscolo a minuscolo (e viceversa) il carattere
                sotto il cursore, e sposta il cursore al prossimo carattere.
                Questo non è un operatore (a meno che no sia impostato
                'tildeop'), quindi non è possibile usarlo in combinazione con
                un comando di movimento. Funziona in Visual_mode e modifica
                il maiuscolo/minuscolo per tutto il testo selezionato.

        I       Avvia l'Insert_mode dopo aver spostato il cursore al primo
                carattere non-blank nella linea.

        A       Avvia l'Insert_mode dopo aver spostato il cursore alla fine
                della linea.
```

==

Capitolo seguente: |usr_05.txt| Configurazioni personali

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

usr_05.txt   Per Vim version 6.2. Ultima modifica: 2003 Dic 20

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

Configurazioni personali

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

  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 legge questo file in fase di avvio.

Se avete problemi nel trovare il vostro file vimrc, usate il comando: **>**

```
:scriptnames
```

Uno dei primi file nella lista dovrebbe chiamarsi ".vimrc" o "_vimrc" e dovrebbe trovarsi nella vostra home directory
   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 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 file vimrc: **>**

```
set incsearch
```

Affinchè 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 not-compatible (vedi |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 Insert_Mode <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 l'Insert mode.
>
```
        set autoindent
```

Questo consente a Vim di usare l'indentazione della riga precedente per la riga appena creata. Per esempio quando si preme <Enter> in Insert_Mode, 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 back-up di un file quando lo si sovrascrive. Ma non entro il sistema VMS, poichè esso già conserva le vecchie versioni dei files. 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 esempio, 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 l'Ex_mode, ma ciò non vi sarà necessario.

>
```
        vnoremap p <Esc>:let current_reg = @"<CR>gvs<C-R>=current_reg<CR><Esc>
```

Questa è una mappatura complessa. Non spiegheremo qui come funziona.
   Ciò che essa ottiene è che il comando "p" sovrascriva in Visual mode il testo selezionato con dell'altro testo preventivamente copiato. Notate come la mappatura possa venire impiegata per fare molte cose complesse. Tuttavia, questa è solo una sequenza di comandi che vengono eseguiti quando lo scrivete.

>
```
        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|.

vimrc-filetype >
```
        filetype plugin indent on
```

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 esempio, lavorando con un file "c", risulta utile per impostare l'opzione 'cindent' per indentare 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 files di indentazione
   Scrivendo codice l'indentazione di una linea può essere spesso calcolata automaticamente. Vim viene fornito con queste regole di indentazione per un certo numero di tipi di file. Vedere |:filetype-indent-on| e 'indentexpr'.

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

Ciò fa si che Vim interrompa il testo per evitare linee che superino la lunghezza di 78 caratteri.
   Ma solo per files 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.
>
```
        autocmd BufReadPost *
            \ if line("'\"") > 0 && 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>
<
```

        Note:
        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. E' il segnale di avvio che fa eseguire il comando quando il tasto viene premuto.
i{<Esc>	Inserisce il carattere {. Il tasto <Esc> termina Insert_mode.
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. Poichè 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 esempio: >

```
        :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 Normal_mode. Altro sulle mappature nella sezione |40.1|.

==
05.4 Inserire un plugin   *add-plugin* *plugin*

Le funzionalità di Vim possono essere estese aggiungendo plugins. 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 plugins.
{not available when Vim was compiled without the |+eval| feature}

Ci sono due tipi di plugins:

   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|.

PLUGINS GLOBALI   *standard-plugin*

Avviando Vim, questi caricherà automaticamente un certo numero di plugins globali.
   Non siete obbligati a fare nulla per ottenere ciò. Aggiungono funzionalità che potrebbero servire a molti, ma che sono state implementate come scripts di Vim anzichè 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 plugins 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 plugins?
- Qualcuno è compreso insieme con Vim. Lo potete trovare nella directory $VIMRUNTIME/macros e nelle sue sub-directories.
- Scaricatelo dalla rete, provate con http://vim.sf.net.
- 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 and 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 plugins): **>**

```
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 PLUGINS *add-filetype-plugin* *ftplugins*

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

```
:filetype plugin on
```

Tutto qui! Vedere |vimrc-filetype|.

Se aveste perso uno dei plugins 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 plugins globali. Guardate se si menziona il tipo del file, così potrete sapere se il plugin sia globale o riferito al tipo del file. Gli scripts in $VIMRUNTIME/macros sono tutti globali, i filetype plugins 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 plugins 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 qualsiasi cosa. Se usaste "otherstuff.vim" potrebbe non funzionare, potrebbe venir caricato per il filetype "otherstuff".

Su MS-DOS non potete usare nomi lunghi. Vi trovereste nei guai agiungendo 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 plugins 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 files di questo tipo utilizzeranno le impostazioni del plugin. La parte <name> del file plugin non è un problema, potete usarla in molti plugins per lo stesso tipo di file. Note Il nome deve terminare in ".vim".

Ulteriori letture:

|filetype-plugins|	Documentazione per i filetype plugins ed informazioni su come evitare che la mappatura causi problemi.
|load-plugins|	Quando i plugins globali vengono caricati all'avvio.
|ftplugin-overrule|	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 plugins 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 sè un file di help.
   Adesso spiegheremo come installarlo, così potrete trovare facilmente aiuto per i vostri nuovi plugins.
  Usiamo il plugin "matchit.vim" come esempio (viene fornito con Vim). Questo plugin fa si che il comando "%" salti ai tags HTML, if/else/endif negli scripts di Vim, etc. Utilissimo, anche se non retrocompatibile (ciò perchè 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 al directory.) >

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

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 files di help locali verrano automagicamente aggiunte a questa sezione. Lì potrete vedere quali file locali di help siano stati aggiunti e saltare ad essi attraversi 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 potreste 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 esempio, 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 esempio, 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 esempio: >

```
: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, affichè 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 Normal mode. Per permettere a <Left> e <Right> di fare ciò nell'Insert mode fate così: >

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

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

VEDERE I TABULATORI

Quando ci sono dei tabulatori entro un file non potete vedere dove siano. Per renderli visibili: >

```
:set list
```

Adesso ogni tabulatore 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 tabulatori entro un file.
Se avete un terminale a colori o state usando la GUI, Vim può mostrare spazi e tabulatori come caratteri evidenziati. Usate l'opzione 'listchars': >

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

Adesso ogni tabulatore verrà mostrato come ">---" 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 to 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 esempio. 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 dovreste 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

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

usr_06.txt   Per Vim version 6.2. Ultima modifica: 2002 Lug 14

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

Usare l'evidenziazione della sintassi

Il testo in bianco e nero è noioso. Col colore, il vostro file prende vita.
E non solo ha un bell'aspetto, ma velocizza anche il vostro lavoro. Cambiate
i colori utilizzati per i diversi tipi di testo. Stampate i vostri testi, con
i colori che vedete a schermo.

  Capitolo seguente: |usr_07.txt| Elaborare più di un file
Capitolo precedente: |usr_05.txt| Configurazioni personali
             Indice: |usr_toc.txt|

==
06.1 Abilitare l'evidenziazione

Basta un semplice comando:

```
:syntax enable
```

Dovrebbe funzionare nella maggior parte dei casi avere per avere i colori nei
vostri file. Vim individuerà automaticamente il tipo di file e caricherà la
giusta evidenziazione della sintassi. I commenti diventeranno blu, le parole
chiave marroni e le stringhe rosse. Questo rende semplice revisionare il
file. Dopo un pò vi renderete conto che il testo in bianco e nero vi
rallenta!

Se volete utilizzare sempre l'evidenziazione della sintassi, mettete il
comando ":syntax enable" nel vostro |vimrc|.

Se volete utilizzarlo solo quando il terminale supporta il colore, potete
inserire in |vimrc| : >

```
if &t_Co > 1
   syntax enable
endif
```

Se volete l'evidenziazione solo nella versione GUI, mettete il comando "syntax
enable" nel vostro |gvimrc| .

==
06.2 Nessun colore o colori sbagliati?

I motivi possono essere diversi:

- Il vostro terminale non supporta il colore.
  Vim utilizzerà il grassetto, il corsivo e il sottolineato, anche se
  il risultato non ha un bell'aspetto. Probabilmente vorrete provare un
  terminale col colore. Per Unix, raccomando xterm del progetto
  XFree86: |xfree-xterm|.

- Il vostro terminale supporta il colore, ma Vim non lo sa.
  Assicuratevi del corretto settaggio di $TERM. Per esempio, usando un
  xterm che supporta il colore: >

```
setenv TERM xterm-color
```

<
  o (a secondo della shell): >

```
TERM=xterm-color; export TERM
```

< Il nome del terminale deve corrispondere a quello che state usando.
  Se ancora non funziona, date un'occhiata a |xterm-color|, che
  illustra alcuni modi di far sì che VIM mostri i colori (non solo per
  un xterm).

- Il tipo di file non viene riconosciuto.

In fondo, Vim non conosce tutti i tipi di file e talvolta è quasi impossibile dire quale sia il linguaggio del file. Provate questo comando: >

```
:set filetype
```
<

Se il risultato è "filetype=" allora il problema è senza dubbio che Vim non sa che tipo di file sia. Potete impostare il tipo file manualmente: >

```
:set filetype=fortran
```
<

Per vedere quali tipi sono disponibili, guardate nella directory $VIMRUNTIME/syntax. Per la GUI potete usare il menu Sintassi. Settare il tipo di file può essere fatto anche con una |modeline|, in modo che il file venga evidenziato ogni volta che lo modificate. Per esempio, questa riga può essere usata in un Makefile (mettetela vicino all'inizio o alla fine del file): >

```
# vim: syntax=make
```
<

Potreste riuscire a determinare da soli il tipo di file. Spesso si può usare l'estensione (dopo il punto).
Vedete |new-filetype| per sapere come dire a Vim di individuare il tipo di file.

- Non è definita una evidenziazione per il vostro tipo di file.
Potreste provare ad usare un tipo simile impostandolo manualmente come spiegato sopra. Se il risultato non fosse soddisfacente, potreste scrivervi il vostro file di sintassi. Vedere |mysyntaxfile|.

Oppure i colori potrebbero essere sbagliati:

- Il testo colorato è molto difficile da leggere.
Vim individua il colore del vostro sfondo. Se è nero (o un altro colore scuro) utilizza colori chiari per il testo. Se è bianco (o un altro colore chiaro) utilizza colori scuri per il testo. Se Vim sbaglia a determinare il colore di sfondo, il testo sarà difficile da leggere. Per risolvere questo inconveniente, impostate l'opzione 'background'. Per uno sfondo scuro: >

```
:set background=dark
```
<

E per uno sfondo chiaro: >

```
:set background=light
```
<

Assicuratevi di impostarlo _prima_ del comando ":syntax enable", altrimenti i colori saranno già stati impostati. Potreste fare un ":syntax reset" dopo aver impostato "background" per far sì che Vim reimposti i colori.

- I colori sono sbagliati quando scorrete dal basso all'alto.
Vim non legge l'intero file per scansionare il testo. Inizia a scansionarlo in qualsiasi punto vi troviate. Questo fa risparmiare un mucchio di tempo, ma a volte i colori risultano sbagliati. Una semplice correzione è premere CTRL-L. O scorrere un pò indietro e poi ancora un pò avanti.
Per una correzione seria, vedete |:syn-sync|. Alcuni file di sintassi hanno un modo per far sì che guardi ben più indietro, vedete l'help dello specifico file di sintassi. Per esempio, |tex.vim| per la sintassi TeX.

==
06.3 Colori Diversi *:syn-default-override*

Se non vi piacciono i colori di default, potete sceglire un altro schema. Nella GUI usate il menu Edit/Color Scheme. Potete anche scrivere il comando: >

```
:colorscheme evening
```

"evening" è il nome dello schema. Ce ne sono diversi altri che potreste voler provare. Guardate nella directory $VIMRUNTIME/colors.

Quando avete trovato lo schema che vi piace aggiungete il comando ":colorscheme" al vostro vimrc.

Potreste anche scrivere da soli il vostro schema. Ecco come:

1. Selezionate uno schema che ci si avvicina. Copiate questo file nella
vostra directory di Vim. Per Unix, dovrebbe bastare questo: >

```
	!mkdir ~/.vim/colors
	!cp $VIMRUNTIME/colors/morning.vim ~/.vim/colors/mine.vim
<
```

   Questo si fa da Vim, visto che conosce il valore di $VIMRUNTIME.

2. Modificate il file dello schema. Queste voci sono utili:

term	attributi in un terminale in bianco e nero
cterm	attributi in un terminale a colori
ctermfg	colore di primo piano in un terminale a colori
ctermbg	colore di sfondo in un terminale a colori
gui	attributi nella GUI
guifg	colore di primo piano nella GUI
guibg	colore di sfondo nella GUI

   Per esempio, per rendere verdi i commenti: >

```
	:highlight Comment ctermfg=green guifg=green
<
```

   Gli attributi utilizzabili per "cterm" e "gui" sono "bold" e "underline".
   Se li volete entrambi, usate "bold,underline". Per maggiori dettagli,
   vedete il comando |:highlight|.

3. Dite a Vim di usare il vostro schema di colori. Mettete questa riga nel
vostro |vimrc|: >

```
	colorscheme mine
```

Se volete vedere come appaiono le combinazioni di colori più usate, utilizzate
questi comandi: >

```
	:edit $VIMRUNTIME/syntax/colortest.vim
	:source %
```

Vedrete del testo in varie combinazioni di colori. Potete quindi scegliere
quelle più leggibili e carine.

==
06.4 Con o senza i colori

Mostrare il testo a colori impiega molte risorse. Se trovate che sia troppo
lento, potete disabilitare l'evidenziazione per un istante: >

```
	:syntax clear
```

Quando modificate un altro file (o lo stesso) i colori torneranno.

:syn-off

Se volete bloccare definitivamente l'evidenziazione usate: >

```
	:syntax off
```

Questo disabiliterà completamente l'evidenziazione della sintassi e la
rimuoverà immediatamente da tutti i buffer.

:syn-manual

Se volete abilitare l'evidenziazione solo per delle determinate righe, usate
questo: >

```
	:syntax manual
```

Questo abiliterà l'evidenziazione della sintassi, ma non la userà
automaticamente quando si inizia a modificare un buffer. Per attivarla nel
buffer corrente, impostate l'opzione 'syntax': >

```
	:set syntax=ON
<
```

==
06.5   Stampare con i colori   *syntax-printing*

Nella versione per MS-Windows potete stampare il file corrente con questo comando: **>**

```
:hardcopy
```

Otterrete la solita finestra di dialogo della stampante, dove potrete selezionare la stampante e alcune impostazioni. Se avete una stampante a colori, l'output su carta dovrebbe risultare lo stesso di quello a schermo con Vim. Ma se utilizzate uno sfondo scuro i colori verranno modificati per avere un buon aspetto su carta bianca.

Ci sono diverse opzioni che possono cambiare il modo in cui Vim stampa:

```
'printdevice'
'printheader'
'printfont'
'printoptions'
```

Per stampare solo un intervallo di righe, utilizzate la modalità Visual per selezionare le linee e poi usate il comando: **>**

```
v100j:hardcopy
```

"v" inizializza il Visual mode. "100j" si sposta in basso di 100 righe, evidenziandole. Infine ":hardcopy" le stampa. Ovviamente potete usare altri comandi per spostarvi in Visual mode.

Questo funziona anche sotto Unix se avete una stampante PostScript. Altrimenti c'è bisogno di un pò più di lavoro. Dovete prima convertire il testo in HTML, e poi stamparlo da un browser come Netscape.

Convertite il file corrente in HTML con questo comando: **>**

```
:source $VIMRUNTIME/syntax/2html.vim
```

Sentirete che sta macinando dati, e potrebbe essere necessario molto tempo per un file di grandi dimensioni.
Poco dopo, la finestra mostrerà il codice HTML. Adesso salvatelo da qualche parte (non importa dove, dopo lo cancellerete):
>

```
:write main.c.html
```

Aprite questo file nel vostro browser preferito e stampatelo da lì. Se tutto va bene, l'output dovrebbe avere lo stesso aspetto che ha dentro Vim. Vedete |2html.vim| per i dettagli. Non dimenticate di cancellare l'HTML quando avete finito.

Invece di stampare, potreste anche mettere il file HTML su un web server, e lasciare che altri vedano il testo a colori.

==========

06.6 Ulteriori letture

|usr_44.txt| Evidenziazione della vostra sintassi
|syntax| Tutti i dettagli.

==========

Capitolo seguente: |usr_07.txt| Elaborare più di un file

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

usr_07.txt   Per Vim version 6.2. Ultima modifica: 2002 Lug 19

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Giuliano Bordonaro

Elaborare più di un file

Non importa quanti file abbiate, potete elaborarli tutti senza lasciare Vim.
Stabilire una lista di file da elaborare e saltare dall'uno all'altro.
Copiare testo da un file ed inserirlo entro un altro.

  Capitolo seguente: |usr_08.txt| Dividere le finestre
Capitolo precedente: |usr_06.txt| Usare l'evidenziazione della sintassi
              Indice: |usr_toc.txt|

==
07.1 Elaborare un altro file

Innanzi tutto bisogna avviare Vim per ogni file da modificare. C'è un modo
semplice. Per iniziare a lavorare su di un altro file usate questo comando: >

```
:edit foo.txt
```

Ovviamente potete utilizzare un qualunque altro nome di file invece di
"foo.txt". Vim chiuderà il file attualmente aperto ed aprirà quello nuovo.
Se il file attualmente aperto avesse delle modifiche non ancora salvate,
comunque, Vim mostrerebbe un messaggio di errore e non aprirebbe il nuovo
file:

```
E37:    No write since last change (use ! to override) ~
        Non salvato dopo l'ultima modifica (usare ! per procedere
        comunque)
```

Note:
Vim fornisce un codice identificativo di errore prima di ciascun
messaggio di errore. Se non doveste comprendere questo messaggio o da
cosa esso fosse causato, vedere nel sistema di aiuto cercando tale
codice identificativo. Nel caso presente: >

```
:help E37
```

Ora vi sono diverse alternative. Potete salvare il file con il comando: >

```
:write
```

Ovvero forzare Vim ad ignorare i cambiamenti ed aprire comunque il secondo
file, usando il carattere (!): >

```
:edit! foo.txt
```

Se voleste modificare un altro file ma non salvare subito le modifiche
dell'attuale, potete nasconderlo: >

```
:hide edit foo.txt
```

Il testo modificato è ancora lì, ma non potete vederlo.
Ciò verrà spiegato più avanti nella sezione |22.4|: La lista dei buffer

==
07.2 Una lista di file

Potete avviare Vim per modificare un gruppo di file. Ad esempio: >

```
vim one.c two.c three.c
```

Questo comando avvia Vim e gli dice che verranno modificati tre file.
   Vim fa vedere solo il primo file. Dopo avere effettuato le dovute
modifiche, per passare al prossimo file usate il comando: >

```
:next
```

Se ci fossero modifiche non salvate nel file corrente, otterreste un messaggio di errore ed il comando ":next" non funzionerebbe. E' esattamente come avveniva con il comando ":edit", descritto nella precedente sezione. Per lasciar perdere le modifiche: **>**

```
:next!
```

Ma di solito vorrete salvare le modifiche prima di passare al prossimo file. Ecco un comando speciale per ottenere ciò: **>**

```
:wnext
```

Questo equivale ad usare due distinti comandi: **>**

```
:write
:next
```

DOVE MI TROVO?

Per vedere quale file della lista sia attualmente in corso di modifica, guardate il titolo della finestra. Potrebbe esserci qualcosa come "(2 of 3)".
   Ciò significa che state modificando il secondo file di una lista che ne comprende tre.
   Se volete vedere l'elenco dei file, usate il comando: **>**

```
:args
```

E' l'abbreviazione di "arguments". Il risultato potrebbe assomigliare a questo: **>**

```
one.c [two.c] three.c
```

Sono i file che sono stati aperti con Vim. Quello su cui state lavorando attualmente, "two.c", è racchiuso tra parentesi quadre.

SPOSTARSI SU ALTRI ARGOMENTI

Per tornare indietro di un file: **>**

```
:previous
```

E' esattamente come il comando ":next", eccetto che va nella direzione opposta. Inoltre, ecco un comando abbreviato per salvare il file prima di muovervi a quello precedente: **>**

```
:wprevious
```

Per andare all'ultimo file della lista: **>**

```
:last
```

E per tornare al primo: **>**

```
:first
```

Però non esistono i comandi ":wlast" o ":wfirst"!

Potete usare un contatore per specificare di quanti file spostarvi con i comandi ":next" e ":previous". Per saltare avanti di due file: **>**

```
:2next
```

SALVATAGGIO AUTOMATICO

Spostandovi tra i file e modificandoli, dovete rammentare di impiegare ":write". Altrimenti apparirà il solito messaggio di errore. Se siete certi di voler salvare comunque le modifiche, potete indicare a Vim di salvare automaticamente, in questo modo: **>**

```
:set autowrite
```

Nel caso in cui invece stiate modificando un file che potreste decidere di non salvare, disattivate il salvataggio automatico con: **>**

```
:set noautowrite
```

LAVORARE CON UN'ALTRA LISTA DI FILE

Potete ridefinire la lista dei file senza uscire da Vim e doverlo poi riavviare. Per editare altri tre file, usate questo comando: **>**

```
:args five.c six.c seven.h
```

Oppure con una wildcard, come nei comandi di shell: **>**

```
:args *.txt
```

Vim si porterà sul primo file dell'elenco. Se il file attuale avesse subito modifiche, potete prima salvarlo, oppure usare ":args!" (con aggiunto il !) per tralasciare le modifiche.

PER MODIFICARE L'ULTIMO FILE?

arglist-quit

Quando si lavora con un elenco di file, Vim prevede che essi debbano essere modificati tutti.
Per prevenire l'eventualità di uscire intempestivamente, nel caso non abbiate raggiunto l'ultimo file dell'elenco, vi verrà mostrato il seguente messaggio di errore: **>**

```
E173: 46 more files to edit
```

Se intendete uscire comunque, ripetete il comando. In questo modo, funzionerà (ovviamente, solo se non saranno stati immessi altri comandi nel frattempo).

==

07.3 Saltare da file a file

Per spostarvi rapidamente tra due file, premete CTRL-^ (nelle tastiere English-US il ^ si trova sopra il tasto del 6, in quelle italiane sopra il tasto della "i accentata"). Ad esempio: **>**

```
:args one.c two.c three.c
```

Attualmente siamo in one.c. **>**

```
:next
```

Adesso ci si trova in two.c. Ora con CTRL-^ si torna a one.c. Un altro CTRL-^ e nuovamente si è in two.c. Ancora CTRL-^ e ci si trova di nuovo in one.c. Se adesso si scrive: **>**

```
:next
```

ci si trova in three.c. Notate come il comando CTRL-^ non cambi l'idea di dove ci si trovi nell'elenco dei file. Solo comandi come ":next" e ":previous" lo fanno.

Il file precedentemente modificato viene chiamato file "alternate". Se si fosse appena avviato Vim, CTRL-^ non funzionerebbe, perchè non esiste un file precedente.

MARCATORI PREDEFINITI

Dopo essere passati ad un altro file, possono essere impiegati due marcatori predefiniti molto utili: **>**

```
`"
```

Ciò riposizionerà il cursore nella posizione che aveva prima che si fosse lasciato quel file. Un altro marcatore ricorda la posizione dove è avvenuta l'ultima modifica: **>**

```
`.
```

Immaginate di lavorare con il file "one.txt". Avete usato il tasto "x" per cancellare un carattere. Poi vi siete spostati sull'ultima riga con "G" e avete salvato il file con ":w". In seguito modificate molti altri file, e poi digitate ":edit one.txt" per ritornare a "one.txt". Se ora usate `" Vim

tornerà all'ultima linea del file. Usando `. tornerete alla posizione dove il carattere era stato cancellato.  Allo stesso modo, spostandovi attraverso il file, `" e `. vi riporteranno nella posizione ricordata. Almeno sino a quando verrà effettuata un'altra modifica o si chiuderà il file.

MARCATURA DI UN FILE

Nel capitolo 4 è stato spiegato come porre un marcatore in un file con "mx" e saltare a quella posizione con "`x". Ciò funziona per un solo file. Se si lavorasse con un altro file e si mettessero marcatori in esso, questi sarebbero specifici per questo file.
   Sinora sono stati usati marcatori con una lettera minuscola. Possono esservi anche marcatori con lettere maiuscole. Questi sono globali, possono essere usati da qualsiasi file. Ad esempio, immaginate di lavorare con il file "foo.txt". Spostatevi a metà del file ("50%") e ponete lì il marcatore F (F per foo): **>**

```
50%mF
```

Adesso modificate il file "bar.txt" e mettete il marcatore B (B per bar) alla sua ultima linea: **>**

```
GmB
```

Adesso potete impiegare il comando "'F" per tornare a metà di foo.txt. Oppure editare un altro file, digitare "'B" e ritrovarvi nuovamente alla fine di bar.txt.

La marcatura dei file verrà ricordata sino a che essa non sia stata posta altrove. Così si può mettere il marcatore, lavorare per ore e poter ancora tornare a quel marcatore.
   E' spesso utile pensare ad un collegamento tra la lettera usata per segnare ed il posto ove essa si trova. Ad esempio, impiegate il marcatore H nell'header di un file , M in un makefile e C in un file in codice C.

Per vedere dove si trovi un marcatore specifico, dovete fornire un argomento al comando ":marks": **>**

```
:marks M
```

Potete fornire anche più argomenti: **>**

```
:marks MCP
```

Non dimenticate che potete usare CTRL-O e CTRL-I per saltare a posizioni più vecchie o più nuove senza porvi dei marcatori.

==
07.4 File di backup

Di norma Vim non genera file di backup. Se si volesse averne, basta eseguire il seguente comando: **>**

```
:set backup
```

Il nome del file di backup è lo stesso del file originale con una ~ aggiunta in coda. Se il file si chiamasse data.txt, ad esempio, il file di backup si chiamerebbe data.txt~.
   Se non gradiste il fatto che i file di backup terminino con ~, potete cambiare l'estensione: **>**

```
:set backupext=.bak
```

Verrà impiegato data.txt.bak invece di data.txt~.
   Un'altra opzione utile è 'backupdir'. Specifica dove scrivere il file di backup. L'impostazione predefinita, ovvero scrivere il backup nella stessa directory del file originale, potrebbe, nella maggior parte dei casi, essere la cosa giusta.

```
Note:
Se l'opzione 'backup' non fosse impostata ma la 'writebackup' sì,
Vim creerà comunque un file di backup. Tuttavia questo verrà
cancellato quando il file sarà stato salvato.  Ciò funziona come
una garanzia contro la perdita del file originale qualora il
salvataggio fallisse per qualche ragione (il disco pieno è una
delle cause più comuni; un colpo di fulmine potrebbe essere
un'altra causa, sebbene meno frequente).
```

RECUPERARE IL FILE ORIGINALE

Se state editando un file sorgente, potreste voler recuperare il file come era prima delle modifiche effettuate. Ma il file di backup viene sovrascritto ogni qualvolta salvate il file. Così esso conterrà solo la versione precedente e non la prima in assoluto.
   Per ottenere che Vim recuperi il file originale, impostate l'opzione 'patchmode'. Ciò specifica l'estensione usata per il primo backup. Solitamente si farà così: **>**

```
:set patchmode=.orig
```

Quando modificate per la prima volta il file data.txt, operando cambiamenti e salvando il file, Vim tiene una copia del file non modificato con il nome "data.txt.orig".
   Se modificate ancora il file, Vim vi informerà che "data.txt.orig" esiste già e lo lascerà così com'è. I backup successivi verranno chiamati "data.txt~" (o in qualunque altro modo aveste specificato mediante 'backupext').
   Lasciando vuoto 'patchmode' (questo è il default), il file originale non verrà recuperato.

==========

07.5 Copiare testo tra più file

Spieghiamo ora come copiare del testo da un file ad un altro. Cominciamo con un esempio facile. Aprite il file che contiene il testo che volete copiare. Portate il cursore all'inizio del testo e premete "v". Ciò avvia il Visual mode. Spostate ora il cursore alla fine del testo e premete "y" (y sta per "yank", cioè copia). Questo copierà il testo selezionato.

Per copiare il paragrafo precedente potete usare: **>**

```
:edit questofile
/Spieghiamo
vjjjj$y
```

Poi aprite il file dove intendete incollare il testo. Portate il cursore sul carattere dopo il quale volete far apparire il testo. Usate "p" per incollarvi il testo: **>**

```
:edit altrofile
/Qui
p
```

Logicamente si possono usare molti altri comandi per copiare (yank) il testo. Ad esempio, per selezionare intere linee, attivate il Visual mode con "V". Oppure, usate CTRL-V per selezionare un blocco rettangolare. Oppure usate "Y" per copiare una sola linea, "yaw" per copiare una parola (yank-a-word), etc.
   Il comando "p" incolla (puts) il testo dopo il cursore. Usate "P" per incollarlo prima del cursore. Si noti che Vim ricorda se si è copiata qualche linea od un blocco, e lo reincolla allo stesso modo.

USARE I REGISTRI

Se si dovessero copiare molti pezzi di testo da un file ad un altro, il passare da un file all'altro e lo scrivere il file destinazione richiede un mucchio di tempo. Per evitare ciò si può copiare ogni pezzo di testo entro un registro.
   Un registro è un posto dove Vim conserva del testo. Per ora useremo dei registri chiamati da a sino a z (in seguito si scoprirà che ne esistono altri). Copiate una frase nel registro f (f per First): **>**

```
"fyas
```

Il comando "yas" copia (yank) una frase, come visto prima. "f indica a Vim quale parte del testo dovrà essere posta entro il registro f. Deve venire prima del comando yank.
   Ora si copino tre intere linee nel registro l (l per line): **>**

```
"l3Y
```

Il numero di linee da copiare può venire anche prima di "l, se necessario.
  Per copiare un blocco di testo nel registro b (for block): **>**

```
        CTRL-Vjjww"by
```

Notate che la specifica del registro "b va posta proprio prima del comando "y". Ciò è necessario. Se venisse posta dopo di esso, non funzionerebbe.
Ora tre frammenti di testo si trovano entro i registri f, l e b. Aprite un altro file e spostatevi in esso nel punto dove volete porre il testo: **>**

```
        "fp
```

Anche ora la specifica "f del registro viene prima del comando "p".
I registri possono essere incollati in ogni ordine. Ed il testo resterà entro i registri sino a quando non vi verrà copiato qualcosa d'altro. Così lo si potrà incollare tutte le volte che si vorrà.

Quando viene cancellato del testo, si può anche specificare un registro. Ciò può essere usato per spostare del testo altrove. Ad esempio, per cancellare una parola (delete-a-word) e scriverla nel registro w: **>**

```
        "wdaw
```

Come sempre la specifica del registro viene prima del comando di cancellazione (delete) "d".

ACCODARE AD UN FILE

Se si volessero riunire diverse linee di testo entro un solo file, si potrebbe usare il seguente comando: **>**

```
        :write >> logfile
```

Verrà scritto il testo del file corrente in coda a "logfile". Così il testo verrà accodato. Ciò eviterà di dover copiare le linee, aprire il log file ed incollarle in esso. Verranno saltati due passaggi. Ma si può accodare soltanto alla fine del file.
Per accodare solo alcune linee, si selezionino nel Visual mode prima di scrivere ":write". Nel capitolo 10 saranno spiegati altri modi per selezionare gruppi di linee.

==
07.6 Visualizzare un file

Talvolta si vuole solo vedere cosa contenga un file, senza volerlo modificare.
C'è il rischio di scrivere ":w" senza pensarci e sovrascrivere il file originale inavvertitamente. Per evitare ciò, aprite il file in sola lettura (read-only).
Per avviare Vim nel modo readonly, si adoperi il seguente comando: **>**

```
        vim -R file
```

Su Unix il seguente comando farebbe la stessa cosa: **>**

```
        view file
```

Il file "file" è ora aperto nel modo read-only. Se si provasse ad usare ":w" si otterrebbe solo un messaggio di errore ed il file non verrebbe sovrascritto.
Se si provasse ad effettuare una modifica al file, Vim darebbe il seguente avviso:

```
        W10: Warning: Changing a readonly file ~
```

Il cambio però può essere fatto. Ciò permette di formattare il file, per esempio, per poterlo leggere facilmente.
Se venissero effettuate delle modifiche ad un file, scordandosi che esso era read-only, si potrebbe comunque tranquillamente scriverlo. Aggiungete al comando un ! per forzare la scrittura.

Se si volesse realmente proibire di modificare un file bisognerebbe scrivere così: **>**

```
        vim -M file
```

Così qualsiasi tentativo di modifica sarà inutile. I file di help sono così, per esempio. Se si prova a modificarli si ottiene questo messaggio di errore:

```
        E21: Non posso fare modifiche, 'modifiable' è inibito ~
```

Si può usare l'argomento −M per avviare Vim in sola lettura. Ciò però soltanto sino a quando lo si voglia; questi comandi rimuovono la protezione: >

```
:set modifiable
:set write
```

==

07.7 Rinominare un file

Un modo semplice per iniziare a scrivere un nuovo testo è impiegare un file esistente che contenga il più possibile di quanto sia necessario. Ad esempio, si inizi a scrivere un nuovo programma per spostare un file. Sapete già di avere un programma che copia dei file, così iniziate con: >

```
:edit copy.c
```

Cancellate tutto ciò che non vi è necessario. Adesso dovete salvare il file con un nuovo nome. Il comando ":saveas" serve a ciò: >

```
:saveas move.c
```

Vim salverà il file con il nome dato ed aprirà quel file. Così la prossima volta che scriverete ":write", Vim salverà "move.c". "copy.c" rimarrà non modificato.
   Quando volete cambiare il nome del file che avete aperto, ma non salvare il file con il vecchio nome, usate questo comando: >

```
:file move.c
```

Vim segnerà il file come "not edited". Significa che Vim sa che questo non è il file che era stato aperto. Quando salverete il file potreste ottenere questo messaggio:

```
E13: File exists (use ! to override) ~
```

Ciò vi eviterà di sovrascrivere accidentalmente un altro file.

==

Capitolo seguente: |usr_08.txt| Dividere le finestre

Segnalare refusi a Bartolomeo Ravera − E−mail: barrav@libero.it

usr_08.txt   Per Vim version 6.2. Ultima modifica: 2003 Giu 03

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Valentino Squilloni

Dividere le finestre

Poter osservare due diversi file, uno sopra l'altro. Oppure vedere contemporaneamente due diversi punti dello stesso file. Confrontare due file diversi mettendoli l'uno affianco all'altro. Tutto ciò è possibile dividendo in più parti la finestra.

  Capitolo seguente: |usr_09.txt| Usare la GUI
Capitolo precedente: |usr_07.txt| Elaborare più di un file
             Indici: |usr_toc.txt|

==========

08.1 Dividere una finestra

Il modo più semplice per aprire una nuova finestra è quello di usare il seguente comando: **>**

```
:split
```

Questo comando divide lo schermo in due finestre e posiziona il cursore nella finestra più in alto:

```
+----------------------------------+
|/* file one.c */                  |
|~                                 |
|~                                 |
|one.c=============================|
|/* file one.c */                  |
|~                                 |
|one.c=============================|
|                                  |
+----------------------------------+
```

Quello che vedrete qui sono due finestre aperte sullo stesso file. La linea con "====" è la linea di status. Essa ci dà informazioni sulla finestra sovrastante. (In pratica la linea di status si presenterà con colori opposti a quelli dello schermo.)
   Le due finestre permettono di vedere due parti dello stesso file. Ad esempio, sarà possibile far mostrare alla finestra in alto la dichiarazione delle variabili di un programma, ed a quella in basso il codice che utilizza quelle variabili.

Il comando CTRL-W w viene usato per saltare fra le finestre. Se siete nella finestra in alto, CTRL-W w salterà alla finestra sottostante. Se siete nell'ultima finestra (quella più in basso) CTRL-W w salterà alla prima. (CTRL-W CTRL-W fa la stessa cosa, nel caso il tasto CTRL venga rilasciato un pelo troppo tardi).

CHIUDERE LA FINESTRA

Per chiudere una finestra, si usa il comando: **>**

```
:close
```

In realtà, ogni comando che chiude l'editing di un file funzionerà, come ":quit" e "ZZ". Ma ":close" previene l' uscita accidentale da Vim quando si chiude l'ultima finestra.

CHIUDERE TUTTE LE ALTRE FINESTRE

Se avete aperto un mucchio di finestre, ma ora volete concentrarvi solo su una

di esse, questo comando vi sarà utile: >

```
:only
```

Questo chiuderà tutte le finestre, a parte quella corrente (quella dove è presente il cursore). Se qualcuna delle altre finestre ha avuto dei cambiamenti, avrete un errore e quelle finestre non verranno chiuse.

==
08.2 Dividere una finestra aprendo un altro file

Il comando seguente apre una seconda finestra e inizia ad editare in essa il file dato: >

```
:split two.c
```

Se state editando one.c, allora il risultato del comando sarà circa così:

```
+----------------------------------+
|/* file two.c */                  |
|~                                 |
|~                                 |
|two.c=============================|
|/* file one.c */                  |
|~                                 |
|one.c=============================|
|                                  |
+----------------------------------+
```

Per aprire una finestra su un nuovo file vuoto, usate questo: >

```
:new
```

Potete ripetere i comandi ":split" e ":new" per creare quante finestre volete.

==
08.3 Dimensioni della finestra

Il comando ":split" può avere un argomento numerico. Se viene specificato, questo sarà l'altezza della nuova finestra. Per esempio, il comando seguente apre una nuova finestra alta tre linee e inizia a editarvi il file alpha.c: >

```
:3split alpha.c
```

Per le finestre già esistenti potete cambiarne le dimensioni in diversi modi. Se avete un mouse funzionante, è facile: basta muovere il cursore sulla linea di status che separa le due finestre, e trascinarla in alto o in basso.

Per aumentare la dimensione di una finestra: >

```
CTRL-W +
```

Per diminuirla: >

```
CTRL-W -
```

Entrambi i comandi prendono un numero e aumentano o diminuiscono l'altezza della colonna di quel numero di linee. In questo modo "4 CTRL-W +" da sì che la finestra diventi 4 linee più alta.

Per impostare l'altezza della finestra ad un numero specificato di linee: >

```
{height}CTRL-W _
```

Cioè: un numero {heigh}, CTRL-W e un underscore (il tasto - con Shift premuto sulle tastiere English-US).
   Per rendere una finestra più alta possibile, si usa il comando CTRL-W _ senza un numero prima.

USARE IL MOUSE

In Vim potete fare un sacco di cose molto velocemente direttamente con la tastiera. Sfortunatamente, i comandi per ridimensionare la finestra richiedono numerose pressioni dei tasti. In questo caso, usare il mouse è più veloce. Posizionate il mouse sulla linea di status. Ora premete il tasto sinistro del mouse e trascinate. La linea di status si muoverà, facendo

diventare la finestra da una parte piu alta e quella dall'altra più bassa.

OPZIONI

L'opzione 'winheight' è usata per impostatare l'altezza minima desiderata di una finestra e 'winminheight' per l'altezza minima consentita di una finestra.
   Allo stesso modo c'è 'winwidth' per la larghezza minima desiderata e 'winminwidth' per la minima larghezza consentita.
   L'opzione 'equalalways', quando è impostata, fa si che Vim aggiusti le altezze delle finestre quando viene chiusa o aperta una finestra per uniformarle.

==
08.4 Tagli verticali

Il comando ":split" crea la finestra nuova sopra a quella corrente. Invece per far comparire la nuova finestra alla sinistra di quella già presente, si usa: **>**

```
:vsplit
```

oppure: **>**

```
:vsplit two.c
```

Il risultato sarà simile a questo:

```
+----------------------------------------+
|/* file two.c */   |/* file one.c */   |
|~                  |~                  |
|~                  |~                  |
|~                  |~                  |
|two.c===============one.c=============|
|                                        |
+----------------------------------------+
```

In pratica, le colonna di | al centro dello schermo sarà in colori opposti a quelli dello schermo. Viene chiamata separatore verticale. Serve a separare le due finestre alla destra e alla sinistra di essa.

C'è anche il comando ":vnew", che taglia verticalmente la finestra aprendo un nuovo file vuoto. Un altro modo di ottenere ciò: **>**

```
:vertical new
```

Il comando ":vertical" può esser inserito prima di un altro comando che divide una finestra. Questo farà sì che quel comando divida la finestra in verticale invece che in orizzontale. (Se il comando non divide finestre, non verrà influenzato da esso).

MUOVERSI FRA LE FINESTRE

Poichè si può dividere più volte la finestra in orizzontale e in verticale, è possibile creare un qualsiasi tipo di schema. Potete usare questi comandi per muovervi fra le finestre:

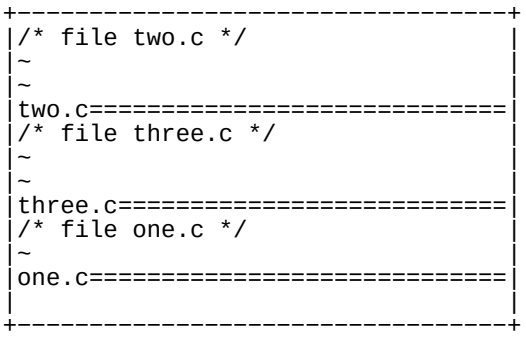
```
CTRL-W h        muove verso la finestra a sinistra
CTRL-W j        muove verso la finestra di sotto
CTRL-W k        muove verso la finestra di sopra
CTRL-W l        muove verso la finestra a destra

CTRL-W t        muove verso la finestra più in alto
CTRL-W b        muove verso la finestra più in basso
```

Noterete che le stesse lettere sono usate per muovere il cursore. E anche le frecce possono essere usate, se preferite.
   Ulteriori comandi per muoversi verso altre finestre: |Q_wi|

==
08.5 Muovere le finestre

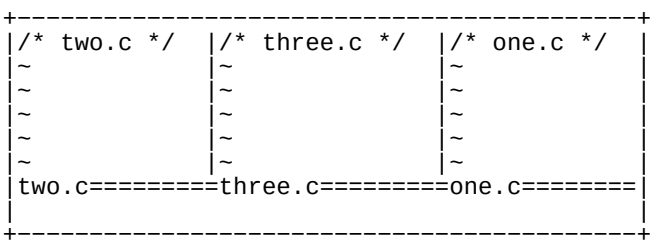
Avete tagliato alcune finestre, ma ora sono nel posto sbagliato. Quindi avrete bisogno di un comando per muovere la finestra in un altro posto. Ad esempio, avete tre finestre messe in questo modo:



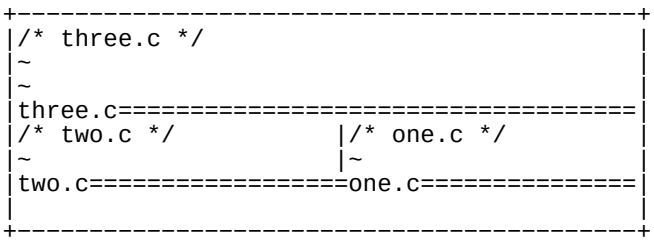
Chiaramente l'ultima in basso dovrebbe stare nel posto più in alto. Allora muovetevi verso quella finestra (usando CTRL-W w) e usate questo comando: **>**

```
CTRL-W K
```

Questo comando usa la lettera maiuscola K. Quello che succede è che la finestra in questione viene spostata nel punto più alto. Noterete che K è usata ancora per muoversi verso l'alto.
Se ci sono dei tagli verticali, CTRL-W K muoverà la finestra corrente nel punto più in alto e le farà occupare l'intera larghezza della finestra di Vim. Se questo è il vostro layout:



Allora usando CTRL-W K sulla finestra al centro (three.c) avrà come risultato:



Gli altri tre comandi simili (e probabilmente ora già ve li immaginate) sono:

```
CTRL-W H        muove la finestra nel punto più a sinistra
CTRL-W J        muove la finestra nel punto più in basso
CTRL-W L        muove la finestra nel punto più a destra
```

==========

08.6 Comandi per tutte le finestre

Quando avete molte finestre aperte e volete uscire da Vim, potete chiudere separatamente ogni finestra. Un modo più veloce per fare la stessa cosa è il comando: **>**

```
:qall
```

Che sta per "quit all". Se qualche finestra contiene delle modifiche, Vim non uscirà. Il cursore sarà automaticamente posizionato in una di queste finestre che hanno subito modifiche. Potete usare ":write" per salvare i cambiamenti fatti, oppure ":quit!" per buttarli via.

Se sapete che ci sono più finestre da salvare, e le volete salvare tutte, usate questo comando: **>**

```
:wall
```

Che sta per ":write all". In realtà va a scrivere solo quei file che hanno subito delle modifiche. Vim sa che non ha senso scrivere un file che non è

stato modificato.
   C'è anche la combinazione di ":qall" e ":wall": il comando "write and quit all": **>**

```
:wqall
```

Che salva tutti i file modificati e chiude Vim.
   Infine, c'è un comando per chiudere Vim e ignorare tutti i cambiamenti fatti: **>**

```
:qall!
```

Attenzione, non c'è il modo di annullare questo comando!

APRIRE UNA FINESTRA PER OGNI ARGOMENTO

Per far sì che Vim apra una finestra per ogni file, lo si fa partire con l'argomento "−o": **>**

```
vim −o one.txt two.txt three.txt
```

Il risultato sarà:

```
+-------------------------------+
|file one.txt                   |
|~                              |
|one.txt========================|
|file two.txt                   |
|~                              |
|two.txt========================|
|file three.txt                 |
|~                              |
|three.txt======================|
|                               |
+-------------------------------+
```

L'argomento "−O" è usato per avere finestre separate verticalmente.
   Quando Vim è già in esecuzione, il comando ":all" apre una finestra per ogni file nella lista degli argomenti. ":vertical all" fa la stessa cosa ma con tagli verticali.

==
08.7 Evidenziare le differenze con vimdiff

C'è un modo speciale di far partire Vim, che mostra le differenze fra due files. Prendiamo ad esempio il file "main.c" e inseriamo qualche carattere in una linea. Salviamo il file con l'opzione 'backup' attiva, così che il file di backup "main.c~" conterrà la versione precedente del file.
   Scrivete questo comando in una shell (non in Vim): **>**

```
vimdiff main.c~ main.c
```

Vim partirà con due finestre una al fianco dell'altra. Voi vedrete solo la linea in cui avete aggiunto i caratteri, e qualche linea sopra e sotto di essa.

```
 VV                    VV
+-----------------------------------------+
|+ +--123 lines: /* a|+ +--123 lines: /* a|  <- fold
|  text              |  text              |
|  text              |  text              |
|  text              |  text              |
|  text              |  changed text      |  <- changed line
|  text              |  text              |
|  text              |  ------------------|  <- deleted line
|  text              |  text              |
|  text              |  text              |
|  text              |  text              |
|+ +--432 lines: text|+ +--432 lines: text|  <- fold
|  ~                 |  ~                 |
|  ~                 |  ~                 |
|main.c~==============main.c==============|
|                                         |
+-----------------------------------------+
```

(Questa figura non mostra le parti evidenziate, usate direttamente vimdiff per una resa migliore.)

Le linee che non erano state modificate sono collassate un una riga. Questo è chiamato una piega chiusa (closed fold). Nella figura sono indicate con "<- fold". Sebbene la singola linea piegata rappresenta 123 linee di testo, queste linee sono uguali in entrambi i file.

La linea contrassegnata con "<- changed line" è evidenziate, e il testo inserito viene visualizzato in un altro colore. Questo mostra in modo chiaro le differenze fra i due file.

La linea che è stata cancellata è mostrata con "---" nella finestra di main.c. Notate il contrassegno "<- deleted line" nella figura. Questi caratteri non sono veramente lì. Servono solo a riempire main.c, in modo che mostri lo stesso numero di linee dell'altra finestra.

LA COLONNA FOLD

Ogni finestra ha una colonna sulla sinistra con lo sfondo leggermente differente. Nella figura sopra queste colonne sono indicate con "VV". Noterete un carattere "+" là, alla testa di ogni piega chiusa. Muovete il mouse su quel "+" e cliccate col bottone sinistro. La piega si distenderà, e potrete vederre il testo che contiene.

La colonna fold contiene un segno di "-" per ogni piegatura aperta. Se cliccate su quel simbolo meno, il fold si chiuderà.

Ovviamente questo sarà possibile solo se avete un mouse funzionante. Altrimenti, potete comunque usare "zo" per aprire una piegatura, e "zc" per chiuderla.

EVIDENZIARE LE DIFFERENZE IN VIM

La modalità diff può essere attivata in un altro modo, direttamente da dentro a Vim. Editate il file "main.c", poi dividete la finestra e mostrate le differenze: **>**

```
:edit main.c
:vertical diffsplit main.c~
```

Il comando ":vertical" viene usato per far sì che la finestra venga tagliata verticalmente. Se lo omettete, avrete un taglio orizzontale.

Se avete un file patch o diff, potete usare un terzo modo per far partire la modalità diff. Innanzitutto editate il file a cui le patch andranno applicate. Poi dite a Vim il nome del file di patch: **>**

```
:edit main.c
:vertical diffpatch main.c.diff
```

ATTENZIONE: Il file patch deve contenere solo una patch per il file che state editando. Altrimenti avrete molti messaggi di errore, e alcuni files potrebbero essere patchati in modo inaspettato.

La patch verrà applicata solo al file all'interno di Vim. Il file sul vostro hard disk resterà immutato (fino a che deciderete di salvare il file).

TENERE UNITO LO SCROLL

Quando i file hanno più di una variazione, potete scrollare nel solito modo. Vim cercherà di tenere allineato l'inizio di entrambe le finestre , per mostrare semplicemente le differenze, lato a lato.

Se volete disabilitare temporaneamente questa caratteristica, usate questo comando: **>**

```
:set noscrollbind
```

SALTARE ALLE DIFFERENZE

Se in qualche maniera avete disabilitato la funzione di piegatura, potrebbe essere difficile trovare le differenze. Usate questo comando per saltare in avanti alla prossima differenza: **>**

```
]c
```

Per andare nell'altra direzione (verso l'alto) usare : **>**

```
[c
```

Mettere come prefisso un numero per saltare più volte.

ELIMINARE LE DIFFERENZE

Potete spostare delle parti di testo da una finestra all'altra. Questo fa sì che siano rimosse le differenze, oppure che siano aggiunte le parti nuove. Vim non tiene aggiornate le differenze in ogni occasione. Per aggiornarle, usate questo comando: **>**

```
:diffupdate
```

Per eliminare una differenza, potete spostare il testo all'interno di un blocco evidenziato da una finestra all'altra. Prendete ad esempio i file "main.c" e "main.c~" citati precedentemente. Spostate il cursore sulla finestra di sinistra, sopra alla linea che nell'altra finestra era stata cancellata. Ora usate questo comando: **>**

```
dp
```

Le differenze verranno rimosse mettendo il testo della finestra corrente nell'altra finestra. "dp" sta per "diff put" (inserisci la differenza).
Potete anche fare la stessa cosa in un altra maniera. Spostate il cursore nella finestra di destra, sulla linea dove era stati inseriti dei "cambiamenti". Ora usate questo comando: **>**

```
do
```

Le differenze in questo caso saranno rimosse prendendo la parte di testo dall'altra finestra. Poichè a questo punto non ci saranno più differenze, Vim racchiuderà tutto il testo in una piega chiusa. "do" sta per "diff obtain" (prendi le differenze). "dg" sarebbe stato meglio, ma aveva già un significato diverso ("dgg" cancella tutto dal cursore fino alla prima linea compresa).

Per maggiori dettagli sulla modalità diff, vedere |vimdiff|.

==
08.8 Varie ed eventuali

L'opzione 'laststatus' può essere usata per specificare se e quando l'ultima finestra debba avere la linea di status:

```
0       mai
1       solo quando ci sono finestre divise (default)
2       sempre
```

Molti comandi che editano un altro file hanno una variante che divide la finestra. Per i comandi nella linea dei comandi questo viene fatto preponendo una "s". Ad esempio: ":tag" salta ad un certo punto contrassegnato, ":stag" divide in due la finestra e salta a quel punto.
Per i comandi in modalità normale viene preposta la combinazione CTRL-W. CTRL-^ salta al file alternato, CTRL-W CTRL-^ divide in due la finestra ed edita il file alternato.

L'opzione 'splitbelow' può essere impostata per far sì che la finestra nuova appaia sotto alla finestra corrente. L'opzione 'splitright' analogamente fa sì che appaia alla destra della finestra corrente.

Quando si divide in due la finestra si può premettere al comando un modificatore per decidere dove apparirà la finestra:

```
:leftabove {cmd}     a sinistra o sopra la finestra corrente
:aboveleft {cmd}     idem
:rightbelow {cmd}    a destra o sotto la finestra corrente
:belowright {cmd}    idem
:topleft {cmd}       nel punto più in alto o a sinistra della
                     finestra di Vim
:botright {cmd}      nel punto più in alto o alla destra della
                     finestra di Vim
```

==

Capitolo seguente: |usr_09.txt| Usare la GUI

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

usr_09.txt   Per Vim version 6.1. Ultima modifica: 2001 Set 03

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Giorgio Luciano

Usare la GUI

Vim funziona come un normale terminale. GVim può fare le stesse cose e qualcosina di più. La GUI offre dei menù, una barra degli strumenti, delle barre di scorrimento ed altri elementi. Questo capitolo è dedicato a tutte queste cose extra che la GUI offre.

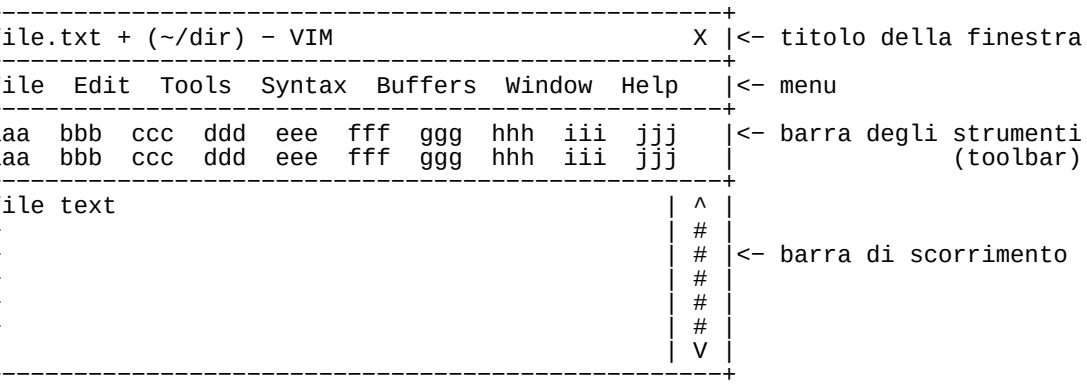
  Capitolo seguente: |usr_10.txt| Fare grandi modifiche
Capitolo precedente: |usr_08.txt| Dividere le finestre
            Indice: |usr_toc.txt|

==
09.1 Parti dell'interfaccia grafica (GUI)

Potreste avere sul desktop un icona per avviare gVim. Altrimenti lo potrebbe fare uno di questi comandi: **>**

```
gvim file.txt
vim -g file.txt
```

Se non funzionasse voi non avete una versione di Vim con interfaccia grafica. Dovete installarne una prima.
   Vim aprirà una finestra e vi farà vedere il "file.txt" in essa. Come appaia la finestra dipende dalla versione di Vim. Potrebbe assomigliare all'immagine seguente (per quanto ciò possa essere visualizzato in ASCII!).



La maggior parte dello spazio a disposizione viene occupato dal file di testo. Questo mostra il file allo stesso modo che in un terminale. Forse con qualche colore diverso ed un altro font.

TITOLO DELLA FINESTRA

Nel punto più alto viene mostrato il titolo della finestra. Viene disegnato dal vostro window system. Vim imposterà il titolo per mostrare il nome del file corrente. Prima viene il nome del file. Poi alcuni caratteri speciali ed il nome della directory tra parentesi. Possono essere presenti questi caratteri speciali:

-	Il file non può essere modificato (i.e., il file di help)
+	Il file contiene dei cambiamenti
=	il file è di sola lettura (read-only)
=+	Il file è di sola lettura e contiene tuttavia dei cambiamenti

Se non viene visualizzato nulla avete un file comune e non modificato.

LA BARRA DEL MENU'

Sapete come funziona un menù, vero? Vim ha i soliti elementi, più qualcosa

d'altro. Esploratelo per avere un'idea di come possiate adoperarlo. Un sottomenù importante è Edit/Global Settings. Dovreste trovare questi comandi:

Toggle Toolbar	per visualizzare o nascondere la barra degli strumenti (toolbar)
Toggle Bottom Scrollbar	per visualizzare o nascondere la barra di scorrimento (scrollbar appear/disappear) in fondo
Toggle Left Scrollbar	per visualizzare o nascondere la barra di scorrimento (scrollbar appear/disappear) a sinistra
Toggle Right Scrollbar	per visualizzare o nascondere la barra di scorrimento (scrollbar appear/disappear) a destra

Nella maggior parte dei sistemi potrete staccare i menù. Selezionate il primo elemento del menù, quello che appare come una linea tratteggiata. Otterrete una finestra separata con gli elementi del menù. Rimarrà appesa sino a quando chiuderete la finestra.

LA BARRA DEGLI STRUMENTI (toolbar)

Contiene le icone dei comandi usati più di frequente. Si spera che le icone siano autoesplicanti. Ci sono i tooltips per ottenere informazioni aggiuntive (muovete il puntatore del mouse su un icona senza cliccare e aspettate per un secondo).

Gli elementi del menù "Edit/Global Settings/Toggle Toolbar" possono essere usati per far scomparire la barra degli strumenti (toolbar). Se non volete la barra degli strumenti usate questo comando nel vostro file vimrc: **>**

```
:set guioptions-=T
```

Ciò rimuove l'attributo 'T' dall'opzione 'guioptions'. Anche altre parti della GUI possono venir abilitate o disabilitate con questa opzione. Consultate l'help per ciò.

LE BARRE DI SCORRIMENTO (scrollbars)

Di default c'è una barra di scorrimento sulla destra. E' la cosa più ovvia. Quando dividete la finestra, ogni singola finestra avrà al propria scrollbar.
   Potete far apparire una scrollbar orizzontale con l'elemento del menù Edit/Global Settings/Toggle Bottom Scrollbar. Ciò è utile in diff mode, o quando l'opzione 'wrap' fosse stata reimpostata (ne parleremo più avanti).

Quando ci sonofinestre affiancate verticalmente, solo la finestra di destra avrà la scrollbar. Comunque, quando muovete il cursore sulla finestra a sinistra, sarà questa ad essere controllata dalla scrollbar. Ciò richiede un po' di tempo per essere effettuato.
   Quando lavorate con finestre separate verticalmente, considerate di aggiungere una scrollbar sulla sinistra. Ciò può essere fatto con un elemento del menù, o con l'opzione 'guioptions': **>**

```
:set guioptions+=l
```

Questo aggiunge l'attributo 'l' alle 'guioptions'.

==========

09.2 Usare il mouse

Gli standard sono meravigliosi. In Microsoft Windows, potete usare il mouse per selezionare in modo standard. Anche X Window ha un sistema standard per usare il mouse. Sfortunatamente questi due standard non sono lo stesso.
   Fortunatamente potete personalizzare Vim. Potete ottenere che il mouse appaia funzionare come un mouse X Window system od un mouse Microsoft Windows. Il seguente comando e farlo funzionare allo stesso modo di Windows.

Potete scegliere di usare i seguenti comandi
Per avere il funzionamento standard di X Window : **>**

```
:behave xterm
```

Il seguente comando farà alvorare il mouse come in Microsoft Windows : **>**

```
:behave mswin
```

Il comportamento di default del mouse sui sistemi UNIX è xterm. Il comportamento di default su un sistema Microsoft Windows viene selezionato durante il processo di installazione. Per i dettagli su cosa siano i due comportamenti vedere |:behave|. Qui segue un sommario.

COMPORTAMENTO DEL MOUSE XTERM

click tasto sinistro	posizione del cursore
trascinamento tasto sinistro	selezione in in Visual mode
click tasto centrale	incolla il testo dalla clipboard
click tasto destro	estende la selezione del testo fino al puntatore del mouse

COMPORTAMENTO DEL MOUSE MSWIN

click tasto sinistro	posizione del cursore
trascinamento tasto sinistro	seleziona testo in Select mode (v. |09.4|)
click tasto sinistro, con shift	estende la selezione del testo fino al puntatore del mouse
click tasto centrale	incolla il testo dalla clipboard
click tasto destro	visualizza un menò a pop-up

Il mouse può venire ulteriormente messo a punto. Provate queste opzioni se volete cambiare il modo in cui funziona il vostro mouse:

'mouse'	in che modo il mouse è usato da Vim
'mousemodel'	che effetto ha il click del mouse
'mousetime'	intervallo di tempo per il doppio click
'mousehide'	nasconde il mouse durante la digitazione
'selectmode'	per selezionare se il mouse parta in Select o Visual mode

==
09.3 Appunti (clipboard)

Nella sezione |04.7| era stato spiegato l'utilizzo fondamentale della clipboard. C'è una cosa essenziale da spiegare su X-window: Ci sono davvero due posti per scambiare del testo tra programmi. MS-Windows non ha ciò.

In X-Windows c'è la "current selection". Questo è il testo attualmente evidenziato. In Vim questa è la Visual area (si suppone che stiate usando le opzioni di default). Potete incollare questa selezione senza ulteriori azioni.
   Per esempio, in questo testo selezione alcune parole col mouse. Vim passerà al Visual mode ed evidenzierà il testo. Ora aprite un altro gVim senza l'argomento filename, in modo che appaia una finestra vuota. Cliccate il bottone centrale del mouse. Il testo selezionato verrà inserito.

La "current selection" rimarrà valida finchè non verrà selezionato altro testo. Dopo avere incollato nell'altro gVim, adesso selezionate qualche carattere su questa finestra. Vi accorgerete che le parole che erano state prima selezionate nell'altra finestra di gVim vengono mostrate in modo diverso. Ciò significa che non si tratta più della selezione corrente.

Non siete costretti a selezionare il testo col mouse, usare i comandi della tastiera per il Visual mode funziona proprio nello stesso modo.

LA VERA CLIPBOARD

Ed ora l'altro posto con cui si può scambiare il testo. Lo chiameremo "vera clipboard" per non fare confusione. Spesso sia la "current selection" che la "vera clipboard" vengono chiamate clipboard, dovrete abituarvici.
   Per mettere del testo nella vera clipboard, selezionate alcune parole in uno dei gVim che avete avviato. Ora usate la voce di menu Edit/Copy. Il testo viene copiato nella vera clipboard. Non potete vederlo se non avete qualche applicazione che mostri i contenuti della clipboard (i.e., klipper di KDE).
   Ora scegliete l'altro gVim, posizionate il cursore da qualche parte ed usate il menù Edit/Paste Vedrete che il testo viene inserito dalla vera clipboard.

USARLE ENTRAMBE

L'uso di una "current selection" e di una "vera clipboard" insieme può

sembrare un po' confusionario. Ma è molto utile. Mostriamolo con un esempio.
Usate un solo gVim con un file di testo ed eseguite queste operazioni:

- Selezione due parole in Visual mode.
- Usate il menù Edit/Copy per mettere queste parole nella clipboard.
- Selezione un'altra parola in Visual mode.
- Usate l'elemento di menù Edit/Paste. Dovrebbe succedere che la singola parola selezionata venga rimpiazzata con le due parole dalla clipboard.
- Muovete il puntatore del mouse e premete il tasto centrale. Vedrete che la parola che avete appena sovrascritto con la clipboard viene inserita qui.

Se usate "current selection" e la "vera clipboard" con attenzione, potete fare molto utile editing con esse.

USARE LA TASTIERA

Se non vi piace usare il mouse, potete accedere alla current selection ed alla "vera clipboard" con due registri. Il registro "* è per la selezione corrente.
   Per far diventare il testo current selection, usate il Visual mode. Per esempio, per selezionare l'intera linea digitate "V".
   Per inserire la current selection prima del cursore digitate: **>**

```
"*P
```

Notate la maiuscola "P". La minuscola "p" mette il testo dopo il cursore.

Il registro "+ è usato per la vera clipboard. Per esempio, per copiare del testo dalla posizione del cursore fino alla fine nella clipboard: **>**

```
"+y$
```

Ricordate, "y" sta per yank, che è il comando per copiare in Vim.
   Per inserire i contenuti della vera clipboard prima del cursore: **>**

```
"+P
```

E' lo stesso che per la "current selection", ma usa il registro (+) invece di quello (*).

==
09.4 Selezioni (Select mode)

E ora quacosa che viene usato più spesso su MS-Windows che in X-Windows.
Ma entrambi lo possono fare. Avete già visto del Visusal mode. Il Select mode è simile al Visual mode, perchè viene anch'esso usato per selezionare del testo. Ma qui c'è però una ovvia differenza: Mentre digitate il testo, il testo selezionato viene cancellato ed il testo battuto lo sostituisce.

Per iniziare a lavorare in Select mode, dovete prima abilitarlo (in MS-Windows è probabilmente sempre abilitato, ma potete farlo comunque): **>**

```
:set selectmode+=mouse
```

Ora usate il mouse per selezionare del testo. E' evidenziato come in Visual mode. Adesso premete una lettera. Il testo selezionato viene cancellato, e la singola lettera lo rimpiazza. Siete nell'Insert mode adesso, così potete continuare a battere.

Poichè battere del testo normale causa la cancellazione del testo selezionato, voi non potete usare i normali comandi di movimento "hjkl", "w", etc. Invece dovrete usare i tasti di funzione shiftati. <S-Left> (shitf sinistro) muove il cursore a sinistra. Il testo selezionato viene cambiato come in Visual mode. Gli altri tasti cursore shiftati fanno ciò che vi attendete. Anche <S-End> e <S-Home> funzionano.

Potete affinare la modalità di lavoro di Select mode con l'opzione 'selectmode'.

==

Capitolo seguente: |usr_10.txt| Fare grandi modifiche

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

usr_10.txt   Per Vim version 6.2. Ultima modifica: 2003 Ott 06

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

Fare grandi modifiche

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

  Capitolo seguente: |usr_11.txt| Recupero dopo un blocco
Capitolo precedente: |usr_09.txt| Usare la GUI
            Indice: |usr_toc.txt|

==
10.1 Registrare e rieseguire comandi

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

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

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

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

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

E quello che volete ottenere è:

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

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

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

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

```
3@a
```

SPOSTARSI ED ESEGUIRE

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

USARE I REGISTRI

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

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

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

AGGIUNGERE AD UN REGISTRO

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

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

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

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

```
"aY
```

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

```
"AY
```

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

==========

10.2 Sostituzione *find-replace*

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

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

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

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

<

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

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

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

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

in:

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

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

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

produce (a partire dalla riga originale):

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

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

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

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

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

A questo punto potete rispondere con una delle seguenti lettere:

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

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

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

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

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

==========

10.3 Ambito di applicazione dei comandi

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

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

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

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

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

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

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

```
:.write altrofile
```

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

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

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

USARE UNO SCHEMA DI RICERCA IN UN INTERVALLO

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

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

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

AGGIUNTE E SOTTRAZIONI

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

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

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

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

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

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

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

USARE I SEGNALIBRI

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

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

VISUAL MODE E INTERVALLI

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

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

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

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

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

```
:'>,$
```

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

UN CERTO NUMERO DI RIGHE

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

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

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

==========

10.4 Il comando global

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

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

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

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

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

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

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

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

==========

10.5 Visual block mode

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

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

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

INSERIMENTO DI TESTO

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

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

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

```
Imain.<Esc>
```

Il risultato è:

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

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

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

          ^^^^ blocco selezionato
```

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

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

Nella riga corta non è stato inserito alcun testo.

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

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

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

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

MODIFICARE IL TESTO

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

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

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

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

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

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

Altri comandi che cambiano i caratteri nel blocco sono:

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

RIEMPIRE CON UN CARATTERE

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

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

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

SPOSTAMENTO

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

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

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

```
:set shiftwidth=4
```

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

UNIRE LE RIGHE

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

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

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

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

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

==========

10.6 Leggere e scrivere parte di un file

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

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

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

```
:read patch
```

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

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

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

```
:0read patch
```

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

SCRIVERE UN GRUPPO DI RIGHE

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

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

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

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

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

AGGIUNGERE AD UN FILE

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

```
:.write raccolta
```

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

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

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

==========

10.7 Formattare un testo

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

```
:set textwidth=72
```

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

```
:set textwidth
```

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

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

```
gqap
```

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

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

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

```
gggqG
```

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

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

==========

10.8 Cambiare Maiuscole/minuscole

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

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

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

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

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

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

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

==
10.9 Usare un programma esterno

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

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

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

```
        !5G
```

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

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

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

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

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

QUANDO NON FUNZIONA

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

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

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

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

LEGGERE L'OUTPUT DEI COMANDI

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

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

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

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

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

INVIARE DEL TESTO AD UN COMANDO

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

```
:write !wc
```

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

```
       4      47     249 ~
```

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

Attenti a questo errore: **>**

```
:write! wc
```

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

RIDISEGNARE LO SCHERMO

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

```
CTRL-L
```

==========

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

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

usr_11.txt   Per Vim version 6.2. Ultima modifica: 2003 Lug 01

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Roberto Franceschini

Recupero dopo un blocco

Il vostro computer si è bloccato? E voi avevate trascorso delle ore a scrivere? Niente paura! Vim salva sufficienti informazioni sul disco fisso da poter recuperare la maggior parte del vostro lavoro. Questo capitolo mostra come recuperare il vostro lavoro e spiega come viene usato il file di swap.

==
11.1 Fondamenti del recupero

Nella maggior parte dei casi recuperare un file è semplice, se conoscete quale file stavate scrivendo (e se il disco fisso funziona ancora). Aprite quel file con Vim aggiungendo l'argomento "-r": >

```
vim -r help.txt
```

Vim leggerà il file di swap (usato per memorizzare il testo che stavate scrivendo) e potrà leggere i pezzi del file originale. Se tutto va bene vedrete questi messaggi (con nomi di file differenti, naturalmente):

```
Uso swap file ".help.txt.swp" ~
File originale "~/vim/runtime/doc/help.txt" ~
Recupero completato. Dovresti controllare se va tutto bene. ~
(Potresti salvare questo file con un altro nome ed eseguire ~
'diff' rispetto al file originale per vedere le differenze) ~
Cancella il file .swp in seguito. ~
```

Per sicurezza, salvate questo file con un nome diverso: >

```
:write help.txt.recovered
```

Confrontate il file con l'originale per verificare avete ottenuto quello che vi aspettavate. Vimdiff è molto utile per ciò |08.7|. Verificate che l'originale non contenga una versione più recente di quella recuperata (se avete salvato poco prima che il computer si bloccasse). E controllate che non manchino righe (qualcosa andato male che Vim non può riparare).
   Se Vim produce dei messaggi di avviso durante il recupero, leggeteli attentamente. Comunque ciò è raro.

E' normale che gli ultimi pochi cambiamenti non possano essere recuperati. Vim invia i cambi al disco quando non scrivete per circa quattro secondi, o dopo che avete battuto circa duecento caratteri. Questo è impostato con le opzioni 'updatetime' e 'updatecount'. Così quando Vim non ha avuto la possibilità di salvare quando il sistema si è bloccato, i cambi successivi all'ultimo invio saranno persi.

Se stavate scrivendo senza un nome di file, date una stringa vuota come argomento: >

```
vim -r ""
```

Dovete essere nella directory giusta, altrimenti Vim non può trovare il file di swap.

==
11.2 Dove si trova il file di swap?

Vim può memorizzare il file di swap in molti posti. Normalmente è nella stessa directory del file originale. Per trovarlo posizionatevi nella directory del file e usate: >

```
vim -r
```

Vim elencherà i file di swap che può trovare. Guarderà anche in altre directory dove potrebbero trovarsi i file di swap per i file della directory corrente. Comunque non troverà file di swap presenti in altre directory, e non controllerà l'albero delle directory.
   Il risultato potrebbe apparire come questo:

```
        Swap files found: ~
           In current directory: ~
        1.    .main.c.swp ~
              proprietario: mool  datato: Tue May 29 21:00:25 2001 ~
                 nome file: ~mool/vim/vim6/src/main.c ~
                modificato: YES ~
               nome utente: mool   nome computer: masaka.moolenaar.net ~
           ID del processo: 12525 ~
           Nella directory ~/tmp: ~
              -- nessuno -- ~
           Nella directory /var/tmp: ~
              -- nessuno -- ~
           Nella directory /tmp: ~
              -- nessuno -- ~
```

Se ci sono più file che possono sembrare quello che cercate, vi verrà presentato un elenco e vi verrà richiesto di digitare il numero di quello che volete usare. Guardate con attenzione la data per decidere quale utilizzare.
   Se non sapete quale usare, provateli uno alla volta e verificate se sono quello cercato.

UTILIZZARE UN FILE DI SWAP SPECIFICO

Se sapete quale file di swap utilizzare, lo potete recuperare dando il nome del filo di swap. Vim troverà il nome del file originale leggendo il file di swap.

Esempio: >

```
        vim -r .help.txt.swo
```

Questo è utile anche quando il file di swap è in una directory diversa da quella che ci si aspetta. Se anche questo non funziona, guardare quali nomi di file Vim restituisce e rinominarli di conseguenza. Controllate l'opzione 'directory' per sapere dove Vim può aver messo il file di swap.

        Note:
        Vim prova a trovare il file di swap cercando nelle directory contenute nell'opzione 'dir' i file che verificano la stringa "filename.sw?".
        Se l'espansione dei caratteri jolly non funziona (ad esempio quando l'opzione 'shell' non è valida), Vim fa un tentativo disperato per trovare il file "filename.swp". Se anche questo fallisce, dovrete dare il nome completo del file di swap per poter recuperare il file.

==
11.3 Bloccato o no?   *ATTENTION* *E325*

Vim prova ad impedirvi di fare stupidaggini. Supponiamo che abbiate innocentemente iniziato a modificare un file e vi aspettiate di visualizzarne il contenuto. Invece Vim produce un lungo messaggio di questo tipo:

```
                E325: ATTENZIONE ~
        Trovato uno swap file di nome ".main.c.swp" ~
              proprietario: mool  datato: Tue May 29 21:09:28 2001 ~
                 nome file: ~mool/vim/vim6/src/main.c ~
                modificato: no ~
               nome utente: mool   nome computer: masaka.moolenaar.net ~
           ID del processo: 12559 (ancora attivo) ~
        Mentre aprivo file  "main.c" ~
                     datato: Tue May 29 19:46:12 2001 ~
 ~
        (1) Un altro programma può essere in edit sullo stesso file. ~
            Se è così, attenzione a non trovarti con due versioni ~
            differenti dello stesso file a cui vengono apportate modifiche. ~
            Esci, o continua con prudenza. ~
 ~
        (2) Una sessione di edit per questo file è finita male. ~
            Se è così, usa ":recover" oppure "vim -r main.c" ~
            per recuperare modifiche fatte (vedi ":help recovery"). ~
            Se hai già fatto ciò, cancella il file di swap ".help.txt.swp" ~
            per non ricevere ancora questo messaggio. ~
```

Ricevete questo messaggio perchè, quando iniziate a modificare un file, Vim controlla se un file di swap per esso esista già. Se ne trovasse uno deve esserci qualcosa di sbagliato. Si può verificare una di queste due situazioni:

1. Un'altra sessione di modifica è attiva per questo file. Cercate nel messaggio la linea con "process ID". Può apparire così:

   ID del processo: 12559 (ancora attivo) ~

   Il testo "(ancora attivo)" indica che il processo che sta utilizzando questo file gira sullo stesso computer. Se state lavorando su un sistema non-Unix non avrete questa informazione. Se state utilizzando questo file in rete non potete vedere questa informazione, in quanto il processo potrebbe girare su un altro computer. In questi due casi dovrete capire da soli in che situazione vi trovate.
   Se un'altra sessione di Vim stesse editando lo stesso file, continuare nella modifica porterà ad averne due versioni. Quella che verrà salvata per ultima sovrascriverà la prima, e le modifiche di questa andranno perse. In questo caso sarebbe opportuno chiudere subito Vim.

2. Il file di swap potrebbe essere il risultato di un blocco precedente di Vim o del computer. Verificate i dati contenuti nel messaggio. Se la data del file di swap è successiva a quella del file che state modificando, ed è presente questa linea:

   modificato: YES ~

   Allora molto probabilmente esisterà una sessione di modifica crashata che varrebbe la pena di salvare.
   Se la data del file è successiva a quella del file di swap, esso può essere stato modificato dopo il blocco (forse l'avete già recuperato, senza poi cancellare il file di swap?), oppure il file è stato salvato prima del blocco, ma dopo l'ultima scrittura sul file di swap (in questo caso siete fortunati, il vecchio file di swap non vi serve neppure). Vim vi avviserà con questa linea aggiuntiva:

   più RECENTE dello swap file! ~

COSA FARE?

SE sono supportate le finestre di dialogo vi verrà chiesto di selezionare una delle cinque scelte:

```
Swap file ".main.c.swp" already exists! ~
[O]pen Read-Only, (E)dit anyway, (R)ecover, (Q)uit, (D)elete it: ~
```

O Aprire il file in sola lettura. Usate questo se volete soltanto vedere il file e non vi serve recuperarlo. Dovreste usare questo quando sapete che qualcun altro sta lavorando sul file, ma voi volete solo consultarlo, senza fare cambi.

E Modificare comunque il file. Impiegatela con prudenza! Se il file fosse aperto da un altro Vim, potreste trovarvi con due versioni del file. Vim proverà ad avvertirvi quando ciò accade, ma è meglio essere sicuri che spiacenti.

R Recuperare il file dal file di swap. Usatelo se sapete che il file di swap contiene delle modifiche che vi interessa recuperare.

Q Quit. Ciò evita di iniziare la modifica del file. Usatelo se c'è un altro Vim che ha aperto lo stesso file.
   Quando avete appena aperto Vim, ciò farà chiudere Vim . Avviando Vim con dei file in molte finestre, Vim esce solo se c'è un file di swap per il primo. Usando un comando di modifica, il file non verrà caricato e voi verrete riportati indietro al file aperto in precedenza.

D Cancellare il file di swap. Usatelo se siete certi che non vi servirà più. Ad esempio, se non avesse subito modifiche, o se il file stesso fosse più nuovo che il file di swap.
   Su Unix questa possibilità viene offerta soltanto se il processo che aveva creato il file di swap non risulta ancora attivo.

Se non avete la finestra di dialogo (avete una versione di Vim che non la supporta), dovrete farlo manualmente. Per recuperare il file, usate questo comando: **>**

```
:recover
```

Vim non può sempre scoprire che esista già un file di swap per un dato file. Questo è il caso in cui l'altra sessione mette i file di swap in un'altra directory o dove il percorso per il file sia diverso quando venga lavorato su macchine diverse. Perciò, non fate conto su Vim che vi da avvisi continuamente.

SWAP FILE NON LEGGIBILE

Talora la linea

```
[non leggibile] ~
```

può apparire sotto il nome dello swap file. Questo può essere positivo o negativo, a seconda del contesto.

E' positivo se una precedente sessione di modifica si è bloccata senza aver fatto alcuna modifica al file. In questo caso una lista directory dello swap file mostrerà che è lungo zero bytes. Potete cancellarlo e andare avanti.

Va un po' male se non siete autorizzati a leggere lo swap file. Potete provare a visualizzare il file in sola-lettura, o abbandonare le mpodifiche. Su sistemi multiutente, se avete fatto voi le ultime modifiche usando un differente nome utente, una chiusura di sessione, seguita da una nuova sessione con l'altro nome utente potrebbe rimediare l'"errore di lettura". Oppure potreste individuare l'utente che ha fatto l'ultima modifica al file (o lo sta modificando adesso) e parlargli...

Va molto male se invece c'è un errore fisico di lettura sul disco che contiene lo swap file. Fortunatamente, questo non capita quasi mai. Potete visualizzare il file in sola-lettura (se ci riuscite), per vedere l'entità delle modifiche che erano state "dimenticate". Se siete responsabili della modifica di quel file, siate pronti a ridigitare le vostre ultime modifiche.

COSA FARE?

Se sono supportate le scelte di dialogo, vi verrà chiesto di scegliere una di queste cinque opzioni:

```
Swap file ".main.c.swp" already exists! ~
[O]pen Read-Only, (E)dit anyway, (R)ecover, (Q)uit, (D)elete it: ~
Swap file ".main.c.swp" già esistente! ~
[O] Apri sola-lettura, (E) Apri comunque, (R)ecupera, (Q) Esci, (D) Cancellalo: ~
```

O Aprire il file in sola lettura. Usate questa opzione se volete solo esaminare il file ma non recuperarlo. Potete utilizzare questa scelta quando sapete che qualcun altro sta modificando il file, e volete solo leggerlo senza fare modifiche.

E Modifica ugualmente il file. Usare con cautela! Se un'altra sessione di Vim sta modificando il file, questo può causare la creazione di due versioni. Vim cercherà di avvisarvi quando questo succede, ma è meglio mettersi al sicuro che doversi pentire.

R Recupera il file dal file di swap. Usate questa opzione se sapete che il file di swap contiene modifiche che volete recuperare.

Q Esci. Questo evita di modificare il file. Usate questa opzione se c'è un'altra sessione che sta modificando lo stesso file.
Se avete appena avviato Vim, questa chiuderà la sessione. Se avete avviato Vim con più file in diverse finestre, chiude la sessione solo se esiste un file di swap per il primo. Se usate un comando di modifica, il file non verrà caricato e verrete riportati al file che stavate modificando.

D Cancella il file di swap. Usate questa opzione se siete sicuri di non averne più bisogno. Ad esempio se non contiene modifiche, o quando il file è più nuovo del file di swap.
In Unix questa scelta è mostrata solo se il processo che ha creato il file di swap non risulta più attivo.

Se non appaiono le scelte di dialogo (state usando una versione di Vim che non le supporta) dovrete fare tutto manualmente. Per recuperare il file, usate

questo comando: **>**

```
:recover
```

Non sempre Vim riesce a trovare un file di swap esistente. Questo avviene quando l'altra sessione di modifica mette il file di swap in una directory diversa, o quando il percorso per il file è differente durante la modifica da un'altro computer. Quindi non fidatevi sempre degli avvisi di Vim.

Se veramente non voleste vedere questo messaggio, potreste aggiungere il flag "A" all'opzione 'shortness'. Ma sarebbe veramente strano se ne aveste bisogno.

==

11.4 Altre letture

|swap-file|	Spiega dove viene creato il file di swap e qual'è il suo nome.
|:preserve|	Salvare manualmente il file di swap sul disco.
|:swapname|	Vedi il nome del file di swap per il file corrente.
'updatecount'	Numero di battute dopo le quali il file di swap viene salvato sul disco.
'updatetime'	Tempo dopo il quale il file di swap viene salvato sul disco.
'swapsync'	Se il disco è synced (sincronizzato?) quando il file di swap viene salvato.
'directory'	Elenco delle directory dove viene memorizzato il file di swap.
'maxmem'	Limite di utilizzo della memoria prima che il file di swap venga salvato.
'maxmemtot'	Come sopra ma per tutti i file aperti.

==

Capitolo seguente: |usr_12.txt| Trucchi ingegnosi

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

usr_12.txt   Per Vim version 6.2. Ultima modifica: 2004 Mag 01

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Rossella Diomede

Trucchi ingegnosi

Combinando diversi comandi potete far fare a Vim quasi ogni cosa. In questo capitolo verrà presentata una serie di combinazioni utili. Utilizzeremo i comandi introdotti nei capitoli precedenti e molti altri.

  Capitolo seguente: |usr_20.txt| Immissione rapida dei comandi sulla linea di comando
Capitolo precedente: |usr_11.txt| Recupero dopo un blocco
             Indice: |usr_toc.txt|

==

12.1 Sostituzione di una parola

Il comando substitute può essere utilizzato per sostituire tutte le occorrenze di una parola con un'altra: >

```
:%s/four/4/g
```

L'intervallo "%" significa sostituirla in ogni riga. Il flag "g" alla fine fa sostituire tutte le parole nella stessa riga.
   Il comando non farà la cosa giusta se il vostro file contenesse anche "thirtyfour". Esso verrebbe sostituito da "thirty4". Per evitare ciò, utilizzate l'elemento "\<" per indicare l'inizio di una parola: >

```
:%s/\<four/4/g
```

Ovviamente, andrà ancora male con "fourty". Utilizzate "\>" per indicare il termine di una parola: >

```
:%s/\<four\>/4/g
```

Se state programmando, potreste voler sostituire "four" nei commenti, ma non nel codice. Poichè ciò è difficile da specificare, aggiungete il flag "c" per far si che il comando substitute chieda quando effettuare la sostituzione: >

```
:%s/\<four\>/4/gc
```

SOSTITUZIONE IN PIU' FILE

Immaginate di voler sostituire una parola in più di un file. Potete aprire ciascun file e digitare manualmente il comando. E' molto più veloce utilizzare record e playback.
   Supponete di avere una directory con file C++, che terminano con ".cpp". C'è una funzione chiamata "GetResp" che volete rinominare "GetAnswer".

```
vim *.cpp               Avviate Vim, definendo la lista di argomenti
                        che contiene tutti i file C++ .  Siete ora
                        nel primo file.
qq                      Iniziate la registrazione nel registro q
:%s/\<GetResp\>/GetAnswer/g
                        Effettuate le sostituzioni nel primo file.
:wnext                  Scrivete il file e passate al successivo.
q                       Interrompete la registrazione.
@q                      Eseguite il registro q.  Ciò ripeterà la
                        sostituzione e il ":wnext".  Potete verificare
                        che non ci sia un messaggio di errore.
999@q                   Eseguite il registro q nei restanti file.
```

Nell'ultimo file avrete un messaggio di errore, in quanto ":wnext" non può passare al file successivo. Ciò fermerà l'esecuzione e tutte le sostituzioni saranno state effettuate.

Note:
Durante la lettura di una sequenza registrata, un errore arresta l'esecuzione.
Pertanto, accertatevi che non ci sia un messaggio di errore durante la registrazione.

C'è un problema: se uno dei file .cpp non contiene la parola "GetResp", avrete un messaggio di errore e la sostituzione verrà interrotta. Per evitare ciò, aggiungete il flag "e" al comando substitute: >

```
:%s/\<GetResp\>/GetAnswer/ge
```

Il flag "e" dice al comando :substitute che non è un errore non trovare corrispondenze.

==========

12.2 Modifica di "Last, First" in "First Last"

Avete una lista di nomi in questo formato:

```
Doe, John ~
Smith, Peter ~
```

La volete modificare in:

```
John Doe ~
Peter Smith ~
```

Ciò può essere fatto con un unico comando: >

```
:%s/\([^,]*\), \(.*\)/\2 \1/
```

Suddividiamolo in più parti. Ovviamente esso inizia con un comando substitute. Il "%" è la lunghezza della riga, che rimane per l'intero file. In tal modo la sostituzione viene effettuata in ogni riga del file.
Gli argomenti del comando substitute sono "/from/to/". Gli slash separano il modello "from" e la stringa "to". Cioè che il modello "from" contiene:

```
\([^,]*\), \(.*\) ~

La prima parte tra \( \) corrisponde a "Last"   \(     \)
    corrisponde a nessuna virgola                    [^,]
    ogni volta                                     *
corrisponde a ", " letteralmente                      ,
La seconda parte tra \( \) corrisponde a "First"        \(  \)
    ogni carattere                                        .
    ogni volta                                              *
```

Nella parte "to" avete "\2" e "\1". Essi si chiamano backreferences. Fanno riferimento al testo cui corrispondono le parti "\(\)" nel modello. "\2" fa riferimento al testo cui corrisponde il secondo "\(\)", che è il nome "First". "\1" fa riferimento al primo "\(\)", che è il nome "Last".
Potete utilizzare più di nove backreferences nella parte "to" del comando substitute. "\0" sta per l'intero modello trovato. Ci sono diversi altri elementi speciali nel comando substitute. Vedere |sub-replace-special|.

==========

12.3 Ordinamento di un elenco

All'interno di un Makefile spesso è presente un elenco di file. Ad esempio:

```
OBJS = \ ~
        version.o \ ~
        pch.o \ ~
        getopt.o \ ~
        util.o \ ~
        getopt1.o \ ~
        inp.o \ ~
        patch.o \ ~
        backup.o ~
```

Per ordinare l'elenco, filtrate il testo attraverso il comando esterno di ordinamento: >

```
/^OBJS
j
:.,/^$/-1!sort
```

Esso va alla prima riga, dove "OBJS" è il primo elemento nella riga. Poi va alla riga successiva e filtra le righe fino ad arrivare ad una riga vuota. Potete anche selezionare le righe con la modalità Visual e poi utilizzare "!sort". Questo è più semplice da digitare, ma presenta maggior lavoro quando ci sono molte righe.
   Il risultato è il seguente:

```
OBJS = \ ~
        backup.o ~
        getopt.o \ ~
        getopt1.o \ ~
        inp.o \ ~
        patch.o \ ~
        pch.o \ ~
        util.o \ ~
        version.o \ ~
```

Attenzione alla presenza di un backslash al termine di ciascuna riga, usato per indicare che la riga continua. Dopo l'operazione si evidenzia un errore. La riga "backup.o" che era alla fine dell'elenco non ha il backslash. Ora, poichè viene inserita nell'elenco in una posizione diversa, è necessario che abbia il backslash.
   La soluzione più semplice è quella di aggiungerla digitando "A \<Esc>". Potete mantenere il backslash nell'ultima riga se siete certi che poi ci sia una riga vuota. In tal modo non avrete più questo problema.

==
12.4 Inversione dell'ordine delle righe

Il comando |:global| può essere combinato con il comando |:move| per spostare tutte le righe prima della riga iniziale, ottenendo come risultato un file invertito. Il comando è: >

```
:global/^/m 0
```

Abbreviato: >

```
:g/^/m 0
```

L'espressione "^" segna l'inizio della riga (anche se la riga è vuota). Il comando |:move| sposta la riga trovata dopo la mitica riga zero, in modo che diventi la prima riga del file. Poichè il comando |:global| non viene confuso dalla numerazione delle righe modificata, |:global| continua a cercare tutte le restanti righe del file ed a porle ciascuna per prima.

Questo comando opera anche su di un intervallo di righe. Per prima cosa spostatevi sulla prima riga e contrassegnatela con "mt". Quindi spostate il cursore sull'ultima riga dell'intervallo e digitate: >

```
:'t+1,.g/^/m 't
```

==
12.5 Conteggio di parole

Capita a volte di dover scrivere un testo con un numero massimo di parole. Vim può contare le parole per voi.
   Se volete contare le parole dell'intero file, utilizzate questo comando: >

```
g CTRL-G
```

Non digitare uno spazio dopo la g, qui è stato usato per rendere il comando facile da leggere.
   Il risultato appare così:

```
Col 1 of 0; Line 141 of 157; Word 748 of 774; Byte 4489 of 4976 ~
```

Potete vedere su quale parola siete (748), ed il numero complessivo di parole nel file (774).

Quando il testo è solo una parte del file, potete spostarvi all'inizio del testo, battere "g CTRL-G", spostarvi alla fine del testo, battere ancora "g CTRL-G", e calcolare così la differenza nella posizione della parola. E' un buon esercizio ma esiste un sistema più semplice. Con il Visual mode, selezionate il testo del quale volete contare le parole. Digitate quindi g CTRL-G. Risultato:

```
Selezionate 5 di 293 righe; 70 di 1884 parole; 359 di 10928 byte ~
```

Per altri metodi relativi al conteggio di parole, righe ed altri elementi, vedere |count-items|.

==
12.6 Ricerca di una pagina man   *find-manpage*

Nel modificare uno script shell o un programma C, state usando un comando o una funzione di cui volete cercare la pagina man (su Unix). Utilizziamo prima un semplice sistema: spostate il cursore sulla parola per la quale si chiede aiuto e premete **>**

```
K
```

Vim eseguirà il programma esterno "man" sulla parola. Se la pagina man è stata trovata, viene visualizzata. Tale sistema utilizza il paginatore normale per scorrere il testo (di solito il programma "more"). Premendo il tasto <Invio> quando terminato, si tornerà a Vim.

Uno svantaggio è quello di non poter vedere contemporaneamente la pagina man ed il testo sul quale state lavorando. Esiste un artificio per far si che la pagina man compaia in una finestra di Vim. Per prima cosa, caricate il plugin del filetype di man: **>**

```
:source $VIMRUNTIME/ftplugin/man.vim
```

Inserite questo comando nel file vimrc se prevedete di effettuare questa operazione spesso. Potete ora utilizzare il comando ":Man" per aprire una finestra su una pagina man: **>**

```
:Man csh
```

Potete scorrere il testo e noterete che questo è evidenziato. Ciò vi consente di trovare l'aiuto che stavate cercando. Utilizzate CTRL-W per spostarvi nella finestra contenente il testo sul quale stavate lavorando.
   Per cercare una pagina man in un sezione specifica, inserite come prima cosa il numero della sezione.
Ad esempio, per cercare "echo" nella sezione 3: **>**

```
:Man 3 echo
```

Per spostarvi ad un'altra pagina man, che è nel testo nel formato tipico "word(1)", premete CTRL-] su di essa. Ulteriori comandi ":Man" utilizzeranno la stessa finestra.

Per visualizzare una pagina man per la parola al di sotto del cursore, utilizzate: **>**

```
\K
```

(Se avete ridefinito il <Leader>, utilizzatelo al posto del backslash). Ad esempio, volete conoscere il valore di ritorno di "strstr()" mentre digitate la riga:

```
if (strstr(input, "aap") == ) ~
```

Spostate il cursore su "strstr" e digitate "\K". Si aprirà una finestra per visualizzare la pagina man per strstr().

==
12.7 Eliminazione di spazi vuoti

Alcuni trovano inutili e brutti gli spazi e le tabulazioni alla fine di una riga. Per eliminare gli spazi alla fine di ciascuna riga eseguite il seguente comando: **>**

```
:%s/\s\+$//
```

Viene usato l'intervallo di riga "%", in tal modo questo funzionerà per l'intero file. Il modello a cui il comando ":substitute" corrisponde è "\s\+$". Esso troverà spazi vuoti (\s), 1 o più (\+), prima della fine della riga ($). Più avanti sarà illustrato come scrivere modelli come questo |usr_27.txt|.
   La parte "to" del comando di sostituzione è vuota: "//". In tal modo essa non viene sostituita con nessun carattere, eliminando effettivamente gli spazi vuoti trovati.

Un altro uso inutile degli spazi è quando vengono posizionati prima di una

tabulazione. Spesso possono essere eliminati senza alterare il numero degli spazi necessari. Ma non sempre è possibile. Pertanto, è preferibile eseguire l'operazione manualmente. Utilizzate questo comando di ricerca: **>**

```
/
```

Non potete vederlo, ma in questo comando c'è uno spazio prima della tabulazione. Quindi è "/<Space><Tab>". Utilizzate ora "x" per eliminare lo spazio e verificate che il numero degli spazi non è stato modificato. Se fosse stato modificato sarebbe necessario inserire una tabulazione. Digitate "n" per trovare la prossima corrispondenza. Ripetete questa operazione finchè non ci saranno più altre corrispondenze.

==========

12.8 Ricerca di una parola all'interno di un file

Se siete utenti di UNIX, potete utilizzare una combinazione di comandi di Vim e grep per modificare tutti i file che contengano una determinata parola. Ciò risulta estremamente utile quando state lavorando su di un programma e volete visualizzare o modificare i file che contengono una variabile specifica.
   Ad esempio, immaginate di voler modificare tutti i file del programma C che contengono la parola "frame_counter". Per fare ciò utilizzate il comando: **>**

```
vim `grep -l frame_counter *.c`
```

Osserviamo il comando nei dettagli. Il comando grep cerca in un gruppo di file una determinata parola. Poichè l'argomento -l è indicato, il comando elencherà soltanto i file che contengono la parola e non le righe corrispondenti. La parola da trovare è "frame_counter". In realtà, essa può essere qualsiasi espressione regolare. (Note: Quella che grep considera un'espressione regolare non è considerata allo stesso modo da Vim.)
   L'intero comando è racchiuso tra apici inversi (`). Questo comunica alla shell di UNIX di eseguire il comando e pretende che i risultati vengano scritti sulla linea di comando. Pertanto ciò che avviene è che il comando grep viene eseguito e produce una lista di file, i quali vengono inseriti nella riga di comando di Vim. Vim modificherà la lista di file data come output da grep. Potete quindi utilizzare comandi come ":next" e ":first" per dare un'occhiata tra i file.

RICERCA DI UNA PAROLA ALL'INTERNO DI UNA RIGA

Il comando sopra descritto cerca esclusivamente i file che contengono la parola. Dovete ora cercare la parola all'interno dei file.
   Vim ha un comando incorporato che potete utilizzare per cercare una determinata stringa all'interno di un gruppo di file. Se volete trovare tutte le ripetizioni di "error_string" in tutti i file del programma C, ad esempio, digitate il seguente comando: **>**

```
:grep error_string *.c
```

Esso fa sì che Vim cerchi la stringa "error_string" in tutti i file (*.c). L'editor aprirà il primo file dove è stata trovata la corrispondenza e posizionerà il cursore sulla prima riga. Per spostarvi alla riga successiva (non importa in quale file), utilizzate il comando ":cnext". Per andare alla corrispondenza precedente, utilizzate il comando ":cprev". Utilizzate ":clist" per visualizzare tutte le corrispondenze ed il punto in cui si trovano.
   Il comando ":grep" utilizza i comandi esterni di grep (su Unix) o di findstr (su Windows). Potete modificare ciò impostando l'opzione 'grepprg'.

==========

Capitolo seguente: |usr_20.txt| Immissione rapida dei comandi sulla linea di comando

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

usr_20.txt   Per Vim version 6.2. Ultima modifica: 2003 Apr 30

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

Immissione rapida dei comandi sulla linea di comando

Vim ha alcune funzionalità generali che rendono più semplice l'immissione di comandi. I comandi possono essere abbreviati, modificati e ripetuti.
Il meccanismo del completamento è disponibile quasi sempre.

  Capitolo seguente: |usr_21.txt| Andarsene e ritornare
Capitolo precedente: |usr_12.txt| Trucchi ingegnosi
             Indice: |usr_toc.txt|

==========

20.1 Elaborazione della linea di comando

Quando, per immettere un comando, digitate il carattere dei due-punti (:) o effettuate la ricerca di una stringa con / o ?, Vim pone il cursore in fondo allo schermo; questa è la posizione in cui inserire i comandi o le chiavi di ricerca, ed è chiamata Command line. Anche quando viene usata per inserire un comando di ricerca.

Il modo più semplice per modificare il comando digitato è quello di premere il tasto <BS>, così da cancellare il carattere a sinistra del cursore. Per cancellare un altro carattere precedentemente digitato, spostate il cursore coi tasti-freccia.
Per esempio, immaginate di aver già digitato il comando:

```
:s/col/pig/
```

Se ora, prima di premere <Invio>, voi voleste che "col" fosse trasformato in "cow", digitate <Left> cinque volte. Il cursore, ora, si trova proprio dopo "col". Per effettuare la correzione digitate "<BS> e "w":

```
:s/cow/pig/
```

Ora potete premere <Enter> direttamente, senza spostare il cursore alla fine della linea prima di eseguire il comando.

Ecco i tasti utilizzati più spesso per spostarsi entro la riga di comando:

<Left>	un carattere a sinistra
<Right>	un carattere a destra
<S-Left> o <C-Left>	una parola a sinistra
<S-Right> o <C-Right>	una parola a destra
CTRL-B o <Home>	all'inizio della linea di comando
CTRL-E o <End>	alla fine della linea di comando

Note:
<S-Left> (tasto-freccia a sinistra con tasto delle Maiuscole premuto) e <C-Left> (tasto-freccia a sinistra con tasto Control premuto) non funzionano su tutte le tastiere. Ciò vale anche per le altre combinazioni ottenute coi tasti Maiuscola e Control.

Per muovere il cursore potete utilizzare anche il mouse.

CANCELLAZIONE

Come già detto, <BS> cancella il carattere precedente al cursore. Per cancellare un'intera parola si usa CTRL-W.

```
/the fine pig ~

             CTRL-W

/the fine ~
```

CTRL-U cancella tutto il testo, permettendo di ricominciare tutto da capo.

SOVRASCRITTURA

Esistono due possibilità per l'inserimento di nuovi caratteri: la prima permette di inserire nuovi caratteri senza cancellare quelli esistenti; la seconda fa sovrascrivere i nuovi caratteri su quelli esistenti. Il tasto <Insert> permette di passare dalla prima modalità alla seconda. Scrivete questo testo:

```
/the fine pig ~
```

Ora muovete il cursore all'inizio della parola "fine" premendo <S-Left> due volte (o <Left> otto volte, se <S-Left> non funziona). Quindi, premete <Insert> per passare alla funzione di sovrascrittura e digitate "great":

```
/the greatpig ~
```

Oops, è andato perso lo spazio. In questo caso non usate <BS>, perché cancellereste la "t" (diversamente dalla modalità di Sovrascrittura). Invece, premete <Insert> per passare dalla sovrascrittura all'inserimento, e digitate lo spazio:

```
/the great pig ~
```

ANNULLAMENTO

Quando volete annullare un comando : oppure / in fase di digitazione, premete CTRL-C o <Esc>.

Note:
<Esc> è il tasto universalmente usato come «uscita». Sfortunatamente, nel buon vecchio Vi la digitazione di <Esc> in una riga di comando eseguiva il comando! Poiché ciò potrebbe essere considerato un bug, Vim usa <Esc> per cancellare il comando.
Ma usando l'opzione 'cpoptions' si può ottenere la compatibilità con VI. <Esc> funziona in modalità compatibile con Vi anche quando si usa un mapping (scritto per Vi).
In definitiva, il metodo che funziona sempre è quello del CTRL-C.

Se siete all'inizio della linea di comando, premendo <BS> annullererete il comando. È equivalente a cancellare i caratteri ":" o "/" posti all'inizio della riga.

==
20.2 Scorciatoie per la linea di comando

Alcuni comandi ":" sono veramente lunghi. Si è già detto che ":substitute" può essere abbreviato come ":s". Questa è una regola generale: tutti i comandi ":" possono essere abbreviati.

Di quanto può essere abbreviato un comando? Ci sono 26 lettere, ma molti più comandi. Per esempio, anche ":set" inizia con ":s", ma ":s" non avvia un comando ":set". Invece ":set" può essere abbreviato con ":se".
   Quando due comandi hanno la stessa forma abbreviata, in realtà essa funzionerà solo per uno di essi. Non c'è un metodo logico per individuarlo: è qualcosa che va imparato. Nei file di help viene indicata la forma abbreviata più corta che funziona. Per esempio: **>**

```
:s[ubstitute]
```

Significa che la forma più corta per ":substitute" è ":s". I caratteri successivi sono opzionali. Quindi funzionano anche ":su" e ":sub".

In questo manuale dell'utente si userà sia il nome intero del comando, sia una forma abbreviata comunque intelligibile. Per esempio, ":function" potrebbe essere abbreviato in ":fu", ma poiché molte persone non capirebbero cosa indica tale abbreviazione, useremo ":fun" (Vim non ha un comando ":funny", altrimenti anche ":fun" potrebbere ingenerare confusione).

Per quel che riguarda gli script di Vim, vi raccomando di scrivere il nome intero dei comandi. Ciò permette di interpretarli più facilmente quando si effettuano modifiche successive. Si può fare eccezione per alcuni comandi usati molto spesso, come ":w" (":write") e ":r" (":read").
   Una forma particolarmente ambigua è ":end", che si adatta a ":endif", ":endwhile" e ":endfunction". Perciò, usate sempre il nome intero.

NOMI BREVI DELLE OPZIONI

Nel manuale dell'utente sono illustrate le opzioni con i nomi lunghi.
Per molte opzioni si possono usare anche nomi brevi. Contrariamente a quel che avviene per i comandi ":", c'è un solo nome breve funzionante. Ad esempio, il nome breve per 'autoindent' è 'ai'. Perciò, i due comandi seguenti fanno lo stesso lavoro: **>**

```
:set autoindent
:set ai
```

Un elenco dei nomi dei comandi (lunghi e brevi) si trova qui: |option-list|.

==
20.3 Completamento della linea di comando

Questa funzionalità di Vim è una di quelle che, da sola, giustifica l'adozione di Vim da parte degli utenti di Vi. Una volta che l'avrete usata, non potrete più farne a meno.

Supponete di avere una directory contenente questi file:

```
info.txt
intro.txt
corpodeltesto.txt
```

Per elaborare l'ultimo file, potete usare il comando: **>**

```
:edit corpodeltesto.txt
```

e noterete che è facile sbagliarne la digitazione. Un modo assai più rapido è:

```
:edit c<Tab>
```

e otterrete lo stesso effetto di prima. Che è successo? Il tasto <Tab> effettua il completamento della parola che precede il cursore. Nel nostro caso: "b".
Vim cerca nella directory e trova soltanto un file il cui nome inizia per "b"; non essendoci ambiguità, Vim completa al posto vostro il nome del file e lo apre.

Ora digitate: **>**

```
:edit i<Tab>
```

Vim, ora, emetterà un segnale acustico e presenterà il comando: **>**

```
:edit info.txt
```

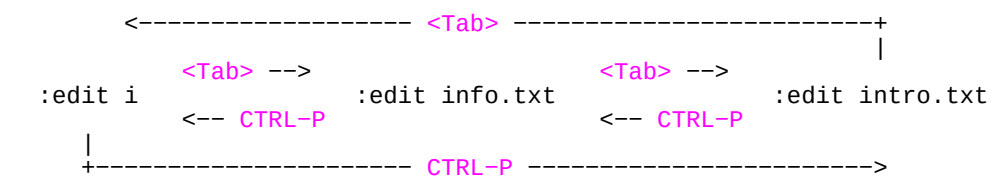
Il segnale acustico significa che Vim ha trovato più di una corrispondenza.
   Per questo motivo aprirà il primo file (in ordine alfabetico) che corrisponde.
   Se premete ancora <Tab>, otterrete: **>**

```
:edit intro.txt
```

Così, se il primo <Tab> non evidenzia il file che stavate cercando, premetelo ancora. Se esistono più nomi corrispondenti, li vedrete tutti, uno per uno.
   Se premete <Tab> sull'ultimo nome corrispondente, tornerete a quello che avevate digitato inizialmente: **>**

```
:edit i
```

Allora si ricomincia daccapo. Così Vim effettua un ciclo attraverso l'elenco delle corrispondenze.
Con CTRL-P potrete scorrere la lista nel verso opposto:



CONTESTO

Quando digitate ":set i" invece che ":edit i" e premete <Tab>, si ottiene: >

```
:set icon
```

Ehi! Perché non compare ":set info.txt"? Ciò avviene perché il completamento di Vim è sensibile al contesto. Il tipo di parole cercate da Vim dipende dal comando iniziale. Vim sa che subito dopo un comando ":set" non potete usare un nome di file; è consentito solo un nome d'opzione.
Anche qui, se digitate più volte il tasto <Tab>, Vim effettuerà una scansione ciclica attraverso tutti i valori corrispondenti.
Ce ne sono parecchi, per cui è meglio digitare altri caratteri: >

```
:set isk<Tab>
```

Diventa: >

```
:set iskeyword
```

Ora digitate "=" e premete <Tab>: >

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

Ciò che accade è che Vim inserisce il vecchio valore dell'opzione. Adesso potete modificarla.
L'uso di <Tab> porta al completamento con ciò che Vim si aspetta debba comparire nella posizione corrispondente. Provate semplicemente a vedere come funziona. In certe situazioni otterrete qualcosa di diverso da quanto desiderato. Ciò è dovuto al fatto che Vim non conosce i vostri desideri; oppure il completamento potrebbe non essere stato implementato per quella determinata situazione. In quel caso otterrete il semplice inserimento di un <Tab> (visualizzato come ^I).

ELENCO DI CORRISPONDENZE

Quando esistono molte corrispondenze, vorrete vederle tutte insieme. Potete farlo premendo CTRL-D. Per esempio, premendo CTRL-D dopo aver digitato >

```
:set is
```

si ottiene: >

```
:set is
incsearch  isfname    isident    iskeyword  isprint
:set is
```

Vim elenca le corrispondenze e ritorna al testo da voi digitato, così che possiate passare in rassegna l'elenco e scegliere ciò che volete. Se non ci sono corrispondenze, potete usare <BS> per correggere la parola. Se le corrispondenze sono molte, prima di premere <Tab> digitate qualche altro carattere di ciò che vi interessa.
Se ci avete fatto caso, noterete che "incsearch" non inizia con "is". In questo caso, "is" è il nome breve di "incsearch" (molte opzioni hanno nomi sia brevi che lunghi). Vim è sufficientemente scaltro da sapere che voi potreste voler espandere il nome breve dell'opzione nel suo nome lungo.

C'E' DI PIU'

Il comando CTRL-L completa la parola trasformandola nella stringa più lunga e non ambigua. Se digitate ":edit i" ed esistono i file "info.txt" e "info_backup.txt" otterrete ":edit info".

Per modificare il meccanismo di completamento, potete usare l'opzione 'wildmode'.
L'opzione 'wildmenu' permette di produrre un elenco corrispondenze in forma di menu.
L'opzione 'suffixes' permette di precisare i file meno importanti che possono essere relegati alla fine dell'elenco.
L'opzione 'wildignore' permette di indicare i file da non elencare affatto.

Altre informazioni in merito si trovano qui: |cmdline-completion|

==
20.4 Cronologia dei comandi

La cronologia dei comandi (o history) è stata brevemente citata nel capitolo 3. La cosa più semplice a riguardo è l'utilizzo del tasto <Up> per richiamare i precedenti comandi; il tasto <Down> permette di scorrere i comandi in avanti.

In realtà di cronologie ne esistono quattro. Qui tratteremo quella dei comandi ":", e quelle dei comandi di ricerca "/" e "?". I comandi "/" e "?" condividono la stessa cronologia, essendo entrambi comandi di ricerca. Le altre due cronologie sono quelle delle espressioni e delle linee di input per la funzione input().
|cmdline-history|

Supponiamo di aver immesso un comando ":set", di aver digitato più di dieci comandi ":" e poi di voler ripetere il comando ":set" di prima. Potreste premere ":" e dieci volte <Up>, ma c'è un modo più rapido: **>**

```
:se<Up>
```

Vim tornerà al precedente comando che inizia con "se". Avete buone probabilità che si tratti del comando ":set" che cercavate. Se non altro, vi sarà risparmiato di premere tante volte il tasto <Up> (a meno che non abbiate dato solo comandi di tipo ":set").

Il tasto <Up> userà il testo digitato fino a quel momento e lo confronterà con le linee presenti nella cronologia. Solo le linee corrispondenti saranno usate.
   Se non trovate le linee che cercavate, usate <Down> e tornerete a quelle digitate inizialmente, per modificarle. Oppure usate CTRL-U per reiniziare tutto da capo.

Per vedere tutte le linee della cronologia: **>**

```
:history
```

Questa è la cronologia dei comandi ":". La cronologia dei comandi di ricerca viene mostrata con il comando: **>**

```
:history /
```

CTRL-P ha lo stesso effetto di <Up>, tranne per il fatto che non importa quel che avete già scritto. Lo stesso vale per CTRL-N e <Down>. CTRL-P sta per "precedente", CTRL-N per "next" (successivo).

==
20.5 Finestra della linea di comando

La digitazione del testo nella linea di comando funziona in modo diverso dalla digitazione in Insert mode: a molti comandi non è permesso di modificare il testo. Per la maggior parte dei programmi questo non è un problema, ma a volte vi troverete a digitare comandi complessi. E' in questo caso che torna utile la finestra della linea di comando.

La finestra della linea di comando si apre con questo comando: **>**

```
q:
```

Con esso, Vim apre una (piccola) finestra in basso, che contiene la cronologia della linea di comando e una linea vuota in fondo:

```
+--------------------------------------+
|other window                          |
|~                                     |
|file.txt==============================|
|:e c                                  |
|:e config.h.in                        |
|:set path=.,/usr/include,,            |
|:set iskeyword=@,48-57,_,192-255      |
|:set is                               |
|:q                                    |
|:                                     |
|command-line==========================|
|                                      |
+--------------------------------------+
```

Ora siete in Normal mode. Qui potete usare i tasti "hjkl" per spostarvi col cursore. Per esempio, spostatevi verso l'alto con "5k" fino alla linea con ":e config.h.in". Digitate "$h" per muovervi fino alla "i" di "in" e digitate "cwout". Ora la linea è stata modificata in:

```
:e config.h.out ~
```

Premete <Enter> e questo comando verrà eseguito, dopodiché la finestra della linea di comando si chiuderà da sé.
   Il comando <Enter> eseguirà la linea su cui si trova il cursore, senza badare se Vim si trova in Insert mode o Normal mode.
   Le modifiche, effettuate in finestra della linea di comando, saranno perdute; non saranno reperibili nella cronologia dei comandi, tranne per il comando da voi eseguito (ciò vale per tutti i comandi mandati in esecuzione).

La finestra della linea di comando è utilissima per ottenere una panoramica della cronologia dei comandi, cercare nei comandi qualcosa di simile a quello che serve, fare le dovute modifiche ed eseguire il tutto. Per trovare quel che si vuole si può utilizzare esplicitamente un comando di ricerca.
   Nell'esempio precedente avreste potuto usare il comando di ricerca "?config" per trovare il comando più recente contenente "config". C'è un che di strano nell'usare un comando di ricerca per effettuare ricerce nella finestra dei comandi.
   Quando digitate questo comando di ricerca non potrete aprire un'altra finestra della linea di comando, perché ne viene supportata solo una.

==

Capitolo seguente: |usr_21.txt| Andarsene e ritornare

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

usr_21.txt   Per Vim version 6.2. Ultima modifica: 2002 Ott 29

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Gianluca Trimarchi

Andarsene e ritornare

Questo capitolo considera come alternare l'uso di altri programmi con Vim.
Sia eseguendo un programma dall'interno di Vim ovvero uscendo da Vim e
ritornandovi in seguito. Inoltre, tratta di come memorizzare lo stato di Vim e
ripristinarlo in seguito.

  Capitolo seguente: |usr_22.txt| Trovare il file da aprire
Capitolo precedente: |usr_20.txt| Immissione rapida dei comandi dalla linea di comando
              Indice: |usr_toc.txt|

==
21.1 Sospendere e ripristinare

Come molti programmi UNIX Vim può essere sospeso premendo CTRL-Z. Vim viene
fermato e venite riportati alla shell da cui è stato avviato. Dopo potete
eseguire qualsiasi altro comando finchè non ne avete abbastanza. Ritornate a
Vim con il comando "fg", >

```
CTRL-Z
{qualsiasi sequenza di comandi della shell}
fg
```

Siete ritornati dove avevate lasciato Vim, non è cambiato nulla.
   Nel caso premere CTRL-Z non funzionasse, potete anche usare ":suspend".
Non dimenticate di riportare Vim in primo piano, potreste perdere qualsiasi
modifica fatta!

Solo UNIX lo permette. Su altri sistemi Vim avvierà una shell per voi. Anche
questa ha la funzionalità di eseguire comandi della shell. Ma è una nuova
shell, non quella da cui avete avviato Vim.
   Quando state eseguendo la GUI non potete tornare alla shell da dove Vim è
stato avviato. CTRL-Z minimizzerà la finestra di Vim.

==
21.2 Eseguire comandi della shell

Per eseguire un singolo comando della shell da Vim usate ":!{command}". Per
esempio, per vedere il contenuto di una directory: >

```
:!ls
:!dir
```

Il primo è per Unix, il secondo per MS-Windows.
   Vim eseguirà il programma. Quando finirà vi verrà chiesto di premere
<Invio>. Questo vi permette di visionare l'output del comando prima di
ritornare al testo su cui stavate lavorando.
  Il "!" è anche usato altrove quando si esegue un programma. Diamo uno
sguardo ad una panoramica:

```
:!{programma}           esegue {programma}
:r !{programma}         esegue {programma} e legge il suo output
:w !{programma}         esegue {programma} e invia il testo come suo
                        input
:[intervallo]!{programma}       filtra il testo attraverso {programma}
```

Notate che la presenza di un intervallo prima di "!{programma}" fa una grossa
differenza. Senza di esso il programma viene eseguito normalmente,
specificando l'intervallo un certo numero di linee di testo verrà filtrato
attraverso il programma.

È possibile eseguire in questo modo molti programmi. Ma una shell è molto
meglio. Potete avviare una nuova shell in questo modo: >

```
:shell
```

È simile ad usare CTRL-Z per sospendere Vim. La differenza è che viene avviata una shell nuova.

Quando usate la GUI, la shell starà usando la finestra di Vim per i suoi input e output. Poichè Vim non è un emulatore di terminale, non funzionerà perfettamente. Se avete problemi, provate ad impiegare l'opzione 'guipty'. Se ancora non dovesse funzionare in modo soddisfacente, avviate un nuovo terminale per eseguire la shell al suo interno. per esempio con: **>**

```
:!xterm&
```

==

21.3 Memorizzare le informazioni; viminfo

Dopo aver scritto per un po', avrete del testo entro i registri, puntatori in vari file, una storia della linea di comando piena di astuti comandi. Quando uscite da Vim tutto ciò viene perso. Ma potete conservarlo!

Il file viminfo è deputato ad immagazzinare le informazioni sullo status:

```
Storia della linea di comando e dei pattern di ricerca
Testo nei registri
Puntatori per vari file
L'elenco dei buffer
Variabili globali
```

Ogni volta che uscite da Vim esso memorizzerà queste informazioni in un file, il file viminfo. Quando Vim viene riavviato, il file viminfo viene letto e le informazioni ripristinate.

L'opzione 'viminfo' per default è impostata per ripristinare un numero limitato di informazioni. Potreste volerla impostare per memorizzare un maggior numero di informazioni. Ciò può avvenire attraverso il seguente comando: **>**

```
:set viminfo=stringa
```

La stringa specifica cosa salvare. La sintassi di questa stringa è un carattere di opzione seguito da un argomento. Le coppie opzione/argomento sono separate da virgole.
   Osservate come potreste costruire la vostra stringa viminfo. Primo, l'opzione ' viene usata per specificare per quanti file salvare i puntatori (a-z). Per questa opzione prendete un bel numero pari (1000, per esempio).
   Adesso il vostro comando potrebbe assomigliare a questo: **>**

```
:set viminfo='1000
```

L'opzione f controlla se memorizzare i puntatori globali (A-Z e 0-9). Se questa opzione è 0, non ne verrà memorizzato nessuno. Se è 1 o non specificate una opzione f, i puntatori verranno memorizzati. Sicuramente vorrete usare questa funzionalità, quindi adesso avrete questo: **>**

```
:set viminfo='1000,f1
```

L'opzione " controlla quante linee vengono salvate per ciascun registro. Per default, vengono salvate tutte le linee. Se è 0, non viene salvato nulla.
   Per evitare di aggiungere migliaia di linee al vostro file viminfo (che potrebbero non venire mai usate e rallenterebbero l'avvio di Vim) usate un massimo di 500 linee: **>**

```
:set viminfo='1000,f1,\"500
<
```

```
Note:
Visto che il carattere " inizia un commento, dev'essere preceduto da
un backslash.
```

Altre opzioni che potreste voler usare:

```
:       numero di linee da salvare dalla storia della linea di comando
@       numero di linee da salvare dalla storia della linea di input
/       numero di linee da salvare dalla storia della ricerca
r       supporto rimovibile, per cui nessun puntatore sarà memorizzato
        (può esser usato più volte)
!       variabili globali che iniziano con una lettera maiuscola e non
        contengono lettere minuscole
h       disattiva l'evidenziazione di 'hlsearch' quando viene avviato
```

	%	l'elenco dei buffer (memorizzato solo quando Vim viene avviato
		senza file come argomenti)
	c	converte il testo usando 'encoding'
	n	il nome usato per il file viminfo (dev'essere l'ultima
		opzione)

Guardate l'opzione 'viminfo' e |viminfo-file| per maggiori informazioni.

Qualora eseguiste più copie di Vim, l'ultima a terminare memorizzerà le sue informazioni. Ciò potrebbe far sì che le informazioni memorizzate dai Vim precedenti vengano perse. Ogni dato può essere memorizzato una sola volta.

RITORNARE DOVE ERAVATE

Siete a metà della scrittura di un file ed è tempo di andarsene in vacanza.
Uscite da Vim e andate a divertirvi, dimenticandovi tutto del vostro lavoro. Dopo qualche settimana avviate Vim, e scrivete:
>

```
'0
```

E siete di nuovo lì dove avevate lasciato Vim. Ora potete continuare col vostro lavoro.
Vim crea un puntatore ogni volta che uscite da Vim. L'ultimo è '0. La posizione a cui puntava '0 diventa '1. E '1 diventa '2, e così via. Il puntatore '9 andrà perso.
Il comando ":marks" è utile per scoprire dove vi porteranno '0 e '9 .

SPOSTARE LE INFORMAZIONI DA UN VIM AD UN'ALTRO

Potete usare i comandi ":wviminfo" e ":rviminfo" per salvare e ripristinare le informazioni quando Vim è ancora in esecuzione. Per esempio è utile per scambiare i contenuti del registro tra due istanze di Vim. Nel primo Vim fate: >

```
:wviminfo! ~/tmp/viminfo
```

E nel secondo Vim fate: >

```
:rviminfo! ~/tmp/viminfo
```

Ovviamente, la "w" sta per "write" e la "r" per "read".
Il carattere ! è usato da ":wviminfo" per forzare la riscrittura di un file già esistente. Quando è omesso, e il file esiste, le informazioni sono fuse in unico file.
Il carattere ! usato per ":rviminfo" significa che verranno usate tutte le informazioni, ciò potrebbe riscrivere informazioni già esistenti. Senza il ! verranno usate solo le informazioni che non sono impostate.
Questi comandi possono anche essere usati per memorizzare informazioni e riutilizzarle in seguito. Potreste creare una directory piena di file viminfo, ognuno contenente informazioni per compiti differenti.

==
21.4 Sessioni

Supponete di avere scritto a lungo, ed è ormai fine giornata. Volete interrompere il lavoro e riprenderlo domani dal punto in cui eravate arrivati. Potete farlo salvando la vostra sessione di lavoro e ripristinandola il giorno seguente.
Una sessione di Vim contiene tutte le informazioni sui file che avete aperto. Sono incluse cose come l'elenco dei file, la disposizione delle finestre, le variabili globali, le opzioni ed altre informazioni. (Cosa esattamente viene memorizzato è controllato dall'opzione 'sessionoptions', descritta più sotto)
Il seguente comando crea un file di sessione: >

```
:mksession vimbook.vim
```

Se dopo volete ripristinare la sessione, potete usare questo comando: >

```
:source vimbook.vim
```

Se volete avviare Vim e ripristinare una specifica sessione, potete usare il seguente comando: >

```
vim -S vimbook.vim
```

Questo dice a Vim di leggere all'avvio uno specifico file. La 'S' sta per sessione (al momento potete utilizzare −S per interpretare qualsiasi script sorgente per Vim, quindi potrebbe star bene anche per "sorgente").

Le finestre che erano aperte sono ripristinate, con la stessa posizione e dimensione di prima. Le mappature dei tasti e i valori delle opzioni sono le stesse di prima.
   Che cosa venga esattamente ripristinato dipende dall'opzione 'sessionoptions'. Il valore di default è "blank,buffers,curdir,folds,help,options,winsize".

blank	mantiene le finestre vuote
buffers	tutti i buffer, non solo quelli dentro ad una finestra
curdir	la directory corrente
folds	annidamenti, anche quelli creati manualmente
help	la finestra d'aiuto
options	tutte le opzioni e le mappature dei tasti
winsize	le dimensioni delle finestre

Cambiatela come preferite. Per esempio, per ripristinare anche la dimensione della finestra di Vim, usate: **>**

```
:set sessionoptions+=resize
```

SESSIONE QUI, SESSIONE LA'

Il modo più ovvio per usare le sessioni è quando si lavora su progetti differenti. Supponete di memorizzare i vostri file di sessione nella directory "~/.vim". Al momento state lavorando sul progetto "segreto" e volete spostarvi sul progetto "noioso": **>**

```
:wall
:mksession! ~/.vim/segreto.vim
:source ~/.vim/noioso.vim
```

La prima usa ":wall" per scrivere tutti i files modificati. Dopo, la sessione corrente viene salvata, usando ":mksession!". La precedente sessione viene sovrascritta. La prossima volta che caricherete la sessione segreta potrete continuare da dove eravate in questo momento. E finalmente caricate la nuova sessione "noioso".

Se aprite delle finestre di help, dividete e chiudete varie finestre, ed alterate in generale l'aspetto delle finestre, potrete tornare all'ultima sessione salvata: **>**

```
:source ~/.vim/noioso.vim
```

Avrete quindi un controllo completo se la prossima volta vorrete continuare da dove vi trovate adesso, salvando la configurazione corrente in una sessione o mantenedo i file di sessione come punto di partenza.
   Un altro modo di usare le sessioni è di creare una disposizione di finestre che vi piaccia usare e salvarla in una sessione. Potrete tornare a questa disposizione quando vorrete.
   Per esempio, questa è una bella disposizione da usare:

```
+----------------------------------------+
|                  VIM − main help file  |
|                                        |
|Move around:  Use the cursor keys, or "h|
|help.txt================================|
|explorer   |                            |
|dir        |~                           |
|dir        |~                           |
|file       |~                           |
|file       |~                           |
|file       |~                           |
|file       |~                           |
|~/=========|[No File]===================|
|                                        |
+----------------------------------------+
```

Ha una finestra d'aiuto all'inizio, così che possiate leggere questo testo. La stretta finestra verticale sulla sinistra contiene un navigatore di file. È un plugin di Vim che elenca il contenuto di una directory. Da li' potete selezionare i file da aprire. Maggiori informazioni in merito nel prossimo capitolo.
   Createla da un Vim appena avviato con: **>**

```
:help
CTRL-W w
:vertical split ~/
```

Potete ridimensionare un po' le finestre a seconda dei vostri gusti. Dopo salvate la sessione con: **>**
>

```
:mksession ~/.vim/mia.vim
```

Ora potete avviare Vim con questa disposizione: **>**

```
vim -S ~/.vim/mia.vim
```

Suggerimento: Per aprire un file che vedere elencato nella finestra del navigatore nella finestra vuota, muovete il cursore sul nome del file e premete "O". Un doppio click col mouse fa la stessa cosa.

UNIX E MS-WINDOWS

Alcune persone devono lavorare un giorno su sistemi MS-Windows e un altro giorno su Unix. Se siete uno di loro, considerate di aggiungere "slash" e "unix" a 'sessionoptions'. I file di sessione saranno scritti in un formato che potrà esser usato su entrambi i sistemi. Questo è il comando da inserire nel vostro file vimrc: **>**

```
:set sessionoptions+=unix,slash
```

Vim quindi userà il formato Unix, perchè il Vim per MS-Windows può leggere e scrivere file Unix, ma il Vim di Unix non può leggere file di sessione nel formato MS-Windows. Similarmente, il Vim di MS-Windows comprende i nomi dei file con / per separare i nomi, ma il Vim di Unix non comprende \.

SESSIONI E VIMINFO

Le sessioni memorizzano molte cose, ma non la posizione dei puntatori, il contenuto dei registri e la storia della linea di comando. Per queste cose avete bisogno delle capacità di viminfo.
   Nella maggior parte delle situazioni vorrete usare le sessioni separatamente da viminfo. Ciò può essere usato per spostarsi su un'altra sessione, ma mantenere la storia della linea di comando. E copiare del testo nei registri in una sessione e riportarlo in un'altra sessione.
   Potreste preferire mantenere le informazioni con la sessione. Dovrete farlo voi stessi allora. Esempio: **>**

```
:mksession! ~/.vim/segreto.vim
:wviminfo! ~/.vim/segreto.viminfo
```

E per ripristinarlo di nuovo: **>**

```
:source ~/.vim/segreto.vim
:rviminfo! ~/.vim/segreto.viminfo
```

==========
21.5 Viste

Una sessione memorizza l'aspetto dell'intero Vim. Quando volete memorizzare le proprietà di una sola finestra, usate una vista.
   La vista si usa quando volete lavorare su di un file in un modo specifico. Per esempio, avete il numero delle linee attivato con l'opzione 'number' e alcuni annidamenti sono definiti. Proprio come con le sessioni, potete memorizzare questa vista su un file e ripristinarla in seguito. Adesso, quando memorizzate una sessione, vengono salvate le viste di ogni finestra.
   Ci sono due modi principali per usare le viste. Il primo è di lasciare che sia Vim a scegliere il nome del file della vista. Potete ripristinare la vista quando in seguito aprirete lo stesso file. Per memorizzare la vista della finestra corrente: **>**

```
:mkview
```

Vim deciderà dove memorizzare la vista. Quando in seguito aprirete lo stesso file potrete riavere la vostra vista con questo comando: **>**

```
:loadview
```

È facile, non è vero?
   Ora vorreste vedere il file senza l' opzione 'number' attivata, o con tutti

gli annidamenti aperti, potete impostare le opzioni per rendere la finestra così. Dopo, memorizzare questa vista con: **>**

```
:mkview 1
```

Ovviamente, potete ricaricarla con: **>**

```
:loadview 1
```

Ora potete cambiare tra le due viste del file usando ":loadview" con, e senza, l' argomento "1".
   In questo modo potete memorizzare fino a dieci viste per lo stesso file, una senza numero e nove numerate da 1 a 9.

UNA VISTA CON NOME

Il secondo metodo principale per usare le viste è di memorizzarle in un file con un nome di vostra scelta. Questa vista potrà essere ricaricata mentre state aprendo un altro file. Vim cambierà il file aperto utilizzando quello specificato nella vista. Potete quindi usarla per cambiare velocemente il file aperto, con tutte le opzioni impostate come quando l'avete salvato.
   Per esempio, per salvare la vista del file corrente: **>**

```
:mkview ~/.vim/principale.vim
```

Potete ripristinarla con: **>**

```
:source ~/.vim/principale.vim
```

==
21.6 Modelines

Quando aprite uno specifico file, potete impostare opzioni per quel particolare file. Scrivere ogni volta questi comandi è noioso. Usare una sessione o vista per aprire un file non funziona quando il file è condiviso tra più persone.
  La soluzione per questa situazione è aggiungere una modeline al file.
  È una linea di testo che dice a Vim i valori delle opzioni da usare solo in questo file.
   Un tipico esempio è un programma in C dove dovete far rientrare le linee con multipli di 4 spazi. Ciò richiede l'impostazione dell'opzione 'shiftwidth' a 4. Questa modeline lo farà:

```
/* vim:set shiftwidth=4: */ ~
```

Inserite questa linea tra le prime o le ultime cinque righe nel file. Quando aprirete il file, noterete che 'shiftwidth' è stata impostata a quattro. Quando aprirete un altro file, verrà reimpostata al valore di default di otto.
   Per alcuni file la modeline si adegua bene ad essere inserita nell'header, sempre che possa esser inserita all'inizio. Per file di testo e altri file dove la modeline fa parte del normale contenuto, inseritela alla fine del file.

L'opzione 'modelines' specifica quante linee all'inizio e alla fine del file sono ispezionate per controllare se hanno una modeline. Per ispezionare dieci linee: **>**

```
:set modelines=10
```

L'opzione 'modeline' può esser usata per disattivare questa capacità. Fatelo quando state lavorando come root o non vi fidate dei file che state aprendo: **>**

```
:set nomodeline
```

usate questo formato per la modeline:

```
qualsiasi-testo vim:set {option}={value} ... : qualsiasi-testo ~
```

Il "qualsiasi-testo" indica che potete inserire qualsiasi testo prima e dopo la parte che userà Vim. Ciò permette di farlo sembrare un commento, come quello che è stato fatto sopra con /* e */.
   La parte " vim:" è quella che permette a Vim di riconoscere la linea. Ci deve essere uno spazio vuoto prima di "vim", oppure "vim" deve stare all'inizio della linea. Usare qualcosa tipo "gvim:" non funzionerà.
   La parte tra i doppi punti è un comando ":set". Funziona nello stesso modo che scrivere il comando ":set", eccezion fatta per il backslash che deve essere inserito prima del doppio punto (altrimenti lo vedrebbe come fine della

modeline).

Un altro esempio:

```
        // vim:set textwidth=72 dir=c\:\tmp:  use c:\tmp here ~
```

C'è un backslash in più prima del doppio punto, così sarà incluso nel comando ":set". Il testo dopo il secondo doppio punto sarà ignorato, quindi potete inserirci un commento.

Per maggiori informazioni guardate |modeline|.

==

Capitolo seguente: |usr_22.txt| Trovare il file da aprire

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

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

usr_22.txt   Per Vim version 6.2. Ultima modifica: 2003 Mar 17

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Stefano Palmeri

Trovare il file da aprire

I file possono venir trovati ovunque. Così, come fare a trovarli? Vim offre vari modi per esplorare l'albero delle directory. Ci sono comandi per saltare ad un file che è menzionato in un altro. E Vim ricorda quali file siano stati modificati in precedenza.

  Capitolo seguente: |usr_23.txt| Modifica di altri file
Capitolo precedente: |usr_21.txt| Andarsene e ritornare
             Indice: |usr_toc.txt|

==

22.1 Il file explorer

Vim ha un plugin che rende possibile visualizzare una directory. Provate questo: >

```
:edit .
```

Tramite la magia dei comandi automatici e degli script di Vim, la finestra verrà riempita con i contenuti della directory. Apparirà come questa:

```
" Press ? for keyboard shortcuts ~
" Sorted by name (.bak,~,.o,.h,.info,.swp,.obj,.orig,.rej at end of list) ~
"= /home/mool/vim/vim6/runtime/doc/ ~
../ ~
check/ ~
Makefile ~
autocmd.txt ~
change.txt ~
eval.txt~ ~
filetype.txt~ ~
help.txt.info ~
```

Potrete vedere queste voci:
1. Un commento su come usare ? per ricevere aiuto per le funzionalità del file explorer.
2. La seconda linea spiega come sono elencati i contenuti della directory. Essi possono essere ordinati in diversi modi.
3. La terza linea è il nome della directory corrente.
4. La voce "../" directory. Questa è la directory genitore.
5. I nomi delle directory.
6. I nomi dei file ordinari. Come rammentato nella seconda linea, alcuni non sono qui ma "alla fine dell'elenco".
7. I nomi dei file meno ordinari. Si suppone che non li usiate spesso, quindi sono stati spostati alla fine.

Se l'evidenziazione della sintassi è abilitata, le diverse parti sono messe in evidenza per poterle individuare più facilmente.

Potete usare i comandi di Vim in Normal mode per muovervi nel testo. Ad esempio, spostatevi su un file e premete <Invio>. Adesso state scrivendo su quel file. Per tornare indietro all'explorer usate di nuovo ":edit .".
  Funziona anche CTRL-O.
   Provate a usare <Invio> mentre il cursore è sul nome di una directory.
  Il risultato è che l'explorer si sposta in quella directory e mostra i suoi contenuti. Premere <Invio> sulla prima directory "../" vi sposta al livello superiore. Premere "-" fa la stessa cosa, senza il bisogno di muovere prima il cursore su "../".

Potete premere ? per avere un piccolo aiuto sulle cose che potete fare nell'explorer.
Questo è ciò che otterrete:

```
" <enter> : open file or directory ~
" o : open new window for file/directory ~
" O : open file in previously visited window ~
```

```
" p : preview the file ~
" i : toggle size/date listing ~
" s : select sort field    r : reverse sort ~
" - : go up one level      c : cd to this dir ~
" R : rename file          D : delete file ~
" :help file-explorer for detailed help ~
```

I primi pochi comandi sono per selezionare il file da vedere. A seconda del comando che userete, il file comparirà da qualche parte:

<Enter>	Usa la finestra corrente.
o	Apre una nuova finestra.
O	Usa la finestra visitata in precedenza.
p	Usa la finestra di anteprima (preview window) e sposta il cursore nella finestra dell'explorer. |preview-window|

I seguenti comandi sono usati per mostrare altre informazioni:

i	Mostra le dimensioni e la data dei file. Usando i di nuovo le informazioni verranno nascoste.
s	Usa il campo in cui si trova il cursore per ordinare. Prima mostrate le dimensioni e la data con i. Dopo spostate il cursore sulla dimensione di un qualsiasi file e premete s. I file adesso saranno ordinati per dimensione. Premete s mentre il cursore si trova su una data e le voci saranno ordinate per data.
r	Inverte l'ordine (sia la dimensione che la data)

Ci sono altri pochi comandi extra:

c	Imposta la directory corrente sulla directory mostrata. Potete poi battere un comando ":edit" per uno dei file senza anteporre il percorso.
R	Rinomina il file sotto il cursore. Vi sarà chiesto il nuovo nome.
D	Elimina il file sotto il cursore. Vi sarà chiesto di confermare questa azione.

==========

22.2 La directory corrente

Proprio come la shell, Vim ha il concetto di directory corrente. Supponete che voi siate nella vostra home directory e vogliate aprire alcuni file nella directory "VeryLongFileName". Potete fare: **>**

```
:edit VeryLongFileName/file1.txt
:edit VeryLongFileName/file2.txt
:edit VeryLongFileName/file3.txt
```

Per evitare troppe battiture, fate questo: **>**

```
:cd VeryLongFileName
:edit file1.txt
:edit file2.txt
:edit file3.txt
```

Il comando ":cd" cambia la directory corrente. Potete vedere quale sia la directory corrente con il comando ":pwd" : **>**

```
:pwd
/home/Bram/VeryLongFileName
```

Vim ricorda l'ultima directory che avete usato. Usate "cd -" per ritornarvi. Esempio: **>**

```
:pwd
/home/Bram/VeryLongFileName
:cd /etc
:pwd
/etc
:cd -
:pwd
/home/Bram/VeryLongFileName
:cd -
:pwd
/etc
```

LA FINESTRA DELLA DIRECTORY LOCALE

Quando dividete una finestra, entrambe le finestre useranno la stessa directory corrente. Quando volete modificare un certo numero di file da qualche altra parte nella nuova finestra, potete far sì che essa usi un'altra directory, senza cambiare la directory corrente nell'altra finestra. Questa si chiama directory locale. **>**

```
:pwd
/home/Bram/VeryLongFileName
:split
:lcd /etc
:pwd
/etc
CTRL-W w
:pwd
/home/Bram/VeryLongFileName
```

Fintanto che il comando ":lcd" non sia stato usato, tutte le finestre condivideranno la stessa directory corrente. Eseguire un comando ":cd" in una finestra cambierà la directory corrente anche nell'altra finestra.
Per una finestra dove sia stato usato il comando ":lcd" verrà ricordata una directory corrente differente. Usare ":cd" o ":lcd" in altre finestre non la cambierà.
Quando si usa il comando ":cd" in una finestra posta in una diversa directory corrente, farà tornare ad usare la directory condivisa.

==========

22.3 Trovare un file

State scrivendo un programma in linguaggio C che contiene questa linea:

```
#include "inits.h" ~
```

Volete vedere cosa c'è in quel file "inits.h". Muovete il cursore sul nome del file e battete: **>**

```
gf
```

Vim troverà il file e ne mostrerà il contenuto.
Cosa succede se il file non è nella directory corrente? Vim userà l'opzione 'path' per trovare il file. Questa opzione è un elenco di nomi di directory nelle quali cercare il vostro file.
Supponete che i vostri file include siano in "c:/prog/include". Questo comando la aggiungerà alla opzione 'path': **>**

```
:set path+=c:/prog/include
```

Questa directory si trova in un percorso assoluto. Non importa dove voi siate, sarà lo stesso posto. Cosa fare se avete collocato dei file in una subdirectory, al di sotto di dov'è il file? Potete specificare un percorso relativo. Questo inizia con un punto: **>**

```
:set path+=./proto
```

Questo dice a Vim di cercare nella directory "proto", sotto la directory dove si trova il file nel quale avete usato "gf". Così, usare "gf" su "inits.h" farà sì che Vim cerchi "proto/inits.h", iniziando nella directory del file.
Senza "./", quindi "proto", Vim dovrebbe cercare nella directory "proto" sotto la directory corrente. La directory corrente, però, potrebbe non essere quella dove il file che state modificando è collocato.

L'opzione 'path' permette di specificare le directory dove cercare i file in molti più modi. Leggete l'aiuto per l'opzione 'path'.
L'opzione 'isfnamè è usata per decidere quali caratteri sono inclusi nel nome del file e quali non lo sono (es., il carattere " nell'esempio in alto).

Quando conoscete il nome del file, ma non deve essere trovato nel file, potete battere questo: **>**

```
:find inits.h
```

Vim userà quindi l'opzione 'path' per trovare il file. Questa è la stessa cosa del comando ":edit", eccetto che per l'uso di 'path'.

Per aprire il file trovato in una nuova finestra usate CTRL-W f anziché "gf",

oppure usate ":sfind" al posto di ":find".

Un bel modo per avviare direttamente Vim per aprire un file che sia ovunque nel 'path' è: >

```
vim "+find stdio.h"
```

Ciò trova "stdio.h" nel vostro valore di 'path'. I doppi apici sono necessari per avere un solo argomento |-+c|.

==

22.4 La lista dei buffer

L'editor Vim usa il buffer del terminale per descrivere un file che si sta aprendo. In realtà, il buffer è una copia del file sul quale voi state lavorando. Quando avrete finito di modificare il buffer, voi scriverete i contenuti del buffer nel file. I buffer non contengono solo i contenuti del file, ma anche tutti i segnalibri, le impostazioni e le altre cose che lo accompagnano.

NASCONDERE I BUFFER

Supponete che voi stiate lavorando sul file one.txt e abbiate bisogno di passare al file two.txt.
Potreste usare semplicemente ":edit two.txt", ma poichè avete fatto delle modifiche in one.txt, quel comando non funzionerà. Inoltre non volete ancora salvare one.txt. Vim ha una soluzione per voi: >

```
:hide edit two.txt
```

Il buffer "one.txt" scompare dallo schermo, ma Vim sa ancora che voi state lavorando su questo buffer, così conserva il testo modificato. Questo viene chiamato buffer nascosto: il buffer contiene il testo, ma voi non potete vederlo.
L'argomento del comando ":hide" è un altro comando. Fa sì che quel comando appaia come se l'opzione 'hidden' fosse impostata. Potete anche voi stessi impostare questa opzione. L'effetto è che quando un buffer viene lasciato, esso diventa nascosto.
State attenti! Quando avete nascosto buffer con cambiamenti, non chiudete Vim senza essere sicuri di avere salvato tutti i buffer.

INATTIVARE I BUFFER

Quando un buffer è stato usato una volta, Vim ricorda alcune informazioni su di esso. Quando non appare in una finestra e non è nascosto, esso è ancora nella lista dei buffer. Questo viene chiamato buffer inattivo. Panoramica:

Attivo	Appare in una finestra, testo caricato.
Nascosto	Non è in una finestra, testo caricato.
Inattivo	Non è in una finestra, testo non caricato.

I buffer inattivi vengono memorizzati, poiché Vim conserva le informazioni che li riguardano, come i segnalibri. Ed anche il ricordare il nome del file è utile, affinché voi possiate vedere su quali file avete lavorato. Ed aprirli ancora.

ELENCARE I BUFFERS

Esaminate la lista dei buffer con questo comando: >

```
:buffers
```

Un comando che fa la stessa cosa, non così ovvio per elencare i buffer, ma molto più corto da battere, è: >

```
:ls
```

L'output potrebbe apparire come questo:

```
  1 #h  "help.txt"                    line 62 ~
  2 %l+ "usr_21.txt"                  line 1 ~
  3     "usr_toc.txt"                 line 1 ~
```

La prima colonna contiene il numero del buffer. Potete usare questo per

ritornare ad un buffer senza doverne battere il nome, guardate sotto.
   Dopo il numero del buffer vengono i flag. Quindi segue il nome del file ed il numero della linea dove era il cursore l'ultima volta.
   I flag che possono apparire sono questi (da sinistra a destra):

```
        u       Il buffer è unlisted (non in lista) |unlisted-buffer|.
         %      Buffer corrente.
         #      Buffer alternativo.
          l     Il buffer è caricato e mostrato.
          h     Il buffer è caricato ma nascosto.
           =    Il buffer è di sola lettura (read-only).
           -    Il buffer non è modificabile, l'opzione 'modifiable' è
                inattiva.
            +   Il buffer è stato modificato.
```

RITORNARE AD UN BUFFER

Potete ritornare ad un buffer tramite il suo numero. Questo evita di dover scrivere il nome del file : **>**

```
        :buffer 2
```

Ma il solo modo di sapere il numero è guardare nell'elenco dei buffer. Potete, invece, usare il nome, o parte di esso: **>**

```
        :buffer help
```

Vim troverà la migliore corrispondenza per il nome che avete battuto. Se c'è un solo buffer che corrisponde al nome, esso verrà usato. In questo caso "help.txt".
   Per aprire un buffer in una nuova finestra: **>**

```
        :sbuffer 3
```

Funziona anche con il nome ugualmente bene.

USARE LA LISTA DEI BUFFER

Potete muovervi nella lista dei buffer con questi comandi:

```
        :bnext          va al buffer successivo
        :bprevious      va al buffer precedente
        :bfirst         va al primo buffer
        :blast          va all'ultimo buffer
```

Per rimuovere un buffer dalla lista, usate questo comando: **>**

```
        :bdelete 3
```

Ancora, ciò funziona anche con un nome.
   Se cancellate un buffer che era attivo (visibile in una finestra), quella finestra verrà chiusa. Se cancellate il buffer corrente, la finestra corrente verrà chiusa. Se fosse l'ultima finestra, Vim cercherebbe un altro buffer da modificare. Voi non potete non avere nulla di aperto!

        Note:
        Anche dopo aver rimosso il buffer con ":bdelete", Vim lo ricorderà. In realtà esso è reso "unlisted" e non compare più nella lista di ":buffers". Il comando ":buffers!" elencherà i buffer unlisted (sì, Vim può fare l'impossibile). Perchè Vim dimentichi veramente un buffer, usate ":bwipe". Vedete anche l'opzione 'buflisted'.

==

Capitolo seguente: |usr_23.txt| Modifica di altri file

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

usr_23.txt   Per Vim version 6.1. Ultima modifica: 2001 Set 03

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Cristian Rigamonti

Modifica di altri file

Questo capitolo tratta l'elaborazione di file diversi da quelli ordinari.
Con Vim potete elaborare file compressi o cifrati, file a cui si accede solo
via internet e, con alcune restrizioni, file binari.

  Capitolo seguente: |usr_24.txt| Inserzione rapida
Capitolo precedente: |usr_22.txt| Trovare il file da aprire
            Indice: |usr_toc.txt|

==
23.1 File DOS, Mac e Unix

Ai vecchi tempi, le macchine telescriventi usavano due caratteri per iniziare
una nuova linea: uno per far ritornare il carrello alla prima posizione
(carriage return, <CR>), un altro per far scorrere la carta (line feed, <LF>).
   Con l'arrivo dei computer, lo spazio di immagazzinamento divenne costoso e
qualcuno decise che non c'era bisogno di due caratteri per l'interruzione di
linea. I programmatori UNIX decisero di usare solo <Line Feed> per
l'interruzione di linea, i programmatori Apple si accordarono per <CR>, quelli
MS-DOS (e Microsoft Windows) decisero di tenere il vecchio <CR><LF>.
   Questo significa che se provate a trasferire un file da un sistema
all'altro, incontrate dei problemi nelle interruzioni di linea. L'editor Vim
riconosce automaticamente i diversi formati di file e si occupa di gestirli in
modo trasparente.
   L'opzione 'fileformats' contiene i vari formati che verranno provati quando
viene elaborato un nuovo file. Il comando seguente, ad esempio, dice a Vim di
provare per primo il formato UNIX e poi quelllo MS-DOS: >

```
:set fileformats=unix,dos
```

Riconoscerete il formato dal messaggio che otterrete quando aprite un file. Se
state usando il formato nativo non vedrete alcun messaggio, questo succede ad
esempio elaborando un file Unix su Unix; se invece aprite un file dos, Vim vi
avvertirà:

```
"/tmp/test" [dos] 3L, 71C ~
```

Per un file Mac vedreste "[mac]".
   Il formato del file riconosciuto sarà immagazzinato nell'opzione
'fileformat'. Per vedere il formato corrente, eseguite il seguente comando: >

```
:set fileformat?
```

I tre nomi usati da Vim sono:

```
unix            <LF>
dos             <CR><LF>
mac             <CR>
```

USARE IL FORMATO MAC

Su Unix, <LF> è usato per interrompere una linea. Non è insolito avere un
carattere <CR> nel mezzo di una linea. Incidentalmente, ciò capita abbastanza
spesso negli script di Vi (e Vim).
   Sul Macintosh, dove <CR> è il carattere di interruzione di linea, è
possibile incontrare un carattere <LF> all'interno di una linea.
   Il risultato è che non è possibile essere sicuri al 100% del fatto che un
file che contiene sia caratteri <CR> che <LF> sia un file Mac o Unix. Vim
assume quindi che su Unix probabilmente non si useranno file Mac e non
controllerà per questo tipo di file. Per controllare comunque anche per
questo tipo di file, aggiungete "mac" a 'fileformats': >

```
:set fileformats+=mac
```

Allora Vim darà uno sguardo al formato del file. Attenzione ai casi in cui Vim veda sbagliato.

FORZARE IL FORMATO

Se usate il buon vecchio Vi e provate a elaborare un file in formato MS-DOS, vi accorgerete che ogni linea finisce con un carattere ^M (^M corrisponde a <CR>).
Il riconoscimento automatico di Vim evita questa situazione. Ma supponiamo che invece vogliate elaborare il file nel suo formato originario: dovete forzare il formato: **>**

```
:edit ++ff=unix file.txt
```

La stringa "++" avvisa Vim che il valore dell'opzione specificata prevarrà sul valore predefinito limitatamente al comando che segue. "++ff" è usato al posto di 'fileformat'; potete anche usare "++ff=mac" o "++ff=dos".
   Questo meccanismo non funziona per tutte le opzioni: al momento sono implementate solo "++ff" e "++enc". Si possono usare anche i nomi completi "++fileformat" e "++encoding".

CONVERSIONE

Potete usare l'opzione 'fileformat' per convertire un file da un formato all'altro. Supponete, ad esempio, di avere un file MS-DOS chiamato README.TXT, che volete convertire in formato UNIX. Iniziate a elaborare il file in formato MS-DOS: **>**

```
vim README.TXT
```

Vim riconoscerà che questo è un file in formato dos. Ora cambiate il formato in UNIX: **>**

```
:set fileformat=unix
:write
```

Il file viene scritto in formato Unix.

==
23.2 File su internet

Qualcuno vi spedisce un messaggio e-mail che fa riferimento ad un file tramite la sua URL. Ad esempio:

```
Trovi le informazioni qui: ~
        ftp://ftp.vim.org/pub/vim/README ~
```

Potreste avviare un programma per scaricare il file, salvarlo sul vostro disco locale ed allora avviare Vim per elaborarlo.
   C'è un modo più semplice. Spostate il cursore su un qualsiasi carattere dell'URL. Poi usate questo comando: **>**

```
gf
```

Con un pò di fortuna, Vim troverà quale programma usare per scaricare il file, scaricarlo ed aprirne la copia. Per aprire il file in una nuova finestra usate CTRL-W f.
   Se qualcosa andasse storto, riceverete un messaggio di errore. E' possibile che l'URL sia sbagliata, che non abbiate il permesso di leggerlo, che la connessione di rete non sia attiva, ecc. Purtroppo è difficile stabilire la causa dell'errore. Dovrete tentare il modo manuale di scaricare il file.

L'accesso dei file via internet funziona col plugin netrw. Al momento sono riconosciuti gli URL con questi formati:

```
ftp://          usa ftp
rcp://          usa rcp
scp://          usa scp
http://         usa wget (sola lettura)
```

Vim non fa lui stesso la comunicazione, ricorre ai programmi menzionati che sono installati sul vostro computer. Sulla maggior parte dei sistemi Unix "ftp" e "rcp" saranno presenti; "scp" e "wget" probabilmente dovranno venire installati.

Vim riconosce queste URL per ciascun comando che inizi ad aprire un nuovo file, anche, ad esempio, con ":edit" e ":split". Funziona anche nei comandi di scrittura, eccetto per http://.

Per maggiori informazioni, anche a proposito delle password, si veda |netrw|.

==
23.3 Cifratura

Alcune informazioni preferite tenerle per voi stessi. Ad esempio, quando state scrivendo una prova d'esame su un computer usato anche dagli studenti. Non volete che i più furbi trovino un modo per leggere le domande prima dell'inizio dell'esame. Vim può cifrare il file, dandovi qualche protezione.
Per iniziare a scrivere un nuovo file con la cifratura, usate l'argomento "-x" per avviare Vim. Esempio: >

```
vim -x exam.txt
```

Vim vi chiede una chiave usata per cifrare e decifrare il file:

```
Enter encryption key: ~
```

Accuratamente scrivete la chiave segreta: non vedrete i caratteri che digitate, saranno sostituiti da asterischi. Per evitare che un errore di battitura causi problemi, Vim vi chiede di immettere ancora la chiave:

```
Enter same key again: ~
```

Ora potete elaborare il file normalmente e scriverci tutti i vostri segreti. Quando avete finito e dite a Vim di uscire, il file viene cifrato e scritto.
Quando aprite il file con Vim, vi chiederà di immettere nuovamente la stessa chiave. Non avrete bisogno di usare l'argomento "-x". Potete anche usare il normale comando ":edit". Vim aggiunge una stringa magica al file, dalla quale riconosce che il file è stato cifrato.
Se tentate di vedere il file con un altro programma, tutto ciò che ottenete è spazzatura. Così, se aprite il file con Vim e immettete la chiave sbagliata, ottenete spazzatura. Vim non ha un meccanismo per controllare se la chiave sia quella giusta (ciò rende molto più difficile tentare di indovinare la chiave).

ATTIVARE E DISATTIVARE LA CIFRATURA

Per disabilitare la cifratura di un file, impostate l'opzione 'key' a una stringa vuota: >

```
:set key=
```

La prossima volta che scriverete il file, verrà fatto senza cifratura.
Impostare l'opzione 'key' per abilitare la cifratura non è una buona idea, visto che la chiave apparirebbe in chiaro. Chiunque fosse dietro di voi potrebbe leggere la chiave.
Per evitare questo problema è stato introdotto il comando ":X". Vi chiede una chiave di cifratura, proprio come fa l'argomento "-x": >

```
:X
Enter encryption key: ******
Enter same key again: ******
```

LIMITI DELLA CIFRATURA

L'algoritmo di cifratura usato da Vim è debole. È abbastanza valido per tenere alla larga il ficcanaso occasionale, ma non lo è abbastanza per un esperto crittologo con molto tempo a disposizione. Dovete anche tenere presente che il file di swap non è cifrato, quindi mentre elaborate un file, gli utenti con privilegi di amministratore possono leggere il testo in chiaro da questo file.
Un modo per evitare che la gente legga il vostro file di swap è quello di non usarne uno. Aggiungendo l'argomento -n alla linea di comando, non viene usato alcun file di swap: Vim terrà tutto in memoria. Ad esempio per elaborare il file cifrato "file.txt" senza usare il file di swap, usate il comando seguente: >

```
vim -x -n file.txt
```

Quando state elaborando un file, il file di swap può esere disabilitato con: >

```
:setlocal noswapfile
```

Poichè manca il file di swap, il recupero diverrà impossibile. Salvate il file un pò più spesso del solito per evitare il rischio di perdere le modifiche effettuate.

Sino a quando è in memoria, il file è in chiaro. Qualunque utente privilegiato può guardare nella memoria dell'editor e scoprire i contenuti del file.
   Se usate un file viminfo, fate attenzione che i contenuti dei registri di testo sono scritti in chiaro.
   Se davvero volete proteggere i contenuti di un file, elaboratelo solo su un computer portatile non connesso in rete, usate buoni strumenti crittografici e tenete il computer chiuso in un posto sicuro quando non lo usate.

==
23.4 File binari

Potete aprire file binari con Vim. Poiché Vim non è davvero fatto per questo, ci sono alcune restrizioni. Ma potete leggere un file, cambiare un carattere e riscriverlo, col risultato che solo quel carattere sarà stato modificato, mentre il file è identico altrove.
   Per essere sicuri che Vim non usi qualcuno dei suoi trucchi nel modo sbagliato, aggiungete l'argomento "-b" all'avvio: **>**

```
vim -b datafile
```

Ciò imposta l'opzione 'binary', che ha l'effetto di disattivare effetti collaterali inaspettati. Ad esempio, 'textwidth' è impostato a zero per evitare la formattazione automatica delle linee. Ed i file vengono comunque letti in formato Unix.

Il Binary mode può essere usato per modificare un messaggio in un programma. Fate attenzione a non inserire o cancellare qualche carattere, il programma non funzionerebbe più. Usate "R" per entrare in modalità sostituzione.

Molti caratteri del file non saranno stampabili. Per vederli in formato Hex: **>**

```
:set display=uhex
```

Altrimenti potete usare il comando "ga" per vedere il valore del carattere sotto il cursore. Il risultato, quando il cursore è su un <Esc>, è questo:

```
<^[>  27,  Hex 1b,  Octal 033 ~
```

Potrebbero non esserci molte interruzioni di linea nel file. Per avere una panoramica del file, disabilitate l'opzione 'wrap': **>**

```
:set nowrap
```

POSIZIONE IN BYTE

Per vedere in quale byte del file vi trovate, usate questo comando: **>**

```
g CTRL-G
```

Il risultato è lungo:

```
    Col 9-16 of 9-16; Line 277 of 330; Word 1806 of 2058; Byte 10580 of 12206 ~
```

Gli ultimi due numeri sono la posizione in byte nel file e il numero totale di byte. Viene tenuto conto di come 'fileformat' modifica il numero di byte usati dalle interruzioni di linea.
    Per spostarvi ad un byte specifico del file, usate il comando "go". Ad esempio, per spostarvi al byte 2345: **>**

```
2345go
```

USARE XXD

Un vero editor binario mostra il testo in due modi: così come è ed in formato hex. Potete farlo in Vim convertendo prima il file con il programma "xxd", fornito con Vim.
   Per prima cosa, aprite il file in modalità binaria: **>**

```
vim -b datafile
```

Ora convertite il file in un hex dump usando xxd: **>**

```
:%!xxd
```

Il testo apparirà così:

```
0000000: 1f8b 0808 39d7 173b 0203 7474 002b 4e49  ....9..;..tt.+NI ~
0000010: 4b2c 8660 eb9c ecac c462 eb94 345e 2e30  K,.`.....b..4^.0 ~
0000020: 373b 2731 0b22 0ca6 c1a2 d669 1035 39d9  7;'1."......i.59. ~
```

Ora potete vedere ed modificare il testo a vostro piacimento. Vim tratta l'informazione come testo ordinario. Modificare il codice hex non modifica automaticamente il carattere stampabile, o quant'altro.
Alla fine convertitelo di nuovo con: **>**

```
:%!xxd -r
```

Solo le modifiche alla parte hex vengono usate. Le modifiche alla parte del testo stampabile sulla destra vengono ignorate.

Per maggiori informazioni, consultare la pagina di manuale di xxd.

==========

23.5 File compressi

Questo è facile: potete elaborare un file compresso come qualsiasi altro file. Il plugin "gzip" si occupa di decomprimere il file quando lo aprite. E di comprimerlo quando lo scrivete.
Attualmente sono supportati questi metodi di compressione:

```
.Z      compress
.gz     gzip
.bz2    bzip2
```

Per comprimere e decomprimere, Vim usa i programmi menzionati sopra. Potrebbe essere necessario installarli prima.

==========

Capitolo seguente: |usr_24.txt| Inserzione rapida

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

usr_24.txt   Per Vim version 6.2. Ultima modifica: 2003 Ago 18

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Giuliano Bordonaro

Inserzione rapida

Quando immettete del testo, Vim vi offre molti modi per ridurre il numero di battute ed evitare errori di battitura. Utilizzate l'Insert mode completion per ripetere le parole battute in precedenza. Accorciate le parole lunghe in parole brevi. Scrivete caratteri che non sono presenti in tastiera.

  Capitolo seguente: |usr_25.txt| Lavorare con testo formattato
Capitolo precedente: |usr_23.txt| Modifica di altri file
              Indice: |usr_toc.txt|

==========

24.1 Effettuare correzioni

Del tasto <BS> abbiamo già parlato. Cancella i caratteri immediatamente precedenti il cursore. Il tasto <Del> fa la stessa cosa per i caratteri posti sotto (dopo) il cursore.
   Avendo scritto qualche parola sbagliata potete usare CTRL-W:

```
        The horse had fallen to the sky ~
                                      CTRL-W
        The horse had fallen to the ~
```

Se aveste sbagliato un'intera linea e voleste ripartire da capo, potreste usare CTRL-U per cancellarla. Ciò conserverebbe il testo posto dopo il cursore e l'indentazione. Solo il testo dal suo inizio alla posizione del cursore verrebbe cancellato. Con il cursore sulla "f" di "fallen" nella linea che segue premendo CTRL-U avverebbe quanto segue:

```
        The horse had fallen to the ~
                     CTRL-U
        fallen to the ~
```

Se trovaste un errore poche parole indietro bisognerebbe spostarvi il cursore per correggerlo. Ad esempio, avendo scritto così:

```
        The horse had follen to the ground ~
```

Volendo cambiare "follen" in "fallen". Con il cursore a fine linea potreste scrivere così per correggerlo: >

```
                                        <Esc>4blraA

<       Uscire dall'Insert mode         <Esc>
        indietro di quattro parole           4b
        vai sulla "o"                          l
        sostituiscila con una "a"              ra
        riavvia l'Insert mode                    A
```

Un altro modo di farlo: >

```
                <C-Left><C-Left><C-Left><C-Left><Right><Del>a<End>

<       indietro di quattro parole          <C-Left><C-Left><C-Left><C-Left>
        vai sopra la "o"                       <Right>
        cancella la "o"                               <Del>
        inserisci una "a"                                  a
        vai a fine linea                                    <End>
```

Questo impiega i tasti speciali per muoversi attorno, restando nell'Insert mode. Ciò assomiglia a quanto fareste con un editor non modale. E' più

facile da ricordare, ma richiede più tempo (dovreste spostarvi tra lettere
e tasti cursore ed il tasto <End> risulta difficile da premere senza guardare
la tastiera).
   Questi tasti speciali sono molto utili per scrivere senza lasciare l'Insert
mode. Allora il fatto di dover scrivere di più non costituisce un problema.
   Una vista d'assieme dei tasti che potete usare nell'Insert mode:

	<C-Home>	va all'inizio del file
	<PageUp>	si sposta di uno schermo verso l'alto
	<Home>		va all'inizio della linea
	<S-Left>	si sposta di una parola a sinistra
	<C-Left>	si sposta di una parola a sinistra
	<S-Right>	si sposta di una parola a destra
	<C-Right>	si sposta di una parola a destra
	<End>		va alla fine della linea
	<PageDown>	si sposta di uno schermo verso il basso
	<C-End>		va alla fine del file

Ce ne sono alcuni in più, vedere |ins-special-special|.

==
24.2 Evidenziare le corrispondenze

Quando scrivete un) potrebbe esservi utile vedere a quale (corrisponda. Per
ottenere che Vim faccia ciò potete usare questo comando: >

	:set showmatch

Quando scrivete un testo come "(example)", appena scritto) Vim sposterà per
un attimo il cursore sul corrispondente (, rimarrà lì per mezzo secondo, poi
tornerà indietro dove stavate scrivendo.
   Se non ci fosse il corrispondente (, Vim emetterebbe un suono. Così
sapreste di aver dimenticato il (da qualche parte, o scritto un) di troppo.
   La corrispondenza viene cercata anche per le coppie [] e {}. Non dovete
aspettare per scrivere la prossima lettera, appena Vim trova la corrispondenza
il cursore tornerà indietro e l'inserzione continuerà come prima.
   Potete modificare questo tempo di verifica con l'opzione 'matchtime'. Ad
esempio, per far si che Vim attenda per un secondo e mezzo: >

	:set matchtime=15

Il tempo va specificato in decimi di secondo.

==
24.3 Completamento

Vim può completare automaticamente le parole durante l'inserzione. Scrivete
la prima parte di una parola, premete CTRL-P, e Vim provvederà al resto.
   Immaginate, ad esempio, di stare scrivendo un programma in C e di voler
scrivere:

	total = ch_array[0] + ch_array[1] + ch_array[2]; ~

Inizierete scrivendo quanto segue:

	total = ch_array[0] + ch_ ~

Ora dite a Vim di completare la parola usando il comando CTRL-P.
Vim cercherà una parola che inizi con ciò che sta davanti al cursore. In
questo caso, si tratta di "ch_", che corrisponde con la parola ch_array.
Così se battete CTRL-P otterrete:

	total = ch_array[0] + ch_array ~

Dopo un minimo sforzo di battitura otterrete (terminando con uno spazio):

	total = ch_array[0] + ch_array[1] + ~

Se adesso battete CTRL-P Vim cercherà ancora una parola che completi quella
prima del cursore. Poichè non c'è nulla davanti al cursore, allora troverà la
prima parola cercando all'indietro, che è "ch_array". Battere ancora CTRL-P
fornirà la prossima parola corrispondente, in questo caso "total". Un terzo
CTRL-P cercherà ancora all'indietro. Se non vi fosse altro, l'editor, avendo
terminato le parole, tornerebbe al testo originale, che è nullo. Un quarto
CTRL-P farà ripartire l'editor con "ch_array".

Per cercare in avanti usate CTRL-N. Quando la ricerca avrà raggiunto la fine
del file, CTRL-N e CTRL-P troveranno le stesse corrispondenze, ma in sequenza

diversa. Significato: CTRL-N vuol dire Next-match e CTRL-P significa Previous-match.

Vim tenterà di tutto per trovare parole da completare. Di default, cercherà nei seguenti posti:

1. File attuale
2. File in altre finestre
3. Altri file caricati (buffer nascosti)
4. File non caricati (buffers inattivi)
5. Tag files
6. Tutti i file #included dal file corrente

OPZIONI

Potete personalizzare l'ordine di ricerca con l'opzione 'complete'.

Uso dell'opzione 'ignorecase'. Quando viene impostata, vengono ignorate le differenze tra maiuscole e minuscole nella ricerca delle corrispondenze.

Un'opzione speciale per il completamento è 'infercase'. Risulta utile per trovare corrispondenze ignorando la distinzione tra maiuscole e minuscole ('ignorecase' deve essere impostata) ma utilizzando l'attributo della parola precedentemente battuta. Così, se scriveste "For", Vim troverebbe la corrispondenza in "fortunately", il risultato sarà "Fortunately".

COMPLETAMENTO DI ELEMENTI SPECIFICI

Se sapete cosa state cercando, potete usare questi comandi per completare a mezzo di un certo tipo di elementi:

CTRL-X CTRL-F	nomi di file
CTRL-X CTRL-L	talune linee
CTRL-X CTRL-D	definizioni di macro (anche in file inclusi)
CTRL-X CTRL-I	file corrente ed inclusi
CTRL-X CTRL-K	parole da un dizionario
CTRL-X CTRL-T	parole da un thesaurus
CTRL-X CTRL-]	marcature
CTRL-X CTRL-V	linea di comando di Vim

Dopo ciascuno di essi CTRL-N può essere usato per trovare la prossima corrispondenza, CTRL-P per quella precedente.
   Una maggiore informazione per ciascuno di questi comandi qui: |ins-completion|.

COMPLETAMENTO DEI NOMI DEI FILE

Prendiamo ad esempio CTRL-X CTRL-F. Serve a trovare nomi di file. Esamina la directory corrente e mostra ciascun file che corrisponda con la parola davanti al cursore.
   Ad esempio, immaginate di avere i seguenti file nella directory corrente:

```
main.c  sub_count.c  sub_done.c  sub_exit.c
```

Entrate in Insert mode ed iniziate a scrivere:

```
The exit code is in the file sub ~
```

A questo punto digitate il comando CTRL-X CTRL-F. Vim completa ora la parola "sub" osservando i file nella directory attuale. La prima corrispondenza è sub_count.c. Ciò non è quello che volevate, così provate il prossimo file con CTRL-N. La corrispondenza è sub_done.c. Scrivendo ancora una volta CTRL-N vi darà sub_exit.c. Il risultato:

```
The exit code is in the file sub_exit.c ~
```

Se il nome del file iniziasse con / (Unix) o C:\ (MS-Windows) potreste trovare tutti i file del file system. Ad esempio, digitate "/u" e CTRL-X CTRL-F. Ciò troverà corrispondenza in "/usr" (ciò in Unix):

```
the file is found in /usr/ ~
```

Se adesso premete CTRL-N tornate a "/u". Invece, per accettare "/usr/" ed andare ad una directory di livello più basso, usate ancora CTRL-X CTRL-F:

```
        the file is found in /usr/X11R6/ ~
```

I risultati dipenderanno da cosa si trova nel vostro file system, ovviamente. Le corrispondenze vengono ordinate alfabeticamente.

==

24.4 Ripetizione di un inserimento

Se premete CTRL-A, l'editor inserisce il testo scritto l'ultima volta che vi trovavate nell'Insert mode.
   Ad esempio, supponete di avere un file che inizi con i seguenti:

```
        "file.h" ~
        /* Main program begins */ ~
```

Modificate questo file inserendo "#include " all'inizio della prima linea:

```
        #include "file.h" ~
        /* Main program begins */ ~
```

Scendete all'inizio della linea seguente con il comando "j^". Inserite una nuova linea "#include". Così scrivete: **>**

```
        i CTRL-A
```

Ne deriverà quanto segue:

```
        #include "file.h" ~
        #include /* Main program begins */ ~
```

E' stato incluso "#include " perchè CTRL-A inserisce il testo dell'inserimento precedente. Ora scrivete "main.h"<Enter> per completare la linea:

```
        #include "file.h" ~
        #include "main.h" ~
        /* Main program begins */ ~
```

Il comando CTRL-@ produce un CTRL-A ed esce dall'Insert mode. E' un modo veloce per fare lo stesso inserimento un'altra volta.

==
24.5 Copiare da un'altra linea

Il comando CTRL-Y inserisce il carattere prima del cursore. Risulta utile dovendo duplicare una linea precedente. Ad esempio, posta questa linea di codice C:

```
        b_array[i]->s_next = a_array[i]->s_next; ~
```

Adesso dovete scrivere la stessa linea, ma con "s_prev" invece di "s_next". Andate a capo e premete CTRL-Y 14 volte, sino a quando giungerete alla "n" di "next":

```
        b_array[i]->s_next = a_array[i]->s_next; ~
        b_array[i]->s_ ~
```

Adesso scrivete "prev":

```
        b_array[i]->s_next = a_array[i]->s_next; ~
        b_array[i]->s_prev ~
```

Continuate premendo CTRL-Y sino al prossimo "next":

```
        b_array[i]->s_next = a_array[i]->s_next;~
        b_array[i]->s_prev = a_array[i]->s_ ~
```

Adesso scrivete "prev;" per terminare.

Il comando CTRL-E fa come CTRL-Y ad eccezione del fatto che inserisce il carattere dopo il cursore.

==
24.6 Inserire un registro

Il comando CTRL-R {register} inserisce i contenuti del registro. Ciò risulta utile per evitare di dover scrivere una parola lunga. Ad esempio, se volete scrivere:

```
        r = VeryLongFunction(a) + VeryLongFunction(b) + VeryLongFunction(c) ~
```

Il nome della funzione è definito entro un file diverso. Aprite questo file e muovete il cursore all'inizio del nome della funzione, copiatelo ora nel registro v: **>**

```
        "vyiw
```

"v è la specificazione del registro, "yiw" sta per yank-inner-word. Ora aprite il file in cui volete inserire la nuova linea e premete le prime lettere:

```
        r = ~
```

Adesso, con CTRL-R v verrà inserito il nome della funzione:

```
        r = VeryLongFunction ~
```

Continuate a scrivere caratteri entro il nome della funzione ed usate ancora due volte CTRL-R v.
   Potreste fare lo stesso per il completamento. Usare un registro è utile se ci fossero troppo parole inizianti con lo stesso carattere.

Se il registro contenesse caratteri come <BS> od altri caratteri speciali, essi verrebbero interpretati come se fossero stati battuti dalla tastiera. Se non volete che ciò accada (volete che venga inserito davvero nel testo il <BS>), usate il comando CTRL-R CTRL-R {register}.

==
24.7 Abbreviazioni

Un'abbreviazione è una parola breve che prende il posto di una lunga. Ad esempio, "ad" sta per "advertisement". Vim permette di scrivere un'abbreviazione e la espanderà automaticamente.
   Per dire a Vim di espandere "ad" in "advertisement" ogni volta che venga inserita, usate il comando seguente: **>**

```
        :iabbrev ad advertisement
```

Adesso, scrivendo "ad", tutta la parola "advertisement" verrà inserita nel testo. Lo otterrete scrivendo un carattere che non fa parte della parola, ad esempio uno spazio:

```
        What Is Entered         What You See
        I saw the a             I saw the a ~
        I saw the ad            I saw the ad ~
        I saw the ad<Space>     I saw the advertisement<Space> ~
```

L'espansione non avviene scrivendo solo "ad". Ciò vi permette di scrivere una parola come "add", che non deve essere espansa. Solo le parole intere vengono prese in esame per le abbreviazioni.

ABBREVIARE DIVERSE PAROLE

E' possibile definire un'abbreviazione che si sviluppi in diverse parole. Ad esempio, per definire "JB" come "Jack Benny", usate il seguente comando: **>**

```
        :iabbrev JB Jack Benny
```

Come programmatore, uso due abbreviazioni abbastanza insolite: **>**

```
        :iabbrev #b /****************************************
        :iabbrev #e <Space>****************************************/
```

Servono per generare "boxed comments". Il commento parte con #b, che disegna la linea sopra. Poi si scrive il testo del commento e si usa #e per disegnare la linea sotto.
   Attenzione al fatto che l'abbreviazione #e inizia con uno spazio. In altre parole, i primi due caratteri sono spazio-asterisco. Di solito Vim ignora gli spazi tra l'abbreviazione e l'espansione. Per evitare questo problema, sillabo "space" come sette caratteri: <, S, p, a, c, e, >.

```
        Note:
        ":iabbrev" è una parola lunga da scrivere.  ":iab" va meglio.
        Ciò significa abbreviare il comando abbreviate!
```

CORREZIONE DEGLI ERRORI DI SCRITTURA

E' comunissimo ripetere spesso lo stesso errore di battitura. Ad esempio, scrivere "teh" invece di "the". Potete rimediare con un'abbreviazione: **>**

```
:abbreviate teh the
```

Potete aggiungerne un'intera lista. Aggiungetene una ogni volta che scoprite un errore comune.

ELENCARE LE ABBREVIAZIONI

Il comando ":abbreviate" elenca le abbreviazioni:

```
:abbreviate
i  #e             ****************************************/
i  #b            /****************************************
i  JB            Jack Benny
i  ad            advertisement
!  teh           the
```

La lettera "i" nella prima colonna indica l'Insert mode. Queste abbreviazioni sono attive soltanto nell'Insert mode. Altri possibili caratteri sono:

```
c       Command-line mode                       :cabbrev
!       Entrambi, Insert e Command-line mode    :abbreviate
```

Poichè le abbreviazioni non sono utili spesso nel Command-line mode, userete preferibilmente il comando ":iabbrev". Ciò eviterà, ad esempio, che "ad" venga espanso quando state scrivendo un comando come: **>**

```
:edit ad
```

CANCELLARE LE ABBREVIAZIONI

Per eliminare un'abbreviazione usate il comando ":unabbreviate". Supponete di avere la seguente abbreviazione: **>**

```
:abbreviate @f fresh
```

La potete rimuovere con il seguente comando: **>**

```
:unabbreviate @f
```

Sino a quando non farete ciò @f verrà espanso in "fresh". Non preoccupatevene, Vim lo capisce comunque (eccetto se aveste un'abbreviazione per "fresh", ma essa fosse molto differente).
   Per eliminare tutte le abbreviazioni: **>**

```
:abclear
```

":unabbreviate" ed ":abclear" sono delle varianti per l'Insert mode ("iunabbeviate ed ":iabclear") e per il Command-line mode (":cunabbreviate" e "cabclear").

RIMAPPATURA DELLE ABBREVIAZIONI

C'è una cosa da tenere in considerazione quando definite un'abbreviazione: La stringa risultante non può essere mappata. Ad esempio: **>**

```
:abbreviate @a adder
:imap dd disk-door
```

Se adesso scriveste @a, otterreste "adisk-doorer". Non è quanto volevate. Per evitarlo, impiegate il comando ":noreabbrev". Fa come ":abbreviate", ma evita che la stringa risultante venga usata per la mappatura: **>**

```
:noreabbrev @a adder
```

Fortunatamente è raro che il risultato di un'abbreviazione venga mappato.

==========

24.8 Inserimento di caratteri speciali

Il comando CTRL-V serve ad inserire letteralmente il prossimo carattere. In altre parole, qualsiasi significato speciale il carattere abbia verrà ignorato. Ad esempio: >

```
CTRL-V <Esc>
```

Inserisce un carattere di escape. Così non dovrete lasciare l'Insert mode. (Non dovete mettere lo spazio dopo CTRL-V, è solo per renderlo più leggibile).

Note:
In MS-Windows CTRL-V viene usato per incollare del testo. Usate CTRL-Q invece di CTRL-V. Su Unix, d'altronde, CTRL-Q non funziona su alcuni terminali perchè ha un significato speciale.

Potete anche usare il comando CTRL-V {digits} per inserire un carattere contenente numeri decimali {digits}. Ad esempio, il carattere numero 127 è il carattere <Del> (ma non necessariamente il tasto <Del>!). Per inserire <Del> scrivete: >

```
CTRL-V 127
```

In questo modo potete inserire caratteri in numero superiore a 255. Quando scrivete meno di due cifre, un carattere non cifra dovrà ultimare il comando. Per evitare di scrivere il carattere non cifra anteponete uno o due zeri per fare tre cifre.
Tutti i seguenti comandi inseriscono un <Tab> seguito da un punto:

```
CTRL-V 9.
CTRL-V 09.
CTRL-V 009.
```

Per inserire un carattere in esadecimale, usate una "x" dopo il CTRL-V: >

```
CTRL-V x7f
```

Ciò va anche oltre i 255 caratteri (CTRL-V xff). Potete usare "o" per scrivere un carattere come numero ottale ed altri due metodi vi consentiranno di scrivere numeri a 16 o 32 bit (e.g., per un carattere Unicode): >

```
CTRL-V o123
CTRL-V u1234
CTRL-V U12345678
```

==
24.9 Digrafia

Taluni caratteri non esistono sulla tastiera. Ad esempio, il carattere di copyright (©). Per scrivere questi caratteri con Vim, utilizzerete la digrafia, ove due caratteri ne rappresentano uno. Per inserire un ©, ad esempio, premerete tre tasti: >

```
CTRL-K Co
```

Per sapere quali digrafie siano disponibili usate il seguente comando: >

```
:digraphs
```

Vim farà vedere la tabella digrafica. Eccone tre linee:

```
  AC ~_ 159   NS |  160   !I ¡  161   Ct ¢  162   Pd £  163   Cu €  164   Ye ¥  165 ~
  BB Š  166   SE §  167   ': š  168   Co ©  169   -a ª  170   << «  171   NO ¬  172 ~
  -- -  173   Rg ®  174   'm ¯  175   DG °  176   +- ±  177   2S ²  178   3S ³  179 ~
```

Ciò mostra, per esempio, che il carattere digrafico che otterrete con CTRL-K Pd è il carattere (£). Si tratta del carattere numero 163 (decimale).
Pd è l'abbreviazione di Pound. I più tanti digrafici sono fatti in modo da darvi un suggerimento circa il carattere che produrranno. Guardando la lista ne capirete la logica.
Si può cambiare il primo ed il secondo carattere, se non esiste un altro digrafico con la stessa combinazione. Così funzionerà anche CTRL-K dP. Se non vi fosse un digrafico per "dP" Vim cercherà anche un digrafico per "Pd".

Note:
I caratteri digrafici dipendono dal character set che Vim pensa usiate. Su MS-DOS è diverso che su MS-Windows. Impiegate spesso ":digraphs" per trovare quali caratteri digrafici siano attualmente disponibili.

Potete definire i vostri digrafici. Esempio: **>**

```
:digraph a" ä
```

Definisce che CTRL-K a" inserisca un carattere ä. Potete anche specificare il carattere con un numero decimale. Ciò definisce il medesimo digrafico: **>**

```
:digraph a" 228
```

Troverete ulteriori informazioni circa i caratteri digrafici in: |digraphs|
   Un altro modo per inserire caratteri speciali è con una keymap. Di più sull'argomento: |45.5|

==
24.10 Comandi in Normal mode

L'Insert mode offre un numero limitato di comandi. In Normal mode è disponibile assai di più. Se ne voleste usare uno, normalmente abbandonereste l'Insert mode con <Esc>, eseguireste il comando in Normal mode, e rientrereste nell'Insert mode con "i" od "a".
   C'è una via più rapida. Con CTRL-O {command} potete eseguire tutti i comandi Normal mode restando nell'Insert mode. Ad esempio, per cancellare dalla posizione del cursore alla fine della linea: **>**

```
CTRL-O D
```

Potete eseguire un solo comando Normal mode in questo modo. Ma potete specificare un registro od un conto. Un esempio più complicato: **>**

```
CTRL-O "g3dw
```

Cancella a partire dalla terza parola entro il registro g.

==

Capitolo seguente: |usr_25.txt| Lavorare con testo formattato

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

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

usr_25.txt   Per Vim version 6.2. Ultima modifica: 2003 Giu 21

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Cristian Rigamonti

Elaborare testo formattato

Difficilmente un testo è composto da una frase per linea. Questo capitolo spiega come interrompere le frasi per adattarle alla pagina ed altre formattazioni. Vim ha anche utili funzioni per elaborare paragrafi di una sola linea e tabelle.

  Capitolo seguente: |usr_26.txt| Ripetizione
Capitolo precedente: |usr_24.txt| Inserzione rapida
            Indice: |usr_toc.txt|

==========

25.1 Interrompere le linee

Vim ha una serie di funzioni che facilitano il trattamento del testo. Di default, l'editor non interrompe le linee automaticamente. In altre parole, dovete premere <Enter> voi stessi. Ciò è utile se state scrivendo programmi e volete essere voi a decidere dove finisce ogni linea. Non va altrettanto bene se state scrivendo della documentazione e volete che il testo occupi al massimo 70 caratteri per linea.
   Impostando l'opzione 'textwidth', Vim inserisce automaticamente le interruzioni di linea. Supponete, ad esempio, di volere una colonna molto stretta di soli 30 caratteri. Dovete eseguire il comando seguente: >

```
:set textwidth=30
```

Ora iniziate a scrivere (il righello è stato aggiunto qui per chiarezza)

```
          1         2         3
12345678901234567890123456789012345
Ho insegnato programmazione pe ~
```

Se ora scrivete la "r", la linea supererà il limite di 30 caratteri. Quando Vim se ne accorge, inserisce un'interruzione di linea e ottenete il seguente:

```
          1         2         3
12345678901234567890123456789012345
Ho insegnato programmazione ~
per un po' ~
```

Continuando, potete scrivere il resto del paragrafo:

```
          1         2         3
12345678901234567890123456789012345
Ho insegnato programmazione ~
per un po'. Una volta sono ~
stato fermato dalla polizia ~
perché davo dei compiti troppo ~
difficili. Storia vera. ~
```

Non dovete battere il ritorno a capo: Vim lo inserisce automaticamente.

Note:
Con l'opzione 'wrap', Vim interrompe le linee solo in fase di visualizzazione, non inserisce interruzioni di linea nel file.

RIFORMATTARE

Vim è un editor non un word processor. In un word processor, se cancellate qualcosa all'inizio di un paragrafo, le interruzioni di linea vengono rielaborate. In Vim non avviene, quindi se cancellate la parola "programmazione" dalla prima linea, vi ritrovate con una linea più corta:

```
          1         2         3
12345678901234567890123456789012345
```

```
        Ho insegnato ~
        per un po'. Una volta sono ~
        stato fermato dalla polizia ~
        perché davo dei compiti troppo ~
        difficili. Storia vera. ~
```

Non è un bel vedere: per ridare forma al paragrafo, usate l'operatore "gq".
   Usiamolo dapprima con una selezione visuale. Partendo dalla prima linea, scrivete: **>**

```
        v4jgq
```

"v" per entrare in Visual mode, "4j" per muovervi alla fine del paragrafo e infine l'operatore "gq". Il risultato è:

```
                 1         2         3
        12345678901234567890123456789012345
        Ho insegnato per un po'. Una ~
        volta sono stato fermato ~
        dalla polizia perché davo dei ~
        compiti troppo difficili. ~
        Storia vera. ~
```

Poiché "gq" è un operatore, potete usare uno dei seguenti tre modi per selezionare il testo su cui operare: con Visual mode, con un movimento e con un oggetto di testo.
   Nell'esempio precedente avremmo potuto usare "gq4j". Che significa scrivere meno, ma bisogna conoscere il numero delle linee. Un comando di spostamento ancora più utile è "}". Questo va alla fine del paragrafo. Così "gq}" formatta il testo dalla posizione del cursore fino alla fine del paragrafo attuale.
   Un oggetto di testo molto utile da usare con "gq" è il paragrafo. Provate: **>**

```
        gqap
```

"ap" sta per "a paragraph". Ciò formatta il testo di un solo paragrafo (separato da linee vuote). Così la parte dopo il cursore.
   Se i vostri paragrafi sono separati da linee vuote, potete formattare l'intero file scrivendo: **>**

```
        gggqG
```

"gg" per spostarvi alla prima linea, "gqG" per formattare fino all'ultima linea.
   Attenzione: se i paragrafi non sono opportunamente separati, verranno uniti. Un errore comune è quello di lasciare una linea con uno spazio o un Tab. Quella è una linea "bianca", ma non vuota.

Vim è in grado di formattare più che il solo testoi semplice. Si veda |fo-table| in proposito. Si veda anche l'opzione 'joinspaces' per cambiare il numero di spazi usati dopo un punto.
   È possibile usare un programma esterno per formattare. Ciò è utile se il vostro testo non può venire correttamente formattato con in comandi disponibili in Vim. Si veda l'opzione 'formatprg'.

==
25.2 Allineare il testo

Per centrare un intervallo di linee, usate il comando seguente: **>**

```
        :{range}center [width]
```

{range} è il solito intervallo da linea di comando. [width], è un'opzionale larghezza di linea da usare per centrare il testo. Se [width] non viene specificata, assume automaticamente il valore di 'textwidth' (se 'textwidth' fosse 0, il valore predefinito è 80).
   Per esempio: **>**

```
        :1,5center 40
```

il risultato sarà il seguente:

```
     Ho insegnato per un po'. Una ~
       volta sono stato fermato ~
    dalla polizia perché davo dei ~
       compiti troppo difficili. ~
             Storia vera. ~
```

ALLINEAMENTO A DESTRA

In modo simile il comando ":right" allinea il testo a destra: >

```
:1,5right 37
```

produrrà:

```
 Ho insegnato per un po'. Una ~
     volta sono stato fermato ~
dalla polizia perché davo dei ~
    compiti troppo difficili. ~
                 Storia vera. ~
```

ALLINEAMENTO A SINISTRA

Infine, c'è il comando: >

```
:{range}left [margin]
```

A differenza di ":center" e ":right", l'argomento di ":left" non è la lunghezza della linea. E' invece il margine sinistro. Se viene omesso, il testo verrà allineato al margine sinistro dello schermo (lo stesso risultato si ottiene indicando un margine zero). Se vale 5, il testo sarà indentato di 5 spazi. Provate ad esempio questi comandi: >

```
:1left 5
:2,5left
```

Il risultato è il seguente:

```
     Ho insegnato per un po'. ~
Una volta sono stato fermato ~
dalla polizia perché davo dei ~
compiti troppo difficili. ~
Storia vera. ~
```

GIUSTIFICARE IL TESTO

Vim non contiene comandi per giustificare il testo. Però c'è un bel pacchetto macro che fa proprio questo. Per usare questo pacchetto, eseguite il comando seguente: >

```
:runtime macros/justify.vim
```

Questo script di Vim definisce un nuovo comando visuale: "_j". Per giustificare un blocco di testo, evidenziatelo in Visual mode ed eseguite "_j".
Maggiori spiegazioni sono contenute nel file. Per andare là, fate "gf" su questo nome: $VIMRUNTIME/macros/justify.vim.

Un'alternativa è filtrare il testo attraverso un programma esterno. Ad esempio: >

```
:%!fmt
```

==
25.3 Indentazione e tabulazione

L'indentazione può venire usata per disallineare una parte del testo rispetto al resto. I testi di esempio in questo manuale, ad esempio, sono indentati di otto spazi od un tab. Potrete normalmente ottenerlo battendo un tab all'inizio di ciascuna linea. Prendete questo testo:

```
la prima linea ~
la seconda linea ~
```

È stato creato scrivendo un tab, del testo, <Enter>, un tab e altro testo.
L'opzione 'autoindent' attiva l'indentazione automatica: >

```
:set autoindent
```

Ogni nuova linea assume la stessa indentazione della precedente. Nell'esempio sopra, il tab dopo <Enter> non sarebbe stato necessario.

AUMENTARE L'INDENTAZIONE

Per aumentare la quantità di indentazione di una linea, usate l'operatore ">". Spesso questo viene usato come ">>", che aggiunge indentazione alla linea corrente.
   Il valore dell'indentazione aggiunta è specificato con l'opzione 'shiftwidth', il cui valore predefinito è 8. Per far sì che ">>" inserisca un'ampiezza di indentazione pari a quattro spazi, ad esempio, scrivete questo: **>**

```
:set shiftwidth=4
```

Se provate a usarlo sulla seconda linea del testo di esempio, ottenete:

```
la prima linea ~
    la seconda linea ~
```

"4>>", invece, aumenterà l'indentazione delle quattro linee successive.

LUNGHEZZA DELLA TABULAZIONE

Se volete ottenere indentazioni multiple di 4, basta impostare 'shiftwidth' a 4; tuttavia, premendo il Tab ottenete ancora un'indentazione di 8 caratteri. Per modificare questo comportamento, impostate l'opzione 'softtabstop': **>**

```
:set softtabstop=4
```

Ciò farà sì che il tasto <Tab> inserisca un'indentazione larga 4 caratteri. Se ci fossero già quattro spazi, verrà usato un carattere <Tab>, (risparmiando così 7 caratteri nel file). (Se invece volete sempre spazi e non caratteri tab, impostate l'opzione 'expandtab').

```
Note:
Potreste impostare l'opzione 'tabstop' a 4.  Tuttavia, se aprite il
file un'altra volta, con 'tabstop' impostato al valore predefinito di
8, il file sarà visualizzato in modo scorretto.  In altri
programmi, e nella stampa, l'indentazione risulterà sbagliata.  Per
questo  motivo è raccomandabile tenere 'tabstop' sempre ad 8.  Questo
è il valore standard ovunque.
```

MODIFICARE LA TABULAZIONE

Se elaborate un file che è stato scritto con una tabulazione 3, in Vim apparirà terribile, visto che Vim usa il valore standard di 8 per la tabulazione. Potreste risolvere impostando 'tabstop' a 3, ma dovreste farlo ogni volta che lavorate su questo file.
   Vim può cambiare l'uso delle tabulazioni nel vostro file. Per prima cosa impostate 'tabstop' in modo che l'indentazione risulti corretta, quindi usate il comando ":retab": **>**

```
:set tabstop=3
:retab 8
```

Il comando ":retab" imposterà 'tabstop' a 8, modificando al contempo il testo in modo che il suo aspetto rimanga inalterato: le sequenze di spazi bianchi saranno trasformate opportunamente in sequenze di tab e spazi per questo. Potete ora salvare il file. La prossima volta che lo aprirete, l'indentazione risulterà corretta senza dover impostare alcuna opzione.
   Attenzione: usando ":retab" su un programma, potreste modificare gli spazi in una costante di stringa. Per questo è buona norma usare "\t" invece del vero tab.

==
25.4 Trattare le linee lunghe

Qualche volta aprirete un file più largo del numero di colonne della finestra. Quando ciò avviene, Vim spezza le linee cosicchè tutto stia sullo schermo.
   Se disattivate l'opzione 'wrap', ogni linea del file verrà mostrata su una linea dello schermo. Allora la fine delle linee lunghe scomparirà sulla destra dello schermo.
   Quando spostate il cursore su un carattere che non può essere visto, Vim farà scorrere il testo fino a mostrarlo. Ciò è come spostare una finestra sul testo in senso orizzontale.
   Di default Vim non mostra una barra di scorrimento nella GUI. Se volete abilitarne una, usate il comando seguente: **>**

```
:set guioptions+=b
```

Una barra di scorrimento orizzontale apparirà in basso nella finestra di Vim.

Se non avete, o non volete usare, una barra di scorrimento, usate i seguenti comandi per far scorrere il testo. Il cursore resterà nello stesso posto, ma verrà spostato sul testo visibile se necessario.

```
        zh              scorre (il testo) a destra
        4zh             scorre a destra di quattro caratteri
        zH              scorre a destra di metà finestra
        ze              scorre a destra finché il cursore è a fine riga
        zl              scroll (il testo) a sinistra
        4zl             scorre a sinistra di quattro caratteri
        zL              scorre a sinistra di metà finestra
        zs              scorre a sinistra finché il cursore è a fine riga
```

Proviamo a mostrarlo con una linea di testo. Il cursore è sulla "e" di "del". La "finestra attuale" sopra la linea indica la parte di testo attualmente visibile. Le "finestre" sotto il testo indicano la parte di testo visibile dopo avere eseguito il comando scritto sulla sinistra.

```
                            |<--finestra attuale-->|
                un testo lungo, parte del quale è visibile nella finestra ~
        ze       |<--    finestra    -->|
        zH        |<--    finestra    -->|
        4zh             |<--    finestra    -->|
        zh                 |<--    finestra    -->|
        zl                   |<--    finestra    -->|
        4zl                     |<--    finestra    -->|
        zL                            |<--    finestra    -->|
        zs                          |<--    finestra    -->|
```

SPOSTARSI CON L'INTERRUZIONE DI LINEA DISATTIVATA

Quando 'wrap' è disattivata ed il testo è stato fatto scorrere orizzontalmente, potete usare i seguenti comandi per spostare il cursore su uno dei caratteri visibili, ignorando il testo al di fuori della finestra, a destra e a sinistra. Questi comandi non fanno mai scorrere il testo:

```
        g0              al primo carattere visibile della linea
        g^              al primo carattere "non bianco" visibile della linea
        gm              a metà della linea
        g$              all'ultimo carattere visibile della linea

                |<--    finestra   -->|
        un testo     lungo, parte del quale è visibile ~
                 g0  g^    gm        g$
```

INTERROMPERE LE LINEE SENZA SPEZZARE LE PAROLE *edit-no-break*

Quando preparate un testo che dovrà essere usato da un altro programma, potreste dover fare dei paragrafi senza linea di interruzione. Uno svantaggio di usare 'nowrap' è che non potreste vedere l'intera frase su cui state lavorando; d'altra parte, quando 'wrap' è attivo, le parole vengono spezzate a metà, rendendone difficile la lettura.

Una buona soluzione per elaborare questo tipo di paragrafi consiste nell'impostare l'opzione 'linebreak'. Vim allora interromperà le linee nel punto giusto mostrando la linea. Il testo nel file rimarrà inalterato.

Senza 'linebreak' un testo potrebbe apparire così:

```
        +----------------------------------+
        |programma per creare automaticame|
        |nte delle lettere. Volevano spedi|
        |re una lettera personalizzata ai |
        |loro 1000 clienti più ricchi. Sfo|
        |rtunatamente per il programmatore|
        +----------------------------------+
```

Dopo: **>**

```
        :set linebreak
```

apparirebbe così:

```
        +----------------------------------+
        |programma per creare              |
        |automaticamente delle lettere.    |
```

```
|Volevano spedire una lettera     |
|personalizzata ai loro 1000      |
|clienti più ricchi.              |
+---------------------------------+
```

Opzioni collegate:
'breakat' specifica i caratteri ove un'interruzione può venire inserita.
'showbreak' specifica una stringa da mostrare all'inizio di una linea interrotta.
Impostate 'textwidth' a zero per evitare che i paragrafi vengano interrotti.

SPOSTARSI SULLE LINEE VISIBILI

I comandi "j" e "k" spostano il cursore alla prossima od alla precedente linea del file. Quando usati su una linea lunga ciò significa spostare molte linee dello schermo alla volta.
Per spostare solo una linea dello schermo, usate i comandi "gj" e "gk". Quando una linea non viene interrotta essi fanno come "j" e "k". Quando la linea viene interrotta, spostano il carattere indicato una linea sopra o sotto.
Potrebbe piacervi usare le seguenti mappature, che assegnano questi comandi di movimento ai tasti cursore: **>**

```
:map <Up> gk
:map <Down> gj
```

TRASFORMARE UN PARAGRAFO IN UNA LINEA

Se volete importare del testo in un programma come MS-Word, ogni paragrafo deve essere formato da una sola linea. Se i vostri paragrafi sono attualmente separati da linee vuote, ecco come trasformare ogni paragrafo in una linea singola: **>**

```
:g/./,/^$/join
```

Sembra complicato. Dividiamolo in parti:

```
:g/./           Un comando ":global" che trova tutte le linee che
                contengono almeno un carattere.
     ,/^$/      Un intervallo che inizia dalla linea attuale (la linea
                non vuota) e termina con una linea vuota.
          join  Il comando ":join" unisce l'intervallo di linee
                formandone una sola.
```

Partendo da questo testo, che contiene otto linee interrotte alla colonna 30:

```
+---------------------------------+
|Un programma per creare          |
|automaticamente delle lettere.   |
|Volevano spedire una lettera     |
|personalizzata.                  |
|                                 |
|Ai loro 1000 clienti più         |
|ricchi. Sfortunatamente per il   |
|programmatore,                   |
+---------------------------------+
```

Otterreste queste due linee:

```
+---------------------------------+
|Un programma per creare automatica|
|mente delle lettere. Volevano sped|
|ire una lettera personalizzata.   |
|Ai loro 1000 clienti più ricchi. S|
|fortunatamente per il programmator|
+---------------------------------+
```

Nota: il tutto non funziona se la linea che separa i paragrafi è "bianca" ma non vuota, ossia se contiene spazi e/o tab. Questo comando funziona con le linee "bianche": **>**

```
:g/\S/,/^\s*$/join
```

Ciò richiede almeno una linea "bianca" o vuota alla fine del file perchè venga unito anche l'ultimo paragrafo.

==
25.5 Elaborare tabelle

Supponete di lavorare su una tabella con quattro colonne:

```
tabella          test 1        test 2      test 3 ~
input A          0.534 ~
input B          0.913 ~
```

Dovete inserire dei numeri nella terza colonna. Potreste spostarvi sulla seconda linea, usare "A", inserire un sacco di spazi e scrivere il testo.
   Per questo tipo di elaborazione esiste una funzione speciale: **>**

```
set virtualedit=all
```

Ora potete muovere il cursore in posizioni dove non vi sia alcun testo. Questo si chiama "spazio virtuale". Elaborare tabelle risulta molto più facile in questo modo.
   Muovete il cursore ricercando l'intestazione dell'ultima colonna: **>**

```
/test 3
```

Ora premete "j" e vi ritrovate esattamente dove dovete inserire il valore per "input A". Scrivendo "0.693" ottenete:

```
tabella          test 1     test 2      test 3 ~
input A          0.534                  0.693 ~
input B          0.913 ~
```

Vim ha riempito automaticamente lo spazio che precede il nuovo testo. Ora, per riempire il campo successivo in questa colonna usate "Bj". "B" vi sposta all'indietro, all'inizio di una parola separata da spazio bianco. Poi "j" vi sposta nella posizione dove può essere inserito il valore per il prossimo campo.

Note:
Potete spostare il cursore ovunque sullo schermo, anche oltre la fine di una linea, ma Vim non inserirà spazi là, finché non inserirete un carattere in quella posizione.

COPIARE UNA COLONNA

Volete aggiungere una colonna, che deve essere la copia della terza colonna e posta prima della colonna "test 1". Potete farlo in sette passi:
1. Spostate il cursore sull'angolo superiore sinistro di questa colonna, ad es. con "/test 3".
2. Premete CTRL-V per entrare in Visual mode a blocchi.
3. Spostate il cursore di due linee verso il basso con "2j". Siete ora in "spazio virtuale": la linea "input B" della colonna "test 3".
4. Spostate il cursore a destra, per includere l'intera colonna nella selezione, più lo spazio che volete tra le colonne. "9l" dovrebbe farlo.
5. Copiate il rettangolo selezionato con "y".
6. Spostate il cursore su "test 1", dove va inserita la nuova colonna.
7. Premete "P".

Il risultato dovrebbe essere:

```
tabella          test 3    test 1     test 2      test 3 ~
input A          0.693     0.534                  0.693 ~
input B                    0.913 ~
```

Notate che l'intera colonna "test 1" è stata spostata a destra, compresa la linea in cui la colonna "test 3" non contiene testo.

Tornate ai movimenti del cursore non virtuali con: **>**

```
:set virtualedit=
```

MODALITÀ DI SOSTITUZIONE VIRTUALE

Lo svantaggio di usare 'virtualedit' è che viene avvertito diverso. Non potete riconoscere tab o spazi oltre la fine delle linee quando spostate il cursore. Può essere usato un altro metodo: il Virtual Replace mode.
   Supponete di avere una linea in una tabella che contenga sia tab che altri caratteri. Usate "rx" sul primo tab:

```
        inp     0.693   0.534   0.693 ~

                |
           rx   |
                V

        inpx0.693   0.534       0.693 ~
```

L'allineamento viene perduto. Per evitare ciò, usate il comando "gr":

```
        inp     0.693   0.534   0.693 ~

                |
          grx   |
                V

        inpx    0.693   0.534   0.693 ~
```

Ciò che avviene è che il comando "gr" si assicura che il nuovo carattere prenda la giusta quantità di spazio dello schermo. Vengono inseriti spazi o tab in più per riempire lo spazio vuoto. Così, ciò che accade ora è che un tab viene sostituito da "x" e che vengono inseriti spazi bianchi per fare sì che il testo dopo di esso mantenga la sua posizione. In questo caso viene inserito un tab.

Se dovete sostituire più di un carattere e usate il comando "R" per entrare in modalità Sostituzione (si veda |04.9|), rovinerete l'allineamento e sostituirete i caratteri sbagliati:

```
        inp     0       0.534   0.693 ~

                |
         R0.786 |
                V

        inp     0.78634 0.693 ~
```

Il comando "gR" invece usa il Virtual Replace mode. Ciò preserva l'allineamento:

```
        inp     0       0.534   0.693 ~

                |
        gR0.786 |
                V

        inp     0.786   0.534   0.693 ~
```

==

Capitolo seguente: |usr_26.txt| Ripetizione

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

usr_26.txt   Per Vim version 6.2. Ultima modifica: 2002 Ott 29

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

Ripetizione

Un lavoro di editing è sempre crudamente privo di struttura. Una modifica spesso necessita di venire eseguita molte volte. In questo capitolo saranno spiegati diversi modi utili per ripetere un cambiamento.

  Capitolo seguente: |usr_27.txt| Comandi di ricerca e modelli
Capitolo precedente: |usr_25.txt| Lavorare con testo formattato
            Indice: |usr_toc.txt|

==
26.1 Ripetizioni in Visual mode

Il Visual mode è molto pratico per effettuare correzioni su qualsiasi numero di righe. Potete vedere il testo evidenziato, così potete verificare se siano state modificate le linee giuste. Ma effettuare la selezione richiede la pressione di alcuni tasti. Il comando "gv" seleziona nuovamente la stessa area di testo. Questo vi permette di fare un'altra operazione sullo stesso testo.
   Supponiamo che abbiate alcune righe in cui vogliate sostituire "2001" con "2002" e "2000" con "2001":

```
The financial results for 2001 are better ~
than for 2000.  The income increased by 50%, ~
even though 2001 had more rain than 2000. ~
                  2000           2001 ~
income          45,403         66,234 ~
```

Per prima cosa sostituite "2001" con "2002". Selezionate le righe in Visual mode, e usate: **>**

```
:s/2001/2002/g
```

Ora usate "gv" per riselezionare lo stesso testo. Non importa dove si trovi il cursore. Usate ":s/2000/2001/g" per effettuare la seconda modifica.
   Ovviamente, potete ripetere queste sostituzioni diverse volte.

==
26.2 Aggiungere e sottrarre

Quando ripetete il cambiamento di un numero in un altro, spesso avete uno scostamento fisso. Nell'esempio sopra, è stato aggiunto uno ad ogni anno. Invece di scrivere un comando diverso per ogni anno, si può usare il comando CTRL-A.
   Usando lo stesso testo di prima, cercate un anno: **>**

```
/19[0-9][0-9]\|20[0-9][0-9]
```

Ora premete CTRL-A. L'anno sarà incrementato di uno:

```
The financial results for 2002 are better ~
than for 2000.  The income increased by 50%, ~
even though 2001 had more rain than 2000. ~
                  2000           2001 ~
income          45,403         66,234 ~
```

Usate "n" per trovare l'anno successivo, e premente "." per ripetere il CTRL-A ("." è un po' più veloce da scrivere). Ripetete "n" e "." per tutti gli anni che ci sono.
   Suggerimento: impostate l'opzione 'hlsearch' per vedere i valori che state per modificare, poi potrete guardare oltre e farlo più velocemente.

L'aggiunta di numeri maggiori di uno può essere fatta premettendo un numero a CTRL-A. Supponete di avere la seguente lista:

```
        1.  item four ~
        2.  item five ~
        3.  item six ~
```

Spostate il cursore su "1." e scrivete: **>**

```
        3 CTRL-A
```

L'"1." cambierà in "4,". Ancora, potete usare "." per ripetere l'operazione sugli altri numeri.
numeri.

Altro esempio:

```
        006     foo bar ~
        007     foo bar ~
```

Usando CTRL-A su questi numeri risulterà:

```
        007     foo bar ~
        010     foo bar ~
```

7 più uno fa 10? Questo è accaduto perchè Vim ha riconosciuto "007" come un numero in rappresentazione ottale, perchè il primo numero è uno zero. Questa notazione è spesso usata nei programmi C. Se non volete che un numero che inizia per zero sia gestito come un numero ottale, usate questo: **>**

```
        :set nrformats-=octal
```

Il comando CTRL-X esegue le sottrazioni nello stesso modo.

==

26.3 Fare una modifica in più file

Supponiamo che abbiate una variabile di nome "x_cnt" e che vogliate cambiarla in "x_counter". Questa variabile viene usata in parecchi dei vostri file C. Dovrete sostituirla in tutti i file. Ecco come fare.
Mettete tutti i file interessati nella lista degli argomenti: **>**

```
        :args *.c
```

<
Questo comando trova tutti i file C e modifica il primo. Ora potete eseguire un comando di sostituzione su tutti gli altri file: **>**

```
        :argdo %s/\<x_cnt\>/x_counter/ge | update
```

Il comando ":argdo" prende come argomento un altro comando. Questo comando verrà eseguito su tutti i file nella list degli argomenti.
Il comando substitute "%s" che segue lavora su tutte le righe. Esso trova la parola "x_cnt" come "\<x_cnt\>". Le "\<" e "\>" vengono usate per trovare solo la parola esatta e non "px_cnt" o "x_cnt2".
Le flag per il il comando includono "g" per sosotituire tutte le "x_cnt" che si presentano nella stessa riga. La flag "e" è usata per evitare un messaggio di errore nel caso in cui "x_cnt" non appaia affatto nel file. Diversamente ":argdo" abortirebbe la sua esecuzione se non venisse trovata "x_cnt" nel primo file.
Le "|" separano i due comandi. Il comando "update" che segue salva il file solo se è cambiato il suo contenuto. Se nessuna "x_cnt" viene cambiata in "x_counter", non succede niente.

C'è anche il comando ":windo", che esegue il suo argomento in tutte le finestre. E ":bufdo" che esegue il suo argomento in tutti i buffer. Fate attenzione con quest'ultimo perchè potreste avere più file di quanti pensiate nella lista dei buffer. Fate un controllo con il comando ":buffers" (o ":ls").

==

26.4 Usare Vim da uno shell script

Supponiamo che abbiate molti file in cui avete bisogno di cambiare la stringa "-person-" con "Jones" e poi stamparla. Come lo fate? Un modo è scrivere molto. L'altro modo è scrivere uno shell script che faccia il lavoro.
L'editor Vim fa un lavoro superbo come editor screen-oriented quando si usano i comandi del Normal mode. Nei processi di batch, tuttavia, i comandi del Normal mode non risultano chiari; così qui userete invece l'Ex mode. Questa modalità fornisce una gradevole interfaccia a linea di comando che facilita l'immissione dei comandi in un file batch. ("Ex command" è solo un altro nome per un comando (:) da linea di comando).

I comandi Ex mode di cui avete bisogno sono i seguenti: >

```
%s/-person-/Jones/g
write tempfile
quit
```

Inserite questi comandi nel file "change.vim". Ora per lanciare l'editor in batch mode, usate questo shell script: >

```
for file in *.txt; do
  vim -e -s $file < change.vim
  lpr -r tempfile
done
```

Il ciclo for-done è un costrutto della shell per ripetere le due righe nel mezzo, sino a che la variabile $file risulta ogni volta impostata con un nome di file diverso.
La seconda riga lancia l'editor Vim in modalità Ex (argomento -e) sul file $file e legge i comandi dal file "change.vim". L'argomento -s dice a Vim di operare in silent mode. In altre parole, non visualizza il :prompt, nè alcun altro prompt.
Il comando "lpr -r tempfile" stampa il risultante "tempfile" e lo cancella (ciò che fa l'argomento -r).

LEGGERE DA STDIN

Vim può leggere testo dallo standard input. Dato che normalmente qui vegono letti i comandi, voi dovete dire a Vim di leggere invece del testo. Ciò è possibile passando l'argomento "-" al posto di un file. Esempio: >

```
ls | vim -
```

Ciò vi permette di modificare l'output del comando "ls", senza prima salvare il testo in un file.
Se usate lo standard input per leggere del testo, potete usare l'argomento "-S" per leggere uno script: >

```
producer | vim -S change.vim -
```

SCRIPT E MODALITA' NORMALE

Se veramente volete usare i comandi del Normal mode in uno script, potete farlo così: >

```
vim -s script file.txt ...
```
<

Note:
"-s" ha un significato diverso quando viene usato senza "-e". Qui significa che utilizza il contenuto di "script" come un insieme di comandi del Normal mode.
Quando è usato con "-e" significa che esegue in silenzio e non usa l'argomento seguente come nome di un file.

I comandi nello script vengono eseguiti nel modo in cui li avete scritti. Non dimenticate che una riga vuota viene interpretata come premere <Enter>. In Normal mode esso sposta il cursore sulla riga successiva.
Per realizzare lo script potete creare il file script e scrivervi i comandi. Avete bisogno di immaginare quale possa essere il risultato, il che può essere un po' difficile. Un altro modo consiste nel registrare i comandi mentre li eseguite manualmente. Questo è quanto dovete fare: >

```
vim -w script file.txt ...
```

Tutti i tasti che verranno premuti saranno scritti nello "script". Se fate un piccolo sbaglio potete pure continuare e ricordarvi di modificare lo script in seguito.
L'argomento "-w" aggiunge tutto ad uno script esistente. Va bene quando volete registrare lo script un pò alla volta. Se voleste iniziare dal niente e scrivere tutto dall'inizio, usate l'argomento "-W". Sovrascriverà ogni file esistente.

==

Capitolo seguente: |usr_27.txt| Comandi di ricerca e modelli

usr_27.txt   Per Vim version 6.2. Ultima modifica: 2003 Ott 28

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Stefano Palmeri

Comandi di ricerca e modelli

Nel capitolo 3 sono stati menzionati pochi semplici modelli di ricerca |03.9|. Vim può eseguire delle ricerche molto più complesse. Questo capitolo spiega quelle usate più spesso. Una dettagliata spiegazione può essere trovata qui: |pattern|

  Capitolo seguente: |usr_28.txt| La piegatura
Capitolo precedente: |usr_26.txt| Ripetizione
            Indice: |usr_toc.txt|

==
27.1 Ignorare le differenze tra i caratteri maiuscoli e minuscoli

Di default, le ricerche di Vim rispettano le differenze tra caratteri maiuscoli e minuscoli. Quindi, "include", "INCLUDE" e "Include" sono tre parole differenti ed una ricerca ne troverà solo una.
   Adesso abilitate l'opzione 'ignorecase': >

```
:set ignorecase
```

Cercate di nuovo "include" e ora corrisponderà a "Include", "INCLUDE" e "InClUDe". (Impostate l'opzione 'hlsearch' per vedere velocemente dove un modello corrisponda.)
   Potete disabilitarla di nuovo con : >

```
:set noignorecase
```

Lasciamola però impostata e cerchiamo "INCLUDE". Troverà esattamente lo stesso testo che "include" aveva trovato . Adesso impostiamo l'opzione 'smartcase' : >

```
:set ignorecase smartcase
```

Se avete un modello con almeno un carattere maiuscolo, la ricerca rispetterà le differenze tra i caratteri maiuscoli e minuscoli. L'idea è che voi non eravate obbligati a scrivere quel carattere maiuscolo, quindi dovete averlo fatto perché volevate cercare una maiuscola. Questo è intelligente!
   Con queste due opzioni impostate troverete le seguenti corrispondenze:

modello	corrispondenze ~
word	word, Word, WORD, WoRd, etc.
Word	Word
WORD	WORD
WoRd	WoRd

CARATTERI MAIUSCOLI E MINUSCOLI IN UN SOLO MODELLO

Se volete ignorare le differenze tra i caratteri maiuscoli e minuscoli per uno specifico modello, potete farlo anteponendo la stringa "\c". Usare "\C" fa sì che il modello rispetti le differenze. Tutto ciò sovrascrive le opzioni 'ignorecase' e 'smartcase'; quando "\c" o "\C" vengono usate i loro valori non contano.

modello	corrispondenze ~
\Cword	word
\CWord	Word
\cword	word, Word, WORD, WoRd, etc.
\cWord	word, Word, WORD, WoRd, etc.

Un grande vantaggio nell'usare "\c" e "\C" è che esse rimangono col modello.

Così se voi ripetete un modello dalla cronologia delle ricerche, succederà la stessa cosa, senza che abbia importanza se 'ignorecase' o 'smartcase' siano state cambiate.

Note:
L'uso di argomenti "\" nei modelli di ricerca dipende dall'opzione 'magic'. In questo capitolo noi assumiamo che 'magic' sia attiva, poichè questa è l'impostazione standard e raccomandata. Se voleste cambiare 'magic', molti modelli di ricerca all'improvviso diventerebbero non validi.

Note:
Se la vostra ricerca richiede più tempo di quanto vi aspettaste, potete interromperla con CTRL-C in Unix e CTRL-Break in MS-DOS e MS-Windows.

==========

27.2 Aggirare (nella ricerca) la fine del file

Di default, una ricerca in avanti inizia a cercare la stringa data dalla posizione corrente del cursore. Essa poi continua fino alla fine del file. Se fino ad allora non ha trovato la stringa, comincia da principio e cerca dall'inizio del file fino alla posizione del cursore.
Tenete a mente che ripetendo il comando "n" per cercare la corrispondenza successiva, potreste eventualmente tornare alla prima corrispondenza. Se non vi accorgete di questo, cercherete all'infinito! Per darvi un consiglio, Vim mostra questo messaggio:

```
search hit BOTTOM, continuing at TOP ~
```

Se usate il comando "?", per cercare nell'altra direzione, ricevete questo messaggio:

```
search hit TOP, continuing at BOTTOM ~
```

Tuttavia, voi non sapete quando siete tornati alla prima corrispondenza. Un modo per saperlo è quello di attivare l'opzione 'ruler': >

```
:set ruler
```

Vim mostrerà la posizione del cursore nell'angolo in basso a destra della finestra (nella linea di stato, se ce n'è una). Appare così:

```
101,29        84% ~
```

Il primo numero è il numero di linea del cursore. Ricordate il numero di linea dal quale siete partiti, affinché possiate controllare se avete passato di nuovo questa posizione.

NON AGGIRARE LA FINE DEL FILE

Per disabilitare l'aggiramento della fine del file in una ricerca , usate il seguente comando: >

```
:set nowrapscan
```

Adesso quando la ricerca raggiunge la fine del file, un messaggio d'errore dice:

```
E385: search hit BOTTOM without match for: forever ~
```

Così potete trovare tutte le corrispondenze andando all'inizio del file con "gg" e continuando a cercare fino a che non vedete questo messaggio.
Se cercate nell'altra direzione, usando "?", voi ricevete:

```
E384: search hit TOP without match for: forever ~
```

==========

27.3 Scostamento

Di default, il comando di ricerca lascia il cursore posizionato all'inizio del modello. Potete dire a Vim di lasciarlo in qualche altro posto specificando uno scostamento. Per il comando di ricerca in avanti "/", lo scostamento è specificato aggiungendo uno slash (/) e lo scostamento: >

```
/comportamento/2
```

Questo comando cerca il modello "comportamento" e poi sposta il cursore all'inizio della seconda linea dopo il modello. Usando questo comando per il paragrafo sopra, Vim trova la parola "comportamento" nella prima linea. Poi il cursore è spostato due linee più sotto e si posa su "uno scostamento".

Se lo scostamento è un semplice numero, il cursore sarà posizionato all'inizio della linea che dista quel numero di linee dalla corrispondenza. Lo scostamento numerico può essere positivo o negativo. Se è positivo, il cursore si sposta in basso di quel numero di linee; se è negativo si sposta verso l'alto.

SCOSTAMENTO DEI CARATTERI

Lo scostamento "e" indica uno scostamento dalla fine della corrispondenza. Esso sposta il cursore sull'ultimo carattere della corrispondenza. Il comando: **>**

```
/const/e
```

mette il cursore sulla "t" di "const".
   Da quella posizione, aggiungendo un numero si sposta in avanti di quel numero di caratteri. Questo comando muove il cursore sul carattere proprio dopo la corrispondenza: **>**

```
/const/e+1
```

Un numero positivo sposta il cursore verso destra, uno negativo lo sposta verso sinistra. Ad esempio: **>**

```
/const/e-1
```

posiziona il cursore sulla "s" di "const".

Se lo scostamento comincia con "b", il cursore si sposta all'inizio del modello. Questo non è molto utile, dal momento che lasciar fuori la "b" fa la stessa cosa. Diventa utile quando si aggiunge o si sottrae un numero. Il cursore dopo va avanti o indietro di quel numero di caratteri. Per esempio: **>**

```
/const/b+2
```

Posiziona il cursore all'inizio della corrispondenza e poi due caratteri verso destra. Così si posa sulla "n".

RIPETIZIONE

Per ripetere una ricerca del precedente modello, ma con un diverso scostamento, lasciate fuori il modello: **>**

```
/that
//e
```

E' uguale a: **>**

```
/that/e
```

Per ripetere con lo stesso scostamento: **>**

```
/
```

"n" fa la stessa cosa. Per ripetere mentre si rimuove un offset usato in precedenza: **>**

```
//
```

CERCARE ALL'INDIETRO

Il comando "?" usa gli offset nello stesso modo, ma dovete usare "?" per separare l'offset dal modello, anziché "/": **>**

```
?const?e-2
```

La "b" e la "e" mantengono il loro significato; essi non cambiano direzione se si usa "?".

POSIZIONE DI PARTENZA

Quando si comincia una ricerca, essa normalmente inizia dalla posizione del cursore. Quando voi specificate uno scostamento di linea, ciò può causare dei problemi. Ad esempio: **>**

```
/const/-2
```

Questo trova la successiva parola "const" e quindi sposta il cursore due linee verso l'alto. Se usate "n" per cercare di nuovo, Vim potrebbe iniziare dalla posizione corrente e trovare la stessa corrispondenza "const". Allora, usando di nuovo lo scostamento, potreste tornare da dove eravate partiti. Dovreste esservi bloccati!
Potrebbe essere peggio: supponete che ci sia un'altra corrispondenza con "const" nella linea successiva. In questo caso ripetendo la ricerca in avanti trovereste questa corrispondenza e spostereste il cursore due linee insù.
Così in realtà spostereste il cursore indietro!

Quando voi specificate uno scostamento di carattere, Vim terrà conto di ciò. In questo modo la ricerca inizia pochi caratteri avanti o indietro, cosicché la stessa corrispondenza non verrà trovata di nuovo.

==========

27.4 Effettuare più volte la ricerca

L'argomento "*" specifica che l'argomento che lo precede può corrispondere un qualsiasi numero di volte.
Così: **>**

```
/a*
```

corrisponde ad "a", "aa", "aaa", etc. Ma anche a "" (la stringa vuota), poichè le zero volte sono incluse.
Il segno "*" influisce solo sull'argomento direttamente prima di esso. Quindi "ab*" corrisponde ad "a", "ab", "abb", "abbb", etc. Per far sì che di un'intera stringa si trovino corrispondenze multiple, essa deve essere raggruppata in un unico argomento. Questo si fa mettendo "\(" prima della stringa e "\)" dopo di essa. Così questo comando: **>**

```
/\(ab\)*
```

corrisponde a: "ab", "abab", "ababab", etc. Corrisponde anche a "".

Per evitare di trovare le corrispondenze con le stringhe vuote, usate "\+". Questo fa sì che l'argomento precedente abbia una o più corrispondenze: **>**

```
/ab\+
```

Corrisponde ad "ab", "abb", "abbb", etc. Non corrisponde ad "a" quando questa non è seguita da una "b".

Per trovare le corrispondenze con un argomento con diverse opzioni, usate "\=". Esempio: **>**

```
/folders\=
```

corrisponde a "folder" e "folders".

CONTEGGI SPECIFICI

Per trovare uno specifico numero di corrispondenze degli argomenti usate il formato "\{n,m}". "n" e "m" sono numeri. L'argomento che lo precede corrisponderà da "n" a "m" volte (incluse |inclusive|). Esempio: **>**

```
/ab\{3,5}
```

corrisponde ad "abbb", "abbbb" and "abbbbb".
Quando "n" è omesso, esso assume il valore di zero. Quando "m" è omesso, esso assume il valore di infinito. Quando ",m" è omesso, esso troverà corrispondenze esattamente "n" volte.
Esempi:

modello	conteggio corrispondenze ~
\{,4}	0, 1, 2, 3 o 4
\{3,}	3, 4, 5, etc.
\{0,1}	0 o 1, lo stesso che \=

```
\{0,}           0 o più, lo stesso che *
\{1,}           1 o più, lo stesso che \+
\{3}            3
```

CERCARE IL MENO POSSIBILE

Fin qui gli argomenti cercano corrispondenze nel maggior numero possibile di caratteri che possono trovare. Per cercare un numero minore di corrispondenze, usate "\{-n,m}". Funziona allo stesso modo di "\{n,m}", eccetto per il fatto che ne viene usata la minore quantità possibile.
Per esempio, usate: **>**

```
/ab\{-1,3}
```

Troverà "ab" in "abbb". In realtà, esso non troverà mai più di una "b", perché non c'è ragione di cercare oltre. Richiede qualcos'altro per forzare la ricerca di corrispondenze oltre il limite più basso.
Le stesse regole si applicano rimuovendo "n" e "m". E' anche possibile rimuoverle entrambe, risultando in "\{-}". Questo trova l'argomento prima di esso zero o più volte, il meno possibile. L'argomento in sé stesso corrisponde sempre zero volte. E' utile quando è combinato con qualcos'altro. Esempio: **>**

```
/a.\{-}b
```

Questo trova le corrispondenze "axb" in "axbxb". Se fosse usato questo modello: **>**

```
/a.*b
```

Esso cercherebbe di trovare corrispondenza con il maggior numero possibile di caratteri con ".*", così troverebbe "axbxb" come un tutt'uno.

==========

27.5 Alternative

L'operatore "or" in un modello è "\|". Esempio: **>**

```
/foo\|bar
```

Questo trova "foo" o "bar". Più alternative possono venire concatenate: **>**

```
/one\|two\|three
```

Cerca "one", "two" e "three".
Per ricercare più volte, tutto deve essere posto tra "\(" e "\)": **>**

```
/\(foo\|bar\)\+
```

Questo trova "foo", "foobar", "foofoo", "barfoobar", etc.
Un altro esempio: **>**

```
/end\(if\|while\|for\)
```

Questo trova "endif", "endwhile" e "endfor".

Un argomento correlato è "\&". Questo richiede che entrambe le alternative corrispondano nello stesso posto. La ricerca risultante usa l'ultima alternativa. Esempio: **>**

```
/forever\&...
```

Questo trova "for" in "forever". Non troverà la corrispondenza in "fortuin", per esempio.

==========

27.6 Intervalli di caratteri

Per cercare "a", "b" o "c" potete usare "/a\|b\|c". Quando voleste cercare tutte le corrispondenze di tutte le lettere dalla "a" alla "z", ciò richiederebbe molto tempo. C'è un metodo più breve: **>**

```
/[a-z]
```

Il costrutto [] trova un singolo carattere. All'interno specificate quali caratteri cercare. Potete includere una lista di caratteri, come questa: **>**

```
        /[0123456789abcdef]
```

Questo troverà le corrispondenze di ogni carattere incluso. Per i caratteri consecutivi potete specificare l'intervallo. "0-3" è uguale a "0123". "w-z" sta per "wxyz". Così lo stesso comando appena visto sopra può essere abbreviato in: **>**

```
        /[0-9a-f]
```

Per trovare proprio il carattere "-" fate in modo che sia il primo o l'ultimo dell'intervallo.
Questi caratteri speciali sono accettati per rendere più facile usarli dentro un intervallo [] (essi in realtà possono essere usati dovunque nel modello di ricerca):

```
        \e      <Esc>
        \t      <Tab>
        \r      <CR>
        \b      <BS>
```

Ci sono alcune situazioni speciali riguardo agli intervalli []; vedete |/[]| per l'intera storia.

INTERVALLI COMPLEMENTARI

Per escludere la ricerca di uno specifico carattere, usate "^" all'inizio di un intervallo. L'argomento di [] quindi cerca tutte le corrispondenze tranne i caratteri inclusi. Esempio: **>**

```
        /"[^"]*"
<
         "        le virgolette
          [^"]    qualsiasi carattere che non siano virgolette
              *   il maggior numero possibile
               "  di nuovo le virgolette
```

Questo corrisponde a "foo" e "3!x", virgolette incluse.

INTERVALLI PREDEFINITI

Un certo numero di intervalli sono usati molto spesso. Vim offre una scorciatoia per questi. Per esempio: **>**

```
        /\a
```

Cerca caratteri alfabetici. Questo equivale a usare "/[a-zA-Z]". Qui di seguito ci sono alcune di queste scorciatoie:

argomento	corrispondenze	equivale a ~
\d	numeri	[0-9]
\D	non-numeri	[^0-9]
\x	numeri esadecimali	[0-9a-fA-F]
\X	numeri non-esadecimali	[^0-9a-fA-F]
\s	spazio bianco	[ ] (<Tab> and <Space>)
\S	caratteri non-bianchi	[^] (not <Tab> and <Space>)
\l	caratteri alfab.	[a-z]
\L	non-lowercase alpha	[^a-z]
\u	uppercase alpha	[A-Z]
\U	non-uppercase alpha	[^A-Z]

Note:
Usare questi intervalli predefiniti funziona molto più velocemente che non gli intervalli di caratteri equivalenti.
Questi argomenti non possono essere usati all'interno di [].
Quindi "[\d\l]" NON trova le corrispondenze di un numero o di lettera minuscola. Usate invece "\(\d\|\l\)".

Vedete |/\s| per avere l'intera lista di questi intervalli.

==========

27.7 Classi di caratteri

L'intervallo di caratteri trova le corrispondenze con una serie prefissata di caratteri. Una classe di caratteri è simile, ma con una differenza essenziale: la serie di caratteri può essere ridefinita senza cambiare il modello di ricerca.

Ad esempio, cercate questo modello: >

```
/\f\+
```

L'argomento "\f" sta per caratteri di nomi di file. Quindi corrisponde a sequenze di caratteri che possono essere il nome di un file.
Quali caratteri possono far parte del nome di un file dipende dal sistema operativo che voi state usando. In MS-Windows, la backslash è inclusa, in Unix non lo è. Questo è specificato con l'opzione 'isfname'. Il valore di default per Unix è: >

```
:set isfname
isfname=@,48-57,/,.,-,_,+,,,#,$,%,~,=
```

Per altri sistemi il valore di default è diverso. Quindi potete creare un modello di ricerca con "\f" per trovare il nome di un file ed esso automaticamente si adatterà al sistema nel quale lo state usando.

Note:
In realtà, Unix permette di usare qualsiasi carattere nel nome di un file, spazi bianchi inclusi. Includere questi caratteri in 'isfname' sarebbe teoricamente corretto, ma renderebbe impossibile trovare la fine del nome del file nel testo. Quindi il valore predefinito di 'isfname' è un compromesso.

Le classi di caratteri sono:

argomento	corrispondenze	opzioni ~
\i	caratteri di identificativi	'isident'
\I	come \i, escludendo i numeri	
\k	caratteri di parole chiave	'iskeyword'
\K	come \k, escludendo i numeri	
\p	caratteri stampabili	'isprint'
\P	come \p, escludendo i numeri	
\f	caratteri di nomi di file	'isfname'
\F	come \f, escludendo i numeri	

==========

27.8 Ricerca di interruzioni di linea

Vim può trovare un modello che includa una interruzione di linea. Dovete specificare dov'è l'interruzione, poichè tutti gli argomenti menzionati fin qui non cercano le interruzioni di linea.
Per verificare una interruzione di linea in una posizione specifica, usate l'argomento "\n" : >

```
/the\nword
```

Questo corrisponderà a una linea che finisca con "the" seguita da una linea che inizi con "word". Per trovare "the word" così com'è, dovete trovare una interruzione o uno spazio bianco.
L'argomento da usare per far questo è "_s": >

```
/the\_sword
```

Per permettere qualsiasi numero di spazi bianchi: >

```
/the\_s\+word
```

Questo trova le corrispondenze anche quando "the " è alla fine di una linea e "   word" si trova all'inizio della successiva.

"\s" trova lo spazio bianco, "_s" trova lo spazio bianco o una interruzione di linea. Similarmente, "\a" corrisponde a un carattere alfabetico e "_a" corrisponde ad un carattere alfabetico o a una interruzione di linea. Le altre classi o intervalli di caratteri possono essere modificati nello stesso modo inserendo un "_".

Molti altri argomenti possono essere creati per trovare una interruzione di linea anteponendo "_". Ad esempio: "_." corrisponde ad ogni carattere od interruzione di linea.

Note:
"_.*" trova qualsiasi cosa fino alla fine del file. State attenti con questo, poichè può rendere un comando di ricerca molto lento.

Un altro esempio è "_[]", un intervallo di caratteri che includa una interruzione di linea: >

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

Questo trova un testo tra virgolette che può essere distribuito in diverse linee.

==
27.9 Esempi

Ecco alcuni modelli di ricerca che voi potreste trovare utili. Mostrano come possono essere combinati gli argomenti menzionati sopra.

TROVARE UNA TARGA AUTOMOBILISTICA DELLA CALIFORNIA

Un semplice numero di targa è "1MGU103". Esso ha un numero, tre lettere maiuscole e tre numeri. Mettiamo questo direttamente in un modello di ricerca: **>**

```
        /\d\u\u\u\d\d\d
```

Un altro metodo è quello di specificare che ci sono tre numeri e lettere con un conteggio: **>**

```
        /\d\u\{3}\d\{3}
```

Usando gli intervalli [] invece: **>**

```
        /[0-9][A-Z]\{3}[0-9]\{3}
```

Quali tra questi potreste usare? Quello che vi ricordate. Il metodo semplice che riuscite a ricordare è molto più veloce del metodo stravagante che non riuscite a ricordare. Se riuscite a ricordarli tutti, evitate l'ultimo, poiché è sia più lungo da battere, sia più lento da eseguire.

TROVARE UN IDENTIFICATIVO

Nei programmi in C (e in molti altri linguaggi del computer), un identificativo inizia con una lettera e più oltre è composto da lettere e numeri. Anche gli underscore possono essere usati. Questi possono essere trovati con: **>**

```
        /\<\h\w*\>
```

"\<" e "\>" sono usati per trovare solo parole intere. "\h" equivale a "[A-Za-z_]" e "\w" sta per "[0-9A-Za-z_]".

        Note:
        "\<" e "\>" dipendono dall'opzione 'iskeyword'. Se essa include "-", per esempio, poi "ident-" non è trovato. In questa situazione usate: **>**

```
                /\w\@<!\h\w*\w\@!
<
```

        Questo controlla se "\w" non trova corrispondenze prima o dopo l'identificativo.
        Vedete |/\@<!| e |/\@!|.

==

Capitolo seguente: |usr_28.txt| La piegatura

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

usr_28.txt Per Vim version 6.2. Ultima modifica: 2003 Dic 21

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Giuliano Bordonaro

La piegatura

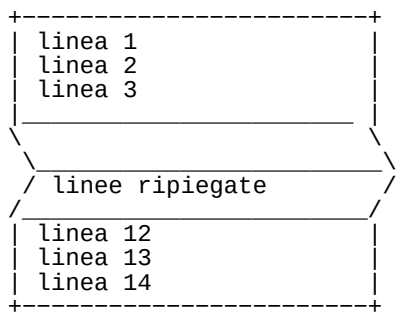
Un testo strutturato può essere diviso in sezioni. E le sezioni in sottosezioni. La piegatura vi consente di mostrare una sezione come una sola linea, consentendo una visione d'assieme. Questo capitolo spiega i diversi modi per farlo.

  Capitolo seguente: |usr_29.txt| Spostarsi attraverso i programmi
Capitolo precedente: |usr_27.txt| Comandi di ricerca e modelli
              Indice: |usr_toc.txt|

==========

28.1 Cosa vuol dire piegatura?

La piegatura serve per mostrare un gruppo di linee del buffer come se fossero una sola riga sullo schermo. Allo stesso modo con cui piegate un foglio di carta per renderlo più corto:



Il testo esiste ancora nel buffer, intatto. Soltanto che le linee visibili hanno una piegatura.

Un vantaggio conseguente alla piegatura è che potete godere di una migliore vista d'assieme della struttura del testo, piegando delle linee di una sezione e sostituendole con una sola linea che indica che lì c'è una sezione.

==========

28.2 Piegatura manuale

Provate: mettete il cursore entro un paragrafo e digitate: **>**

```
zfap
```

Vedrete che il paragrafo viene sostituito da una linea evidenziata. Avete effettuato una piegatura. |zf| è un operatore ed |ap| la selezione di un oggetto di testo. Potrete impiegare l'operatore |zf| con tutti i comandi di movimento per creare una piegatura del testo su cui vi siete spostati. |zf| lavora anche nel modo visuale.

Per rivedere il testo, dispiegatelo digitando: **>**

```
zo
```

E potrete richiudere la piegatura con: **>**

```
zc
```

Tutti i comandi di piegatura cominciano con "z". Con un po' di fantasia "z"

ricorda un foglio di carta piegato visto da un lato. La lettera che segue la "z" ha un significato mnemonico per aiutarvi a rammentare i comandi:

```
zf      F-old creation  (Creare una piegatura)
zo      O-pen a fold    (Riaprire una piegatura)
zc      C-lose a fold   (Chiudere una piegatura)
```

La piegatura può venire annidata: una regione di testo che contenga delle piegature può essere nuovamente piegata. Ad esempio, potete piegare tutti i paragrafi di questa sezione e poi farlo con le sezioni costituenti questo capitolo. Provate. Vedrete che aprendo la piegatura che contiene l'intero capitolo verranno conservate le pieghe nidificate così come erano, alcune potrebbero essere aperte ed altre invece chiuse.

Supponete di aver fatto diverse piegature e di voler invece vedere tutto il testo. Potreste portarvi su ogni piega e digitare "zo". Per fare prima usate questo comando: **>**

```
zr
```

Questo R-idurrà la piegatura. Il contrario è: **>**

```
zm
```

Questo piega M-ore (nuovamente). Potete ripetere "zr" e "zm" per aprire e chiudere le pieghe annidate di molti livelli.

Se aveste annidato attraverso molti livelli, potreste aprirli tutti con: **>**

```
zR
```

Questo R-iduce le piegature non lasciandone alcuna. E potete chiudere tutte le pieghe con: **>**

```
zM
```

Questo piegherà M-olto e M-olto di più.

Potete togliere rapidamente le piegature con il comando |zn|. E |zN| riporta la piegatura come era. |zi| cambia tra le due. Questo è un modo proficuo di lavorare:
- creare le piegature per avere la vista d'assieme del vostro file
- spostarvi dove vi pare per fare il vostro lavoro
- fare |zi| per cercare del testo e lavorarci
- fare |zi| nuovamente per tornare indietro e reiniziare lo spostamento

Maggiori informazioni sulla piegatura manuale sono nel manuale di riferimento: |fold-manual|

==
28.3 Lavorare con le piegature

Quando una piegatura è chiusa, i comandi di movimento come "j" e "k" la attraversano come se fosse una sola linea vuota. Ciò vi consente di spostarvi rapidamente sopra il testo ripiegato.

Potete copiare, cancellare ed incollare le piegature come se si trattasse di una sola linea. Ciò è utilissimo per riordinare delle funzioni entro un programma. Prima assicuratevi che ciascuna piega contenga tutta una funzione (o poco meno) selezionando il corretto 'foldmethod'. Poi cancellate la funzione con "dd", spostate il cursore ed incollatela con "p". Se alcune linee della funzione fossero prima o dopo la piegatura, potreste usare la selezione Visual:
- ponete il cursore sulla prima linea da spostare
- premete "V" per entrare nel modo Visual
- portate il cursore sull'ultima linea che volete spostare
- premete "d" per cancellare le linee selezionate.
- spostate il cursore sulla nuova posizione e "p"ut (incollate) qui le linee.

E' talvolta difficile vedere o ricordare dove si trovi una piegatura, ossia dove un comando |zo| potrebbe davvero operare. Per vedere le piegature definite: **>**

```
:set foldcolumn=4
```

Ciò mostrerà una piccola colonna alla sinistra della finestra per indicare le piegature. Un "+" viene mostrato per una piegatura chiusa. Un "-" è posto all'inizio di ciascuna piega aperta e "|" alle linee seguenti della piega.

Potete usare il mouse per aprire una piega cliccando sul "+" nella colonna della piegatura. Cliccare sul "−" o sul "|" chiuderà una piegatura aperta.

  Per aprire tutte le pieghe sulla linea del cursore usate |zO|.
  Per chiudere tutte le pieghe sulla linea del cursore usate |zC|.
  Per eliminare una piega sulla linea del cursore usate |zd|.
  Per eliminare tutte le pieghe sulla linea del cursore usate |zD|.

Se siete nell'Insert mode, la piega alla linea del cursore non è mai chiusa. Ciò vi consente di vedere ciò che state scrivendo!

Le piegature vengono aperte automaticamente quando si salta in giro o si sposta il cursore a sinistra od a destra. Ad esempio, il comando "0" apre la piega sotto il cursore (se 'foldopen' contiene "hor", che è il default). L'opzione 'foldopen' può essere cambiata per aprire pieghe per comandi specifici. Se volete che venga sempre aperta la linea sotto il cursore, fate così: **>**

```
:set foldopen=all
```

Attenzione: non potrete spostarvi entro una piega chiusa allora. Potreste voler usare ciò solo temporaneamente e poi tornare al settaggio di default: **>**

```
:set foldopen&
```

Potete far si che le piegature si chiudano automaticamente quando uscite da esse: **>**

```
:set foldclose=all
```

Ciò riapplicherà 'foldlevel' a tutte le piegature che non contengono il cursore. Dovrete provarlo per sentire se vi piace. Usate |zm| per piegare ancora e |zr| per piegare di meno (ridurre le pieghe).

La piega è localizzata nella finestra. Ciò vi permette di aprire due finestre sullo stesso buffer, una con le pieghe e l'altra senza. Od una con tutte le pieghe chiuse ed una con tutte aperte.

==
28.4 Salvataggio e ripristino delle piegature

Quando lasciate un file (iniziando ad editarne un altro), lo stato delle piegature viene perduto. Ritornando allo stesso file successivamente tutte le piegature aperte o chiuse manualmente saranno tornate al proprio default. Quando le pieghe sono state create manualmente, tutte le pieghe vanno perdute! Per salvare le piegature usate il comando |:mkview|: **>**

```
:mkview
```

Ciò scriverà nel file le impostazioni e le altre cose che influenzano la vista. Potete modificare quanto salvato con l'opzione 'viewoptions'. Riaprendo successivamente lo stesso file, potrete caricare nuovamente le viste: **>**

```
:loadview
```

Potete salvare fino a dieci viste di un file. Ad esempio, salvare le impostazioni attuali come terza vista e caricare la seconda: **>**

```
:mkview 3
:loadview 2
```

Notare che quando aggiungete o cancellate delle linee la vista potrebbe diventare non valida.
Così osservate l'opzione 'viewdir', che indica dove le viste sono state salvate. Potreste voler cancellare delle vecchie viste.

==
28.5 Piegature secondo le indentazioni

Definire le pieghe con |zf| costa un sacco di lavoro. Se avete strutturato il vostro testo dando un'indentatura più ampia alle componenti di livello più basso, potrete usare il metodo di piegatura delle indentazioni. Ciò creerà le piegature per tutti gli insiemi di linee aventi la stessa indentatura.
  Linee con un'indentatura più ampia diverranno piegature annidate. Ciò funziona assai bene con molti linguaggi di programmazione.

Provatelo impostando l'opzione 'foldmethod': >

```
:set foldmethod=indent
```

Così potrete usare i comandi |zm| e |zr| per piegare maggiormente o ridurre
le piegature. E' facile da vedere in questo testo di esempio:

```
This line is not indented
        This line is indented once
                This line is indented twice
                This line is indented twice
        This line is indented once
This line is not indented
        This line is indented once
        This line is indented once
```

Notare che la relazione tra l'estensione dell'indentatura e la profondità
della piegatura dipendono dall'opzione 'shiftwidth'. Ciascun valore
'shiftwidth' dell'indentatura aggiunge un'unità alla profondità della
piegatura. Ciò viene chiamato livello di piegatura.

Usando i comandi |zr| e |zm| adesso aumentate o diminuite il valore
dell'opzione 'foldlevel'. Potete anche impostarlo direttamente: >

```
:set foldlevel=3
```

Ciò significa che tutte le piegature con tre volte l'indentatura 'shiftwidth'
o più verranno chiuse. Più basso sarà il livello di piegatura,
tanto più piegature verranno chiuse. Quando 'foldlevel' è zero tutte le
piegature vengono chiuse. |zM| pone 'foldlevel' a zero. Il comando opposto
|zR| pone 'foldlevel' al livello più profondo presente entro il file.

Così ci sono due modi per aprire e chiudere le piegature:
(A) Impostando il livello di piegatura.
    Fornisce un velocissimo modo di "zooming out" per vedere la struttura del
    testo, spostare il cursore, e rieffettuare lo "zoom in" del testo.

(B) Usando i comandi |zo| e |zc| per aprire o chiudere specifiche piegature.
    Consente di aprire soltanto le piegature volute, lasciando chiuse le
    altre.

Ciò può venire combinato: potete prima chiudere molte piegature usando |zm|
poche volte e poi aprire una piega specifica con |zo|. Od aprire tutte le
piegature con |zR| e poi chiuderne alcune con |zc|.

Ma non potete definire delle piegature manualmente se il 'foldmethod' è
"indent", poiché ciò entra in conflitto con la relazione tra l'indentatura ed
il livello di piegatura.

Di più circa le piegature da indentatura nel manuale di riferimento:
|fold-indent|

==
28.6 Piegature mediante marker

Dei marker entro il testo vengono usati per specificare l'inizio e la fine di
una regione di piegatura. Ciò dà un esatto controllo su quali linee siano
incluse entro una piegatura. Lo svantaggio è che il testo necessita di essere
modificato.

Provatelo: >

```
:set foldmethod=marker
```

Testo di esempio, come apparirebbe in un programma C:

```
/* foobar () {{{ */
int foobar()
{
        /* return a value {{{ */
        return 42;
        /* }}} */
}
/* }}} */
```

Notate che la linea piegata mostrerà il testo prima del marker. Ciò è molto
utile per capire cosa contenga la piegatura.

E' fastidioso quando i marker non siano più accoppiati correttamente dopo aver spostato qualche linea. Ciò può essere evitato usando marker numerati. Esempio:

```
        /* global variables {{{1 */
        int varA, varB;

        /* functions {{{1 */
        /* funcA() {{{2 */
        void funcA() {}

        /* funcB() {{{2 */
        void funcB() {}
        /* }}}1 */
```

Ad ogni marker numerato comincia una piegatura del livello specificato. Ciò farà che qualsiasi piegatura di livello più alto finisca qui. Potete usare proprio dei marker numerati di partenza per definire tutte le piegature. Solo quando voleste esplicitamente terminare una piegatura prima che ne inizi un'altra dovrete aggiungere un marker di fine.

Di più circa le piegature con i marker nel manuale di riferimento:
|fold-marker|

==
28.7 Piegature secondo la sintassi

Per ogni linguaggio Vim usa un file di sintassi diverso. Ciò definisce i colori per le diverse componenti entro il file. Se state leggendo questo testo su di un terminale che supporti i colori, i colori che vedete vengono fatti con il file di sintassi "help".
   Nei files di sintassi è possibile aggiungere oggetti di sintassi che abbiano l'argomento "fold". Questi definiscono una regione di piegatura. Ciò comporta di scrivere un file di sintassi ed aggiungergli questi oggetti. Non è semplice da fare. Ma una volta fatto, tutte le piegature avvengono automaticamente.
   Poniamo di usare un file di sintassi esistente. Così non vi è più nulla da spiegare. Potete aprire e chiudere piegature come spiegato sopra. Le piegature verranno create e distrutte automaticamente scrivendo il file.

Di più circa le piegature secondo sintassi nel manuale di riferimento:
|fold-syntax|

==
28.8 Piegature secondo espressione

E' come la piegatura da indentatura ma, invece di impiegare l'indentatura di una linea, una funzione utente verrà chiamata a calcolare il livello di piegatura di una linea. Potete usarlo per del testo ove qualcosa indichi quali linee debbano stare insieme. Un esempio è costituito da un messaggio e-mail dove il testo quotato viene indicato da un ">" prima della linea. Per piegare queste quote usate questo: **>**

```
        :set foldmethod=expr
        :set foldexpr=strlen(substitute(substitute(getline(v:lnum),'\\s','',\"g\"),'[^>].
*','',''))
```

Potete provarlo con il testo seguente:

```
> quoted text he wrote
> quoted text he wrote
> > double quoted text I wrote
> > double quoted text I wrote
```

Spiegazione per la 'foldexpr' usata nell'esempio (inside out):

getline(v:lnum)	prende la linea corrente
substitute(...,'\\s','','g')	toglie tutti gli spazi bianchi dalla linea
substitute(...,'[^>].*','',''))	toglie tutto ciò che segue '>'
strlen(...)	misura la lunghezza della stringa, pari al numero di '>'s trovati

Notare che una barra rovesciata deve essere inserita davanti ad ogni spazio, virgolette doppie e barra rovesciata per il comando ":set". Se ciò vi confonde usate **>**

```
        :set foldexpr
```

per vedere l'attuale valore risultante. Per correggere un'espressione complicata usate il completamento automatico da linea di comando: **>**

```
:set foldexpr=<Tab>
```

Dove <Tab> è il vero Tab. Vim completerà il valore precedente che potrete allora modificare.

Di più circa la piegatura secondo espressione nel manuale di riferimento: |fold-expr|

==
28.9 Piegatura delle linee non modificate

Risulta utile se impostate l'opzione 'diff' nella stessa finestra. Il comando |vimdiff| lo farà per voi. Esempio: **>**

```
setlocal diff foldmethod=diff scrollbind nowrap foldlevel=1
```

Fatelo in ogni finestra che mostri una versione diversa dello stesso file. Vedrete chiaramente le differenze tra i files, sintanto che il testo non modificato è piegato.

Per maggiori dettagli vedere |fold-diff|.

==
28.10 Quale metodo di piegatura usare?

Tutte queste possibilità vi faranno chiedere quale metodo possiate scegliere. Purtroppo non esiste una regola d'oro. Ecco alcuni suggerimenti.

Se c'è un file di sintassi con le piegature per il linguaggio che state utilizzando, questa è probabilmente la scelta migliore. Se non ce n'è uno potreste provare a scriverlo. Ciò richiede una buona conoscenza delle stringhe di ricerca. Non è facile, ma quando funziona non dovete definire le piegature manualmente.

Piegare manualmente delle regioni scrivendo comandi può essere usato per il testo non strutturato. Allora usate il comando |:mkview| per salvare e ripristinare le vostre piegature.

Il metodo dei marker richiede che voi modifichiate il file. Se condividete i files con altri o dovete rispettare gli standard della ditta, potreste non essere in grado di aggiungerne.
   Il vantaggio principale dei marker è che li potete mettere esattamente dove li volete. Ciò evita che qualche linea vada persa quando tagliate ed incollate delle piegature. E potete aggiungere un commento su cosa sia contenuto della piegatura.

La piegatura secondo l'indentazione è qualcosa che funziona in molti files, ma non sempre benissimo. Usatela quando non potete usare uno degli altri metodi. Comunque è molto utile per il contorno. Poi potete usare specificamente una sola 'shiftwidth' per ogni livello di annidamento.

Piegare mediante espressioni può fare le piegature in quasi tutti i testi strutturati. È anche semplice da specificare, particolarmente se l'inizio e la fine di una piegatura può essere trovato facilmente.
   Se usate il metodo "expr" per definire le piegature, ma non vi vengono come vorreste potete passare al metodo "manual". Ciò non rimuoverà le piegature già definite. Così potrete cancellare od aggiungere piegature manualmente.

==

Capitolo seguente: |usr_29.txt| Spostarsi attraverso i programmi

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

usr_29.txt   Per Vim version 6.2. Ultima modifica: 2003 Dic 24

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Antonio Colombo

Spostarsi attraverso i programmi

Chi ha creato Vim è un programmatore di computer. Non sorprende quindi che Vim contenga molte funzionalità utili nella scrittura di programmi. Spostatevi per trovare dove vengono definiti o usati degli identificatori. Visualizzate dichiarazioni in una finestra apposita. Ulteriori possibilità sono descritte nel prossimo capitolo.

  Capitolo seguente: |usr_30.txt| Editare programmi
Capitolo precedente: |usr_28.txt| La piegatura
             Indice: |usr_toc.txt|

==

29.1 Usare i tags

Cos'è una tag? È un posto dove è definito un identificatore. Un esempio è la definizione di una funzione in un programma C o C++. Una lista di tags è il contenuto di un file di tag. Il file può essere usato da Vim per saltare direttamente da un posto qualsiasi alla tag, il posto dove un identificatore è definito.
   Per generare il file di tag per tutti i file C nella directory corrente, usate il comando seguente: **>**

```
ctags *.c
```

[Il nome del file generato è sempre "tags" - NdT]
"ctags" è un programma a parte. Quasi tutti i sistemi Unix lo hanno già installato. Se ancora non l'avete, potete trovare "Exuberant ctags" qui:

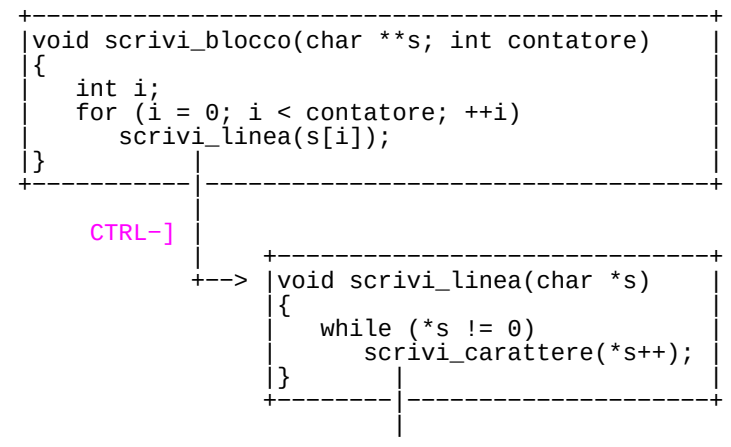
http://ctags.sf.net ~

Adesso, quando siete in Vim e volete andare a una definizione di funzione, potete raggiungerla velocemente usando il comando seguente: **>**

```
:tag startlist
```

Questo comando troverà una funzione "startlist" anche se è in un altro file.
   Il comando CTRL-] salta alla tag della parola che è sotto il cursore. Questo facilita l'esplorazione di una "giungla" di codice C. Supponete, ad es., di trovarvi in una funzione "scrivi_blocco". Potete vedere che chiama "scrivi_linea". Ma cosa fa "scrivi_linea"? Posizionando il cursore sulla chiamata a "scrivi_linea" e battendo CTRL-], saltate alla definizione di questa funzione.

La funzione "scrivi_linea" chiama "scrivi_carattere". Per capire cosa faccia "scrivi_carattere", posizionate il cursore sulla chiamata a "scrivi_carattere" e battete CTRL-]. Siete ora alla definizione di "scrivi_carattere".



```
                          CTRL-] |
                                 |    +-------------------------------------+
                                 +--> |void scrivi_carattere(char c)        |
                                      |{                                    |
                                      |    putchar((int)(unsigned char)c);  |
                                      |}                                    |
                                      +-------------------------------------+
```

Il comando ":tags" visualizza la lista di tag attraverso cui siete passati:

```
        :tags
          # A  tag          DA__  linea in file/testo ~
          1  1 scrivi_linea        8  scrivi_blocco.c ~
          2  1 scrivi_carattere    7  scrivi_linea.c ~
        > ~
>
```

Ora tornate indietro. Il comando CTRL-T torna alla tag precedente. Nell'esempio sopra tornate a una funzione "scrivi_linea", nella chiamata a "scrivi_carattere".

Questo comando accetta come argomento un contatore, che indica di quante tag tornare indietro. Siete andati avanti, e poi indietro. Andiamo avanti ancora. Il comando seguente va alla prima tag di una lista: **>**

```
        :tag
```

Potete premettergli un contatore e saltare in avanti di alcuni tag. Ad es.: ":3tag". Anche con CTRL-T si può specificare un contatore.

Questi comandi quindi vi permettono di scendere lungo una cascata di chiamate con CTRL-] e di tornare indietro ancora con CTRL-T. Usate ":tags" per vedere dove vi trovate.

DIVIDERE LA FINESTRA

Il comando ":tag" rimpiazza il file nella finestra corrente con quello che contiene la nuova funzione. Supponete di voler vedere non solo la vecchia funzione ma anche quella nuova. Potete dividere in due la finestra usando il comando ":split" seguito dal comando ":tag". Vim ha un comando "scorciatoia" che fa entrambe le cose: **>**

```
        :stag nome_tag
```

Per dividere in due la finestra corrente e saltare alla tag sotto il cursore usate questo comando: **>**

```
        CTRL-W ]
```

Se un contatore è specificato, la nuova finestra sarà alta quel numero di linee.

PIU' FILE DI TAG

Quando avete file in molte directory, potete create un file di tag in ognuna di esse. Vim sarà capace solo di saltare a tags all'interno di quella directory.

Per trovare più file di tag, impostate la opzione 'tags' per includere tutti i file di tag che vi interessano. Esempio: **>**

```
        :set tags=./tags,./../tags,./*/tags
```

Questo trova un file di tag nella stessa directory del file corrente, nel livello di directory superiore e in tutte le sub-directory.

In questo modo di usano parecchi file di tag, ma potrebbero servirene altri. Ad es., modificando un file in "~/proj/src", non trovereste il file di tag "~/proj/sub/tags". In vista di questa situazione Vim permette di cercare un intero albero di directory, per utilizzarne i file di tag. Esempio: **>**

```
        :set tags=~/proj/**/tags
```

UN SOLO FILE DI TAG

Quando Vim deve cercare in parecchi posti per trovare file di tag, potete udire il vostro hard disk che rumoreggia. La faccenda può diventare lunga. In questo caso è meglio impiegare lo stesso tempo per generare un unico grosso file di tag. Potreste lasciar girare un simile lavoro durante la notte.

È necessario il programma "Exuberant ctags", prima citato. Provvede un argomento per cercare attraverso un intero albero di directory: >

```
cd ~/proj
ctags -R .
```

Il bello della faccenda è che "Exuberant ctags" riconosce vari tipi di file. E quindi è possibile far questo non solo con programmi C e C++, ma anche per script Eiffel and perfino per script Vim. Si veda la documentazione di ctags per come funziona la cosa.
Ora dovete solo dire a Vim dov'è il vostro grosso file di tag: >

```
:set tags=~/proj/tags
```

CORRISPONDENZE MULTIPLE

Quando una funzione è definita più volte (o un metodo in parecchie classi), il comando ":tag" salterà alla prima delle definizioni. Se c'è una corrispondenza nel file corrente, quella è usata per prima.
Potete poi saltare ad altre corrispondenze per la stessa tag con: >

```
:tnext
```

Ripetete questo comando per trovare ulteriori corrispondenze. Se sono tante, potete scegliere quella a cui saltare: >

```
:tselect nome_tag
```

Vim vi farà scegliere da una lista:

```
  # pri tipo tag               file ~
  1 F   f    mch_init          os_amiga.c ~
               mch_init() ~
  2 F   f    mch_init          os_mac.c ~
               mch_init() ~
  3 F   f    mch_init          os_msdos.c ~
               mch_init(void) ~
  4 F   f    mch_init          os_riscos.c ~
               mch_init() ~
Batti n. di scelta (<INVIO> per lasciar perdere): ~
```

Potete ora scegliere il numero (nella prima colonna) della corrispondenza a cui volete saltare. L'informazione nelle altre colonne vi dà una buona idea di dove dove la corrispondenza è definita.

Per muoversi fra la tag corrispondenti, si possono usare questi comandi:

```
:tfirst                 vai alla prima corrispondenza
:[count]tprevious       vai alla [contatore] corrispondenza precedente
:[count]tnext           vai alla [contatore] prossima corrispondenza
:tlast                  vai alla ultima corrispondenza
```

Se [count] è omesso ne viene usato uno solo.

TROVARE NOMI TAG

Usare il completamento della linea comandi permette di evitare di immettere un lungo nome di tag. Immettete solo la parte iniziale e battete <Tab>: >

```
:tag scrivi_<Tab>
```

Vi verrà segnalata la prima corrispondenza. Se non è quella giusta, battete <Tab> finchè non trovate quella buona.
Talora conoscete solo parte del nome di una funzione. Oppure avete tante tag che iniziano con la stessa stringa, ma finiscono in modo diverso. Allora potete dire a Vim di usare un'espressione per trovare la tag.
Supponete di voler saltare a una tag che contiene "blocco". Prima battete questo: >

```
:tag /blocco
```

Adesso usate il completamento della linea comandi: battete <Tab>. Vim troverà tutte le tag che contengono "blocco" e userà la prima corrispondenza.
La "/" prima del nome della tag dice a Vim che quel che segue non è un nome di tag, ma un'espressione. Potete usate tutto quel che mettete in un'espressione di ricerca qui. Ad es., supponete di voler selezionare una tag

che comincia con "scrivi_": >

```
:tselect /^scrivi_
```

L'accento circonflesso "^" specifica che il nome della tag inizia con "scrivi_". Altrimenti una corrispondenza verrebbe trovata anche nel mezzo del nome di tag. Allo stesso modo il carattere "$" posto alla fine richiede solo corrispondenze che si trovino nella parte finale del nome di una tag.

UN NAVIGATORE DI TAG

Dato che CTRL-] vi porta alla definizione dell'identificatore sotto il cursore, potete usate una lista di nomi di identificatore come un indice. Ecco un esempio.
Prima create una lista di identificatori (ciò richiede "Exuberant ctags"): >

```
ctags --c-types=f -f funzioni *.c
```

Adesso avviate Vim senza un file, ed editate questo file in Vim, in una finestra separata verticalmente: >

```
vim
:vsplit funzioni
```

La finestra contiene una lista di tutte le funzioni. Ci sono anche altre informazioni, ma potete ignorarle. Battete ":set ts=99" per fa un po' di "pulizia".
In questa finestra, definite la mappatura: >

```
:nmap <buffer> <CR> 0ye<C-W>w:tag <C-R>"<CR>
```

Muovete il cursore sulla linea che contiene una funzione a cui volete saltare. Poi premete <Invio>. Vim passerà all'altra finestra e salterà alla funzione che avete selezionato.

ARGOMENTI CORRELATI

Potete impostare 'ignorecase' per non dover battere maiuscole e minuscole nei nomi di tag.

L'opzione 'tagbsearch' dice se il file di tag è già in ordine alfabetico o no. Il default è di supporre che il file di tag sia già ordinato, cosa che rende la ricerca molto più veloce, ma non funziona se il file di tag non è ordinato.

L'opzione 'taglength' si può usare per dire a Vim il numero di caratteri significativi in un nome di tag.

Quando usate il programma SNiFF+, potete usate l'interfaccia fra SNiFF e Vim. Si veda |sniff|. SNiFF+ è un programma a pagamento.

Cscope è un programma free. Non solo trova i posti dove un identificatore è stato dichiarato, ma anche dove viene usato. Si veda |cscope|.

==
29.2 La finestra di anteprima

Quando modificate del codice che contiene una chiamata a funzione, dovete passargli gli argomenti corretti. Per sapere che valori passare potete guardare a come la funzione è definita. Il meccanismo delle tag si presta molto bene ad effettuare questo controllo. Preferibilmente la definizione viene visualizzata in un'altra finestra. Per questo si può usare la finestra di anteprima.
Per aprire una finestra di anteprima per visualizzare la funzione "scrivi_carattere": >

```
:ptag scrivi_carattere
```

Vim aprirà una finestra, e salterà alla tag "scrivi_carattere". Poi vi riporterà indietro alla posizione di partenza. Così potete continuare a modificare il file senza dover ricorrere al comando CTRL-W.
Se il nome di una funzione ricorre nel testo, potete vedere la sua definizione nella finestra di anteprima battendo: >

```
CTRL-W }
```

Esiste uno script che automaticamente visualizza il testo dove è stata

definita la parola sotto il cursore. Si veda |CursorHold-example|.

Per chiudere la finestra di anteprima, usate questo comando: >

```
:pclose
```

Per modificare un file nella finestra di anteprima, usate ":pedit". Questo può tornare utile per modificare un file "header" [contenente definizioni che di solito includono più programmi - Ndt], ad es.: >

```
:pedit definizioni.h
```

Infine, ":psearch" si può usare per trovare una parola nel file corrente e nei file da questo inclusi e visualizzare la corrispondenza nella finestra di anteprima. Questo è particolarmente utile se si usano funzioni di libreria, per le quali non avete un file di tag. Ad es.: >

```
:psearch popen
```

Ciò mostrerà il file "stdio.h" nella finestra di anteprima, col prototipo di funzione per popen():

```
FILE    *popen __P((const char *, const char *)); ~
```

Potete specificare l'altezza della finestra di anteprima, quando è aperta, con l'opzione 'previewheight'.

==
29.3 Muoversi all'interno di un programma

Poiché un programma ha una struttura, Vim può riconoscere parti di esso. Ci sono comandi che si possono usare per muoversi all'interno.
   I programmi C spesso contengono costrutti del tipo:

```
#ifdef USE_POPEN ~
    fd = popen("ls", "r") ~
#else ~
    fd = fopen("tmp", "w") ~
#endif ~
```

Ma molto più lunghi, e possibilmente indentati. Posizionate il cursore su "#ifdef" e battete %. Vim salterà ad "#else". Battendo % ancora arrivate a "#endif". Un altro % vi riporta indietro a "#ifdef".
   Quando il costrutto è annidato, Vim troverà l'elemento corrispondente. Questa è una buona maniera per controllare se avete dimenticato un "#endif".
   Quando siete nel bel mezzo di un "#if" - "#endif", potete saltare all'inizio dello stesso con: >

```
[#
```

Se non siete dopo un costrutto "#if" o "#ifdef" Vim manda un segnale acustico di errore. Per raggiungere il prossimo "#else" o "#endif" usate: >

```
]#
```

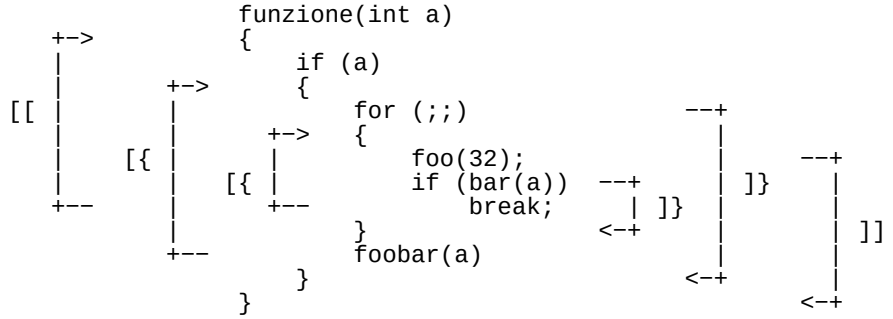
Questi due comandi saltano qualsiasi blocco "#if" - "#endif" che incontrano. Ad es.:

```
#if defined(HAS_INC_H) ~
    a = a + inc(); ~
# ifdef USE_THEME ~
    a += 3; ~
# endif ~
    set_width(a); ~
```

Col cursore sull'ultima linea, "[#" vi porta alla prima linea. Il blocco "#ifdef" - "#endif" nel mezzo viene saltato.

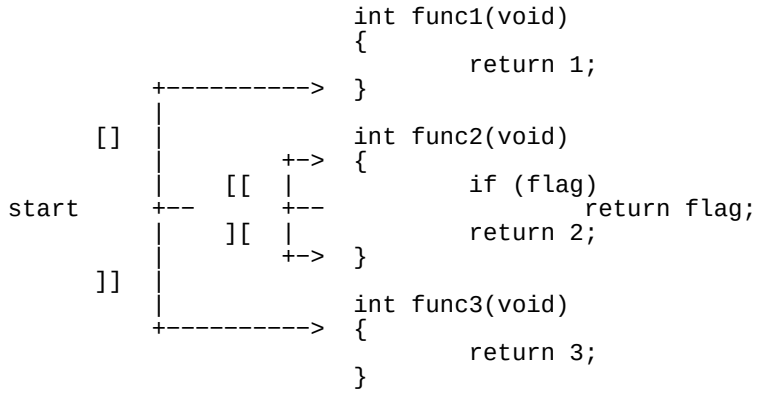
MUOVERSI ALL'INTERNO DI BLOCCHI DI PROGRAMMA

Nel codice C, i blocchi sono racchiusi fra {}. Questi blocchi possono essere anche molto lunghi. Per spostarvi all'inizio del blocco più esterno, usate il comando "[[". Usate "]]" per portarvi alla fine. Questi comandi suppongono che "{" e "}" siano nella prima colonna.
   Il comando "[{" vi porta all'inizio del blocco corrente. Coppie di {} allo stesso livello vengono saltate. "]}" salta alla fine.
   Una panoramica:



Quando scrivete in C++ o Java, il blocco più esterno {} è per la "class". Il livello successivo di {} è per un "method". Se siete da qualche parte all'interno di una "class", usate "[m" per trovare il precedente inizio di un "method". "]m" trova la prossima fine di un "method".

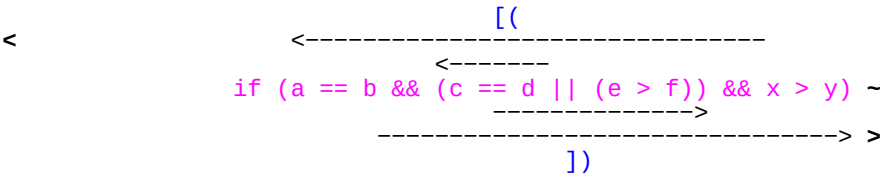
Inoltre, "[]" va all'indietro alla fine di una funzione e "][" va avanti alla fine di una funzione. La fine di una funzione è definito da una "}" nella prima colonna.



Non dimenticate che potete anche usare "%" per muovervi tra (), {} e [] corrispondenti. Questo funziona anche quando sono separati da parecchie linee di codice.

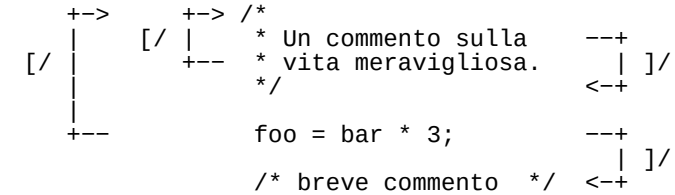
MUOVERSI TRA LE PARENTESI

I comandi "[(" e "])" funzionano in modo simile a "[{" e "]}", solo che si muovono fra coppie di () invece che di {}.



MUOVERSI TRA I COMMENTI

Per muoversi all'indietro verso l'inizio di un commento, usate "[/". Muovetevi in avanti verso la fine di un commento con "]/". Questo funziona solo per commenti delimitati da /* - */.



==========

29.4 Trovare identificatori globali

State modificando un programma C e vi domandate se una variabile è dichiarata

come "int" o "unsigned". Un modo veloce per accertarlo è quello di usare il comando "[I".
Supponete il cursore sia sulla parola "column". Battete: >

```
[I
```

Vim elencherà le linee corrispondenti che riesce a trovare. Non solo nel file corrente, ma anche in tutti i file inclusi (e nei file da questi ultimi a loro volta inclusi, etc.). Il risultato è del tipo:

```
structs.h ~
 1:   29     unsigned     column;    /* column number */ ~
```

Il vantaggio rispetto ad usare le tag o la finestra di anteprima è che i file inclusi sono a loro volta controllati. Il più delle volte si arriva così alla dichiarazione che interessa. Anche quando il file di tag non è aggiornato. Anche quando non esiste un file di tag per i file inclusi.
Perché il comando "[I" funzioni, servono tuttavia alcuni prerequisiti. Innanzitutto, l'opzione 'include' deve specificare come un file è incluso. Il valore predefinito funzione per C e C++. Va cambiato in maniera opportuna, se state modificando file scritti in altri linguaggi.

LOCALIZZARE I FILE INCLUSI

Vim troverà i file inclusi nelle directory specificate con l'opzione 'path'. Se una directory non è listata, alcuni file inclusi non potranno essere trovati. Potete accertarvene con questo comando: >

```
:checkpath
```

Verranno elencati i file inclusi che non si riescono a trovare. Anche i file inclusi nei file che [invece] si riescono a trovare. Un esempio di output:

```
--- File inclusi non trovati nel percorso --- ~
<io.h> ~
vim.h --> ~
  <functions.h> ~
  <clib/exec_protos.h> ~
```

Il file "io.h" è incluso dal file corrente e non può venire trovato. "vim.h" invece può essere trovato, e quindi ":checkpath" lo legge e controlla cosa a sua volta includa. I file "functions.h" e "clib/exec_protos.h", inclusi da "vim.h" non vengono trovati.

```
Note:
Vim non è un compilatore. Non interpreta istruzioni "#ifdef".
Questo significa che ogni "#include" viene controllata, anche
se viene dopo una istruzione "#if NEVER". [Ossia, si
controllano tutte le "#include", sia quelle in uso che quelle
inutilizzate. - NdT]
```

Per correggere i file che non possono essere trovati, aggiungete una directory all'opzione 'path'. Un buon posto per scoprire dove sia il Makefile. Cercate linee che contengano parametri "-I", tipo "-I/usr/local/X11". Per aggiungere questa directory usate: >

```
:set path+=/usr/local/X11
```

Quando ci sono tante subdirectory, potete usare la "wildcard" "*". Ad es.: >

```
:set path+=/usr/*/include
```

Questo potrebbe trovare dei file in "/usr/local/include" e anche in "/usr/X11/include".

Se lavorate su un progetto con un intero albero nidificato di file inclusi, la notazione "**" è utile. Utilizzandola, verranno cercate tutte le sottodirectory. Ad es.: >

```
:set path+=/projects/invent/**/include
```

Così si troveranno file nelle directory:

```
/projects/invent/include ~
/projects/invent/main/include ~
/projects/invent/main/os/include ~
etc.
```

Ci sono anche ulteriori possibilità. Si veda l'opzione 'path' per maggiori informazioni.
   Se voler vedere quali file inclusi sono stati trovati, usate questo comando: **>**

```
:checkpath!
```

Otterrete una (...lunga) lista dei file inclusi, i file che questi a loro volta includono, e così via. Per non allungare troppo la lista, Vim indica "(Già elencato)" per file che sono già stati censiti, e non lista di nuovo tutti i file che questi includono.

SALTARE A UNA CORRISPONDENZA

"[I" produce una lista con solo una linea di testo. Quando volete dare un'occhiata più da vicino al primo elemento, potete saltare a quella linea col comando: **>**

```
[<Tab>
```

Potete anche usare "[CTRL-I", poiché CTRL-I ha lo stesso effetto che battere <Tab>.

La lista che "[I" produce ha un numero all'inizio di ogni linea. Quando volete saltare a un altro elemento diverso dal primo, digitate quel numero prima del comando: **>**

```
3[<Tab>
```

Salterà al terzo elemento nella lista. Ricordate che potete usate CTRL-O per tornare indietro dove vi trovavate.

COMANDI CORRELATI

```
[i              lista solo la prima corrispondenza
]I              lista solo elementi sotto il cursore
]i              lista solo il primo elemento sotto il cursore
```

TROVARE IDENTIFICATORI PARTICOLARI

Il comando "[I" trova qualsiasi identificatore. Per trovare solo macro, definite con "#define" usate: **>**

```
[D
```

Di nuovo, anche i file inclusi vengono controllati. L'opzione 'define' specifica com'è formata una linea che definisce elementi da ricercare tramite "[D". Potreste cambiare questa definizione per farla funzionare con linguaggi diversi da C o C++.
   I comandi correlati a "[D" sono:

```
[d              lista solo la prima corrispondenza
]D              lista solo elementi sotto il cursore
]d              lista solo il primo elemento sotto il cursore
```

==
29.5 Trovare identificatori locali

Il comando "[I" ricerca nei file inclusi. Per cercare solo nel file corrente, e saltare alla prima istruzione dove la parola sotto il cursore è usata: **>**

```
gD
```

Pensate a "gD" come: Go_to Definition [Vai alla definizione]. Questo comando è molto utile per trovare una variabile o funzione che sia stata dichiarata localmente ("static", nel gergo C). Ad es. (il cursore è su "contatore"):

```
   +->   static int contatore = 0;
   |
   |     int get_contatore(void)
gD |     {
   |         ++contatore;
   +--       return contatore;
         }
```

Per delimitare ulteriormente la ricerca, e considerare solo la funzione corrente, usate questo comando: **>**

```
        gd
```

Questo comando partirà dall'inizio della funzione corrente a trovare la prima occorrenza della parola sotto il cursore. In realtà, si posiziona all'indietro fino a trovare un linea vuota sopra la "{" a colonna 1. Da lì viene effettuata una ricerca in avanti dell'identificatore. Ad es. (cursore su "indice"):

```
                int find_entry(char *name)
                {
           +->      int indice;
           |
        gd |        for (indice = 0; indice < lunghezza_tabella; ++indice)
           |            if (strcmp(table[indice].name, name) == 0)
           +--              return indice;
                }
```

==========

Capitolo seguente: |usr_30.txt| Editare programmi

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

usr_30.txt   Per Vim version 6.2. Ultima modifica: 2004 Gen 17

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 l'indentazione 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 {arguments}
```

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' ~
Hit ENTER or type command to continue ~
```

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;                                  |
      cursor -> |       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 riguardo 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 settarlo 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, poichè 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 d'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|.

REINDIRIZZAMENTO 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 Indentazione dei files in C

Un programma è molto più facile da capire quando le linee siano state indentate 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 l'indentazione 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 indentata ad un livello aggiuntivo.

```
                                if (flag)
        Indent automatico  --->     do_the_work();
        Unindent automatico <--  if (other_flag) {
        Indent automatico  --->     do_file();
        Mantenimento  indent        do_some_more();
        Unindent automatico <--  }
```

Quando scrivete qualcosa tra parentesi graffe ({}), il testo verrà indentato all'inizio ma non alla fine. L'unindent verrà fatto dopo aver battuto '}', poichè Vim non può indovinare cosa state per scrivere.

Un effetto collaterale dell'indentazione automatica è che vi aiuta a trovare rapidamente gli errori nel codice. Quando battete una } per concludere una funzione, il solo vedere che l'indentazione automatica 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 indent. Così, se vedete più spazio bianco di quello che vi aspettavate, controllate le linee precedenti.

Quando avete del codice che sia mal formattato. od avete inserito e cancellato delle linee, dovrete reindentarlo. l'operatore "=" lo fa. La forma più semplice è: **>**

```
==
```

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

```
=a{
```

Se avete del codice veramente mal indentato, potete reindentare l'intero file con: **>**

```
gg=G
```

Comunque, non fatelo con dei file che avete indentato con cura manualmente. L'indentazione automatica fa un buon lavoro, ma in certe situazioni potreste volerla evitare.

IMPOSTARE LO STILE DI INDENTAZIONE

Persone diverse hanno differenti stili di indentazione. Per default, Vim fa un gradevole buon lavoro di indentazione, nel modo in cui lo fa il 90% dei programmatori. Comunque ci sono diversi stili; quindi, se volete, potete personalizzare lo stile di indentazione 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 Indentazione automatica

Non volete attivare l'opzione 'cindent' manualmente ogni volta che editate un file in C. Ecco come farlo avvenire 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 indentazione per questo tipo di file. La distribuzione di Vim include un buon numero di questi tipi per vari linguaggi di programmazione. Il file di indentazione poi si preoccuperà di predisporre la giusta indentazione per il file corrente.

Se non vi piace l'indentazione automatica, potete disattivarla: **>**

```
:filetype indent off
```

Se non vi piace l'indentazione per un determinato tipo di file, ecco come

evitarla. 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 dovreste usare "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 l'indentazione, potreste scrivere un vostro indent file. Come farlo è spiegato qui: |indent-expression|.

==
30.4 Altre indentazioni

La forma più semplice di indentazione automatica è quella dell'opzione 'autoindent'. Usa l'indentazione della riga precedente. Un po' più furba è l'opzione 'smartindent'. E' utile per i file che non hanno un indent file. 'smartindent' non è intelligente come 'cindent' ma lo è sempre più di 'autoindent'.
   Con 'smartindent' impostata viene aggiunto un livello extra di indentazione 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 tolta tutta l'indentazione. Viene fatto perchè le direttive di preprocessore così inizieranno tutte alla colonna 1. L'indentazione viene ripristinata alla linea successiva.

CORREZIONE DELLE INDENTAZIONI

Quando state usando 'autoindent' o 'smartindent' per ottenere l'indentazione della prima linea, molte volte dovrete aggiungere o rimuovere un valore di 'shiftwidth' dell'indentazione. Un modo rapido per farlo è usare i comandi CTRL-D e CTRL-T nell'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'indentazione in più. Scrivete CTRL-T. Il risultato:

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

Il comando CTRL-T, nell'Insert mode, aggiunge un solo valore di 'shiftwidth' all'indentatura, nulla in quella ove vi trovate.
   Continuate a scrivere la seconda linea, <Enter> e la terza linea. Questa volta l'indentazione è OK. Allora <Enter> e la prossima linea. Adesso avrete questo:

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

Per rimuovere l'indentatura superflua nell'ultima linea premete CTRL-D. Ciò cancellerà un solo valore di 'shiftwidth', non ha importanza dove vi troviate 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 indentare. Una combinazione utile è: **>**

```
        >i{
```

Ciò aggiungerà un'indentatura 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).
   Indentare 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 indentare di quattro, non potete usare un carattere di <Tab> per fare la vostra indentazione. Ci sono due modi per gestire la cosa:

1. Usare un mix di <Tab> e di spazi. Poichè 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 di <Tab> che sia possibile e riempie il resto con degli spazi.
   Usando il backspace opera più o meno in altra maniera. Un <BS> cancellerà sempre il numero di spazi previsto da 'softtabstop'. Così si adoperano il più possibile i <Tabs> e gli 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 indentare una linea ed un vero <Tab> quando si batte dopo il primo carattere non-blank. 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'indentazione, 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 le proprie indentazioni per usare gli spazi invece dei tabulatori. Comunque tutti i tabulatori che vengono dopo un carattere non spazio vuoto vengono lasciati. Se volete convertire anche questi ultimi agiungete un !: **>**

```
        :%retab!
```

Ciò è un po più pericoloso perchè 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 ~
         */ ~
```

Affinchè ciò funzioni debbono esserci alcuni flags 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 flags.

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++vien impiegato //, nei Makefiles #, negli scripts di Vim ". Ad esempio per far capire a Vim i commenti di C++: **>**

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

I due punti separano i flags 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 ne 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 poichè 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' affinchè 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

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

usr_31.txt   Per Vim version 6.2. Ultima modifica: 2003 Ott 21

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Roberto Franceschini

Sfruttare la GUI

Vim funziona bene in un terminale, ma la sua GUI ha qualche componente in più. Un navigatore di file può venir impiegato per i comandi che usano un file. Finestre di dialogo per scegliere tra più alternative. Tasti di scelta rapida per accedere a voci di menu rapidamente.

Capitolo seguente: |usr_40.txt| Definire nuovi comandi
Capitolo precedente: |usr_30.txt| Editare programmi
Indice: |usr_toc.txt|

==

31.1 Il Navigatore

Quando utilizzate la voce di menu File/Open... ottenete un navigatore di file. Questo rende semplice trovare il file che volete modificare. Ma cosa fare se voleste dividere la finestra per aprire un'altro file? Non c'è una voce di menu per questo. Potreste usare prima Window/Split e poi File/Open..., ma è lavoro in più.

Poichè digitate la maggior parte dei comandi in Vim, è possibile anche aprire il navigatore digitando un comando. Per fare in modo che il comando split utilizzi il navigatore, anteponete "browse": **>**

```
:browse split
```

Selezionate un file ed il comando ":split" verra eseguito su di esso. Se chiudete la finestra di dialogo non succederà niente, la finestra non verrà divisa.

Potete anche specificare un argomento per il nome del file. Questo verrà utilizzato per indicare al navigatore dove iniziare la ricerca. Ad esempio: **>**

```
:browse split /etc
```

Il navigatore si aprirà iniziando la ricerca nella directory "/etc".

Potete anteporre il comando ":browse" a qualunque comando che apra un file.

Se non viene specificata alcuna directory, Vim deciderà dove iniziare la ricerca. Per default utilizza l'ultima directory utilizzata la sessione precedente. Quindi se voi utilizzate ":browse split" e selezionate un file in "/usr/local/share", la prossima volta che utilizzerete ":browse" la ricerca inizierà nuovamente in "/usr/local/share".

Ciò può essere modificato con l'opzione 'browsedir'. Può assumere uno di questi tre valori:

```
last            Utilizza l'ultima directory selezionata (default)
buffer          Utilizza la stessa directory del buffer corrente
current         Utilizza la directory corrente
```

Ad esempio, se voi siete nella directory "/usr", e state editando il file "/usr/local/share/readme", allora il comando: **>**

```
:set browsedir=buffer
:browse edit
```

Avvierà il navigatore in "/usr/local/share". In alternativa: **>**

```
:set browsedir=current
:browse edit
```

Avvierà il navigatore in "/usr".

Note:
Per evitare l'utilizzo del mouse, molti navigatori permettono di muoversi usando combinazioni di tasti. Dato che queste sono differenti per ogni sistema, non verranno spiegate qui. Vim utilizza un navigatore standard quando possibile, e la documentazione del

vostro sistema dovrebbe contenere una spiegazione delle scorciatoie da tastiera.

Se non utilizzate l'interfaccia grafica, potete utilizzare la finestra del "file explorer" per selezionare i file come in un navigatore. Questo comunque non funziona per il comando ":browse". Vedere |file-explorer|.

==
31.2 Conferme

Vim vi protegge dal sovrascrivere accidentalmente un file o da altri modi di perdere delle modifiche. Se fate qualcosa che potrebbe essere sbagliato, Vim produce un messaggio di errore e vi suggerisce di aggiungere ! se veramente volete proseguire.
Per evitare di digitare nuovamente il comando con !, potete fare in modo che Vim vi presenti una finestra di dialogo. Potete quindi scegliere "OK" o "Cancel" per dire a Vim cosa volete.
Ad esempio, voi state aprendo un file e vi apportate delle modifiche. Iniziate la modifica di un altro file con: **>**

```
:confirm edit foo.txt
```

Vim farà apparire una finestra di dialogo che appare simile a questa:

```
+-----------------------------------+
|                                   |
|   ?   Save changes to "bar.txt"?  |
|                                   |
|   YES   NO             CANCEL     |
+-----------------------------------+
```

Adesso fate la vostra scelta. Se volete salvare le modifiche, selezionate "YES". Se volete abbandonare le modifiche per sempre: "NO". Se avete dimenticato cosa stavate facendo e volete verificare le modifiche effettuate usate "CANCEL". Tornerete allo stesso file, con le modifiche ancora lì.

Come ":browse", il comando ":confirm" può essere anteposto alla maggior parte dei comandi che aprano un altro file. E possono essere anche combinati: **>**

```
:confirm browse edit
```

Se il buffer corrente fosse stato modificato, ciò genererebbe una finestra di dialogo. Quindi apparirebbe un navigatore per scegliere il file da aprire.

Note:
Nelle finestre di dialogo potete usare la tastiera per selezionare la scelta. Solitamente il tasto <Tab> e le frecce cambiano la scelta. Premendo <Enter> selezionate la scelta. Questo, comunque, dipende dal sistema che utilizzate.

Il comando ":confirm" funziona anche quando non state usando la GUI. Invece di aprire una finestra di idalogo, Vim stamperà il messaggio in fondo alla finestra e vi chiederà di premere un tasto per effettuare la scelta. **>**

```
:confirm edit main.c
<       Save changes to "Untitled"? ~
        [Y]es, (N)o, (C)ancel:  ~
```

Ora potete premere il tasto per la scelta. Non dovrete premere <Enter>, a differenza delle altre battute dalla riga di comando.

==
31.3 Scelte rapide

La tastiera si usa per tutti i comandi di Vim. I menu forniscono un modo più semplice per selezionare i comandi, senza sapere come vengono chiamati. Ma dovete spostare la vostra mano dalla tastiera ed afferrare il mouse.
I menu possono spesso essere selezionati anche da tastiera. Dipende dal vostro sistema, ma quasi sempre funziona così. Usate il tasto <Alt> assieme alla lettera sottolineata del menu. Per esempio <A-w> (<Alt> e w) aprirà il menu Finestra.
Nel menu Finestra , "split" ha la <p> sottolineata. Per selezionarlo premete <Alt> e quindi <p> assieme.

Dopo la prima selezione di un menu con il tasto <Alt>, potete usare le frecce per muovervi attraverso i menu. La freccia a destra seleziona un submenu e la freccia a sinistra lo chiude. Anche <Esc> chiude un menu. <Enter> seleziona una scelta.

Esiste un conflitto utilizzando il tasto <Alt> per scegliere le componenti dei menu ed impiegando le combinazioni del tasto <Alt> per le mappature.
L'opzione 'winaltkeys' dice a Vim cosa può fare con il tasto <Alt>.
   Il valore di default "menu" è la scelta migliore: se la combinazione di tasti è una scelta rapida per il menu non può essere mappata. Tutti gli altri tasti sono disponibili per mappature.
   Il valore "no" non consentirà l'uso di alcun tasto <Alt> per i menu. Così dovrete usare il mouse per i menu, e potranno essere mappate tutte le combinazioni con <Alt>.
   Se impostata a "yes" significa che potrà utilizzare le combinazioni con <Alt> per i menu. Qualche combinazione del tasto <Alt> potrà eseguire anche azioni diverse dal selezionare i menu.

==
31.4 Posizione e dimensione della finestra

Per leggere la posizione corrente della finestra di Vim usate: >

```
:winpos
```

Questo funzionerà solo nella GUI. L'output può essere simile a questo:

```
Window position: X 272, Y 103 ~
```

La posizione è data in pixel dello schermo. Ora potete usare i numeri per muovere la finestra di Vim. Per esempio, per muoverla cento pixel a sinistra: >

```
:winpos 172 103
```
<

Note:
Può esserci una leggera differenza tra la posizione riportata e dove la finestra si posiziona. Questo a causa del bordo della finestra. Questo è aggiunto dal programma gestore delle finestre.

Potete utilizzare questo comando nello script di avvio per posizionare la finestra in una posizione specifica.

La dimensione della finestra di Vim è misurata in caratteri. Quindi dipende dalla dimensione del font in uso. Potete visualizzare la dimensione corrente con questo comando: >

```
:set lines columns
```

Per cambiare la dimensione impostate le opzioni 'lines' e/o 'columns' ad un nuovo valore: >

```
:set lines=50
:set columns=80
```

La visualizzazione della dimensione in un terminale funziona proprio come nella GUI. Nella maggior parte dei terminali non è però possibile impostare la dimensione.

Puoi avviare la versione X-Windows di gvim con un argomento per specificare la dimensione e la posizione della finestra: >

```
gvim -geometry {width}x{height}+{x_offset}+{y_offset}
```

{width} e {height} sono in caratteri, {x_offset} e {y_offset} sono in pixel. Esempio: >

```
gvim -geometry 80x25+100+300
```

==
31.5 Varie

Potete utilizzare gvim per scrivere un messaggio di posta elettronica. Nel vostro programma di posta elettronica dovete selezionare gvim come editor per i messaggi. Quando proverete, probabilmente non funzionerà: Il programma di posta pensa che la scrittura del messaggio sia finita, mentre gvim sta girando ancora!
   Succede che gvim si scollega dalla shell nella quale è stato avviato. Ciò va bene quando avviate gvim in un terminale, per poter fare altro in quel terminale. Ma quando volete davvero aspettare che gvim abbia finito, dovete fare in modo che non si scolleghi. L'argomento "-f" lo fa: >

```
gvim -f file.txt
```

La "-f" sta per "foreground" (primo piano). Ora Vim bloccherà la shell da cui è stato avviato fino a che avrete finito di scrivere e siete usciti.

AVVIO RITARDATO DELLA GUI

In Unix è possibile avviare Vim in un terminale. Questo serve se volete avviare diversi programmi nella stessa shell. Se voi state lavorando su di un file e, ad un certo punto, volete passare alla GUI, potete avviarla con: **>**

```
:gui
```

Vim aprirà la finestra della GUI e non userà più il terminale. Potete quindi utilizzare il terminale per altri scopi. L'argomento "-f" viene qui usato per avviare la GUI in primo piano. Potete anche usare ":gui -f".

IL FILE DI AVVIO DI GVIM

Quando gvim viene avviato, legge il file gvimrc. Questo è simile al file vimrc usato quando lanciate Vim. Il file gvimrc può essere utilizzato per impostazioni e comandi che vengono usati solo quando la GUI deve ancora venire avviata. Per esempio, potete impostare l'opzione 'lines' per avere una diversa dimensione della finestra: **>**

```
:set lines=55
```

Non dovete fare ciò entro un terminale, in quanto la sua dimensione è fissa (ad eccezione di xterm che supporta il ridimensionamento).
Il file gvimrc viene cercato nella stessa posizione di vimrc. Normalmente il suo nome è "~/.gvimrc" in Unix e "$VIM/_gvimrc" in MS-Windows.
Se per qualche ragione non voleste utilizzare il file gvimrc normale, potreste specificarne un'altro con l'argomento "-U": **>**

```
gvim -U thisrc ...
```

Questo permette di avviare gvim per differenti tipi di lavori. Potete, per esempio, impostare un'altra dimensione di font.
Per evitare definitivamente la lettura del file gvimrc: **>**

```
gvim -U NONE ...
```

==========

Capitolo seguente: |usr_40.txt| Definire nuovi comandi

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

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

usr_40.txt   Per Vim version 6.2. Ultima modifica: 2004 Gen 17

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.

  Capitolo seguente: |usr_41.txt| Preparare uno script Vim
Capitolo precedente: |usr_31.txt| Sfruttare la GUI
              Indice: |usr_toc.txt|

==
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. E' un meccanismo semplice, ma potente.
   La forma più semplice è che ad un singolo tasto venga attribuito il significato di una sequenza di tasti. Poichè 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 sù 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 battete "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.

REMAPPING

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

RECURSIVE MAPPING

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 files 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 and Operator-pending
:vunmap         Visual
:nunmap         Normal
:ounmap         Operator-pending
:unmap!         Insert and 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 %<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 si 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 battuto 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 si 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.

Note:
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 battere 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          E' ammesso un intervallo; il default è la linea
                attuale.
−range=%        E' ammesso un intervallo; il default è l'intero file.
−range={count}  E' 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={number}	Il comando inizia a contare dal valore di default {number}. 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 un registro senza nome.) La specifica del registro è disponibile come <reg> (a.k.a. <register>).
-complete={type}	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 file venga salvato. Ciò farà sì che avvenga: >

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

"FileWritePre" è l'evento tramite il quale viene fatto agire questo comando: Proprio prima di salvare un file. Il "*" è un modello che fa corrispondere il nome del file. In questo caso tutti i file corrispondono.
   Avendo abilitato questo comando, quando eseguite uno ":write", Vim cerca qualsiasi autocomando che corrisponda con FileWritePre e lo esegue, e allora esegue ":write".
   La forma generale del comando :autocmd command è quella che segue: **>**

```
:autocmd [group] {events} {file_pattern} [nested] {command}
```

Il nome [group] è facoltativo. Viene usato per gestire e chiamare i comandi (maggiori particolari su di ciò più avanti). Il parametro {events} è un elenco di eventi (separati da una virgola) che fa partire il comando.
   {file_pattern} è il nome di un file, di solito con le wildcards. Ad esempio, usando "*.txt" fa si che l'autocomando venga utilizzato con tutti i file il cui nome termini in ".txt". Il flag facoltativo [nested] consente l'annidamento degli autocomandi (vedere in avanti) ed, in ultimo, {command} 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 assembly 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.

PATTERNS

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 le consuete wildcards per i files. 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").

```
Note:
Se si lavora con un filesystem che impiega la barra rovesciata per
separare i files, 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 {command} 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 {group}, 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 si che gli eventi li facciano partire aggiungete l'argomento "nested": >

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

L'ESECUZIONE DEGLI AUTOCOMANDI

E' 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("<afile>: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 IL MODO COMANDI NORMALE

I comandi eseguiti da un autocomando sono comandi a linea di comando. Se voleste usare dei comando in Normal mode, si può 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 scripts 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 si 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

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

usr_41.txt   Per Vim version 6.2. Ultima modifica: 2004 Feb 20

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Antonio Colombo

Preparare uno script Vim

Il linguaggio script di Vim è usato per il file di inizializzazione vimrc, nei file di sintassi, e molto altro. Questo capitolo spiega gli elementi che possono venir usati in uno script di Vim. Ce ne sono molti, così questo capitolo è lungo.

  Capitolo seguente: |usr_42.txt| Aggiungere nuovi menù
Capitolo precedente: |usr_40.txt| Definire nuovi comandi
             Indice: |usr_toc.txt|

==========

41.1 Introduzione *vim-script-intro*

Il vostro primo incontro con gli script di Vim è il file vimrc. Vim lo legge all'avvio e ne esegue i comandi. Potete impostare delle opzioni con il valore che preferite, ed usare al suo interno ogni comando che cominci per ":" (questi comandi sono talora designati come comandi Ex o comandi della linea di comando).

Anche i file di sintassi sono degli script Vim. E lo sono pure i file che impostano opzioni per uno specifico tipo di file. Una macro complessa può essere definita da un file script di Vim separato. Potete pensare ad altri usi voi stessi.

Partiamo da un semplice esempio: **>**

```
:let i = 1
:while i < 5
:  echo "il contatore è" i
:  let i = i + 1
:endwhile
```
<

Note:
I ":" non sono obbligatori in questo caso. Vanno necessariamente usati solo se immettete un comando direttamente. In un file di script Vim possono essere omessi. Noi li useremo comunque, per sottolineare che questi sono comandi coi ":" e per distinguerli dai comandi che si danno in Normal mode.

Il comando ":let" assegna un valore a una variabile. In forma generale: **>**

```
:let {variabile} = {espressione}
```

In questo caso il nome della variabile è "i" e l'espressione è un valore semplice, il numero 1.

Il comando ":while" inizia un ciclo. In forma generale: **>**

```
:while {condizione}
:  {comandi}
:endwhile
```

I comandi fino ad ":endwhile" che chiude il ciclo sono eseguiti finché la condizione rimane verificata. La condizione qui usata è l'espressione "i < 5". Questa espressione è vera finché la variabile i assume valori inferiori a 5.

Il comando ":echo" visualizza gli argomenti che gli vengono passati. In questo caso la stringa di caratteri "il contatore è" ed il valore della variabile i. Poiché i vale 1, visualizzerà:

```
il contatore è 1 ~
```

Poi c'è un altro ":let i =". Il valore usato è l'espressione "i + 1". Questo aggiunge 1 alla variabile i ed assegna il nuovo valore alla variabile stessa.
   Il risultato del codice di esempio è:

```
        il contatore è 1 ~
        il contatore è 2 ~
        il contatore è 3 ~
        il contatore è 4 ~
```

        Note:
        Se scriveste un ciclo che duri più del previsto, potete interromperlo
        battendo CTRL-C (CTRL-Break in ambiente MS-Windows)

TRE TIPI DI NUMERI

I numeri possono essere decimali, esadecimali od ottali. Un numero esadecimale inizia con "0x" o con "0X". Ad esempio "0x1f" è 31. Un numero ottale inizia con uno 0. "017" è 15. Attenzione: non mettete uno zero davanti ad un numero decimale, per non farlo interpretare come un numero ottale!
   Il comando ":echo" stampa sempre numeri decimali. Ad es.: **>**

```
        :echo 0x7f 036
<       127 30 ~
```

Un numero diventa negativo quando è preceduto da un segno "-". Questo vale anche per numeri esadecimali ed ottali. Un segno "-" indica anche una sottrazione. Confrontate questo esempio con il precedente: **>**

```
        :echo 0x7f -036
<       97 ~
```

Gli spazi bianchi in una espressione vengono ignorati. Comunque se ne raccomanda l'uso per separare gli elementi e rendere l'espressione di più facile lettura. Per esempio, per evitare confusione con un numero negativo, mettete un spazio fra il segno "-" e il numero che lo segue: **>**

```
        :echo 0x7f - 036
```

==
41.2 Variabili

Un nome di variabile è composto di lettere ASCII, cifre ed il carattere "_". Il primo carattere del nome non può essere un numero. Nomi validi di variabile sono:

```
        contatore
        _aap3
        nome_molto_lungo_di_variabile_con_sottolineature
        LunghezzaFunz
        LUNGHEZZA
```

Nomi di variabile non validi sono "pippo+pluto" e "6var".
   Queste variabili sono globali [valgono per tutti i buffer di una sessione Vim - NdT]. Per vedere una lista di tutte le variabili correntemente definite usate questo comando: **>**

```
        :let
```

Potete usare variabili globali in qualsiasi posto. Questo implica anche che quando la variabile "contatore" è usata in un file di script, potrebbe anche essere usata in un altro file. Questo porta quanto meno a confusione, od al peggio crea problemi veri. Per evitarli, potete usare una variabile locale per un file di script, mettendogli come prefisso "s:". Ad esempio, uno script contiene questi comandi: **>**

```
        :let s:contatore = 1
        :while s:contatore < 5
        :  source altro.vim
        :  let s:contatore = s:contatore + 1
        :endwhile
```

Poiché "s:contatore" vale solo per questo script, potete essere certi del fatto che lo script "altro.vim" non ne cambierà il valore. Se "altro.vim" usa una variabile "s:contatore", si tratterà di una copia diversa, propria di quello script. Per saperne di più sulle variabili locali ad uno script si veda: |script-variable|.

Ci sono ulteriori tipi di variabili, si veda |internal-variables|. Quelle usate più spesso sono:

```
        b:name          variabile locale propria di un buffer
        w:name          variabile locale propria di una finestra (window)
        g:name          variabile globale (anche in una funzione)
        v:name          variabile predefinita da Vim
```

ANNULLARE VARIABILI

Le variabili occupano memoria e vengono visualizzate nell'output del comando ":let". Per cancellare una variabile usate il comando ":unlet". Ad es.: **>**

```
        :unlet s:contatore
```

Questo cancella la variabile di script locale "s:contatore" e libera la memoria che essa occupava. Se non siete sicuri dell'esistenza della variabile, e non volete vedere un messaggio di errore nel caso la variabile non esista, aggiungete al comando il "!" **>**

```
        :unlet! s:contatore
```

Quando uno script Vim termina, le variabili locali che ha usato non saranno annullate automaticamente. La prossima volta che lo script verrà eseguito, potrà ancora usare il vecchio valore della variabile. Ad es.: **>**

```
        :if !exists("s:contatore_chiamate")
        :  let s:contatore_chiamate = 0
        :endif
        :let s:contatore_chiamate = s:contatore_chiamate + 1
        :echo "Chiamato" s:contatore_chiamate "volte"
```

La funzione "exists()" controlla l'esistenza di una variabile. Il suo argomento è il nome della variabile di cui volete controllare l'esistenza. Non la variabile in sé! Se scriveste: **>**

```
        :if !exists(s:contatore_chiamate)
```

Allora il valore di s:contatore_chiamate sarebbe usato come nome della variabile la cui esistenza viene controllata da exists(). Questo non è ciò che volete.

   Il punto esclamativo ! nega un valore. Quando il valore della funzione era "vero" (true) diventa "falso" (false). Quando era "falso" diventa "vero". Potete leggerlo come la parola "non" (not). Ossia "if !exists()" si può leggere come "if not exists()", ovvero "se è falso il valore ritornato da exists()".

   Vim considera "vero" qualsiasi valore diverso da zero. Solo lo zero è falso.

STRINGHE VARIABILI E COSTANTI

Finora alle variabili sono stati assegnati solo valori numerici. Si possono usare come valori anche stringhe di caratteri. Numeri e stringhe sono i due soli tipi di variabile che Vim consente. Il tipo della variabile è dinamico, viene impostato ogni volta che si assegna un valore alla variabile stessa con ":let:".

   Per assegnare il valore di una stringa ad una variabile, dovete usare una costante di tipo stringa. Ne esistono due tipi. La prima è una stringa di caratteri, racchiusa in doppi apici: **>**

```
        :let nome = "pietro"
        :echo nome
<       pietro ~
```

Se volete inserire un doppio apice all'interno della vostra stringa, metteteci davanti un "\" (barra retroversa). **>**

```
        :let nome = "\"pietro\""
        :echo nome
<       "pietro" ~
```

Per evitare di usare "\", potete racchiudere la stringa fra apici: **>**

```
        :let nome = '"pietro"'
        :echo nome
```

```
<       "pietro" ~
```

All'interno di una stringa racchiusa fra apici, tutti i caratteri restano invariati. Lo svantaggio è quello di non poter inserire un apice da solo. Un "\" viene anch'esso preso alla lettera, così non lo potete usare per cambiare il significato del carattere che lo segue.
   Nelle stringhe racchiuse fra doppi apici è possibile usare i caratteri speciali. Eccone qualcuno utile:

```
        \t              <Tab>   tabulatore
        \n              <NL>    a capo
        \r              <CR>    <Invio>
        \e              <Esc>   escape
        \b              <BS>    "cancella un carattere"
        \"              "       doppi apici
        \\              \       barra retroversa
        \<Esc>          <Esc>   escape (forma alternativa)
        \<C-W>          CTRL-W carattere "iniziale" per spostarsi tra finestre
```

Gli ultimi due sono solo a titolo di esempio. La forma "\<nome_car>" si può usare per inserire il carattere speciale "nome_car".
   Si veda |expr-quote| per la lista completa degli elementi speciali di una stringa.

==
41.3 Espressioni

Vim ha un modo ricco ma semplice per gestire le espressioni. Potete leggere qui la definizione: |expression-syntax|. Qui mostreremo gli elementi più comuni.
   I numeri, le stringhe e le variabili citate, sono già espressioni. Così ovunque ci si aspetti un'espressione, potete usare un numero, una stringa od una variabile. Altri elementi fondamentali in una espressione sono:

```
        $NOME           variabile d'ambiente
        &nome           opzione
        @r              registro
```

Esempio: >

```
        :echo "Il valore di 'tabstop' è" &ts
        :echo "La tua home directory [Unix] è" $HOME
        :if @a > 5
```

La forma &nome si può usare per salvare il valore di una opzione, impostare l'opzione a un nuovo valore, fare qualcosa, e ripristinare il valore precedente. >

```
        :let salva = &ic
        :set noic
        :/L'Inizio/,$delete
        :let &ic = save_ic
```

Questo serve ad essere certi che l'espressione "L'Inizio" sia usata con l'opzione 'ignorecase' non impostata. Alla fine, prende il valore che l'utente aveva impostato.

OPERAZIONI MATEMATICHE

Diventa più interessante se combiniamo questi elementi fondamentali. Cominciamo con la operazioni matematiche.

```
        a + b           addizione
        a - b           sottrazione
        a * b           moltiplicazione
        a / b           divisione
        a % b           modulo (resto della divisione)
```

Valgono le solite regole di precedenza. Ad es.: >

```
        :echo 10 + 5 * 2
<       20 ~
```

Il raggruppamento viene fatto con le parentesi. Niente sorprese qui. Ad es.: >

```
        :echo (10 + 5) * 2
<       30 ~
```

Le stringhe si possono riunire con ".". Ad es.: >

```
        :echo "foo" . "bar"
<       foobar ~
```

Quando si danno più argomenti al comando ":echo", questi vengono separati tra loro con uno spazio. Nell'esempio l'argomento è una singola espressione, così non viene inserito alcuno spazio.

L'espressione condizionale viene presa dal linguaggio C:

```
        a ? b : c
```

Se "a" è vera, si usa "b", altrimenti si usa "c". Ad es.: >

```
        :let i = 4
        :echo i > 5 ? "i è grande" : "i è piccolo"
<       i è piccolo ~
```

Le tre parti dei costrutti vengono sempre valutate prima, per cui potreste vederle funzionare come:

```
        (a) ? (b) : (c)
```

==

41.4 Condizioni

Il comando ":if" esegue i comandi che lo seguono, fino all'":endif", solo se la condizione è verificata. La forma generica è:

```
        :if {condizione}
           {comandi}
        :endif
```

Solo quando l'espressione {condizione} viene considerata vera (valore diverso da zero) i {comandi} vengono eseguiti. Questi debbono essere comandi validi. Se contengono spazzatura, Vim non riesce a trovare l'":endif".
   Potete anche usare ":else". La forma generica è:

```
        :if {condizione}
           {comandi}
        :else
           {comandi}
        :endif
```

Il secondo {comandi} viene eseguito solo se il primo non lo è.
   Infine, abbiamo l'":elseif":

```
        :if {condizione}
           {comandi}
        :elseif {condizione}
           {comandi}
        :endif
```

Questo equivale ad usare ":else" e poi "if", ma senza la necessità di inserire un ulteriore ":endif".
   Un esempio utile per il vostro file vimrc è verificare l'opzione 'term' e fare qualcosa in funzione del suo valore: >

```
        :if &term == "xterm"
        :  " Fai qualcosa per xterm
        :elseif &term == "vt100"
        :  " Fai qualcosa per un terminale vt100
        :else
        :  " Fai qualcosa per gli altri tipi di terminale
        :endif
```

OPERAZIONI LOGICHE

Ne abbiamo già utilizzate alcune negli esempi precedenti. Quelle di più frequente utilizzo sono:

```
        a == b          uguale a
        a != b          differente da
        a >  b          maggiore di
        a >= b          maggiore o uguale a
```

```
        a <  b          minore di
        a <= b          minore o uguale a
```

Il risultato è 1 se la condizione è verificata, altrimenti è 0. Ad es.: **>**

```
        :if v:version >= 600
        :  echo "Congratulazioni"
        :else
        :  echo "State usando una vecchia, versione, passate alla nuova"
        :endif
```

Qui "v:version" è una variabile definita da Vim, che assume il valore della versione di Vim. 600 sta per la versione 6.0. La versione 6.1 ha il valore 6.01. Questo è molto utile per scrivere uno script che debba funzionare con molte versioni di Vim. |v:version|

Gli operatori logici lavorano sia con numeri che con stringhe. Confrontando due stringhe, viene usata la differenza matematica. Questa confronta il valore dei byte, ma può non andare bene per alcuni linguaggi.
   Confrontando una stringa con un numero, la stringa viene prima convertita a numero. Questo è un po' arbitrario, perché quando una stringa non appare come un numero, viene utilizzato il numero 0. Ad es.: **>**

```
        :if 0 == "uno"
        :  echo "sì"
        :endif
```

Ciò darà come risultato "sì", poiché "uno" non appare come un numero, e così viene convertito nel numero zero.

Per le stringhe ci sono due ulteriori elementi;

```
        a =~ b          corrisponde a
        a !~ b          non corrisponde a
```

L'elemento di sinistra "a" viene usato come una stringa. L'elemento di destra "b" viene usato come espressione, simile a quelle usate per la ricerca. Ad es.: **>**

```
        :if str =~ " "
        :  echo "str contiene uno spazio"
        :endif
        :if str !~ '\.$'
        :  echo "str non finisce con un punto"
        :endif
```

Notate l'uso della stringa inclusa fra apici semplici nell'espressione. Ciò è utile perché le barre retroverse devono essere scritte due volte all'interno di una stringa inclusa fra doppi apici, e le espressioni regolari tendono a contenere parecchie barre retroverse.

   L'opzione 'ignorecase' viene usata nel confronto tra stringhe. Se non volete ciò, aggiungete "#" per tener conto di maiuscole e minuscole, e "?" per non farlo. Così "==?" verifica se due stringhe siano uguali, trascurando maiuscole e minuscole. E "!~#" controlla se una espressione non corrisponde, controllando anche maiuscole e minuscole. Per la tabella completa degli operatori, si veda |expr-==|.

ALTRI CICLI

Il comando ":while" è già stato citato. Ci sono altri due comandi che si possono usare tra ":while" e ":endwhile":

```
        :continue               Ritorna all'inizio del ciclo "while"; il ciclo
                                inizia una nuova iterazione.
        :break                  Salta in avanti fino a ":endwhile"; il ciclo
                                termina l'esecuzione.
```

Ad es.: **>**

```
        :while contatore < 40
        :  call fa_qualche_controllo()
        :  if salta_un_giro
        :    continue
        :  endif
        :  if abbiamo_finito
        :    break
        :  endif
```

```
        :  sleep 50m
        :endwhile
```

Il comando ":sleep" fa fare a Vim un pisolino. "50m" significa "per cinquanta millisecondi". Un altro esempio è ":sleep 4", che fa dormire per quattro secondi.

==
41.5 Esecuzione di una espressione

Finora i comandi nello script venivano eseguiti direttamente da Vim. Il comando ":execute" permette di eseguire il risultato di una espressione. È questa una maniera molto potente per costruire comandi ed eseguirli.
   Un esempio è per saltare ad una tag, che sia contenuta in una variabile:
 >

```
        :execute "tag " . nome_tag
```

Il "." si usa per concatenare la stringa "tag " con il valore della variabile "nome_tag". Supponiamo che "tag_name" abbia come valore "get_cmd", allora il comando che verrà eseguito è: **>**

```
        :tag get_cmd
```

Il comando ":execute" può solo eseguire i comandi che iniziano con ":". Il comando ":normal" esegue i comandi che si possono dare in Normal mode. Comunque, l'argomento di ":normal" non è una espressione, ma i caratteri del comando letterale. Ad es.: **>**

```
        :normal gg=G
```

Qeusto comando salta alla prima linea e formatta tutte le linee usando l'operatore "=".
   Per far funzionare ":normal" con un'espressione, combinatelo con ":execute".
Ad es.: **>**

```
        :execute "normal " . comandi_normali
```

La variabile "comandi_normali" deve contenere comandi del Normal mode.
   Assicuratevi che l'argomento di ":normal" sia un comando completo. Altrimenti Vim raggiungerà la fine dell'argomento, ma non eseguirà il comando. Ad es. se voi iniziate in Insert mode, dovete anche uscire da Insert mode. Ad esempio: **>**

```
        :execute "normal Inuovo testo \<Esc>"
```

Ciò inserisce "nuovo testo " nella linea corrente. Si noti l'uso del tasto speciale "\<Esc>". Ciò evita di dover immettere un vero carattere <Esc> nel vostro script.

==
41.6 Utilizzo funzioni

Vim definisce parecchie funzioni e fornisce un grande numero di funzionalità in questo modo. Qualche esempio verrà dato in questa sezione. Potete trovare qui la lista completa: |functions|.

Una funzione si invoca col comando ":call". I parametri vengono passati alla funzione fra parentesi, separati da virgole. Ad es.: **>**

```
        :call search("Data: ", "W")
```

Questo invoca la funzione search(), con argomenti "Date: " e "W". La funzione search() usa il proprio primo argomento come una espressione da ricercare e il secondo come un indicatore. L'indicatore "W" significa che la ricerca non aggiri la fine del file.

Una funzione può essere invocata all'interno di una espressione. Ad es.: **>**

```
        :let line = getline(".")
        :let repl = substitute(line, '\a', "*", "g")
        :call setline(".", repl)
```

La funzione getline() ottiene una linea dal file corrente. Il suo argomento è una specificazione del numero di linea. In questo caso viene usato "." che significa la linea in cui è posizionato il cursore.
   La funzione substitute() fa qualcosa di simile al comando ":substitute". Il primo argomento è la stringa in cui effettuare la sostituzione. Il

secondo è l'espressione da cercare, il terzo argomento la stringa con cui rimpiazzarla. Gli ultimi argomenti, per finire, sono indicatori.
   La funzione setline() imposta la linea, specificata dal primo argomento, ad una nuova stringa, il secondo argomento. In questo esempio, la linea sotto il cursore viene rimpiazzata con il risultato della funzione substitute(). Così l'effetto dei tre comandi è uguale a: **>**

```
:substitute/\a/*/g
```

L'uso delle funzioni si fa più interessante quando fate più lavoro prima e dopo la invocazione di substitute().

FUNZIONI *function-list*

Ci sono parecchie funzioni. Le citiamo qui sotto, raggruppate secondo il loro impiego. Potete trovare una lista in ordine alfabetico qui: |functions|. Usate CTRL-] sul nome di funzione per saltare ad una documentazione più dettagliata su di essa.

Manipolazione di stringhe:

char2nr()	ottiene il valore ASCII di un carattere
nr2char()	ottiene un carattere dal suo valore ASCII
escape()	premette '\' a caratteri in una stringa
strtrans()	modifica una stringa rendendola stampabile
tolower()	passa una stringa a caratteri minuscoli
toupper()	passa una stringa a caratteri maiuscoli
match()	si posiziona ad una espressione in una stringa
matchend()	si posiziona a fine espressione in una stringa
matchstr()	trova una espressione in una stringa
stridx()	primo indice ad una stringa in una stringa
strridx()	ultimo indice ad una stringa in una stringa
strlen()	lunghezza di una stringa
substitute()	sostituisce una espressione in una stringa
submatch()	ottiene un dato valore in una ":substitute"
strpart()	ottiene una parte di una stringa
expand()	espande caratteri speciali
type()	restituisce il tipo di una variabile
iconv()	converte testo da una codifica a un'altra

Modificare il testo nel buffer corrente:

byte2line()	dice a che linea si trova un certo byte
line2byte()	dice posizione in byte di una data linea
col()	numero di colonna del cursore o marcatore
virtcol()	colonma sullo schermo del cursore o marcatore
line()	numero di linea del cursore o marcatore
wincol()	numero colonna del cursore nella finestra
winline()	numero linea del cursore nella finestra
cursor()	posiziona il cursore a linea/colonna
getline()	ottiene una linea dal buffer
setline()	sostituisce una linea nel buffer
append()	aggiungere {stringa} sotto linea {numero}
indent()	fa rientrare una data linea
cindent()	fa rientrare una data linea (stile C)
lispindent()	fa rientrare una data linea (stile Lisp)
nextnonblank()	trova la prossima linea non vuota
prevnonblank()	trova la linea non vuota precedente
search()	trova una espressione nel testo
searchpair()	trova l'altro capo, ad es. di un "if"

Funzioni di sistema e manipolazione di file:

browse()	visualizza un file
glob()	valorizza espressioni regolari (wildcard)
globpath()	valorizza espressione lista-di-directory
resolve()	trova nome file dato puntatore (shortcut)
fnamemodify()	modifica nome file
executable()	controlla esistenza programma eseguibile
filereadable()	controlla se possibile leggere un file
filewritable()	controlla se possibile scrivere un file
isdirectory()	controlla se una data directory esiste
getcwd()	ottiene directory di lavoro corrente
getfsize()	ottiene lunghezza di un file
getftime()	ottiene data ultima modifica di un file
localtime()	ottiene la data corrente
strftime()	converte una data in formato stampabile
tempname()	ottiene nome di un file temporaneo
delete()	cancella un file
rename()	rinomina un file

```
        system()                ottiene codice ritorno da comando shell
        hostname()              ottiene nome del computer

Buffer, finestre e lista argomenti:
        argc()                  numero argomenti nella lista argomenti
        argidx()                posizionamento corrente nella lista argomenti
        argv()                  ottiene un argomento dalla lista argomenti
        bufexists()             controlla se un buffer esiste
        buflisted()             controlla se un buffer esiste ed è listato
        bufloaded()             controlla se un buffer esiste ed è caricato
        bufname()               ottiene nome di un dato buffer
        bufnr()                 ottiene numero buffer di un dato buffer
        winnr()                 ottiene numero finestra di finestra corrente
        bufwinnr()              ottiene numero finestra di un dato buffer
        winbufnr()              ottiene numero buffer di una data finestra
        getbufvar()             ottiene valore di variabile da un dato buffer
        setbufvar()             imposta variabile in un dato buffer
        getwinvar()             ottiene valore variabile da una data finestra
        setwinvar()             imposta variabile in una data finestra

Piegature:
        foldclosed()    controlla se piegatura chiusa in una linea data
        foldclosedend()         come foldclosed() ma restituisce ultima linea
        foldlevel()             controlla livello piegatura in una linea data
        foldtext()              specifiche linea che indica piegatura chiusa

Evidenziazione sintattica:
        hlexists()              controlla se esiste gruppo evidenziazione
        hlID()                  ottiene ID di un gruppo evidenziazione
        synID()                 ottiene ID sintattico ad una data posizione
        synIDattr()             ottiene un dato attributo di un ID sintattico
        synIDtrans()            ottiene traduzione di un ID sintattico

Storia:
        histadd()               aggiunge un elemento a una storia
        histdel()               toglie un elemento a una storia
        histget()               ottiene un elemento da una storia
        histnr()                ottiene indice più alto di una storia

Interazione:
        confirm()               chiede all'utente di fare una scelta
        getchar()               ottiene un carattere dall'utente
        getcharmod()            ottiene modificatori ultimo carattere immesso
        input()                 ottiene una linea dall'utente
        inputsecret()           ottiene una linea dall'utente senza mostrarla
        inputdialog()           ottiene una linea dall'utente usando GUI
        inputresave             salva caratteri immessi e pulisce
        inputrestore()          ripristina caratteri immessi

Vim server:
        serverlist()            restituisce lista nomi server
        remote_send()           invia caratteri comando a server Vim
        remote_expr()           valuta una espressione in un server Vim
        server2client()         invia risposta a un cliente di un server Vim
        remote_peek()           controlla se esiste risposta da un server Vim
        remote_read()           legge risposta da Vim server
        foreground()            sposta finestra Vim in primo piano
        remote_foreground()     sposta finestra server Vim in primo piano

Varie:
        mode()                  ottiene modalità di edit corrente
        visualmode()            dice ultima modalità visuale utilizzata
        hasmapto()              controlla se esiste mappatura
        mapcheck()              controlla se esiste mappatura corrispondente
        maparg()                ottiene valore di una mappatura
        exists()                controlla se variabile, funzione, etc. esiste
        has()                   controlla se Vim supporta una estensione data
        cscope_connection()     controlla se esiste una connessione cscope
        did_filetype()          controlla se già usato autocomando tipo_file
        eventhandler()          controlla se invocato da un gestore eventi
        getwinposx()            ottiene posizione X della finestra GUI Vim
        getwinposy()            ottiene posizione Y della finestra GUI Vim
        winheight()             ottiene altezza di una data finestra
        winwidth()              ottiene larghezza di una data finestra
        libcall()               richiama una funzione in una libreria esterna
        libcallnr()             come sopra, la funzione restituisce un numero
        getreg()                ottiene contenuto di un registro
        getregtype()            ottiene tipo di un registro
```

```
        setreg()                imposta contenuto e tipo di un registro
```

==

41.7 Definizione funzioni

Vim vi consente di definire le proprie funzioni. La dichiarazione di una funzione comincia così: **>**

```
        :function {nome}({var1}, {var2}, ...)
        :  {corpo_della_funzione}
        :endfunction
<
```

        Note:
        I nomi di funzione devono cominciare con una lettera maiuscola.

Definiamo una breve funzione che restituisce il minore di due numeri. Comincia con questa linea:

```
        :function Min(num1, num2)
```

Ciò dice a Vim che la funzione si chiama "Min" ed ha due argomenti: "num1" e "num2".
   La prima cosa di cui avete bisogno è di tentare di vedere quale numero sia il più piccolo:
   >

```
        :  if a:num1 < a:num2
```

Il prefisso speciale "a:" dice a Vim che la variabile è un argomento della funzione. Assegniamo alla variabile "minore", il valore del numero più piccolo: **>**

```
        :  if a:num1 < a:num2
        :    let minore = a:num1
        :  else
        :    let minore = a:num2
        :  endif
```

La variabile "minore" è una variabile locale. Variabili usate all'interno di una funzione sono locali se non hanno prefissi come "g:", "a:", o "s:".

        Note:
        Per accedere ad una variabile globale dall'interno di una funzione dovete premettere "g:" alla variabile stessa. Così "g:contatore" all'interno di una funzione viene usato per la variabile globale "contatore", e "contatore" è un'altra variabile, locale per la funzione.

Usate il comando ":return" per restituire il numero più piccolo all'utente. Infine, indicate la fine della funzione: **>**

```
        :  return minore
        :endfunction
```

La definizione completa della funzione è la seguente: **>**

```
        :function Min(num1, num2)
        :  if a:num1 < a:num2
        :    let minore = a:num1
        :  else
        :    let minore = a:num2
        :  endif
        :  return minore
        :endfunction
```

Una funzione definita dall'utente viene invocata esattamente nello stesso modo con cui vengono invocate le funzioni predefinite di Vim. Solo il nome è diverso. La funzione Min si può usare così: **>**

```
        :echo Min(5, 8)
```

Solo adesso la funzione verrà eseguita, e le linee saranno interpretate da Vim. Se ci sono errori, come usare una variabile o una funzione indefinite, si otterrà un messaggio di errore. Quando si definisce una funzione, errori di questo tipo non vengono rilevati.

Quando una funzione trova ":endfunction" o se ":return" viene usato senza un argomento, la funzione restituisce zero.

Per ridefinire una funzione già esistente, usate il "!" nel comando ":function": >

```
:function!  Min(num1, num2, num3)
```

USARE UN INTERVALLO

Al comando ":call" si può dare un intervallo di linee su cui agire. Ciò può avere uno o due significati. Quando una funzione è stata definita con la parola chiave "range", terrà conto dell'intervallo di linee stesso.
Alla funzione verranno passate le variabili "a:firstline" and "a:lastline". Queste avranno i numeri di linea dall'intervallo in cui la funzione è stata chiamata. Ad es: >

```
:function Conta_parole() range
:  let n = a:firstline
:  let quante = 0
:  while n <= a:lastline
:    let quante = quante + Wordcount(getline(n))
:    let n = n + 1
:  endwhile
:  echo "trovate " . quante . " parole"
:endfunction
```

Potete chiamare questa funzione con: >

```
:10,30call Conta_parole()
```

Verrà eseguita una volta sola, e scriverà il numero di parole.
L'altro modo per usare un intervallo di linee consiste nel definire una funzione senza la parola chiave "range". La funzione sarà chiamata una volta per ogni linea nell'intervallo, con il cursore posizionato su quella linea.
Ad es.: >

```
:function  Numero()
:  echo "La linea numero " . line(".") . " contiene: " . getline(".")
:endfunction
```

Se chiamate questa funzione con: >

```
:10,15call Numero()
```

la funzione verrà invocata sei volte.

NUMERO VARIABILE DI ARGOMENTI

Vim vi consente di definire funzione che abbia un numero variabile di argomenti. Il comando seguente, ad es., definisce una funzione che deve avere almeno 1 argomento (inizio) e può avere più di 20 argomenti in aggiunta al primo: >

```
:function Mostra(inizio, ...)
```

La variabile "a:1" contiene il primo argomento facoltativo, "a:2" il secondo, e così via. La variabile "a:0" contiene il numero di argomenti aggiuntivi.
Ad es.: >

```
:function Mostra(inizio, ...)
:  echohl Title
:  echo "Mostra è " . a:inizio
:  echohl None
:  let indice = 1
:  while indice <= a:0
:    echo "  Arg " . indice . " è " . a:{index}
:    let indice = indice + 1
:  endwhile
:  echo ""
:endfunction
```

La funzione usa il comando ":echohl" per specificare l'evidenziazione da usare per il successivo comando ":echo" command. ":echohl None" torna alla visualizzazione normale. Il comando ":echon" si comporta come ":echo", ma non scrive il carattere per andare a capo.

LISTA DELLE FUNZIONI

Il comando ":function" lista i nomi e gli argomenti di tutte le funzioni definite dall'utente: >

```
        :function
<       function Show(start, ...) ~
        function GetVimIndent() ~
        function SetSyn(name) ~
```

Per vedere cosa fa una funzione, usate il suo nome come argomento per ":function": >

```
        :function SetSyn
<       1     if &syntax == '' ~
        2       let &syntax = a:name ~
        3     endif ~
           endfunction ~
```

DEBUGGING

Il numero linea è utile quando ricevete un messaggio di errore o in fase di debug. Si veda |debug-scripts| a proposito del debug.
Potete anche impostare l'opzione 'verbose' a 12 o più per vedere tutte le chiamate di funzione. Impostatelo a 15 o più per visualizzare ogni linea eseguita.

CANCELLARE UNA FUNZIONE

Per cancellare la funzione Mostra(): >

```
        :delfunction Mostra
```

Se la funzione non esiste, viene visualizzato un messaggio di errore.

==
41.8 Note varie

Cominciamo con un esempio: >

```
        :try
        :   read ~/templates/pascal.tmpl
        :catch /E484:/
        :   echo "Spiacente, il file_modello Pascal non è disponibile."
        :endtry
```

Il comando ":read" non può essere eseguito se il file non esiste. Invece di lasciar visualizzare un messaggio di errore, questo codice cattura l'errore e dà all'utente un messaggio più comprensibile.

Per i comandi compresi fra ":try" e ":endtry" gli errori sono trasformati in eccezioni. Una eccezione è una stringa di caratteri. Nel caso di un errore la stringa contiene il messaggio di errore. E ogni messaggio di errore ha un identificativo di messaggio. In questo caso, l'errore che intercettiamo contiene l'identificativo "E484:". Questo identificativo è una costante (il testo del messaggio può variare, ad es. può essere tradotto).

Quando il comando ":read" genera un altro errore, la stringa "E484:" non sarà contenuta nel messaggio di errore. Ragion per cui questa eccezione non sarà intercettata, e quindi riceveremo il relativo messaggio di errore.

Si potrebbe essere tentati di scrivere: >

```
        :try
        :   read ~/templates/pascal.tmpl
        :catch
        :   echo "Spiacente, il file modello Pascal non è disponibile."
        :endtry
```

Così si intercettano tutti gli errori. In questo modo però non si vedono errori "utili" tipo: "E21: Non posso fare modifiche, 'modifiable' è inibito"

Un altro meccanismo utile è il comando ":finally": >

```
        :let tmp = tempname()
        :try
        :   exe ".,$write " . tmp
```

```
        :   exe "!filter " . tmp
        :   .,$delete
        :   exe "$read " . tmp
        :finally
        :   call delete(tmp)
        :endtry
```

Questo script Vim fa passare le linee dal cursore fino alla fine del file attravrso il comando "filter", che ha come argomento un nome di file.
  Sia che il filtro funzioni, sia se qualche errore accade tra ":try" e ":finally" o l'utente cancella l'operazione di filtro premendo CTRL-C, la chiamata "call delete(tmp)" è eseguita sempre. Questo consente di essere sicuri di non lasciarsi dietro file temporanei di lavoro.

Maggiore informazione a riguardo della gestione eccezioni si trova nel manuale: |exception-handling|.

==
41.9 Osservazioni varie

Diamo qui una serie di osservazioni relative agli script Vim. Sono spiegate anche altrove, ma sono qui riunite per convenienza.

Il carattere di fine linea dipende dal sistema operativo. Per Unix si usa un solo carattere <NL>. Per MS-DOS, Windows, OS/2 etc. si usa, <CR><LF>.
  Questo è importante qunado si usano mappature che finiscono con <CR>.
  Si veda |:source_crnl|.

SPAZI BIANCHI

Linee bianche possono essere inserite dappertutto, e vengono ignorate.

Spazi bianchi prima di un testo (spazi e/o TAB) sono sempre ignorati. Gli spazi bianchi fra parametri (ad es. tra il 'set' e 'cpoptions' nell'esempio qui sotto) sono ridotti ad un unico spazio, che serve da separatore.
  Gli spazi bianchi dopo l'ultimo carattere (visibile) possono essere ignorati oppure no, a seconda della situazione. Si veda sotto.

Per un comando ":set" che usa il segno "=" (uguale), tipo: **>**

```
        :set cpoptions    =aABceFst
```

lo spazio bianco subito PRIMA del segno "=" è ignorato. Ma non è consentito alcuno spazio bianco DOPO il segno "="!

Per inserire uno spazio bianco nel valore di una opzione, bisogna farlo precedere da un "\" (barra retroversa) come nell'esempio seguente: **>**

```
        :set tags=il\ mio\ bel\ file
```

Lo stesso esempio scritto come **>**

```
        :set tags=Il mio bel file
```

produrrà un errore, perché viene interpretato come: **>**

```
        :set tags=il
        :set mio
        :set bel
        :set file
```

COMMENTI

Il carattere " (il doppio apice) inizia un commento. A partire da questo carattere il resto della linea è considerato un commento ed è ignorato, tranne che nei comandi che non considerano i commenti, come mostrato negli esempi qui sotto. Un commento può cominciare in qualsiasi posizione della linea.

Bisogna fare attenzione ai commenti per alcuni comandi. Ad es.: **>**

```
        :abbrev dev development         " abbreviazione
        :map <F3> o#include             " inserisci include
        :execute cmd                    " eseguilo
        :!ls *.c                        " lista i files C
```

La abbreviazione 'dev' sarè espansa come 'development   " abbreviazione'.

La mappatura di <F3> sarà costituita dall'intera linea dopo 'o# ....' compreso '" inserisci include'. Il comando "execute" darà un errore. Il comando "!" passerà tutto quel che lo segue allo shell, che darà un errore perché manca un carattere '"' che chiuda il primo '"'.
Non sono permessi commenti dopo i comandi ":map", ":abbreviate", ":execute" e "!" (ci sono alcuni altri comandi con la stessa limitazione).
Per i comandi ":map", ":abbreviate" and ":execute" si può però scrivere: **>**

```
:abbrev dev development|" abbreviazione
:map <F3> o#include|" inserisci include
:execute cmd                    |" eseguilo
```

Il carattere '|' separa un comando da quello successivo. Ed il comando successivo è solo un commento.

Si noti che non c'è alcuno spazio prima del '|' nella abbreviazione e nella mappatura. Per questi comandi, ogni carattere fino a fine linea o a '|' è compreso. Per questo motivo, non sempre è chiaro se sono presenti degli spazi dopo l'ultimo carattere visibile sulla linea; **>**

```
:map <F4> o#include
```

Per evitare questi problemi, potete attivare l'opzione 'list' mentre modificate degli script Vim.

TRABOCCHETTI

Problemi anche maggiori sono presenti nell'esempio seguente: **>**

```
:map ,ab o#include
:unmap ,ab
```

Il comando "unmap" non funzionerà, perché tenta di togliere la mappatura di ",ab " (c'è uno spazio bianco dopo il "b" − NdT). Questa mappatura non esiste. Sarà quindi visualizzato un messaggio di errore, difficile da comprendere, perché lo spazio bianco finale in ":unmap ,ab " non è visibile.

Lo stesso capita quando si usa un commento dopo un comando 'unmap': **>**

```
:unmap ,ab     " commento
```

Qui il commento verrà ignorato. Comunque, Vim tenterà di togliere la mappatura ',ab ', che non esiste. Riscrivetelo come: **>**

```
:unmap ,ab|    " commento
```

RITORNARE DOVE ERAVAMO

Talora volete fare una modifica, e ritornare al punto in cui si trovava il cursore prima della modifica. Anche un ripristino della posizione relativa dello schermo sarebbe gradita, con la stessa linea di prima visualizzata in cima alla finestra.
L'esempio che segue copia la linea sotto il cursore, la mette sopra la prima linea del file, e torna al punto di partenza: **>**

```
map ,p ma"aYHmbgg"aP`bzt`a
```

Spiegazione: **>**

```
ma"aYHmbgg"aP`bzt`a
<       ma                      marcatura "a" di posizione corrente
          "aY                   copia linea corrente nel registro "a"
             Hmb                marcatura "b" su linea in cima alla pagina
                gg              vai alla prima linea del file
                  "aP           mettici sopra la linea nel registro "a"
                      `b        vai alla linea in cima alla pagina
                        zt      posiziona testo nella finestra come prima
                          `a    torna alla posizione in cui era il cursore
```

PER LA DISTRIBUZIONE

Per evitare che i vostri nomi di funzione entrino in conflitto con funzioni che altri hanno preparato, usate questo schema:
− Premettete una vostra stringa ad ogni nume funzione. Io uso spesso una abbreviazione. Ad es., "OW_" è usato per le funzioni Opzioni Window.
− Mettete la definizione delle vostre funzioni in un unico file. Impostate una

variabile globale per indicare che la funzione è stata caricata. Quando volete caricare di nuovo il file, cancellate prima le funzioni. Ad es: **>**

```
" Questa è la funzione XXX

if exists("XXX_caricata")
  delfun XXX_uno
  delfun XXX_due
endif

function XXX_uno(a)
        ... corpo della funzione ...
endfun

function XXX_due(b)
        ... corpo della funzione ...
endfun

let XXX_caricata = 1
```

==========

41.10 Scrivere un plugin *write-plugin*

Potete scrivere uno script Vim in modo tale che chi vuole lo possa usare. Questo si chiama un "plugin". Gli utenti Vim possono copiare il vostro script Vim nella loro directory dei plugin e cominciare ad usare le sue funzioni. Si veda |add-plugin|.

Ci sono di fatto due tipi di plugin:

  global plugin: Per ogni tipo di file.
filetype plugin: Solo per file di un dato tipo.

In questa sezione è spiegato solo il "global plugin". La maggior parte delle cose dette vale anche per i "filetype plugin". Le informazioni relative ai "filetype plugin" sono nella sezione seguente |write-filetype-plugin|.

NOME

innanzitutto dovete scegliere un nome per il vostro plugin. Lo scopo per cui è stato scritto il plugin dovrebbe essere riconoscibile dal nome che ha. E dovrebbe essere poco probabile che qualcun altro scriva un plugin con lo stesso nome, ma che faccia cose differenti. E per favore, limitate il nome a 8 caratteri, per evitare problemi sui vecchi sistemi Windows.

Uno script che corregge errori di battitura potrebbe essere chiamato "typecorr.vim". Lo useremo qui come esempio.

Se il plugin deve poter essere usato da tutti, deve seguire alcune linee guida. Le vediamo una alla volta. L'esempio completo del plugin è riportato alla fine.

CORPO

Cominciamo dal corpo del plugin, le linee che fanno il lavoro vero: **>**

```
 14     iabbrev li il
 15     iabbrev altor altro
 16     iabbrev voule vuole
 17     iabbrev sopratutto
 18             \ soprattutto
 19     let s:contatore = 4
```

La lista vera dovrebbe essere molto più lunga, naturalmente.

La numerazione delle linee è stata aggiunta per uso didattico, non dovete metterla nel vostro file plugin!

INTESTAZIONE

Probabilmente aggiungerete nuove correzioni al vostro plugin, e vi ritroverete con diverse versioni sparse qua e là. E quando distribuirete il vostro file plugin, la gente vorrà sapere chi ha scritto questo meraviglioso plugin e a chi possono essere inviate considerazioni al riguardo. Ragion per cui,

dovreste mettere una intestazione all'inizio del vostro plugin: >

```
  1     " Plugin generale di Vim per correggere errori di battitura
  2     " Ultima Modifica:      2000 Ott 15
  3     " Manutentore:  Bram Moolenaar <Bram@vim.org>
```

A proposito di copyright e licenze: Poiché i plugin sono molto utili, e non ha gran senso restringerne la distribuzione, siete invitati a considerare l'idea di dichiarare il vostro plugin di pubblico dominio, oppure di usare la licenza Vim |license|. Una breve nota al riguardo vicino all'inizio del plugin dovrebbe essere sufficiente. Ad es.: >

```
  4     " Licenza:      Questo file è di pubblico dominio.
```

CONTINUAZIONE LINEA, AL NETTO DA EFFETTI SECONDARI   *use-cpo-save*

Nella linea 18 qui sopra, il meccanismo usato per la continuazione è descritto in |line-continuation|. Utenti che hanno attivato l'opzione 'compatible' avranno del problemi, e otterranno un messaggio di errore. Non possiamo semplicemente cambiare l'opzione 'compatible', perché questo avrebbe molti altri effetti oltre a quello desiderato. Per evitare di toccare 'compatible' imposteremo invece l'opzione 'cpoptions' al suo valore predefinito in Vim e la riporteremo poi al valore che aveva. Questo ci consente di usare le linee di continuazione e nello stesso tempo rende lo script utilizzabile da un numero molto maggiore di utenti. Si fa in questo modo: >

```
 11     let s:salva_cpo = &cpo
 12     set cpo&vim
 ..
 42     let &cpo = s:salva_cpo
```

Prima si memorizza il valore precedente di 'cpoptions' nella variabile s:salva_cpo. Alla fine del plugin questo valore viene ripristinato.

Notate l'uso di una variabile locale allo script |s:var|. Una variabile globale con lo stesso nome potrebbe essere già in uso da qualche altra parte. Usate sempre delle variabili locali allo script per azioni limitate solo all'interno dello script.

NON CARICARE

È possibile che un utente non voglia sempre caricare questo plugin. Oppure l'amministratore del sistema l'ha messo nella directory dei plugin di uso comune a tutti gli utenti, ma un utente vuole usare il suo proprio plugin.
  Quindi all'utente deve essere offerta la possibilità di evitare l'uso di questo specifico plugin. Questo si ottiene così: >

```
  6     if exists("loaded_typecorr")
  7       finish
  8     endif
  9     let loaded_typecorr = 1
```

Si evita così anche che, nel caso lo script sia caricato una seconda volta, si riceva un messaggio di errore perché la funzione erà già definita, e possano presentarsi dei problemi per degli autocomandi aggiunti più di una volta.

MAPPATURE

Per rendere il plugin più interessante, introduciamo una mappatura che aggiunga una correzione per la parola che si trova sotto il cursore. Potremmo scegliere una combinazione di tasti per questa mappatura, ma l'utente magari la usa già per fare qualcosa d'altro. Per permettere all'utente di definire la sequenza di tasti che vuole usare per una mappatura all'interno di un plugin, si può usare l'elemento <Leader>: >

```
 22       map <unique> <Leader>a  <Plug>TypecorrAdd
```

Il comando "<Plug>TypecorrAdd" rende questo possibile, come vedremo dopo.

L'utente può dare alla variabile "mapleader" la combinazione di tasti con cui vuole che cominci questa mappatura. Se l'utente ha impostato: >

```
        let mapleader = "_"
```

la mappatura definirà "_a". Se l'utente non ha fatto nulla, il valore predefinito (che è "\", barra retroversa) sarà utilizzato. In quel caso la mappatura definita sarà "\a".

Note Poiché <unique> è usato, sarà inviato un messaggio di errore nel caso la stessa mappatura sia già stata definita. |:map-<unique>|

E se l'utente volesse definire una sua sequenza di tasti? Glielo si può permettere con questo meccanismo: >

```
 21     if !hasmapto('<Plug>TypecorrAdd')
 22       map <unique> <Leader>a  <Plug>TypecorrAdd
 23     endif
```

Qui controlliamo se una mappatura per "<Plug>TypecorrAdd" esiste già, e definiamo la mappatura con "<Leader>a" solo se non esiste. L'utente ha quindi la possibilità di mettere nel suo file vimrc: >

```
        map ,c  <Plug>TypecorrAdd
```

E quindi la sequenza di tasti mappata sarà ",c" invece che "_a" o "\a".

PARTI

Se uno script si allunga, potrebbe essere desiderabile dividere il lavoro in parti. Potete usare sia delle funzioni che delle mappature per questo. Ma non volete che queste funzioni e mappature interferiscano con altre, contenute in altri script. Per esempio, potreste definire una funzione Add(), ma un altro script potrebbe tentare di definire la stessa funzione. Per evitare questo, definiamo la funzione come locale, esistente solo all'interno del nostro script, premettendo "s:" al suo nome.

Definiano una funzione che aggiunge una nuova correzione di errore di battitura: >

```
 30     function s:Add(errato, corretto)
 31       let buono = input("Battere correzione per " . a:errato . ": ")
 32       exe ":iabbrev " . a:errato . " " . buono
 ..
 36     endfunction
```

Ora possiamo richiamare la funzione s:Add() dall'interno dello script. Se un altro script definisce ancora s:Add(), esisterà solo dentro quello script, e può essere chiamata solo daqlla script in cui è stata definita. Ci può anche essere una funzione globale. Ci può anche essere una funzione globale Add() (senza "s:"), che è distinta dalle altre due.

<SID> si può usare senza mappature. Genera un ID dello script, che identifica lo script in cui si trova. Nel nostro plugin di correzione errori di battitura lo usiamo in questo modo: >

```
 24     noremap <unique> <script> <Plug>TypecorrAdd  <SID>Add
 ..
 28     noremap <SID>Add  :call <SID>Add(expand("<cword>"), 1)<CR>
```

Ovvero, quando un utente batte "\a", viene richiamata questa sequenza: >

```
        \a  ->  <Plug>TypecorrAdd  ->  <SID>Add  ->  :call <SID>Add()
```

Se un altro script mappasse <SID>Add, un altro script ID sarebbe generato, e quindi verrebbe definito un'altra mappatura.

Note Usiamo qui <SID>Add() invece che s:Add(). Questo perché la mappatura viene immessa dall'utente, ossia dall'esterno dello script. Il <SID> si traduce nell'ID dello script, in modo che Vim sappia in che script cercare la funzione Add().

La cosa è un po' complicata, ma serve per far sì che il plugin funzioni assieme ad altri plugin. La regola fondamentale è che voi usiate <SID>Add() nelle mappature e s:Add() da altre parti (nello script stesso, in autocomandi o in comandi utente).

Possiamo anche rendere disponibile la mappatura in un menu: >

```
 26     noremenu <script> Plugin.Aggiungi\ Correzione      <SID>Add
```

Il menu "Plugin" è raccomandato per aggiungere elementi di menu a plugin. In

questo caso c'è solo un elemento. Se si aggiungono più elementi, la creazione di un sottomenu è auspicabile. Ad esempio, "Plugin.CVS" potrebbe essere usato per un plugin che offre le operazioni CVS "Plugin.CVS.checkin", "Plugin.CVS.checkout", etc. [CVS è un software per gestire sorgenti, "checkin" inserisce un sorgente, "checkout" lo preleva per aggiornarlo − NdT]

Note Nella linea 28 ":noremap" è usato per evitare che qualsiasi altra mappatura provochi problemi. Qualcuno potrebbe aver ridefinito ":call", ad esempio. Nella linea 24 usiamo anche ":noremap", ma vogliamo che "<SID>Add" sia rimappato. Per questo "<script>" è usato qui. Questo permette mappature che valgono solo all'interno dello script. |:map−<script>| La stessa cosa è fatta nella linea 26 per ":noremenu". |:menu−<script>|

<SID> E <Plug> *using−<Plug>*

Sia <SID> che <Plug> sono usati per evitare che mappature di combinazioni di tasti entrino in conflitto con altri che devono essere usati solo da altre mappature. Note Questa è la differenza fra usare <SID> e <Plug>:

<Plug> è visibile all'esterno dello script. È usato per mappature per cui l'utente voglia utilizzare una particolare sequenza di tasti. <Plug> è un codice speciale che un tasto della tastiera non è in grado di produrre.
Per rendere molto difficile che altri plugin usino la stessa sequenza di caratteri, usate questa struttura:
<Plug> nome_script nome_mappatura
Nel nostro esempio il nome_script è "Typecorr" e il nome_mappatura è "Add".
Il risultato è "<Plug>TypecorrAdd". Solo il primo carattere del nome_script e del nome_mappatura deve essere maiuscolo, in modo da poter capire dove comincia il nome_mappatura.

<SID> è l'ID dello script, un identificatore unico a quello script. Internamente Vim traduce <SID> in "<SNR>123_", dove "123" può essere un numero qualsiasi. Per cui una funzione "<SID>Add()" avrà come nome "<SNR>11_Add()" in uno script, e "<SNR>22_Add()" in un altro. Potete vederlo se usate il comando ":function" per ottenere una lista di funzioni. La traduzione di <SID> nelle mappature è esattamente la stessa, ed è questo il modo con cui potete invocare una funzione locale ad uno script da una mappatura.

COMANDI UTENTE

Aggiungiamo ora un comando utente per aggiungere una correzione: >

```
 38     if !exists(":Correzione")
 39       command −nargs=1  Correzione  :call s:Add(<q−args>, 0)
 40     endif
```

Il comando utente viene definito solo se non esiste già un comando con lo stesso nome. Altrimenti otterremmo un messaggio di errore. Sostituire il comando utente già esistente col nostro, scrivendo ":command!" non è una buona idea, in quanto l'utente si domanderebbe perché il comando che LUI ha definito non funziona. |:command|

VARIABILI NEGLI SCRIPT

Quando una variabile ha un nome che comincia con "s:" è una variabile dello script. Si può solo usare all'interno di uno script, Non esiste all'esterno dello script. Questo consente di evitare problemi quando si utilizza lo stesso nome di variabile in script differenti. La variabile sarà disponibile per tutto il tempo in cui si usa Vim. E la stessa variabile verrà riutilizzata se si ricarica lo stesso script una seconda volta. |s:var|

Il bello è che queste variabili possono anche essere usate in funzioni, autocomandi e comandi utente che sono definiti nello script. Nel nostro esempio di script possiamo aggiungere alcune linee per contare il numero di correzioni: >

```
 19     let s:contatore = 4
 ..
 30     function s:Add(from, correct)
 ..
 34       let s:contatore = s:contatore + 1
 35       echo s:contatore . " correzioni finora"
```

```
 36     endfunction
```

Dapprima s:contatore è inizializzato a 4 nello script stesso. Quando, più tardi, la funzione s:Add() viene invocata, incrementa s:count. Non importa da DOVE la funzione è stata chiamata, poiché, essendo stata definita nello script, userà solo variabili interne a questo script.

IL RISULTATO

Questo è lo script completo risultante: **>**

```
  1     " Plugin generale di Vim per correggere errori di battitura
  2     " Ultima Modifica:      2000 Ott 15
  3     " Manutentore:  Bram Moolenaar <Bram@vim.org>
  4     " Licenza:      Questo file è di pubblico dominio.
  5
  6     if exists("loaded_typecorr")
  7       finish
  8     endif
  9     let loaded_typecorr = 1
 10
 11     let s:salva_cpo = &cpo
 12     set cpo&vim
 13
 14     iabbrev li il
 15     iabbrev altor altro
 16     iabbrev voule vuole
 17     iabbrev sopratutto
 18             \ soprattutto
 19     let s:contatore = 4
 20
 21     if !hasmapto('<Plug>TypecorrAdd')
 22       map <unique> <Leader>a  <Plug>TypecorrAdd
 23     endif
 24     noremap <unique> <script> <Plug>TypecorrAdd  <SID>Add
 25
 26     noremenu <script> Plugin.Aggiungi\ Correzione      <SID>Add
 27
 28     noremap <SID>Add  :call <SID>Add(expand("<cword>"), 1)<CR>
 29
 30     function s:Add(errato, corretto)
 31       let buono = input("Battere correzione per " . a:errato . ": ")
 32       exe ":iabbrev " . a:errato . " " . buono
 33       if a:corretto | exe "normal viws\<C-R>\" \b\e" | endif
 34       let s:contatore = s:contatore + 1
 35       echo s:contatore . " correzioni finora"
 36     endfunction
 37
 38     if !exists(":Correzione")
 39       command -nargs=1  Correzione  :call s:Add(<q-args>, 0)
 40     endif
 41
 42     let &cpo = s:salva_cpo
```

La linea 33 non è stata ancora spiegata. Applica la nuova correzione alla parola sotto il cursore. Il comando |:normal| è usato per fare uso della nuova abbreviazione. Note mappature e abbreviazioni sono espanse qui, anche se la funzione è stata invocata da una mappatura definita come ":noremap".

L'uso di "unix" come opzione 'fileformat' è raccomandato. Gli script Vim in questo modo funzioneranno anche in tutti gli altri ambienti software. Script che abbiano 'fileformat' impostato a "dos" non funzionano sotto Unix.
  Si veda anche |:source_crnl|. Per essere sicuri di usarlo, scrivete, prima di scrivere il file che contiene lo script: **>**

```
        :set fileformat=unix
```

DOCUMENTAZIONE *write-local-help*

É una buona idea preparare anche un po' di documentazione per il vostro plugin. Questo è ancora più necessario quando il suo comportamente può essere personalizzato dall'utente. Si veda |add-local-help| per come aggiungere la vostra documentazione a quella di Vim.

Qui vedete un semplice esempio di file di aiuto, di nome "typecorr.txt": **>**

```
 1     *typecorr.txt*  Plugin per correzione di errori di battitura.
 2
 3     Se vi capita di fare errori di battitura, questo plugin li correggerà
 4     automaticamente.
 5
 6     Al momento le correzioni sono poche.  Potete aggiungerne, se volete.
 7
 8     Mappatura:
 9     <Leader>a   o   <Plug>TypecorrAdd
10             Aggiunge una correzione per la parola sotto il cursore.
11
12     Comandi:
13     :Correzione {parola}
14             Aggiunge una correzione per {parola}.
15
16                                                      *typecorr-settings*
17     Questo plugin non ha alcuna impostazione particolare.
```

La prima linea è l'unica per la quale il formato è importante. Sarà questa ad essere estratta dal file di aiuto per essere inserita nella sezione "LOCAL ADDITIONS" del file help.txt |local-additions|. Il primo "*" deve essere nella prima colonna della prima linea di testo. Dopo aver aggiunto il vostro file di aiuto date il comando ":help" per controllare che le linee in questione siano correttamente allineate.

Potete aggiungere altre tag (racchiuse fra **) nel vostro file di aiuto. Ma state attenti ad evitare nomi di tag di help già esistenti. Potreste inserire il nome del vostro plugin nei loro nome, come "typecorr-settings" nel nostro esempio.

L'uso di puntatori ad altre parti del file di help (racchiusi fra ||) è auspicabile. Questo facilita l'utente nel reperimento dell'aiuto a loro associato.

DETERMINAZIONE DEL TIPO FILE *plugin-filetype*

Se il tipo del vostro file non è già determinato da Vim, dovreste creare un piccolo script che lo determini in un file a parte. Di solito lo script consiste in un autocomando che imposta il tipo file quando il nome del file corrisponde a un dato modello.
Ad es.: **>**

```
        au BufNewFile,BufRead *.foo                     set filetype=foofoo
```

Scrivete il file contenente quest'unica linea come "ftdetect/foofoo.vim" nella prima directory che compare in 'runtimepath'. In Unix questa sarebbe "~/.vim/ftdetect/foofoo.vim". La convenzione è di usare il nome del tipo file come nome dello script.

I controlli possono essere molto più sofisticati, se volete, come ad es. una ispezione del contenuto del file per riconoscere il linguaggio.
SI veda anche |new-filetype|.

SOMMARIO *plugin-special*

Lista delle particolarità che contraddistinguono un plugin:

s:name	variabili interne allo script.
<SID>	Script-ID, usato per mappature e funczioni interne allo script.
hasmapto()	Funzione per accertare se l'utente ha già definito mappature per funzionalità contenute nello script.
<Leader>	Valore di "mapleader", che l'utente definisce come sequenza di tasti con cui iniziano le mappature del plugin.
:map <unique>	Emette un messaggio se una mappatura esiste già.
:noremap <script>	Usa solo mappature interne allo script, non mappature valide globailmente.
exists(":Cmd")	Controlla se un comando utente esiste già.

==
ftplugin
41.11 Scrivere un plugin per un tipo_file *write-filetype-plugin*

A plugin per un tipo_file è simile a un plugin globale, ma imposta opzioni e definisce mappature solo per il buffer corrente. Si veda |add-filetype-plugin| per informazioni sull'uso di questo tipo di plugin.

Leggete prima la sezione precedente sui plugin globali |41.10|. Tutto quel che si dice lì vale anche per i plugin per un tipo_file. Ci sono alcune regole in più, che sono spiegate qui di seguito. La caratteristica fondamentale è che un plugin per un tipo_file dovrebbe avere effetto solo sul buffer corrente.

DISABILITAZIONE

Se state scrivendo un plugin per un tipo_file che potrà essere usato da molta gente, dovete concedere loro la possibilità di evitare di usarlo. Mettete questo in testa al plugin: **>**

```
" Usa solo se non esiste già un latro plugin attivo
if exists("b:did_ftplugin")
  finish
endif
let b:did_ftplugin = 1
```

Occorre usare questa tecnica per evitare che lo stesso plugin sia eseguito due volte per lo stesso buffer (succede quando si usa un comando ":edit" senza fornirgli alcun argomento).

A questo punto gli utenti possono diabilitare completamente il plugin predefinito creando un plugin per un tipo_file composto solo da questa linea:

```
let b:did_ftplugin = 1
```

Per ottenere questo, la directory che contiene il vostro plugin per un tipo_file deve venire prima della directory $VIMRUNTIME nella list di directory contenuta nella opzione 'runtimepath'!

Se volete usare il plugin predefinito, ma cambiare una delle impostazioni, potete inserire l'impostazione da cambiare in uno script: **>**

```
setlocal textwidth=70
```

Poi lo scrivete nella directory "after" ["dopo"], così che venga eseguito DOPO il plugin di tipo_file predefinito (in questo caso "vim.vim").
Si veda |after-directory|. In Unix questa directory è: "~/.vim/after/ftplugin/vim.vim". Note Il plugin predefinito avrà impostato "b:did_ftplugin", ma la cosa viene qui ignorata.

OPZIONI

Per far sì che il plugin per un tipo_file agisca solo sul buffer corrente usate il comando:

```
:setlocal
```

per impostare le opzioni. Ed impostate solo le opzioni che agiscono solo sul buffer corrente (si veda l'aiuto relativo a ogni singola opzione per controllare quali lo facciano). Quando si usa |:setlocal| per opzioni globali o per opzioni che riguardano una particolare finestra, il valore potrebbe cambiare in numerosi buffer, e non è questo l'obiettivo che si dovrebbe avere in un plugin per un tipo_file.

Quando una opzione ha un valore che è una lista di indicatori o elementi, potete utilizzare la sintassi "+=" e "-=" per non modificare i valori già impostati. Tenete presente che l'utente possa aver già cambiato il valore di una opzione. Ripristinare il valore predefinito e poi cambiarlo come serve è spesso una buona idea. Ad es.: **>**

```
:setlocal formatoptions& formatoptions+=ro
```

MAPPATURE

Per far sì che le mappature siano applicate solo nel buffer corrente usate il

comando: >

```
:map <buffer>
```

Questo va combinato con la mappatura in due passi spiegata più sopra.
  Ecco un esempio di come si definisce una funzionalità in un plugin per un tipo_file: >

```
if !hasmapto('<Plug>JavaImport')
  map <buffer> <unique> <LocalLeader>i <Plug>JavaImport
endif
noremap <buffer> <unique> <Plug>JavaImport oimport ""<Left><Esc>
```

|hasmapto()| è usato per controllare se l'utente ha già definito una mappatura per <Plug>JavaImport. In caso negativo, il plugin per un tipo_file definisce la mappatura predefinita. Questa inizia con |<LocalLeader>|, che permette all'utente di scegliere il tasto (o i tasti) con cui vuole che inizino le mappature dei plugin per un tipo_file. Il valore predefinito è la barra rovesciata ("\").
  "<unique>" è usato per mandare un messaggio di errore se la mappatura esiste già o entra in conflitto con una mappatura esistente.
  |:noremap| è usato per evitare che ogni altra mappatura che l'utente abbia definito possa interferire. Potreste voler usare ":noremap <script>" per far sì che si possano ri-mappare mappature definite in questo script, che comincino con <SID>.

L'utente deve avere la possibilità di disabilitare una mappatura in un plugin per un tipo_file, senza disabilitarlo completamente. Ecco un esempio di come si può fare questo per il tipo_file "mail" (messaggi di posta elettronica): >

```
" Aggiungi mappature, a meno che l'utente preferisca evitarlo.
if !exists("no_plugin_maps") && !exists("no_mail_maps")
  " Citate un testo inserendo "> "
  if !hasmapto('<Plug>MailQuote')
    vmap <buffer> <LocalLeader>q <Plug>MailQuote
    nmap <buffer> <LocalLeader>q <Plug>MailQuote
  endif
  vnoremap <buffer> <Plug>MailQuote :s/^/> /<CR>
  nnoremap <buffer> <Plug>MailQuote :.,$s/^/> /<CR>
endif
```

Due variabili globali sono usate:
no_plugin_maps   disabilita mappature per ogni plugin di un tipo_file
no_mail_maps   disabilita mappature per un dato tipo_file

COMANDI UTENTE

Per aggiungere un comando utente per un dato tipo_file, in modo che possa solo essere usato in un buffer, usate l'argomento "-buffer" di |:command|.
  Ad es.: >

```
:command -buffer  Make  make %:r.s
```

VARIABILI

Un plugin per un tipo_file sarà utilizzato in ogni buffer contenente un file di quel tipo. La variabile interna di script |s:var| sarà la stessa per tutti i buffer in cui il plugin viene utilizzato. Usate variabili interne al buffer |b:var| se volete che una variabile sia usata sono all'interno di un unico buffer.

FUNZIONI

Quando si definisce una funzione, questa va definita soltanto una volta. Ma il plugin di tipo_file verrà invocato ogni volta che verrà aperto un file con quel dato tipo file. La tecnica seguente fa sì che la funzione venga definita una volta sola. >

```
:if !exists("*s:Func")
:  function s:Func(arg)
:    ...
:  endfunction
:endif
```
<

ANNULLARE *undo_ftplugin*

Quando l'utente imposta ":setfiletype xyz" l'effetto del tipo_file "precedente" dovrebbe essere annullato. Impostate la variabile b:undo_ftplugin con i comandi che annulleranno le impostazioni di un vostro plugin per un tipo_file. Ad es.: >

```
let b:undo_ftplugin = "setlocal fo< com< tw< commentstring<"
        \ . "| unlet b:match_ignorecase b:match_words b:match_skip"
```

L'uso di ":setlocal" con "<" subito dopo il nome dell'opzione reimposta per quell'opzione il suo valore globale. Nella maggior parte dei casi è questa la maniera migliore di ripristinare i valori "precedenti" delle opzioni.

Occorre togliere la indicazione "C" da 'cpoptions' per consentire la continuazione della linea, come detto prima |use-cpo-save|.

NOME FILE

Il tipo_file deve far parte del nome del plugin |ftplugin-name|. Usate una di queste tre forme:

```
.../ftplugin/roba.vim
.../ftplugin/roba_mia.vim
.../ftplugin/roba/tua.vim
```

"roba" è il tipo_file, "mia" e "tua" sono nomi a piacere.

SOMMARIO *ftplugin-special*

Lista delle particolarità che contraddistinguono un plugin di tipo_file:

<LocalLeader>	Valore di "maplocalleader", che l'utente definisce come la combinazione di tasti con cui iniziano le mappatur per un plugin di tipo_file.
:map <buffer>	Definisce una mappatura che vale solo all'interno del buffer.
:noremap <script>	Ri-mappa solo mappature definite in questo script e che cominciano con <SID>.
:setlocal	Imposta una opzione solo per il buffer corrente.
:command -buffer	Definisce un comando utente valido solo in questo buffer.
exists("*s:Func")	Controlla se una funzione è stata già definita.

Si veda anche |plugin-special|, che descrive le proprietà specifiche dei plugin.

==

41.12 Scrivere un plugin per un compilatore *write-compiler-plugin*

Un plugin per un compilatore imposta le opzioni da usare con un dato compilatore. L'utente può invocarlo con il comando |:compiler|. L'uso principale che se ne fa è per impostare le opzioni 'errorformat' e 'makeprg'.

Un esempio è la maniera più semplice di introdurlo. Questo comando modificherà tutti i plugin predefiniti per i compilatori: >

```
:next $VIMRUNTIME/compiler/*.vim
```

Usate |:next| per passare al file di plugin successivo.

La sola particolarità notevole di questi file è un meccanismo che permette a un utente di modificare o di sostituire il file di plugin predefinito. I file predefiniti cominciano con: >

```
:if exists("current_compiler")
:  finish
:endif
:let current_compiler = "mine"
```

Quando scrivete un file per un compilatore e lo mettere nella vostra propria directory di files di esecuzione ("runtime directory") (ad es., ~/.vim/compiler in Unix), voi potete impostare la variabile "current_compiler" in modo che il file predefinito non esegua (in seguito) le impostazioni che contiene.

Quando scrivete un plugin per un compilatore da distribuire con Vim o da mettere in una directory di esecuzione ("untime directory") a livello di sistema, usate il meccanismo delineato più sopra. Se "current_compiler" era già stato impostato da un plugin dell'utente, questo verrà ritenuto valido e non sostituito in alcun modo.

Qaundo scrivete un plugin di compilatore per modificare impostazioni contenute nel plugin predefinito, non controllate la variabile "current_compiler". Un plugin di questo tipo dovrebbe venire eseguito per ultimo, e quindi dovrebbe trovarsi in una directory in fondo alla lista di directory specificata in 'runtimepath'. In Unix il suo nome potrebbe essere ~/.vim/after/compiler.

==

Capitolo seguente: |usr_42.txt| Aggiungere nuovi menù

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

usr_42.txt   Per Vim version 6.2. Ultima modifica: 2002 Ott 08

VIM USER MANUAL - by Bram Moolenaar
Traduzione di questo capitolo: Luca Ferraro

Aggiungere nuovi menù

Da ora sapete che Vim è molto flessibile. Ciò include i menu usati nella GUI. Potete definire voi stessi le opzioni del vostro menu per rendere alcuni comandi più facilmente accessibili. Ciò soltanto per i felici utenti del mouse!

  Capitolo seguente: |usr_43.txt| Utilizzo dei tipi di file
Capitolo precedente: |usr_41.txt| Preparare uno script Vim
             Indice: |usr_toc.txt|

==

42.1 Introduzione

I menù che Vim usa sono definiti nel file "$VIMRUNTIME/menu.vim". Se volete scrivere i vostri menu personali, potreste prima dare un'occhiata a questo file.
   Per definire un elemento del menù, utilizzate il comando ":menu". La forma fondamentale di questo comando è molto semplice: **>**

```
:menu {elemento-menù} {comando}
```

Il {elemento-menù} descrive in quale posizione del menù inserire l'elemento. Un tipico {elemento-menù} è "File.Save", che rappresenta l'elemento "Save" sotto il menù "File". Un punto viene utilizzato per separare i nomi. Un esempio: **>**

```
:menu File.Save  :update<CR>
```

Il comando ":update" salva il file quando esso sia stato modificato.
   Potete aggiungere un altro livello: "Edit.Settings.Shiftwidth" definisce un sotto menù "Settings" nel menù "Edit", con l'elemento "Shiftwidth". Potreste utilizzare anche livelli inferiori. Non usatelo troppo, vi toccherebbe spostare il mouse un bel po' per poter usare così un elemento.
   Il commando ":menu" è molto simile al comando ":map" : il lato sinistro specifica quale elemento verrà lanciato ed il destro definisce i caratteri che verranno eseguiti. {keys} sono caratteri, vengono usati solo come li avevate scritti. Così nell'Insert mode, quando {keys} è puro testo, questo testo viene inserito.

TASTI DI SCELTA RAPIDA

Il carattere ampersand (&), viene utilizzato per indicare un tasto di scelta rapida. Per esempio, potete utilizzare Alt-F per selezionare il menù "File" e quindi premere S per il comando "Save". (L'opzione 'winaltkeys' può peraltro disabilitare questa possibilità!). Comunque il {menu-item} appare come "&File.&Save". I caratteri acceleratori veranno sottolineati nel menù.
   Dovrete fare attenzione che ogni tasto venga usato una sola volta in ciascun menù. Altrimenti non sapreste quale dei due potrebbe venire usato. Vim non vi avvisa di ciò.

PRIORITA'

La definizione corrente dell'elemento di menù File.Save è come segue: **>**

```
:menu 10.340 &File.&Save<Tab>:w  :confirm w<CR>
```

Il numero 10.340 è chiamato il numero di priorità. Questo viene utilizzato dall'editor per decidere in quale posizione inserire l'elemento di menù. Il primo numero (10) indica la posizione sulla barra dei menù. I menù con i valori più bassi vengono posizionati a sinistra, quelli con valori più alti a destra.
   Questi sono le priorità usate per i menù standard:

```
          10    20     40     50      60       70              9999
```

```
+-------------------------------------------------------------+
| File  Edit  Tools  Syntax  Buffers  Window           Help |
+-------------------------------------------------------------+
```

Notate che al menù Help viene dato un numero molto grande, per farlo apparire il più a destra possibile.
   Il secondo numero (340) determina la posizione dell'elemento nel menù a discesa. I numeri più bassi vanno in alto, quelli più alti in fondo. Queste sono le priorità nel menù File:

```
                    +-----------------+
    10.310          |Open...          |
    10.320          |Split-Open...    |
    10.325          |New              |
    10.330          |Close            |
    10.335          |---------------- |
    10.340          |Save             |
    10.350          |Save As...       |
    10.400          |---------------- |
    10.410          |Split Diff with  |
    10.420          |Split Patched By |
    10.500          |---------------- |
    10.510          |Print            |
    10.600          |---------------- |
    10.610          |Save-Exit        |
    10.620          |Exit             |
                    +-----------------+
```

Notate che c'è un intervallo tra i numeri. Qui è dove potete inserire i vostri elementi personali, se lo volete davvero (spesso è meglio lasciare stare i menù standard ed aggiungere un nuovo menù per i vostri elementi personali).
   Quando create un sotto-menù, potete aggiungere un altro ".numero" alla priorità. Così ogni nome nel {menu-item} ha i propri numeri di priorità.

CARATTERI SPECIALI

Il {menu-item} in questo esempio è "&File.&Save<Tab>:w". Ciò evidenzia un punto importante: {menu-item} deve essere una sola parola. Se volete mettere un punto, spazio o tabulatore nel nome usate la notazione <> (<space> e <tab>, ad esempio) od usare l'escape backslash (\). **>**

```
        :menu 10.305 &File.&Do\ It\.\.\. :exit<CR>
```

In questo esempio, l'elemento di menù "Do It..." contiene uno spazio ed il comando è ":exit<CR>".

Il carattere <Tab> nel nome del menù viene usato per separare la parte che definisce il nome del menù da quella che da un'indicazione all'utente. La parte dopo il <Tab> viene visualizzata allineata nel menù. Nel menù File.Save il nome usato è "&File.&Save<Tab>:w". Così il nome del menù è File.Save e l'indicazione è ":w".

SEPARATORI

Le linee di separazione, utilizzare per raggruppare assieme elementi di menù, possono essere definite usando un nome che inizi e finisca in un '-'. Ad esempio "-sep-". Se si usano diversi separatori i nomi debbono essere differenti. Altrimenti i nomi non vanno.
   Il comando da un separatore non verrà mai eseguito, ma voi dovete definirne comunque uno. Un solo due punti va bene. Esempio: **>**

```
        :amenu 20.510 Edit.-sep3- :
```

==
42.2 Comandi dei menù

Potete definire elementi del menù che esistono solo per alcune modalità. Ciò funziona proprio come le variazioni nel comando ":map" :

```
        :menu           Normal, Visual and Operator-pending mode
        :nmenu          Normal mode
        :vmenu          Visual mode
        :omenu          Operator-pending mode
        :menu!          Insert and Command-line mode
        :imenu          Insert mode
```

```
:cmenu          Command-line mode
:amenu          All modes
```

Per evitare che i comandi di un elemento del menù vengano mappati, usate il comando ":noremenu", ":nnoremenu", ":anoremenu", etc.

UTILIZZO DI :AMENU

Il comando ":amenu" è leggermente diverso. Esso assume che le {keys} che voi date debbano essere eseguite in Normal mode. Quando Vim è in Visual od Insert mode quando il menù viene usato, Vim prima deve tornare al Normal mode. ":amenu" inserisce un CTRL-C o un CTRL-O per voi. Per esempio, se usate questo comando: **>**

```
:amenu  90.100 Mine.Find\ Word  *
```

Allora i comandi del menù risultanti saranno:

```
Normal mode:            *
Visual mode:            CTRL-C *
Operator-pending mode:  CTRL-C *
Insert mode:            CTRL-O *
Command-line mode:      CTRL-C *
```

Se siete in modalità Command-line, il CTRL-C abbandonerà il comando sinora scritto. In Visual e Operator-pending mode CTRL-C fermerà la modalità. In Insert mode CTRL-O eseguirà il comando e quindi ritornerà in Insert mode.
CTRL-O funziona per un solo comando. Se avete necessità di usare due o più comandi, inseriteli in una funzione e quindi chiamate questa funzione. Esempio: **>**

```
:amenu  Mine.Next\ File  :call <SID>NextFile()<CR>
:function <SID>NextFile()
:  next
:  1/^Code
:endfunction
```

Questa opzione del menù va al prossimo file nella lista degli argomenti con ":next". Poi cerca la linea che inizia con "Code".
Il <SID> prima del nome della funzione è l'ID dello script. Ciò rende la funzione locale allo script di Vim corrente. Ciò evita problemi quando una funzione con lo stesso nome viene definita in un altro file di script. Vedere |<SID>|.

MENU' SILENTI

Il menù esegue le {keys} come le avete scritte. Per un comando ":" ciò significa che vedrete il comando scritto sulla linea di comando. Se è un comando lungo, allora apparirà il prompt hit-Enter. Ciò potrebbe essere molto fastidioso!
Per evitare ciò rendete muto il menù. Ciò viene fatto con l'argomento <silent>. Ad esempio, prendete al chiamata a NextFile() nell'esempio precedente. Quando usate questo menù vedrete ciò sulla linea di comando:

```
:call <SNR>34_NextFile() ~
```

Per evitare la comparsa del testo, inseriamo "<silent>" come primo argomento: **>**

```
:amenu <silent> Mine.Next\ File :call <SID>NextFile()<CR>
```

Non utilizzate "<silent>" troppo spesso. Non è necessario per comandi brevi. Se realizzate dei menù per qualcun altro, poter vedere il comando eseguito darà un suggerimento su ciò che egli può avere scritto, invece di usare il mouse.

ELENCARE I MENU'

Quando un comando del menù viene usato senza la parte {keys}, esso elenca i menù già definiti. Potete specificare un questo si comporta richiamando il contenuto di un menù già definito. Potete specificare un {menu-item} od una parte di esso, per elencare menù specifici. Esempio: **>**

```
:amenu
```

Ciò elenca tutti i menù. Che è una lista lunga! Meglio specificate il nome

di un menù per ottenere una lista più corta: >

```
:amenu Edit
```

Questo mostra solo gli elementi del menù "Edit" per tutte le modalità. Per mostrare solo uno specifico elemento del menù per l'Insert mode: >

```
:imenu Edit.Undo
```

Attenti a scrivere esattamente il nome giusto. Qui contano maiuscole e minuscole. Ma l'&' per gli acceleratori può venire omesso. Il <Tab> e ciò che viene dopo può ugualmente venir lasciato fuori.

CANCELLARE I MENU'

Per cancellare un menù, si utilizza lo stesso comando usato per elencarli, ma con "menu" cambiato in "unmenu". Così ":menu" diventa ":unmenu", ":nmenu" diventa ":nunmenu", ecc. Per eliminare l'elemento "Tools.Make" per l'Insert mode: >

```
:iunmenu Tools.Make
```

Potete cancellare un intero menù, con tutti i propri elementi, usando il nome del menù stesso: Esempio: >

```
:aunmenu Syntax
```

Ciò cancella il menù Syntax e tutti i suoi elementi.

==========

42.3 Vario

Potete cambiare l'aspetto dei menù tramite delle flags in 'guioptions'. Nel valore di default esse sono tutte incluse. Potete rimuovere una flag con un comando del tipo: >

```
:set guioptions-=m
```

<

m	Quando rimossa la barra dei menù non viene mostrata.
M	Quando rimossa non vengono caricati i menù di default.
g	Quando rimossa i menù inattivi sono completamente rimossi (Non funziona su tutti i sistemi).
t	Quando rimossa il tearoff non viene abilitato.

La linea punteggiata in cima ai menù non è una linea di separazione. Quando selezionate questo elemento il menù biene "teared-off": Viene mostrato entro una finestra separata. Questo viene chiamato menù tearoff. E' utile quando usate lo stesso menù spesso.

Per la traduzione degli elementi dei menù, vedete |:menutrans|.

Poichè bisogna utilizzare il mouse per selezionare gli elementi dei menù, è bene usare il comando ":browse" per selezionare un file. Ed il comando ":confirm" per avere una finestra di dialogo invece di un messaggio di errore, ad esempio, quando il buffer corrente ha subito modifiche. Questi due possono venir combinati: >

```
:amenu File.Open  :browse confirm edit<CR>
```

":browse" fa apparire un navigatore di file per selezionare il file da aprire. ":confirm" fa apparire una finestra di dialogo se il file corrente ha subito modifiche. Potete scegliere se salvare le modifiche, gettarle via o cancellare il comando.
Per elementi più complicati, le funzioni confirm() e inputdialog() possono venir utilizzate. I menù di default contengono qualche esempio.

==========

42.4 Toolbar ed i menù popup

Ci sono due menù speciali: ToolBar e PopUp. Elementi che iniziano con questi nomi non compaiono nella barra del menù normale.

TOOLBAR

La toolbar compare solo se la flag "T" è inclusa nell'opzione 'guioptions'.
   La toolbar utilizza icone piuttosto che testo per rappresentare il comando. Per esempio, il {menu-item} chiamato "ToolBar.New" provoca la comparsa dell'icona "New" sulla toolbar.
   Vim ha un set di 28 icone built-in. Potete trovare una tabella qui: |builtin-tools|. La maggior parte di queste è già usata nella toolbar di default. Potete ridefinirne l'azione (dopo che i menù di default siano a punto).
   Potete aggiungere una nuova bitmap per un elemento della toolbar. Oppure definire un nuovo elemento per la toolbar con una bitmap. Per esempio, definiamo un nuovo elemento di toolbar con: **>**

```
:tmenu ToolBar.Compile  Compila il file corrente
:amenu ToolBar.Compile  :!cc % -o %:r<CR>
```

Ora bisogna creare una nuova icona. Per gli utenti di MS-Windows, questa deve essere in formato bitmap, con il nome "Compile.bmp". Per Unix viene usato il formato XPM, il nome è "Compile.xpm". Le dimensioni devono essere 18x18 pixels (sotto Windows è possibile utilizzare anche altre dimensioni, ma apparirà noioso).
   Mettete le bitmap nella directory "bitmaps" entro una delle directory del 'runtimepath'. Ad esempio, per Unix "~/.vim/bitmaps/Compile.xpm".

Potete definire anche delle didascalie per le icone della toolbar. Una didascalia è una breve testo che spiega cosa farà un elemento della toolbar. Per esempio "Open file". Apparirà quando il puntatore del mouse si trova sull'elemento, senza spostarsi per un attimo. Ciò è molto utile per capire l'immagine se non fosse chiara. Esempio: **>**

```
:tmenu ToolBar.Make  Run make in the current directory
<
```

Note:
Fate molta attenzione alle maiuscole/minuscole. "Toolbar" e "toolbar" sono diverse da "ToolBar"!

Per eliminare una didascalia, usate il comando |:tunmenu|.

L'opzione 'toolbar' può essere utilizzata per mostrare del testo invece di una bitmap, oppure entrambi testo ed una bitmap. Molti usano solo la bitmap poichè il testo prende più spazio.

MENU' POPUP

I menù popup compaiono dove si trova il mouse. In MS-Windows li attivate cliccando con il tasto destro del mouse. Allora potete selezionare un elemento con il tasto sinistro del mouse. In Unix i menù popup si attivano cliccando e rilasciando il tasto destro del mouse.
   I menù popup compaiono solo se l'opzione 'mousemodel' è stata impostata su "popup" o su "popup_setpos". La differenza tra le due è che "popup_setpos" muove il cursore verso la posizione in cui si trova il puntatore del mouse. Quando si clicca entro una selezione, questa verrà utilizzata non modificata. Quando c'è una selezione, ma cliccate fuori di essa, la selezione viene rimossa.
   Esistono menù popup separati per ogni modalità. Così non ci sono mai elementi inattivi come nei menù normali.

Qual è il significato della vita, dell'universo e di ogni cosa? *42*
Douglas Adams, la sola persona che ha conosciuto cosa fosse davvero questa questione è ormai morto, sfortunatamente. Così adesso potreste chiedervi quale sia il significato della morte...

==

Capitolo seguente: |usr_43.txt| Utilizzo dei tipi di file

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

usr_43.txt   Per Vim version 6.2. Ultima modifica: 2002 Lug 14

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

Utilizzo dei tipi di file

Quando state scrivendo un file di un certo tipo, per esempio un programma C o uno script di shell, spesso usate lo stessa scelta di impostazioni e di mappature. Rapidamente diverrete stanchi di impostarli manualmente ogni volta. Questo capitolo spiega come farlo automaticamente.

  Capitolo seguente: |usr_44.txt| Evidenziazione della vostra sintassi
Capitolo precedente: |usr_42.txt| Aggiungere nuovi menù
            Indice: |usr_toc.txt|

==
43.1 Plugin per un tipo di file   *filetype-plugin*

Come iniziare ad usare i plugin per i tipi di file è già stato discusso qui: |add-filetype-plugin|. Ma probabilmente non siete soddisfatti dalle impostazioni di default, perché sono state ridotte al minimo. Supponiamo che per i file in C vogliate impostare l'opzione 'softtabstop' a 4 e definire una mappatura per inserire un commento di tre linee. Potete farlo con due sole operazioni:

                                                        your-runtime-dir
1. Creare una vostra runtime directory. Su Unix questa è abitualmente
   "~/.vim". In questa directory create la directory "ftplugin": >

        mkdir ~/.vim
        mkdir ~/.vim/ftplugin
<
   Se non siete su Unix, controllate il valore dell'opzione 'runtimepath' per vedere dove Vim cercherà la directory "ftplugin": >

        set runtimepath

< Usare normalmente il primo nome di directory (quello che precede la prima virgola). Oppure potete premettere un nome di directory all'opzione 'runtimepath' nel vostro file |vimrc| se non vi piace il valore di default.

2. Creare il file "~/.vim/ftplugin/c.vim", con il contenuto: >

        setlocal softtabstop=4
        noremap <buffer> <LocalLeader>c o/**************<CR><CR>/<Esc>

Provate ora ad editare un file C. Potreste accorgervi che l'opzione 'softtabstop' è impostata a 4. Ma quando editate un altro file è riportata al valore di default che è zero. Questo succede perché si è usato il comando ":setlocal". Questo imposta l'opzione 'softtabstop' solo localmente al buffer. Appena aprite un altro buffer, viene impostata al valore proprio di quel buffer. Per un nuovo buffer prenderà il valore di default o quello impostato dall'ultimo comando ":set".

Analogamente, la mappatura per "\c" sparirà quando si lavora in un altro buffer. Il comando ":map<buffer>" crea una mappatura locale al buffer corrente. Questo succede con ogni comando di mappatura: ":map!", ":vmap", ecc. Il |<LocalLeader>| è sostituito con il valore di "maplocalleader".

Trovate degli esempi di plugin per i tipi di file in questa directory: >

        $VIMRUNTIME/ftplugin/

Ulteriori approfondimenti su come scrivere plugin per i tipi di file si trovano qui: |write-plugin|.

==
43.2 Aggiunta di un tipo di file

Se usate un tipo di file che non viene riconosciuto da Vim, ecco come si può farglielo riconoscere. Avete bisogno di una vostra runtime directory. Vedere sopra |your-runtime-dir|.

Creare un file "filetype.vim" che contenga un autocomando per il vostro

tipo di file. (Gli autocomandi sono stati spiegati nella sezione |40.3|).
Esempio: >

```
augroup filetypedetect
au BufNewFile,BufRead *.xyz     setf xyz
augroup END
```

Questo riconoscerà tutti i file che terminano in ".xyz" come tipo di file
"xyz". Il comando ":augroup" pone questo autocomando nel gruppo
"filetypedetect". Questo permette di rimuovere tutti gli autocomandi di
riconoscimento del tipo di file facendo ":filetypeoff". Il comando "setf"
imposta l'opzione 'filetype' al suo argomento, a meno che non fosse già
impostata. Questo evita che 'filetype' sia impostato due volte.

Potete usare molti schemi differenti per individuare il nome del vostro file.
Si possono includere anche i nomi delle directory. Vedere |autocmd-patterns|.
Per esempio che i file in "/usr/share/scripts/" siano tutti file "ruby", ma
non abbiano l'estensione attesa. Basta aggiungere una riga all'esempio di
sopra: >

```
augroup filetypedetect
au BufNewFile,BufRead *.xyz                     setf xyz
au BufNewFile,BufRead /usr/share/scripts/*      setf ruby
augroup END
```

Comunque, se ora aprite il file /usr/share/scripts/README.txt, non è un file
di Ruby. Il pericolo di uno schema che finisce in "*" è che individua
immediatamente troppi tipi di file. Per evitare problemi con ciò, mettete il
file filetype.vim in un'altra directory, una che sia alla fine di
'runtimepath'. Per Unix, ad esempio, potreste usare
"~/.vim/ultimo/filetype.vim".
   Ora mettete il riconoscimento dei file di testo in ~/.vim/filetype.vim: >

```
augroup filetypedetect
au BufNewFile,BufRead *.txt                     setf text
augroup END
```

Questo file viene trovato nel 'runtimepath' per primo. Poi scrivete questo in
~/.vim/ultimo/filetype.vim, che viene trovato per ultimo: >

```
augroup filetypedetect
au BufNewFile,BufRead /usr/share/scripts/*      setf ruby
augroup END
```

Quello che succede ora è che Vim cerca i file "filetype.vim" in ogni directory
del 'runtimepath'. Per primo viene trovato ~/.vim/filetype.vim.
L'autocomando per individuare i file *.txt è definito lì. Poi Vim trova il
file filetype.vim in $VIMRUNTIME, che è a metà strada nel 'runtimepath'.
Infine viene trovato ~/.vim/ultimo/filetype.vim e viene aggiunto l'autocomando
per individuare i file ruby in /usr/share/scripts.
   Ora, quando editate /usr/share/scripts/README.txt, gli autocomandi vengono
provati nell'ordine in cui sono stati trovati. Lo schema *.txt è soddisfatto,
e così si esegue "setf text" e il tipo di file è impostato a "text". Anche lo
schema per ruby è soddisfatto, e anche "setf ruby" viene eseguito. Ma siccome
'filetype' è stato già impostato a "text", questa seconda volta non succede
nulla.
   Quando invece si apre /usr/share/scripts/foobar vengono eseguiti gli stessi
autocomandi. Ma solo quello per ruby è sodisfatto, e "setf ruby" imposta
'filetype' a ruby.

RICONOSCIMENTO PER CONTENUTO

Se il vostro file non può essere riconosciuto semplicemente dal nome,
potreste essere in grado di riconoscerlo dal suo contenuto. Per esempio,
molti file di script iniziano con una linea del tipo:

```
#!/bin/xyz ~
```

Per riconoscere questo file di comandi create un file "scripts.vim" nella
vostra runtime directory (lo stesso posto in cui va filetype.vim). Potrebbe
apparire così: >

```
if did_filetype()
  finish
endif
if getline(1) =~ '^#!.*[/\\]xyz\>'
  setf xyz
```

```
        endif
```

Il primo controllo con did_filetype() serve ad evitare di controllare i contenuti dei file per i quali filetype era già stato riconosciuto dal nome. Questo evita di sprecare del tempo a controllare il file quando il comando "setf" non farà nulla.
   Il file scripts.vim viene letto da un autocomando nel default file filetype.vim. Quindi l'ordine di verifica è:

1. i file filetype.vim prima di $VIMRUNTIME in 'runtimepath'
2. la prima parte di $VIMRUNTIME/filetype.vim
3. tutti i file scripts.vim in 'runtimepath'
4. il resto di $VIMRUNTIME/filetype.vim
5. i file filetype.vim dopo $VIMRUNTIME in 'runtimepath'

Se questo non vi basta, aggiungete un autocomando che sia soddisfatto da ogni file, e che legga uno script od esegua una funzione che controlli il contenuto del file.

==

Capitolo seguente: |usr_44.txt| Evidenziazione della vostra sintassi

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

usr_44.txt   Per Vim version 6.2. Ultima modifica: 2002 Ott 10

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Giuliano Bordonaro

Evidenziazione della vostra sintassi

Vim nasce con la proprietà di evidenziare circa duecento tipi di file diversi.
Se il file che state modificando non fosse incluso, leggete questo capitolo
per scoprire come ottenere che anche questo tipo di file venga evidenziato.
Guardate anche |:syn-define| nel manuale di riferimento.

  Capitolo seguente: |usr_45.txt| Selezionate la vostra lingua
Capitolo precedente: |usr_43.txt| Utilizzo dei tipi di file
             Indice: |usr_toc.txt|

==
44.1 Fondamentali comandi della sintassi

Usando un file di sintassi esistente per iniziare risparmierete un sacco di
tempo. Provateci trovando un file di sintassi entro $VIMRUNTIME/syntax per un
linguaggio che sia simile. Questi files vi mostreranno l'aspetto normale di
un file di sintassi. Per capirlo dovete leggere quanto segue.

Cominciamo con gli argomenti fondamentali. Prima incominciamo definendo una
nuova sintassi, dovete togliere qualsiasi vecchia definizione: **>**

```
:syntax clear
```

Ciò non è necessario con il file di sintassi finale, ma molto utile quando
state sperimentando.

In questo capitolo ci sono ulteriori semplificazioni. Se scrivese un file di
sintassi che venga usato da altri, leggete comunque tutto sino alla fine per
scoprire i dettagli.

ELENCARE GLI ELEMENTI DEFINITI

Per vedere quali elementi di sintassi siano attualmente definiti usate questo
comando: **>**

```
:syntax
```

Potete usarlo per vedere quali elementi di sintassi siano stati sinora
definiti. Risulta utile per fare esperimenti con un nuovo file di sintassi.
Mostra anche i colori usati per ciascun elemento, il che aiuta a comprendere
di cosa si tratta.
   Per elencare gli elementi di un gruppo di sintassi specifico usate: **>**

```
:syntax list {group-name}
```

Ciò può anche venire impiegato per elencare dei clusters (spiegati in |44.8|).
Soltanto aggiungendo un @ nel nome.

CONFRONTO DEI CARATTERI

Alcuni linguaggi non sono sensibili al carattere come il Pascal. Altri,
come il C, sono sensibili al carattere. Dovete specificare quale sia il tipo
con i seguenti comandi: **>**

```
:syntax case match
:syntax case ignore
```

L'argomento "match" significa che Vim confronterà il carattere degli elementi di sintassi. Perciò "int" è diverso da "Int" ed "INT". Se l'argomento "ignore" viene impiegato tutti gli esempi seguenti saranno equivalenti: "Procedure", "PROCEDURE" e "procedure".
I comandi ":syntax case" possono apparire dovunque in un file di sintassi e riferirsi alle definizioni di sintassi che seguono. Il più delle volte avete solo un comando ":syntax case" nel vostro file di sintassi; se lavoraste con un tipo insolito di linguaggio contenente sia elementi sensibili al carattere che altri che non lo siano, comunque, avete la facoltà di disseminare il comando ":syntax case" per tutto il file.

==
44.2 Parole chiave

I più diffusi elementi fondamentali della sintassi sono le parole chiave. Per definire una parola chiave usate la seguente forma: **>**

```
:syntax keyword {group} {keyword} ...
```

Il {group} è il nome del gruppo di sintassi. Con il comando ":highlight" potete assegnare un colore ad un {group}. L'argomento {keyword} è precisamente una keyword. Ecco qualche esempio: **>**

```
:syntax keyword xType int long char
:syntax keyword xStatement if then else endif
```

Questo esempio usa i nomi di gruppo "xType" e "xStatement". Per convenzione ogni nome di gruppo è preceduto dal tipo di file per il linguaggio che viene definito. Questo esempio definisce la sintassi per il linguaggio x (linguaggio di esempio senza un nome particolarmente significativo). In un file di sintassi per scripts "csh" potrà essere usato il nome "cshType". Così il prefisso è uguale al valore di 'filetype'.
Questi comandi fanno si che le parole "int", "long" e "char" vengano evidenziate in un modo e le parole "if", "then", "else" ed "endif" lo siano diversamente. Ora dovete collegare i nomi del gruppo x ai nomi standard di Vim. Potete farlo con i seguenti comandi: **>**

```
:highlight link xType Type
:highlight link xStatement Statement
```

Ciò dice a Vim di evidenziare "xType" come "Type" e "xStatement" come "Statement". Vedere |group-name| peri nomi standard.

KEYWORDS INSOLITE

I caratteri usati in una parola chiave debbono essere entro l'opzione 'iskeyword'. Se usate un altro carattere la parola non potrà trovare mai una corrispondenza. Vim non fornirà un messaggio per informarvi di ciò.
Il linguaggio x usa il carattere '-' nelle parole chiave. Ecco come viene fatto:
>

```
:setlocal iskeyword+=-
:syntax keyword xStatement when-not
```

Il comando ":setlocal" viene usato per cambiare 'iskeyword' soltanto per il buffer corrente. Ora cambierà il comportamento di comandi come "w" e "*". Se non volete ciò, non definite una parola chiave ma usate un confronto (spiegato nella prossima sezione).

Il linguaggio x ammette delle abbreviazioni. Ad esempio "next" può venire abbreviato in "n", "ne" o "nex". Potete definirlo usando questo comando:
>

```
:syntax keyword xStatement n[ext]
```

Questo non corrisponde con "nextone", le parole chiave corrispondono sempre soltanto con parole intere.

==
44.3 Raffronti

Pensiamo di definire qualcosa di più complesso. Volete verificare le corrispondenze di identificatori ordinari. Per farlo definite un elemento di confronto per la sintassi. Questo trova corrispondenza con ogni parola che consista di soli caratteri minuscoli: **>**

```
:syntax match xIdentifier /\<\l\+\>/
```

<
	Note:
	Le keywords prevalgono su di ogni altro elemento di sintassi. Così le
	parole chiave "if", "then", etc., saranno keywords, come definito
	prima dal comando ":syntax keyword", anche se mantengono il modello
	per xIdentifier.

La parte finale è un modello, come esso viene usato per la ricerca. Il //
viene impiegato in aggiunta al modello (come avviene in un comando
":substitute". Potete utilizzare qualsiasi altro carattere, come un più o le
virgolette.

Adesso definiamo un confronto per un commento. Nel linguaggio x si tratta
di qualsiasi cosa a partire da # sino alla fine della linea: >

```
	:syntax match xComment /#.*/
```

Poichè potete usare qualunque modello di ricerca, potete evidenziare cose
molto complicate con un elemento di confronto. Vedere |pattern| sui modelli
per la ricerca.

==
44.4 Regioni

Nel linguaggio x di esempio le stringhe vengono incluse entro virgolette
doppie ("). Per evidenziare stringhe definite una regione. Serve una regione
di inizio (double quote) ed una di fine (double quote). La definizione è come
segue: >

```
	:syntax region xString start=/"/ end=/"/
```

Le direttive "start" ed "end" definiscono i modelli usati per trovare l'inizio
o la fine della regione. Ma come fare con stringhe come questa?

```
	"A string with a double quote (\") in it" ~
```

Ciò crea un problema: Le virgolette doppie nel mezzo della stringa faranno
terminare la regione. Dovrete dire a Vim di saltare ogni virgoletta doppia
preceduta da \ entro la stringa. Fatelo con la parola chiave skip: >

```
	:syntax region xString start=/"/ skip=/\\"/ end=/"/
```

La backslash doppia trova una backslash singola, poichè la backslash è un
carattere speciale nei modelli per la ricerca.

Quando usare una regione in luogo di una verifica di corrispondenza? La
differenza principale consiste nel fatto che un elemento di confronto è
costituito da un unico modello che deve trovare una corrispondenza esatta.
Una regione inizia dove si trova il modello "start". Se il modello "end"
viene trovato o no non importa. Così quando l'elemento dipendesse dalla
corrispondenza con il modello "end" non potreste utilizzare le regioni.
Altrimenti le regioni sono generalmente più facili da definire. Ed è più
semplice per impiegare elementi annidati, come spiegato nella prossima
sezione.

==
44.5 Elementi annidati

Osservate questo commento:

```
	%Get input  TODO: Skip white space ~
```

Volete evidenziare TODO in grosse lettere gialle, nonostante si trovi dentro
un commento che viene evidenziato in blu. Per consentire a Vim di saperlo
definite i seguenti gruppi di sintassi: >

```
	:syntax keyword xTodo TODO contained
	:syntax match xComment /%.*/ contains=xTodo
```

Nella prima linea l'argomento "contained" dice a Vim che questa keyword può
esistere soltanto entro altri elementi di sintassi. La linea successiva ha
"contains=xTodo". Ciò indica che l'elemento di sintassi xTodo è dentro di
essa. Il risultato è che la linea di commento nel suo insieme viene
riscontrata con "xComment" e viene fatta blu. La parola TODO entro essa trova
corrispondenza con xTodo ed evidenziata in giallo (L'evidenziazione per xTodo
era stata fatta per questo).

ANNIDAMENTO RICORSIVO

Il linguaggio x definisce blocchi di codice tra parentesi graffe. Ed un blocco di codice può contenere altri blocchi di codice. Ciò può venir definito così:
>
```
:syntax region xBlock start=/{/ end=/}/ contains=xBlock
```

Supponiamo che abbiate questo testo:

```
while i < b { ~
        if a { ~
                b = c; ~
        } ~
} ~
```

Inizialmente uno xBlock comincia dalla { nella prima linea. Nella seconda linea si trova un'altra {. Poichè ci troviamo entro un elemento xBlock ed esso contiene solo se stesso, un elemento annidato xBlock inizierà qui. Così la linea "b = c" lè dentro la regione xBlock di secondo livello. Allora una } viene trovata nella linea successiva e corrisponde con il modello di fine della regione. Ciò termina l'xBlock annidato. Poichè la } si trova nella regione annidata, viene nascosta dalla regione del primo xBlock. Così all'ultima } termina la regione del primo xBlock.

TROVARE LA FINE

Considerate i seguenti due elementi di sintassi: >

```
:syntax region xComment start=/%/ end=/$/ contained
:syntax region xPreProc start=/#/ end=/$/ contains=xComment
```

Definite un commento come qualsiasi cosa a partire da % sino al termine della linea. Una direttiva di preprocessore è qualunque cosa a partire da # sino al termine della linea. Poichè potete avere un commento su di una linea di preprocessore, la definizione di preprocessore include un argomento "contains=xComment". Adesso vediamo cosa succede con questo testo:

```
#define X = Y  % Comment text ~
int foo = 1; ~
```

Ciò che vedete è che anche la seconda linea viene evidenziata come xPreProc. La direttiva preprocessore potrebbe terminare alla fine della linea. Ciò perchè avete usato "end=/$/". Così cosa è andato male?
Il problema è il commento contenuto. Il commento parte con % e finisce alla fine della linea. Dopo la fine del commento la sintassi del preprocessore continua. Ciò succede dopo che si è vista la fine della linea, così viene inclusa anche la linea successiva.
Per evitare questo problema e che un elemento di sintassi contenuto si mangi la necessaria fine della linea, impiegate l'argomento "keepend". Ciò si occuperà del fatto di trovare una doppia fine della linea: >

```
:syntax region xComment start=/%/ end=/$/ contained
:syntax region xPreProc start=/#/ end=/$/ contains=xComment keepend
```

CONTENIMENTO DI MOLTI ELEMENTI

Potete usare l'argomento contains per specificare che tutto può essere compreso. Ad esempio: >

```
:syntax region xList start=/\[/ end=/\]/ contains=ALL
```

Tutti gli elementi di sintassi verranno compresi entro soltanto questo. Esso contiene anche se stesso, ma non alla stessa posizione (che potrebbe causare un loop senza fine).
Potete specificare che alcuni gruppi non siano compresi. Così comprende tutti i gruppi ma solo quelli che vengono elencati:
>
```
:syntax region xList start=/\[/ end=/\]/ contains=ALLBUT,xString
```

Con l'elemento "TOP" potete inserire tutti gli elementi che non abbiano un argomento "contained". "CONTAINED" viene usato solo per includere elementi con un argomento "contained". Vedere |:syn-contains| per i dettagli.

==
44.6 Seguendo i gruppi

Il linguaggio x comprende dichiarazioni in questa forma:

```
        if (condition) then ~
```

Volete evidenziare i tre elementi in modo diverso. Ma "(condition)" e "then" potrebbero apparire anche altrove, dove sono richieste diverse evidenziazioni. Potete fare così: **>**

```
 :syntax match xIf /if/ nextgroup=xIfCondition skipwhite
 :syntax match xIfCondition /([^)]*)/ contained nextgroup=xThen skipwhite
 :syntax match xThen /then/ contained
```

L'argomento "nextgroup" specifica quale argomento possa essere il successivo. Ciò non è necessario. Se nessuno degli elementi che avete specificato venisse trovato non succederebbe nulla. Ad esempio, in questo testo:

```
        if not (condition) then ~
```

L'"if" corrisponde con xIf. "not" non trova corrispondenze con la specificata xIfCondition di nextgroup, così solo l'"if" viene evidenziato.

L'argomento "skipwhite" dice a Vim che gli spazi bianchi (spazi e tabulazioni) possono apparire tra gli elementi. Argomenti analoghi sono "skipnl", che consente un'interruzione di linea tra gli elementi, e "skipempty", che permette linee vuote. Notate che "skipnl" non salta una linea vuota, talvolta trova corrispondenza dopo l'interruzione di linea.

==========

44.7 Altri argomenti

MATCHGROUP

Quando definite una regione l'intera regione viene evidenziata secondo il nome di gruppo specificato. Per evidenziare il testo incluso tra parentesi () con il gruppo xInside, ad esempio, usate il comando seguente: **>**

```
        :syntax region xInside start=/(/ end=/)/
```

Immaginiamo che vogliate evidenziare le parentesi in modo diverso. Potete farlo con molte contorte dichiarazioni di regione, o potete usare l'argomento "matchgroup". Ciò dice a Vim di evidenziare l'inizio e la fine di una regione con un diverso gruppo di evidenziazione (in questo caso, il gruppo xParen): **>**

```
        :syntax region xInside matchgroup=xParen start=/(/ end=/)/
```

L'argomento "matchgroup" viene applicato alla corrispondenza di start e di end che lo seguono. Nell'esempio precedente sia start che end vengono evidenziati con xParen. Per evidenziare end con xParenEnd: **>**

```
        :syntax region xInside matchgroup=xParen start=/(/
                \ matchgroup=xParenEnd end=/)/
```

Un effetto collaterale dell'uso di "matchgroup" è che gli elementi contenuti non trovino corrispondenza all'inizio od alla fine della regione. L'esempio per "transparent" usa ciò.

TRANSPARENT

In un file in linguaggio C vorreste evidenziare il testo () dopo un "while" in modo differente dal testo tra parentesi dopo un "for". In entrambi i casi potrebbero esservi annidati degli elementi (), che dovrebbero essere evidenziati allo stesso modo. Dovete essere certi che l'evidenziazione di () cessi con la corrispondenza). Ecco un modo per farlo:
>

```
        :syntax region cWhile matchgroup=cWhile start=/while\s*(/ end=/)/
                \ contains=cCondNest
        :syntax region cFor matchgroup=cFor start=/for\s*(/ end=/)/
                \ contains=cCondNest
        :syntax region cCondNest start=/(/ end=/)/ contained transparent
```

Ora potete dare con cWhile e cFor diverse evidenziazioni. L'elemento cCondNest può apparire in uno di essi, ma escludere l'evidenziazione dell'elemento che è contenuto. L'argomento "transparent" causa ciò.
   Notare che l'argomento "matchgroup" ha lo stesso gruppo che l'elemento itself. Perchè definirlo allora? Bene, l'effetto collaterale di usare matchgroup is che elementi contenuti non vengono trovati allora con l'elemento start. Ciò evita che il gruppo cCondNest trovi la (appena dopo il "while" od

il "for". Se ciò avvenisse potrebbe accoppiare tutto il testo sino alla corrispondenza con) e la regione continuerebbe dopo di esso. Adesso cCondNest trova la corrispondenza solo con il modello start, dopo la prima (.

OFFSETS

Poniamo che vogliate definire una regione per il testo tra (e) dopo un "if". Ma non volete includere l'"if" o le (e). Potete fare questo specificando gli offsets per i modelli. Esempio: **>**

```
:syntax region xCond start=/if\s*(/ms=e+1 end=/)/me=s-1
```

L'offset per il modello di start è "ms=e+1". "ms" sta per Match Start. Ciò definisce un offset per l'inizio della corrispondenza. Normalmente la verifica parte dal modello di ricerca. "e+1" significa che la verifica adesso partirà al termine del modello di ricerca e precisamente da un carattere dopo.
   L'offset per il modello end è "me=s-1". "me" sta per Match End. "s-1" significa l'inizio dal modello di ricerca ed allora un carattere prima. Il risultato è che in questo testo:

```
if (foo == bar) ~
```

Soltanto il testo "foo == bar" verrà evidenziato come xCond.

Altro sugli offsets qui: |:syn-pattern-offset|.

ONELINE

L'argomento "oneline" indica che la regione non attraversa il limite di una linea. Ad esempio: **>**

```
:syntax region xIfThen start=/if/ end=/then/ oneline
```

Ciò definisce una regione che inizia dall'"if" e termina al "then". Ma se non ci fosse un "then" dopo l'"if", la regione non corrisponderebbe.

Note:
Usando "oneline" la regione non inizia se il modello end non si trova nella stessa linea. Senza "oneline" Vim non potrà trovare una corrispondenza per il modello di end pattern. La regione inizia lo stesso quando il modello di end non venisse trovato nel resto del file.

LINEE CHE VANNO A CAPO E COME EVITARLE

Adesso le cose diventano un po più complesse. Definiamo una linea di preprocessor. Questa inizia con un # nella prima colonna e continua sino alla fine della linea. Una linea che finisca con \ rende la linea successiva una linea di continuazione. Il modo per gestire ciò è consentire che un elemento di sintassi contenga un modello di continuazione: **>**

```
:syntax region xPreProc start=/^#/ end=/$/ contains=xLineContinue
:syntax match xLineContinue "\\$" contained
```

In questo caso sebbene xPreProc verifichi normalmente una sola linea, il gruppo in essa contenuto (vale a dire xLineContinue) gli consente di procedere per più di una sola linea. Ad esempio, verificherà entrambe queste linee:

```
#define SPAM  spam spam spam \ ~
                bacon and spam ~
```

In questo caso è quanto volevate. Se non fosse così potreste imporre alla regione di essere su di una sola linea aggiungendo "excludenl" al modello contenuto. Ad esempio, volete evidenziare "end" in xPreProc, ma soltanto alla fine della linea. Per evitare che xPreProc continui anche sulla linea successiva, come fa xLineContinue, usate "excludenl" in questa maniera: **>**

```
:syntax region xPreProc start=/^#/ end=/$/
        \ contains=xLineContinue,xPreProcEnd
:syntax match xPreProcEnd excludenl /end$/ contained
:syntax match xLineContinue "\\$" contained
```

"excludenl" deve essere posto prima del modello. Se c'è "xLineContinue" non si può usare "excludenl", una corrispondenza con esso estenderebbe xPreProc alla prossima linea come prima.

==
44.8 Clusters

Una delle cose che dovete sapere quando iniziate a scrivere un file di sintassi è che state per generare molti gruppi di sintassi. Vim vi consente di definire una collezione di gruppi di sintassi che viene chiamata cluster.
   Immaginate di avere un linguaggio che contenga cicli di for, dichiarazioni if, cicli while e funzioni. Ognuno di essi contiene gli stessi elementi di sintassi: numeri ed identificatori. Potete definirli così: **>**

```
:syntax match xFor /^for.*/ contains=xNumber,xIdent
:syntax match xIf /^if.*/ contains=xNumber,xIdent
:syntax match xWhile /^while.*/ contains=xNumber,xIdent
```

Vi tocca ripetere lo stesso "contains=" ogni volta. Se volete aggiungere un altro elemento contenuto, dovete aggiungerlo tre volte. I clusters di sintassi semplificano queste definizioni consentendovi di avere un solo cluster che stia per molti gruppi di sintassi.
   Per definire un cluster per i due elementi che i tre gruppi contengono usate il seguente comando: **>**

```
:syntax cluster xState contains=xNumber,xIdent
```

I cluster vengono usati entro altri elementi di sintassi proprio come ogni gruppo di sintassi. Il loro nome inizia con @. Potete definire i tre gruppi in questo modo: **>**

```
:syntax match xFor /^for.*/ contains=@xState
:syntax match xIf /^if.*/ contains=@xState
:syntax match xWhile /^while.*/ contains=@xState
```

Potete aggiungere nuovi nomi di gruppo a questo cluster con l'argomento "add": **>**

```
:syntax cluster xState add=xString
```

Potete togliere dei gruppi di sintassi da questo elenco altrettanto facilmente: **>**

```
:syntax cluster xState remove=xNumber
```

==
44.9 Inserimento di un altro file di sintassi

La sintassi del linguaggio C++ è un superset del linguaggio C. Poichè non volete scrivere due files di sintassi potrete fare leggere il file di sintassi per C++ entro uno per C usando il comando seguente: **>**

```
:runtime! syntax/c.vim
```

Il comando ":runtime!" cerca 'runtimepath' per tutti i files "syntax/c.vim". Ciò fa si che la sintassi C venga definita come per i files C. Se aveste sostituito il file di sintassi c.vim, od aggiunto elementi con un altro file, questo verrebbe caricato lo stesso correttamente.
   Dopo aver caricato gli elementi di sintassi C gli elementi specifici per C++ potranno essere definiti. Ad esempio, aggiungere keywords che non vengono usate in C: **>**

```
:syntax keyword cppStatement    new delete this friend using
```

Funzionerà esattamente come in ogni altro file di sintassi.

Adesso consideriamo il linguaggio Perl. Consiste di due parti distinte: una sezione di documentazione in formato POD, ed un programma scritto nello stesso Perl. La sezione POD comincia con "=head" e finisce con "=cut".
   Volete definire la sintassi POD in un solo file ed usarlo dal file di sintassi di Perl. Il comando ":syntax include" legge in un file di sintassi ed immagazzina gli elementi che questo ha definito in un cluster di sintassi. Per Perl le dichiarazioni sono come segue: **>**

```
:syntax include @Pod <sfile>:p:h/pod.vim
:syntax region perlPOD start=/^=head/ end=/^=cut/ contains=@Pod
```

Quando in un file Perl viene trovato "=head", la regione perlPOD inizia. In questa regione è contenuto il cluster @Pod. Tutti gli elementi definiti come elementi di massimo livello nei files di sintassi pod.vim si troveranno qui. Quando verrà trovato "=cut", la regione finirà e ritorneremo agli elementi

definiti nel file di Perl.
   Il comando ":syntax include" è abbastanza intelligente da ignorare un comando ":syntax clear" nel file incluso. Ed un argomento come "contains=ALL" conterrà solo elementi definiti nel file in esso incluso, non nel file che lo include.
   La parte "<sfile>:p:h/" usa il nome del file corrente (<sfile>), lo espande al suo path completo (:p) ed allora prende l'intestazione (:h). Ciò risulta nel nome della directory del file. Ciò causa che il file pod.vim venga incluso nella directory stessa.

==
44.10 Sincronizzazione

I compilatori la fanno semplice. Partono dall'inizio del file e lo interpretano direttamente. Vim non fa ciò così facilmente. Deve partire dal mezzo, dove si è iniziato a lavorare. Come fare a dirgli dove si trova?
   Il segreto sta nel comando ":syntax sync". Spiega a Vim come capire dove si trovi. Ad esempio, il comando che segue dice a Vim di cercare all'indietro l'inizio o la fine di un commento in stile C ed iniziare da li a colorare la sintassi: **>**

```
        :syntax sync ccomment
```

Potete affinare questo processo con qualche argomento. L'argomento "minlines" dice a Vim il minimo numero di linee da vedere all'indietro, e "maxlines" dice all'editor il massimo numero di linee da scandire.
   Ad esempio, il comando seguente dice a Vim di osservare almeno 10 linee prima della cima dello schermo: **>**

```
        :syntax sync ccomment minlines=10 maxlines=500
```

Se non riesce a capire dove si trovi entro questo spazio comincerà a guardare sempre più distante sino a riuscirvi. Ma non guarderà indietro più di 500 linee. (Un grosso "maxlines" rallenta il processo. Uno piccolo potrebbe causare il fallimento di sincronizzazione.)
   Per sincronizzare un po più rapidamente dite a Vim quali elementi di sintassi possono essere saltati. Tutte le corrispondenze e le regioni che richiedono solo di essere usate mentre viene mostrato il testo possono essere date con l'argomento "display".
   Di default, il commento che deve essere trovato verrà colorato come parte del gruppo Comment syntax. Se voleste colorare le cose in altro modo potete specificare un gruppo di sintassi differente: **>**

```
        :syntax sync ccomment xAltComment
```

Se il vostro linguaggio di programmazione non prevede commenti in stile C potete provare un altro metodo di sincronizzazione. Il modo più semplice consiste nel dire a Vim di spaziare all'indietro di un certo numero di linee e tentare di riuscire a capire le cose da li. Il comando seguente dice a Vim di andare indietro di 150 linee ed iniziare ad interpretare da lì: **>**

```
        :syntax sync minlines=150
```

Un valore grosso di "minlines" può rendere Vim più lento, specialmente scorrendo il file all'indietro.
   In ultimo, potete specificare un gruppo di sintassi per cercare usando questo comando:
>

```
        :syntax sync match {sync-group-name}
                \ grouphere {group-name} {pattern}
```

Esso dice a Vim che quando vede {pattern} il gruppo di sintassi chiamato {group-name} comincia subito dopo il modello dato. Il {sync-group-name} viene usato per dare un nome a questa specifica di sincronizzazione. Ad esempio il linguaggio di scripting sh comincia una dichiarazione if con "if" e lo finisce con "fi":

```
        if [ --f file.txt ] ; then ~
                echo "File exists" ~
        fi ~
```

Per definire una direttiva "grouphere" per questa sintassi usate il seguente comando: **>**

```
        :syntax sync match shIfSync grouphere shIf "\<if\>"
```

L'argomento "groupthere" dice a Vim che il modello finisce un gruppo. Ad esempio, la fine del gruppo if/fi è come segue: **>**

```
:syntax sync match shIfSync groupthere NONE "\<fi\>"
```

In questo esempio il NONE dice a Vim che non siete entro una regione di sintassi speciale. Particolarmente non vi trovate entro un blocco if.

Potete anche definire corrispondenze e regioni che siano senza "grouphere" od argomenti di "groupthere". Questi gruppi sono per gruppi di sintassi saltati durante la sincronizzazione. Ad esempio quanto segue salta ogni cosa entro {}, anche se corrisponderebbe normalmente con altri metodi di sincronizzazione: **>**

```
:syntax sync match xSpecial /{.*}/
```

Di più circa la sincronizzazione nel manuale di riferimento : |:syn-sync|.

==========

44.11 Installare un file di sintassi

Quando il vostro nuovo file di sintassi è pronto per essere usato mettetelo entro una directory chiamandola "syntax", nel 'runtimepath'. Per Unix potrebbe essere "~/.vim/syntax".
  Il nome del file di sintassi deve essere uguale al tipo di file con aggiunto ".vim". Così per il linguaggio x il path completo potrebbe essere:

```
~/.vim/syntax/x.vim ~
```

Dovete anche far si che il tipo di file venga riconosciuto. Vedere |43.2|.

Se il vostro file lavora bene potreste desiderare di renderlo disponibile per altri utenti di Vim. Prima leggete la prossima sezione per essere certi che il vostro file funzioni bene per altri. Allora inviate un messaggio e-mail al manutentore di Vim: <maintainer@vim.org>. Così spiegate come il tipo di file può essere riconosciuto. Con un po' di fortuna il vostro file verrà incluso nella prossima versione di Vim!

AGGIUNGERE AD UN FILE DI SINTASSI ESISTENTE

Stiamo immaginando che stiate aggiungendo un file di sintassi completamente nuovo. Quando un file di sintassi esistente funziona, ma sono stati persi alcuni elementi, potete aggiungere degli elementi in un file separato. Ciò evita di cambiare i file di sintassi distribuiti che possono andare perduti installando una nuova versione di Vim.
   Scrivete comandi di sintassi nel vostro file possibilmente usando nomi di gruppo dalla sintassi esistente. Ad esempio per aggiungere nuovi tipi di variabile al file di sintassi di C:
>

```
:syntax keyword cType off_t uint
```

Scrivete il file con lo stesso nome del file di sintassi originale. In questo caso "c.vim". Piazzatelo entro una directory vicino alla fine di 'runtimepath'. Ciò fa si che venga caricato dopo il file di sintassi originale. Per Unix potrebbe essere:

```
~/.vim/after/syntax/c.vim ~
```

==========

44.12 Aspetto di un file di sintassi portatile

Non sarebbe bello se tutti gli utenti di Vim si scambiassero i file di sintassi? Per renderlo possibile il file di sintassi deve seguire poche linee guida.

Cominciate con un header che spieghi a cosa serve il file di sintassi, chi lo mantiene e quale sia l'ultimo aggiornamento. Non includete troppe informazioni circa la storia delle modifiche, poca gente lo leggerà. Esempio: **>**

```
" Vim syntax file
" Language:     C
" Maintainer:   Bram Moolenaar <Bram@vim.org>
" Last Change:  2001 Jun 18
" Remark:       Included by the C++ syntax.
```

Usate lo stesso aspetto degli altri file di sintassi. Usando un file di sintassi esistente come esempio risparmierete un mucchio di tempo.

Scegliete un buono, descrittivo nome per il vostro file di sintassi. Usate

lettere minuscole e cifre. Non fatelo troppo lungo, viene usato in molti posti: Il nome del file di sintassi "nome.vim", 'filetype', b:current_syntax l'inizio di ogni gruppo di sintassi (nameType, nameStatement, nameString, etc).

Iniziate con una prova per "b:current_syntax". Se è definito, qualche altro file di sintassi, prima in 'runtimepath' era già stato caricato. Perchè sia compatibile con Vim 5.8 usate: **>**

```
        if version < 600
          syntax clear
        elseif exists("b:current_syntax")
          finish
        endif
```

Mettete "b:current_syntax" al nome della sintassi alla fine. Non dimenticate che i file inclusi fanno ciò anche, potreste dover resettare "b:current_syntax" se includete due files.

Se volete che il vostro file di sintassi funzioni con Vim 5.x, aggiungete una prova per v:version. Vedere yacc.vim per un esempio.

Non includete alcunchè sia una preferenza dell'utente. Non impostate 'tabstop', 'expandtab', etc. Ciò appartiene al plugin di un filename.

Non includete mappature od abbreviazioni. Aggiungete soltanto l'impostazione 'iskeyword' se è realmente necessario per riconoscere le keywords.

Evitate di usare colori specifici. Collegatevi ai gruppi di evidenziazione standard ogni qual volta sia possibile. Non scordate che taluni usano colori di sfondo diversi od hanno solo otto colori a disposizione.
Per compatibilità all'indietro con Vim 5.8 si usa questa costruzione: **>**

```
        if version >= 508 || !exists("did_c_syn_inits")
          if version < 508
            let did_c_syn_inits = 1
            command -nargs=+ HiLink hi link <args>
          else
            command -nargs=+ HiLink hi def link <args>
          endif

          HiLink nameString     String
          HiLink nameNumber     Number
          ... etc ...

          delcommand HiLink
        endif
```

Aggiungete l'argomento "display" agli elementi che non si usano quando si effettua una sincronizzazione, per velocizzare lo scorrimento all'indietro e CTRL-L.

==

Capitolo seguente: |usr_45.txt| Selezionate la vostra lingua

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

usr_45.txt   Per Vim version 6.2. Ultima modifica: 2002 Ott 08

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Paolo Giovannelli

Selezionate la vostra lingua

I messaggi in Vim possono essere dati in molte lingue. Questo capitolo spiega come cambiare quale venga usata. Così vengono illustrati i diversi modi per lavorare con file in varie lingue.

  Capitolo seguente: |usr_90.txt| Installare Vim
Capitolo precedente: |usr_44.txt| Evidenziazione della vostra sintassi
              Indice: |usr_toc.txt|

==

45.1 Lingua per i messaggi

Quando avviate Vim, esso controlla l'ambiente di lavoro per trovare quale lingua stiate utilizzando. Generalmente ciò funziona bene ed ottenete dei messaggi nella vostra lingua (se sono disponibili). Per vedere quale sia la lingua corrente, usate questo comando: **>**

```
:language
```

Se risponde con "C", ciò significa che viene impiegato il default, ovvero l'inglese.

> Note:
> L'utilizzo di lingue diverse è possibilie solo se Vim era stato compilato appositamente per gestirlo. Per scoprire se funziona, utilizzate il comando ":version" e verificate l'output per "+gettext" e "+multi_lang". Se ci sono, siete a posto. Altrimenti, se vedete "-gettext" o "-multi_lang" dovrete cercare un altro Vim.

Cosa fare se voleste i vostri messaggi in una lingua diversa? Ci sono diversi modi. Quale potreste usare dipende dalle capacità del vostro sistema.
   Il primo modo consiste nell'impostare l'ambiente di lavoro per la lingua desiderata prima di avviare VIM. Esempio per Unix: **>**

```
env LANG=de_DE.ISO_8859-1  vim
```

Ciò funziona solamente se la lingua è presente nel vostro sistema. Il vantaggio è che tutti i messaggi della GUI e gli oggetti nelle librerie useranno anch'essi la lingua giusta. Uno svantaggio è che dovete farlo prima di avviare VIM. Se invece volete cambiare la lingua mentre Vim è in esecuzione, potete usare il secondo metodo: **>**

```
:language fr_FR.ISO_8859-1
```

In questo modo potete provare diversi nomi per la vostra lingua. Otterrete un messaggio di errore quando tale funzione non fosse supportata dal vostro sistema. Non ottenete alcun messaggio d'errore quando i messaggi tradotti non siano disponibili. VIM automaticamente ritornerà ad usare l'inglese.
   Per sapere quali lingue sono supportate cercate la cartella ove esse sono elencate. Nel mio sistema, ad esempio, è "/usr/share/locale". Su alcuni sistemi "/usr/lib/locale". La pagina del manuale per "setlocale" potrebbe darvi un'indicazione su dove essa si trovi nel vostro sistema.
   State attenti a battere il nome esattamente come potrebbe essere. Le maiuscole e le minuscole sono importanti, ed i caratteri '-' e '_' vengono facilmente confusi.

Potete anche impostare una lingua diversa per i messaggi, il testo in lavorazione ed il formato della data. Vedere |:language|.

REALIZZARE DA SOLI LA TRADUZIONE DEI MESSAGGI

Se i messaggi tradotti non fossero disponibili nella vostra lingua, potreste scriverli voi stessi. Per far ciò, procuratevi il codice sorgente di Vim ed il pacchetto gettext GNU. Dopo aver scompattato i sorgenti, potrete trovare le istruzioni nella cartella src/po/README.txt.

Non è particolarmente difficile fare la traduzione. Non avrete bisogno di essere programmatori. Dovrete semplicemente conoscere sia l'inglese che la lingua in cui state traducendo, ovviamente.
Quando sarete soddisfatti con la traduzione pensate a renderla disponibile ad altri. Caricatela su vim-online (http://vim.sf.net) o spedendo un mail al mantainer Vim <maintainer@vim.org> o entrambe le cose.

==
45.2 Lingua per i menù

I menù di default sono in inglese. Per poter impiegare la vostra lingua locale devono essere tradotti. Normalmente ciò viene svolto automaticamente per voi se l'ambiente di lavoro è impostato per la vostra lingua, proprio come per i messaggi. Non sarà necessario fare di più per questo. Ma ciò funziona solo se le traduzioni nella vostra lingua sono disponibili.
Supponete di essere in Germania, di avere impostato la lingua tedesca e di preferire "file" a "Datei". Potreste ripassare ai menù in inglese in questo modo: **>**

```
:set langmenu=none
```

E'anche possibile specificare una lingua: **>**

```
:set langmenu=nl_NL.ISO_8859-1
```

Come sopra la differenza tra "-" e "_" è rilevante. Ad ogni modo, la differenza tra le maiuscole e le minuscole è ignorata in questo contesto.
L'opzione 'langmenu' deve essere impostata prima che i menù vengano caricati. Dopo aver definito i menù infatti, cambiare 'langmenu' non sortisce alcun effetto. Comunque, inserite il comando per impostare 'langmenu' nel vostro file "vimrc".
Se realmente volete cambiare la lingua dei menù mentre utilizzate Vim, potete farlo in questo modo: **>**

```
:source $VIMRUNTIME/delmenu.vim
:set langmenu=de_DE.ISO_8859-1
:source $VIMRUNTIME/menu.vim
```

C'è solo uno svantaggio: Tutti i menù definiti da voi se ne andranno. Dovrete perciò ridefinirli.

REALIZZARE DA SOLI LA TRADUZIONE DEI MENU'

Per sapere quali traduzioni dei menù sono disponibili, guardate in questa cartella:

```
$VIMRUNTIME/lang ~
```

I file sono denominati menu_{language}.vim. Se non vedete la lingua che intendete utilizzare, potete fare voi stessi le traduzioni. Il modo più semplice per farlo è copiare uno dei file lingua esistenti e modificarlo.
Prima trovate il nome della vostra lingua con il comando ":language". Usate questo nome, ma con tutte le lettere rese minuscole. Allora copiate il file nella vostra cartella di runtime, quella trovata prima in 'runtimepath'. Ad esempio, in Unix sarebbe: **>**

```
:!cp $VIMRUNTIME/lang/menu_ko_kr.euckr.vim ~/.vim/lang/menu_nl_be.iso_8859-1.vim
```

Troverete suggerimenti per la traduzione in "$VIMRUNTIME/lang/README.txt".

==
45.3 Utilizzare un'altra codifica.

Vim osserva che i file che state per aprire sono codificati per la vostra lingua. Per molte lingue europee tale codifica è "latin1". Allora ogni byte è un solo carattere. Ciò significa che sono possibili 256 differenti caratteri. Per le lingue asiatiche ciò non è sufficiente. Queste usano principalmente una codifica a doppio byte, che rende disponibili oltre 10000 possibili caratteri. Ciò non è ancora sufficiente quando un testo contenga molte lingue diverse. Qui è dove entra Unicode. E' stato progettato per includere tutti i caratteri utilizzati nelle lingue comunemente usate. Questa è la "super codifica che sostituisce tutte la altre". Ma non è ancora molto utilizzata.
Fortunatamente, Vim supporta tre tipi di codifica. E, con alcune restrizioni, potete utilizzarle anche quando il vostro ambiente di lavoro usa un'altra lingua rispetto al testo.

Ciononostante, quando aprite solo dei file codificati nella vostra lingua, il default dovrebbe lavorare bene e non dovrete fare altro. Ciò che segue è importante solo quando lavorate con lingue differenti.

Note:
L'utilizzo di molteplici codifiche funziona solo se Vim era stato compilato per gestirle. Per scoprire se ciò funziona, utilizzate il comando ":version" e cercate nell'output "+multi_byte". Se lo trovate significa che siete a posto. Se invece vedete "-multi_byte" significa che dovrete cercare un'altra versione di Vim.

USARE UNICODE NELLA GUI

Il bello di Unicode è che le altre codifiche possono essere convertite ad esso e viceversa senza perdere informazioni. Quando impostate Vim per utilizzare Unicode internamente, potrete lavorare con file in qualsiasi codifica.
Sfortunatamente, il numero di sistemi che supportano Unicode è ancora limitato. Così è improbabile che la vostra lingua lo utilizzi. Dovrete dire a Vim che volete usare Unicode, e come interfacciarsi con il resto del sistema.
Iniziamo con la versione GUI di Vim, che può usare i caratteri Unicode. Ciò potrebbe funzionare: **>**

```
:set encoding=utf-8
:set guifont=-misc-fixed-medium-r-normal--18-120-100-100-c-90-iso10646-1
```

L'opzione 'encoding' dice a Vim quale codifica di caratteri utilizzate. Ciò si applica al testo nei buffer (i file su cui state lavorando), ai registri, agli script di Vim, etc. Potete guardare 'encoding' come un'impostazione per gli internals di Vim.
Questo esempio presuppone che abbiate questo font installato nel vostro sistema. Il nome nell'esempio è per il sistema Xwindow. Questo font si trova in un pacchetto che viene utilizzato per migliorare xterm con il supporto Unicode. Se non avete questo font, potete trovarlo qui:

```
http://www.cl.cam.ac.uk/~mgk25/download/ucs-fonts.tar.gz ~
```

Per MS-Windows, alcuni font hanno un numero limitato di caratteri Unicode. Porvate usando il font "Courier New". Potete usare il menù Edit/Select Font per selezionare e provare i font disponibili. Tuttavia solo i font con ampiezza fissa posso essere utilizzati. **>**

```
:set guifont=courier_new:h12
```

Se non dovesse funzionare, cercate di ottenere un fontpack. Se Microsoft non l'ha spostato, potete trovarlo qui:

```
http://www.microsoft.com/typography/fontpack/default.htm ~
```

Ora avete detto a Vim di usare Unicode internamente e di visualizzare il testo con un font Unicode. I caratteri battuti ancora arrivano nel vostro linguaggio originale. Ciò richiede di convertirli in Unicode. Dite a Vim il linguaggio da cui covertire con l'opzione 'termencoding'. Potete farlo così: **>**

```
:let &termencoding = &encoding
:set encoding=utf-8
```

Ciò assegna il vecchio vaore di 'encoding' a 'termencoding', prima di impostare 'encoding' ad utf-8. Dovrete controllare se questi comandi funzionano davvero con la vostra installazione. Potrebbe funzionare particolarmente bene usando un metodo di input per una lingua asiatica, e volete scrivere del testo in Unicode.

USARE UNICODE IN UN TERMINALE UNICODE

Esistono terminali che supportano Unicode direttamente. Lo standard xterm che viene con Xfree è uno di essi. Usiamolo come esempio.
Prima di tutto, xterm deve essere compilato con il supporto ad Unicode. Guardate |UTF8-xterm| per sapere come controllare ciò e come compilarlo in caso di necessità.
Avviate xterm con l'argomento "-u8". Potreste anche avere bisogno di specificare un font. Esempio: **>**

```
xterm -u8 -fn -misc-fixed-medium-r-normal--18-120-100-100-c-90-iso10646-1
```

Ora potete avviare Vim entro questo terminale. Impostate 'encoding' ad

'utf-8' come prima. E questo è tutto.

USARE UNICODE IN UN TERMINALE NORMALE

Supponiamo che vogliate lavorare con dei file Unicode, ma non abbiate un terminale con lo supporti. Potete fare ciò con Vim, tuttalpiù i caratteri non supportati dal terminale non verranno visualizzati. L'aspetto del testo verrà preservato. **>**

```
:let &termencoding = &encoding
:set encoding=utf-8
```

Questo è lo stesso che era stato usato per la GUI. Ma funziona diversamente: Vim convertirà il testo visualizzato prima di inviarlo al terminale. Ciò evita che il display visualizzi strani caratteri.
   Affinchè ciò funzioni è necessario che la conversione tra 'termencoding' ed 'encoding' sia possibile. Vim convertirà da latin1 ad Unicode, cosicchè funzioni sempre. Per altre conversioni è richiesta la caratteristica |+iconv|.
   Provate a modificare un file con caratteri Unicode. Noterete che Vim mette un punto interrogativo (oppure una sottolineatura o qualche altro carattere) nei posti dove c'è un carattere che il terminale non può visualizzare. Muovete il cursore sino al punto interrogatico ed usate questo comando: **>**

```
ga
```

Vim visualizzerà una linea con il codice del carattere. Ciò vi da un'idea di che carattere sia. Potrete poi trovarlo in una tabella Unicode. Potreste realmente leggere un file in questo modo se aveste molto tempo a disposizione.

Note:
Poichè 'encoding' è usato per tutto il testo entro Vim, modificarlo invaliderebbe tutto il testo non ASCII. Potetete notare ciò usando i registri ed il file 'viminfo' (ad esempio, un modello di ricerca memorizzato). Si raccomanda perciò di impostare 'encoding' nel vostro file vimrc, e lasciarlo stare.

==
45.4 Elaborare files con una codifica differente

Supponete di avere impostato Vim per l'uso di Unicode e che vogliate modificare un file Unicode a 16 bit. Sembra semplice, giusto? Bene, Vim in realtà usa internamente la codifica utf-8, perciò la codifica a 16 bit deve essere convertita. Così c'è una differenza tra il set di caratteri (Unicode) e la codifica (Utf-8 o 16 bit).
   Vim cercherà di individuare il tipo di file con il quale state lavorando. Esso usa nomi di codifica situati nell'opzione 'fileencodings'. Quando si usa Unicode, il valore standard è:"ucs-bom,utf-8,latin1". Ciò significa che Vim controlla il file per vedere se contiene una di queste codifiche:

ucs-bom	Il file deve iniziare con un 'Byte Order Mark (BOM)'. Ciò consente la rilevazione delle codifiche unicode a 16 bit, 32 a bit ed utf-8
utf-8	utf-8 unicode. E' rifiutata quando una sequenza di bytes è illegale in utf-8.
latin1	La vecchia codifica ad 8-bit. Funziona sempre.

Quando iniziate a lavorare su un file Unicode a 16 bit, nel quale è presente un BOM, Vim individua ciò e converte il file in utf-8 quando lo legge. L'opzione 'fileencoding' (senza la s finale) è impostata al valore individuato. In questo caso è "ucs-2le". Ciò significa Unicode, due byte ed un 'little-endian'. Questo formato di file è comune in MS-Windows, (ad esempio per i file di registro).
   Quando si scrive un file, Vim confronta 'fileencoding' con 'encoding'. Se sono differenti, il testo sarà convertito.
   Un valore nullo in 'fileencoding' significa che nessuna conversione deve essere eseguita.

Se il valore di 'fileencodings' di default non andasse bene per voi, allora impostatelo per le codifiche che volete che Vim tenti. Solo quando un valore venisse trovato non valido verrebbe usato quello successivo. Impostare "latin1" come primo non funziona, perchè non è mai illegale. Un esempio, per tornare al giapponese quando il file non ha un BOM e non è utf-8: **>**

```
        :set fileencodings=ucs-bom,utf-8,sjis
```

Vedere |encoding-values| per i valori suggeriti. Altri valori potrebbero funzionare altrettanto bene. Ciò dipende dal tipo di conversione disponibile.

FORZARE UNA CODIFICA

Se il rilevamento automatico non funzionasse, dovrete dire a Vim quale codifica sia usata dal file. Esempio: **>**

```
        :edit ++enc=koi8-r russian.txt
```

La parte "++enc" specifica il nome della codifica da usare solamente per questo file. Vim convertirà il file dalla codifica specificata, il russo in questo esempio, in 'enconding'. 'fileencoding' verrà così impostato per la codifica specificata, cosicchè la conversione inversa può essere fatta quando si scrive il file.
   Lo stesso argomento può essere utilizzato scrivendo un file. In tal modo potete realmente usare Vim per convertire un file. Esempio: **>**

```
        :write ++enc=utf-8 russian.txt
<
```

        Note:
        La conversione può causare una perdita di caratteri. Convertire da una codifica qualsisasi in Unicode e viceversa è generalmente libera da questi problemi, a meno che ci siano caratteri illegali. La conversione da Unicode in altre codifiche spesso perde informazioni quando ci sia più di un linguaggio nel file.

==
45.5 Impostare la lingua del testo

Le tastiere dei computer non hanno più di un centinaio di tasti. Alcune lingue hanno un centinaio di caratteri, Unicode ne ha più di 10000. Così come potete battere questi caratteri?
   Prima di tutto, quando non usate troppi caratteri speciali, potete usare i digrafici. Questo argomento è già stato spiegato in |24.9|.
   Quando utilizzate una lingua che usa molti più caratteri di quanti ne abbia la vostra tastiera, vorrete usare un Input Method (IM). Ciò richiede di imparare la traduzione dai tasti battuti ai caratteri risultanti. Quando vi serve un IM, ne avrete probabilmente già uno installato nel vostro sistema. Potrebbe funzionare con Vim come con altri programmi. Per i dettagli vedere |mbyte-XIM| per il sistema XWindow e |mbyte-IME| per MS-Windows.

KEYMAPS

Per alcune lingue, il set di caratteri è diverso da quello latino, ma impiega un numero simile di caratteri. E' perciò possibile mappare i tasti ai caratteri. Vim utilizza le 'keymap' per questo.
   Supponete di voler scrivere in ebraico. Potete caricare la keymap così: **>**

```
        :set keymap=hebrew
```

Vim cercherà di trovare un file di keymap per voi. Ciò dipende dal valore di "encoding". Se nessun file corrispondente venisse trovato, otterreste un messaggio di errore.

Ora potete digitare in ebraico in Insert mode. In Normal mode e quando battete un comando ":", Vim automaticamente passerà all'inglese. Potete usare questo comando per passare dall'ebraico all'inglese: **>**

```
        CTRL-^
```

Questo funziona solo in Insert mode che in Comand-line mode. In Normal mode fa qualcosa di completamente diverso (salta ad un altro file).
   L'uso della keymap è indicato nel messaggio della modalità, se avete l'opzione 'showmode' attivata. Nella gui Vim indicherà l'uso delle keymap con un colore diverso del cursore.
   Potete anche cambiare l'uso della keymap con le opzioni 'iminsert' e 'imsearch'.

Per vedere la lista delle mappature, usate questo comando : **>**

```
        :lmap
```

Per trovare quali file di keymap siano disponibili, nella GUI potete usare il menù Edit/Keymap. Altrimenti potete usare questo comando: >

```
:echo globpath(&rtp, "keymap/*.vim")
```

CREARE DA SOLI LE KEYMAPS

Potete creare da soli i vostri file di keymap. Non è particolarmente difficile. Iniziate con un file di keymap simile a quello della lingua che volete impiegare. Copiatelo nella cartella "keymap" nella vostra directory di runtime. Ad esempio, in Unix potreste usare la cartella "~/.vim/keymap".
   Il nome del file di keymap deve assomigliare a questo:

```
keymap/{name}.vim ~
```

oppure

```
keymap/{name}_{encoding}.vim ~
```

{name} è il nome della keymap. Scegliete un nome che sia ovvio, ma differente dalle keymap esistenti (a meno che non vogliate sostituire un file di keymap esistente).
{name} non può contenere un underscore. Volendo, aggiungete la codifica usata dopo un underscore. Esempi:

```
keymap/hebrew.vim ~
keymap/hebrew_utf-8.vim ~
```

I contenuti del file dovrebbero spiegarsi da sè. Guardate qualcuna delle keymap distribuite con VIM. Per i dettagli, vedere |mbyte-keymap|.

ULTIMA RISORSA

Se tutti gli altri metodi fallissero, potrete comunque inserire qualsiasi carattere con CTRL-V:

```
encoding   type                 range ~
8-bit      CTRL-V 123           decimal 0-255
8-bit      CTRL-V x a1          hexadecimal 00-ff
16-bit     CTRL-V u 013b        hexadecimal 0000-ffff
31-bit     CTRL-V U 001303a4    hexadecimal 00000000-7fffffff
```

Non digitate gli spazi. Vedere |i_CTRL-V_digit| per i dettagli.

==

Capitolo seguente: |usr_90.txt| Installare Vim

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

usr_90.txt   Pe Vim versione 6.2. Ultima modifica: 2002 Lug 14

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Paolo Giovannelli

Installare Vim

install

Prima di poter utilizzare Vim è necessario installarlo. Dipende dal vostro sistema se essa sia semplice o facile. Questo capitolo fornisce alcuni suggerimenti e spiega inoltre come vada fatto l'aggiornamento del programma ad una nuova versione.

Capitolo precedente: |usr_45.txt| Selezionate la vostra lingua
Indice: |usr_toc.txt|

==

90.1 Unix

Per prima cosa dovrete decidere se state installando Vim per tutto il sistema o per un solo utente. La procedura è quasi la stessa, ma la directory dove Vim viene installato è diversa.
   In una installazione per tutto il sistema la directory di partenza "/usr/local/" viene spesso utilizzata. Ma ciò può variare per il vostro sistema. Cercate di trovare dove sono installati altri pacchetti.
   Quando installate per un solo utente, potete utilizzare la vostra directory "home". I file saranno collocati in sottodirectory quali "bin" e "shared/vim".

DA UN PACCHETTO

Potete ottenere i binari precompilati per molti sistemi UNIX diversi. C'è una lunga lista con i link su questa pagina:

        http://www.vim.org/binaries.html ~

Dei volontari mantengono i binari, così questi sono spesso poco aggiornati. E' una buona idea compilare la propria versione UNIX utilizzando i sorgenti. Così, la creazione dell'editor dal sorgente vi consente di controllare quali siano le caratteristiche da compilare. Tuttavia ciò richiede un compilatore.

Se utilizzate una distribuzione Linux, il programma "vi" è probabilmente una versione minimale di Vim. Non esegue l'evidenziazione della sintassi, ad esempio. Cercate di trovare un altro pacchetto Vim nella vostra distribuzione, o cercate sul sito Web.

DAI SORGENTI

Per compilare ed installare Vim, avrete bisogno di:

- Un compilatore C (GCC di preferenza)
- Il programma GZIP (potete ottenerlo su www.gnu.org)
- I sorgenti Vim e gli archivi runtime

Per ottenere gli archivi Vim, cercate in questo file un mirror vicino a voi, consentirà il più veloce dei download:

        ftp://ftp.vim.org/pub/vim/MIRRORS ~

Od utilizzate l'home site ftp.vim.org se pensate sia veloce abbastanza. Andate alla directory "unix" e lì troverete un elenco di file. Il numero di versione è dentro al nome del file. Prenderete la versione più recente.
   Potete ottenere i file per UNIX in due modi: Un grande archivio contenente tutto, oppure quattro archivi più piccoli ognuno dei quali può essere cwntenuto in un floppy disk. Per la versione 6.1 l'unico grosso è denominato:

        vim-6.1.tar.bz2 ~

Avrete bisogno del programma bzip2 per decomprimerlo. Se non lo avete, prendete i quattro file più piccoli, che possono essere decompressi con gzip.

Per Vim 6.1 essi si chiamano:

```
vim-6.1-src1.tar.gz ~
vim-6.1-src2.tar.gz ~
vim-6.1-rt1.tar.gz ~
vim-6.1-rt2.tar.gz ~
```

COMPILARE

Prima create una directory di partenza dove lavorare, ad esempio: >

```
mkdir ~/vim
cd ~/vim
```

Qui ora decomprimete gli archivi. Se avete l'unico grosso archivio, lo potete decomprimere in questo modo: >

```
bzip2 -d -c path/vim-6.1.tar.bz2 | tar xf -
```

Sostituite "path" con il percorso della directory dove avete scaricato i file.

```
gzip -d path/vim-6.1-src1.tar.gz | tar xf -
gzip -d path/vim-6.1-src2.tar.gz | tar xf -
gzip -d path/vim-6.1-rt1.tar.gz | tar xf -
gzip -d path/vim-6.1-rt2.tar.gz | tar xf -
```

Se siete soddisfatti delle impostazioni di default ed il vostro ambiente è installato correttamente, dovreste poter compilare Vim solo con: >

```
cd vim61/src
make
```

Il programma make configurerà e compilerà ogni cosa. In seguito spiegheremo come compilare usando altre impostazioni.
Se ci fossero problemi nella compilazione, esaminate attentamente i messaggi di errore. Lì dovrebbero esserci le spiegazioni di cosa sia andato male. Con la speranza di riuscire a correggerli. Potreste dover disabilitare alcune caratteristiche per essere far compilare Vim. Controllate nel Makefile i suggerimenti specifici per il vostro sistema.

TEST

Ora potete provare se la compilazione abbia funzinato: >

```
make test
```

Ciò avvierà una serie di script di verifica per controllare che Vim funzioni come ci si aspetta. Vim verrà avviato diverse volte ed apparirà ogni genere di testo o di messaggio. Se tutto va bene alla fine vedrete:

```
test results: ~
ALL DONE ~
```

Se ci fossero uno o due messaggi relativi a test falliti, Vim potrebbe funzionare lo stesso, ma non perfettamente. Se vedete molti messaggi di errore o Vim non riuscisse a terminare, c'è qualcosa di sbagliato. O provate ad uscirne da soli o cercate qualcuno che possa farlo per voi. Potreste guardare in |maillist-archive| per una soluzione. Se invece tutto andasse male potreste chiedere sulla |maillist| di Vim se qualcuno può aiutarvi.

INSTALLAZIONE

install-home

Se volete installare nella vostra directory home, modificate il Makefile e cercate una linea

```
prefix = $(HOME) ~
```

Rimuovete la # all'inizio della linea.
Quando installate per l'intero sistema, Vim avrà molto probabilmente già scelto una buona directory di installazione per voi. Potete anche specificarne una, guardate sotto. Dovrete diventare root per quanto segue.

Per installare Vim battete: >

```
make install
```

Ciò dovrebbe collocare tutti i file importanti al giusto posto. Ora potete provare ad avviare vim per verificare che funzioni. Usate due semplici test per accertarvi che Vim trovi i propri file di runtime: **>**

```
:help
:syntax enable
```

Se non dovesse funzionare, usate questo comando per verificare dove Vim ricerchi i propri file di runtime: **>**

```
:echo $VIMRUNTIME
```

Potete anche avviare Vim con l'argomento "-V" per sapere cio che accade all'avvio: **>**

```
vim -V
```

Non dimenticate che il manuale utente presuppone che usiate Vim in un certo modo. Dopo aver installato Vim, seguite le istruzioni in |not-compatible| per far funzionare Vim come spiegato in questo manuale.

SELEZIONARE LE CARATTERISTICHE

Vim offre diversi modi per scegliere le che. Uno dei più semplici consiste nel modificare il Makefile. Ci sono molti suggerimenti ed esempi al riguardo. Spesso potete abilitare o disabilitare una caratteristica eliminando il commento ad una linea.
   Un'alternativa è di avviare separatamente "configure". Ciò vi consente di specificare le opzioni di configurazione manualmente. Lo svantaggio consiste nella necessità di dover riuscire a capire esattamente cosa battere.
   Seguono alcuni degli argomenti di configurazione più interessanti. Essi possono anche venir abilitati dal Makefile.

--prefix={directory}	Cartella dove installare Vim.
--with-features=tiny	Compila con molte impostazioni disabilitate.
--with-features=small	Compila con molte impostazioni disabilitate.
--with-features=big	Compila con più impostazioni abilitate.
--with-features=huge	Compila con ulteriori impostazioni abilitate. Vedere la |+feature-list| per sapere quali impostazioni vengono abilitate nei diversi casi.
--enable-perlinterp	Abilita l'interfaccia Perl. Esistono impostazioni simili per for ruby, python e tcl.
--disable-gui	Non compila l'interfaccia GUI.
--without-x	Non compila le caratteristiche della X-Windows. Quando entrambi gli argomenti sono utilizzati, Vim non si connette al server X, il che rende l'avvio più veloce.

Per vedere l'intera lista usate: **>**

```
./configure -help
```

Potete trovare un pò di spiegazioni per ciascuna caratteristica e link per ulteriori informazioni qui: |feature-list|.
   Per gli audaci: modificate il file "feature.h". Potete anche modificare il codice sorgente!

==

90.2 MS-Windows

Ci sono due modi per installare Vim per Microsoft Windows. Potete decomprimere diversi archivi, oppure utilizzare un grosso archivio autoinstallante. Molti utenti dotati di computer recenti preferiscono il

secondo metodo. Per il primo avrete bisogno di:

- Un archivio con i binari di Vim.
- L'archivio di runtime di Vim.
- un programma per decomprimere i file zip.

Per ottenere gli archivi Vim, cercate in questo file il mirror
più vicino a voi, garantirà un download più veloce:

	ftp://ftp.vim.org/pub/vim/MIRRORS ~

Oppure potete utilizzare il sito ftp.vim.org, se per voi è abbastanza veloce.
Recatevi nella directory "pc" nella quale troverete un elenco di file. Il
numero di versione è compreso nel nome del file. Utilizzerete la versione più
recente. Noi utilizzeremo "61" qui, che è la versione 6.1.

gvim61.exe	L'archivio autoinstallante.

Ciò è tutto cio di cui avrete bisogno se optate per il secondo modo. Lanciate
l'eseguibile e seguite i prompt.

Per il primo metodo dovete scegliere uno degli archivi binari. Questi sono
disponibili:

gvim61.zip	La normale versione MS-Windows con GUI.
gvim61ole.zip	La versione MS-Windows GUI con supporto OLE. Richiede più memoria, consente l'interfacciamento con altre applicazioni OLE.
vim61w32.zip	La versione console per MS-Windows a 32 bit. Per l'utilizzo in una console Win NT/2000/XP. Non funziona bene su Win 95/98.
vim61d32.zip	La versione per MS-DOS a 32 bit. Per l'utilizzo in una console Win 95/98.
vim61d16.zip	La versione per MS-DOS a 16 bit. Esclusivamente per i sistemi più vecchi. Non supporta i nomi dei file lunghi.

Vi serve solo uno di questi. Tuttavia potete installare sia una versione con
GUI che una a console. Dovrete sempre prendere l'archivio con i propri file
di runtime.

vim61rt.zip	I file di runtime.

Usate il vostro programma di unzip per decomprimere i file. Ad esempio,
utilizzando il programma "unzip": **>**

```
cd c:\
unzip path\gvim61.zip
unzip path\vim61rt.zip
```

Ciò spacchetterà i file nella directory "c:\vim\vim61". Se avete già una
directory "vim", dovrete spostarvi nella directory immediatamente sopra
questa.
   Ora andate nella directory "vim\vim61" ed avviate il programma di
installazione: **>**

```
install
```

Guardate con attenzione i messaggi e selezionate le opzioni che intendete
utilizzare. Selezionando "do it" il programma di installazione eseguirà le
azioni da voi selezionate.
   Il programma di installazione non sposta i file di runtime. Essi rimangono
nella directory dove li avrete decompressi.

Nel caso non foste soddisfatti delle caratteristiche incluse nei binari,
potete provare a compilare Vim voi stessi. Recuperate l'archivio dei sorgenti
dallo stesso posto dove si trovano i binari. Avrete bisogno di un compilatore
per il quale esiste un makefile. Microsoft Visual C funziona, ma è costoso.
Si può utilizzare il compilatore gratuito a riga di comando Borland 5.5., così
come gli altrettanto gratuiti compilatori MingW e Cygwin. Consultate il file
src/INSTALLpc.txt per i suggerimenti.

==
90.3 Aggiornamento

Se utilizzate una sola versione di Vim e ne volete installare un'altra, ecco

cosa fare.

UNIX

Digitando "make install" i file di runtime verranno copiati in una directory specifica per questa versione. Così non sovrascriveranno una versione precedente. Ciò rende possibile usare due o più versioni di Vim l'una accanto all'altra.
L'eseguibile "Vim" tuttavia sovrascriverà una versione precedente. Se non vi interessa mantenere la versione più vecchia, eseguire "make install" andrà bene. Potete cancellare manualmente i vecchi file di runtime. Cancellate la directory con il numero di versione del caso e tutti i file in essa.
Ad esempio: **>**

```
rm -rf /usr/local/share/vim/vim58
```

Normalmente non ci sono file modificati sotto questa directory. Se avete modificato il file "filetype.vim", ad esempio, sarà meglio che uniate le modifiche nella nuova versione prima di cancellarla.

Se siete prudenti e volete provare la nuova versione prima di passare ad essa, installate la nuova versione sotto un altro nome. Dovrete specificare un argomento di configure. Ad esempio: **>**

```
./configure --with-vim-name=vim6
```

Prima di avviare "make install", potete usare "make -n install" per assicurarvi che nessun file esistente possa venir sovrascritto.
Quando alla fine decidete di passare alla nuova versione, basterà rinominare il file binario in "vim". Ad esempio: **>**

```
mv /usr/local/bin/vim6 /usr/local/bin/vim
```

MS-WINDOWS

L'aggiornamento è quasi uguale all'installazione di una nuova versione. Soltanto decomprimete i file nello stesso luogo della versione precedente. Una nuova directory verrà creata, ad esempio ,"vim61", per i file della nuova versione. I vostri file di runtime, i file vimrc, viminfo, etc verranno lasciati stare.
Se volete avviare la nuova versione acanto a quella vecchia, dovrete lavorare un pò. Non eseguite il programma di installazione, che sovrascriverebbe qualche file della vecchia versione. Lanciate i nuovi binari specificando il percorso completo. Il programma dovrebbe rintracciare automaticamente i file di runtime necessari per la versione corretta. Ad ogni modo, ciò non funzionerebbe se aveste impostato la variabile $VIMRUNTIME da qualche parte.
Se siete soddisfatti dell'aggiornamento, potete cancellare i file della versione precedente.

Consultate |90.5|.

==
90.4 Problemi comuni di installazione.

Questa sezione descrive alcuni problemi comuni che capitano durante l'installazione di Vim e fornisce alcune soluzioni. Contiene anche delle risposte a molte delle domande sull'installazione.

Q: Non ho i privilegi di root. Come posso installare Vim? (Unix)

Usate il seguente comando di configurazione per installare Vim in una directory chiamata $HOME/vim: **>**

```
./configure -prefix=$HOME
```

Ciò vi fornisce una copia personale di Vim. Dovrete inserire $HOME/bin nel vostro path per eseguire l'editor. Consultate anche |install-home|.

Q: I colori sul mio schermo non sono corretti. (Unix)

Controllate le impostazioni del vostro terminale utilizzando il seguente comando in una shell: **>**

```
echo $TERM
```

Se il tipo di terminale ottenuto non è va bene, correggetelo. Per ulteriori suggerimenti consultate |06.2|. Un'altra soluzione è di usare sempre la versione GUI di VIM, denominata gvim. Cio evita la necessità di una corretta installazione del terminale.

Q: I miei tasti Backspace e Delete Keys non funzionano correttamente

La definizione dei codici che i tasti inviano è piuttosto incerta per bs <bs> e Delete <Del>. Prima di tutto, controllate le impostazioni di $TERM. Se non c'è nulla di sbagliato, provate questo: >

```
:set t_kb=^V<BS>
:set t_kD=^V<Del>
```

Nella prima riga dovrete premere CTRL-V ed in seguito il tasto backspace.
Nella seconda linea dovrete premere CTRL-V e poi il tasto Delete.
Potete inserire queste righe nel vostro file vimrc, consultate |05.1|.
Uno svantaggio è che non funzionerà quando utilizzerete un altro terminale qualche volta. Guardate qui per soluzioni alternative: |:fixdel|.

Q: Uso RedHat Linux. Posso utilizzare la versione di Vim fornita nel sistema?

Di default Redhat installa una versione minimale di Vim. Controllate i vostri pacchetti RPM per reperire "Vim-enhanced-version.rpm" ed installatelo.

Q: Come posso attivare la colorazione della sintassi? Come faccio funzionare i plugin?

Usate lo script di esempio vimrc. Potete trovare la spiegazione sul suo uso qui: |not-compatible|.

Vedere il capitolo 6 per le informazioni circa la evidenziazione della sintassi:|usr_06.txt|.

Q: Qual'è un buon file vimrc da usare?

Vedere il sito www.vim.org per diversi buoni esempi.

Q: Dove trovo un buon plug-in per VIM?

Vedere il sito Vim.online: http://vim.sf.net. Molti utenti hanno inviato utili script e plugin.

Q: Dove posso trovare altri suggerimenti?

Vedere il sito Vim-online: http://vim.sf.net. C'è un archivio di suggerimenti per gli utenti di Vim. Potete anche consultare |maillist-archive|.

==
90.5 Disinstallare VIM.

Nell'improbabile ipotesi che vogliate disinstallare Vim completamente, questo è ciò che dovete fare.

UNIX

Se avete installato Vim come pacchetto, controllate il vostro gestore dei pacchetti per sapere come rimuoverlo.
   Se avete installato vim dai sorgenti potete usare il seguente comando: >

```
make uninstall
```

Comunque, se avete cancellato i file originali o avete usato un archivio fornitovi da qualcuno, non potete farlo. Dovrete cancellare i file a mano, di seguito un esempio di con "/usr/local" usato come root: >

```
rm -rf /usr/local/share/vim/vim61
rm /usr/local/bin/eview
rm /usr/local/bin/evim
rm /usr/local/bin/ex
rm /usr/local/bin/gview
```

```
rm /usr/local/bin/gvim
rm /usr/local/bin/gvim
rm /usr/local/bin/gvimdiff
rm /usr/local/bin/rgview
rm /usr/local/bin/rgvim
rm /usr/local/bin/rview
rm /usr/local/bin/rvim
rm /usr/local/bin/rvim
rm /usr/local/bin/view
rm /usr/local/bin/vim
rm /usr/local/bin/vimdiff
rm /usr/local/bin/vimtutor
rm /usr/local/bin/xxd
rm /usr/local/man/man1/eview.1
rm /usr/local/man/man1/evim.1
rm /usr/local/man/man1/ex.1
rm /usr/local/man/man1/gview.1
rm /usr/local/man/man1/gvim.1
rm /usr/local/man/man1/gvimdiff.1
rm /usr/local/man/man1/rgview.1
rm /usr/local/man/man1/rgvim.1
rm /usr/local/man/man1/rview.1
rm /usr/local/man/man1/rvim.1
rm /usr/local/man/man1/view.1
rm /usr/local/man/man1/vim.1
rm /usr/local/man/man1/vimdiff.1
rm /usr/local/man/man1/vimtutor.1
rm /usr/local/man/man1/xxd.1
```

MS-WINDOWS

Altrimenti, se avete installato Vim con l'archivio auto-installante potete avviare il programma "uninstall-gui" situato nella stessa directory degli altri programmi di Vim, ad esempio "c:\vim\vim61". Potete anche lanciarlo dal menù start se vi avete installato i collegamenti a Vim. Ciò rimuoverà la maggior parte dei file, voci nei menù e collegamenti sul desktop. Alcuni file possono rimanere comunque in quanto richiedono il riavvio di Windows prima di essere cancellati.
Vi sarà data la possibilità di rimovere l'intera directory di Vim. Essa probabilmente conterrà il vostro file vimrc ed altri file di runtime che avrete creato, perciò state attenti.

Se avete installato Vim dagli archivi zip, il metodo migliore è utilizzare il programma "uninstal" (notate la mancanza della "l" alla fine). Potete trovarlo nella stessa directory del programma "install", ad esempio "c:\vim\vim61". Ciò potrebbe funzionare anche dalla solita pagina "install/remove software".
Ad ogni modo, ciò rimuove solo le voci del registro relative a Vim. Dovrete poi cancellare i file a mano. Selezionate semplicemente la directory "vim\vim61" e cancellateli ricorsivamente. Non ci dovrebbero essere file modificati da voi, ma potreste voler controllare ciò prima.
La directory "vim" probabilmente contiene il vostro file vimrc ed altri file di runtime da voi creati. Potreste volerli conservare.

==

Indice: |usr_toc.txt|

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

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

message.txt   For Vim version 6.2. Ultima modifica: 2004 Apr 25

VIM REFERENCE MANUAL   by Bram Moolenaar
Traduzione di questo capitolo: Antonio Colombo

Questo file contiene una lista di messaggi, di errore e non, prodotti da Vim.
I messaggi sono listati in ordine alfabetico [...inglese -NdT].
Si può usare se il messaggio vi risulta poco chiaro. La lista è comunque
incompleta.

==
1. Messaggi passati   *:messages* *:mes* *message-history*

Il comando ":messages" si può usare per visualizzare i messaggi ricevuti in
precedenza. Questo può tornare utile quando dei messaggi siano stati
sovrascritti o troncati, a seconda del valore dall'opzione 'shortmess'.

Il numero di messaggi memorizzati è fissato in 20.

Se state usando messaggi tradotti, la prima linea stampata vi dice chi
mantiene i messaggi o la traduzione. L'indirizzo e-mail può permettervi di
contattare il manutentore, se rilevate degli errori.

Per trovare aiuto riguardo a uno specifico messaggio (di errore o meno), usate
l'identificativo posto all'inizio del messaggio. Per esempio per ottenre
aiuto sul messaggio: >

        E72: Close error on swap file

oppure (tradotto): >

        E72: Errore durante chiusura swap file

Usate: >

        :help E72

Se siete pigri, potete scrivere a lettere minuscole [ed abbreviare - NdT]: >

        :h e72

==
2. Messaggi di errore   *error-messages*

Quando viene visualizzato un messaggio di errore, ma viene rimosso prima che
siate riusciti a leggerlo, lo potete rivedere ancora con: >
  :echo errmsg
o vedere una lista di messaggi recenti con: >
  :messages

LISTA DI MESSAGGI
                        E222 *E228* *E232* *E256* *E293* *E298* *E304* *E317*
                        E318 *E356* *E438* *E439* *E440* *E316* *E320* *E322*
                        E323 *E341* *E473* *E570* >
  Aggiunto al buffer di lettura
  makemap: modo non consentito
  Non riesco a creare 'BalloonEval' con sia messaggio che callback
  ERRORE processore Hangul
  il blocco non era riservato
  Non riesco a leggere blocco numero 0?
  ml_timestamp: Non riesco a leggere blocco 0??
  ID blocco puntatori errato {N}
  Aggiornati troppi blocchi?
  ERRORE get_varp
  u_undo: numeri linee errati
  lista 'undo' non valida
  linea di 'undo' mancante
  ml_get: non riesco a trovare la linea {N}
  Non riesco a trovare la linea {N}
  numero linea non ammissibile: {N} dopo la fine
  contatore linee errato nel blocco {N}

  Errore interno
  errore irreparabile in cs_manage_matches

Errore interno in Vim. Se riuscite a riprodurlo, siete pregati di inviare un rapporto sull'errore. Si veda |bugs|.

>
  ATTENZIONE
  Trovato uno swap file di nome ...

Si veda |ATTENTION|.

E92 >
  Buffer {N} non trovato

Il buffer richiesto non esiste. Questo messaggio può essere invianto anche dopo la cancellazione di un buffer contenente un mark o referenziato in qualche altro modo. |:bwipeout|

E95 >
  C'è già un buffer con questo nome

Non si possono avere due buffer con lo stesso nome.

E72 >
  Errore durante chiusura swap file

Lo |swap-file|, usato per tenere una copia del testo in modifica non è stato chiuso regolarmente. Di solito non c'è da preoccuparsi.

E169 >
  Comando troppo ricorsivo

Capita quando un comand Ex esegue un comando Ex che esegue un comando Ex, etc. Questo è permesso fino a 200 volte. Superata questa soglia, siamo di solito in presenza di un loop infinito. Probabilmente è in esecuzione un comando |:execute| o |:source|.

E254 >
  Non riesco ad allocare il colore {nome}

Il colore di nome {nome} è sconosciuto. SI veda |gui-colors| per una lista di colori disponibili sulla maggior parte delle piattaforme.

E458 >

  Non riesco ad allocare elemento di colormap, possibili colori errati

Il messaggio indica che non ci sono abbastanza colori disponibili a Vim. Vim funziona lo stesso, ma alcuni colori non corrisponderanno alle specifiche. Provate a chiudere applicazioni che usano parecchi colori, o a farle partire dopo aver fatto partire gvim.
Netscape è un noto consumatore di colori. Potete evitare l'inconveniente chiedendogli di usare la sua mappa interna di colori: >
        netscape -install
O ditegli di limitarsi a un dato numero di colori (64 dovrebbero bastare): >
        netscape -ncols 64
La cosa si può anche definire nel vostro file Xdefaults: >
        Netscape*installColormap: Yes
oppure >
        Netscape*maxImageColors: 64
<
E79 >
  Non posso espandere 'wildcard'

Un nome di file contiene una combinazione strana di caratteri che Vim tenta di espandere come 'wildcard' senza riuscirci. Questo NON vuol dire che non ci siano nomi file corrispondenti all'espressione usata, ma solo che l'espressione stessa non è una espressione valida.

E459 >
  Non posso tornare alla directory precedente

Durante l'espansione di un nome file Vim non riesce a tornare alla directory precedentemente usata. Tutti i nome file in uso potrebbero di conseguenza non essere più validi! E' necessario (per usarla) di avere il permesso di esecuzione sulla directory corrente.

E190 *E212* >
  Non riesco ad aprire "{nome_file}" in scrittura
  Non posso aprire il file in scrittura

Pwr qualche ragione il file che state scrivendo non può essere creato o riscritto. Una ragione potrebbe essere che non siete autorizzati a scrivere nella directory o che il nome del file non è valido.

E166 >
  Non posso aprire il file collegato ('linked') in scrittura

State tentando di scrivere un file che non può essere riscritto, e il file è un 'link' (un 'hard link' [fra due file nello stesso File System − NdT] o un 'link simbolico' [fra due file nello stesso o in due diversi File System − Ndt]). Una scrittura del file potrebbe ancora essere possibile, ma Vim non sa se volete eliminare il 'link' all'altro file e scrivere il file come file separato, o se volete cancellare il file stesso e scrivere un nuovo file che lo sostituisca. Se veramente si vuole scrivere il file usando questo nome, occorre prima cancellare a mano il 'link' o il file "collegato", oppure cambiare le autorizzazioni di accesso al file, in modo che Vim sia in grado di riscriverlo.

E46 >
  Non posso impostare la variabile read−only "{nome}"

Si sta tentando di assegnare un valore a un argomento di funzione |a:var| o a una variabile interna di Vim |v:var| in sola lettura ('read−only').

E90 >
  Non riesco a scaricare l'ultimo buffer

Vim richiede sempre che ci sia un buffer da caricare, altrimenti non ci sarebbe nulla da visualizzare nella schermata.

E40 >
  Non riesco ad aprire il file errori {nome_file}

Usando i comandi ":make" o ":grep": Il file usato per salvare i messaggi di errore o l'output del comando grep non può essere aperto. Ci possono essere molti motivi per questo:
− 'shellredir' è impostato in maniera incorretta.
− Lo shell cambia directory, e quindi il file di errore viene scritto in un'altra directory. Questo si può correggere cambiando 'makeef', ma così il comando viene ancora eseguito nella directory sbagliata.
− 'makeef' è impostato in maniera incorretta.
− I programmi indicati in 'grepprg' o 'makeprg' non si sono potuti eseguire. Questo non è sempre facile da determinare (specialmente in MS−Windows). Controllare il vostro $PATH.

>
  Non riesco ad aprire il file C:\TEMP\{nome_file}.TMP

In MS−Windows, questo messaggio è visualizzato quando si intende leggere l'output di un comando, ma il comando non è stato eseguito con successo. Le cause possono essere molteplici. Controllare le opzioni 'shell', 'shellquote', 'shellxquote', 'shellslash' e quelle ad esse collegate. Potrebbe anche darsi che il comando esterno non sia stato trovato, non c'è un messaggio di errore specifico per segnalare quest'ultima eventualità.

E12 >
  Comando non ammesso da exrc/vimrc nella dir. in uso o nella ricerca tag

Alcuni comandi non sono permessi per motivi di sicurezza. Questi comandi provengono da un file .exrc or .vimrc nella directory corrente, oppure da un file di tag. Si veda anche 'secure'.

E74 >
  Comando troppo complesso

Una mappatura ha prodoto una stringa di comandi molto lunga. Potrebbe dipendere da una mappatura che indirettamente invoca se stessa.

>
  ERRORE DI CONVERSIONE

Quando si sta scrivendo un file ed il testo "ERRORE DI CONVERSIONE" viene visualizzato, significa che alcuni bit sono andati persi nella conversione del testo dalla codifica UTF−8, utilizzata internamente da Vim, alla codifica

usata nel file. Il file verrà marcato come "modificato" [cioè come se non fosse stato ancora salvato − NdT]. Se la perdita di informazione è rilevante, impostare l'opzione 'fileencoding' ad un valore differente, in grado di gestire i caratteri nel buffer, e riscrivere amcora. Se invece la cosa non ha importanza si può abbandonare il buffer oppure resettare l'opzione 'modified'.

E302 >
  Non riesco a rinominare lo swap file

Quando il nome del file in modifica viene cambiato, Vim prova a rinominare anche il relativo |swap-file|.
Se non ci riesce, continua ad usare il vecchio swap file. Messaggio quasi sempre innocuo.

E43 *E44* >
  Stringa di confronto danneggiata
  Programma 'regexp' corrotto

Qualcosa è andato storto all'interno di Vim, ed ha generato un'espressione regolare non valida. Se riuscite a riprodurre questo problema, per favore, inviateci un rapporto. Si veda |bugs|.

E208 *E209* *E210* >
  Errore in scrittura di "{nome_file}"
  Errore in chiusura di "{nome_file}"
  Errore in lettura di "{nome_file}"

Vim sta tentando di rinominare un file, ma un semplice 'rename' non basta. Allora si tenta di copiare il file, ma anche questo per qualche motivo non funziona. Il risultato può essere che sia il file d'origine che quello di destinazione esistono, ed il file di destinazione può risultare danneggiato.

>
  Vim: Errore leggendo l'input, esco...

Vim non riesce a leggere dei caratteri immessi dall'utente, quando ha bisogno di input. Vim è bloccato, la sola cosa che può fare è terminare l'esecuzione. Può succedere quando sia 'stdin' che 'stdout' sono rediretti, durante l'esecuzione di uno 'script', il quale al suo interno non chiude Vim.

E47 >
  Errore leggendo il file errori

Vim non è riuscito a leggere il file errori. La cosa NON è correlata con la mancata individuazione di un messaggio di errore.

E80 >
  Errore in scrittura

La scrittura di un file non è andata a buon fine. Il file è probabilmente incompleto [dopo essere stato riscritto NdT].

E13 *E189* >
  File esistente (aggiungi ! per riscriverlo)
  "{nome_file}" esiste (aggiungi ! per eseguire comunque)

Vim protegge dalla riscrittura involontaria di un file, Se volete davvero scriverci sopra, usate lo stesso comando, ma aggiungete un "!" subito dopo il comando stesso.
Esempio: >
        :w /tmp/test
diviene: >
        :w! /tmp/test
<
E139 >
  File già caricato in un altro buffer

Tentativo di scrivere un file con un nome che è già in uso in un altro buffer. Così verreste ad avere due versioni dello stesso file [in modifica NdT].

E142 >
  File non scritto: Scrittura inibita da opzione 'write'

L'opzione 'write' è inibita. Per questo motivo tutti i comandi che tentano di scrivere un file generano questo messaggio. La cosa potrebbe dipendere da una opzione |-m| specificata alla partenza di Vim. Potete abilitare l'opzione 'write' dando il comando ":set write".

E25 >
GUI non utilizzabile: Non abilitata in compilazione

La versione di Vim che state usando non contiene la parte GUI. Ragion per cui "gvim" e ":gui" non sono disponibili.

E49 >
Quantità di 'scroll' non valida

Valore non permesso rilevato per le opzioni 'scroll', 'scrolljump' o 'scrolloff'.

E17 >
"{nome_file}" è una directory

Tentativo di scrivere un file con lo stesso nome di una directory esistente. Non è sensato. Probabilmente si deve aggiungere in fondo un nome file.

E19 >
'Mark' con numero linea non valido

Uso di un 'mark' che indica un numero di linea inesistente. Ciò può accadere quando avete un 'mark' che si riferisce ad un altro file, dal quale qualche altro programma ha tolto delle linee.

E219 *E220* >
Manca {.
Manca }.

Usando la notazione {} per indicare un nome file, c'è una "{" senza una corrispondente "}" o viceversa. Andrebbe usato così: {foo,bar}. Questo indica sia "foo" che "bar".

E315 >
ml_get: numero linea non valido:

Questo è un errore interno di Vim. Cercate di capire come riprodurlo, ed inviate un rapporto di errore. Si veda |bugreport.vim|.

E173 >
ancora {numero} file da elaborare

Avete chiesto di uscire da Vim, ma almeno uno dei file sui quali avete chiesto di lavorare non è stato acceduto. Vim evita di uscire prima di aver finito di lavorare sui file richiesti. Si veda |argument-list|. Se volete uscire lo stesso da Vim, ripetete il comando, che questa volta verrà eseguito.

E23 *E194* >
Nessun file alternato
Nessun nome file alternativo da sostituire a '#'

Il file alternativo non è ancora stato definito. Si veda |alternate-file|.

E32 >
Manca nome file

Il buffer in uso non ha un nome. Per scriverlo, usate ":w nome_file". Oppure assegnate un nome al buffer con ":file nome_file".

E141 >
Manca nome file per il buffer {numero}

Uno dei buffer modificati non ha un nome file associato. Ragion per cui non può essere scritto. Dovete assegnare un nome file al buffer: >
:buffer {numero}
:file {nome_file}
<

E33 >
Nessuna espressione regolare precedente di 'substitute'

Se si usa il carattere '~' in una espressione regolare, questo viene sostituito dall'espressione precedentemente utilizzata in un comando ":substitute". Se non si è ancora usato il predetto comando, la sostituzione non può avvenire. Si veda |/~|.

E35 >
Nessuna espressione regolare precedente

Specificando una espressione di ricerca vuota, viene usata l'ultima che era stata specificata. Se non si era fatta in precedenza alcune ricerca, non ne esiste nessuna.

E24 >
  Abbreviazione inesistente

Avete usato un comando ":unabbreviate" specificando un argomento che non è una delle abbrevizioni definite. Tutte le variazioni di questo comando danno questo stesso messaggio: ":cunabbrev", ":iunabbrev", etc. Controllate se avete messo degli spazi alla fine della abbreviazione.

>
  /dev/dsp: No such file or directory

Errore possibile solo usando la GUI GTK con supporto Gnome. Gnome tenta di usare un dispositivo audio, ma questo non è presente. Potete ignorare questo errore.

E31 >
  Mapping inesistente

Avete usato un comando ":unmap" specificando un argomento che non è il nome di una mappatura definita. Tutte le variazioni di questo comando danno lo stesso messaggio: ":cunmap", ":unmap!", etc. Controllate se avete messo degli spazi alla fine della mappatura.

E37 *E89* >
  Non salvato dopo modifica (aggiungi ! per eseguire comunque)
  Buffer {numero} non salvato dopo modifica (aggiungi ! per eseguire comunque)

Avete chiesto di uscire senza salvare le modifiche |abandon| per un file che è stato modificato. Vim tenta di evitarvi di perdere il lavoro fatto. Potete scrivere il file modificato con ":w", o, se questa è la vostra intenzione, uscire senza salvare, |abandon|, perdendo tutte le modifiche. Ciò si ottiene aggiungendo un carattere '!' in fondo al comando che avete appena dato.
As esempio: >
        :e un_altro_file
diventa: >
        :e! un_altro_file
<

E162 >
  Buffer "{nome}" non salvato dopo modifica

Il messaggio viene inviato se volete uscire da Vim, dopeo aver modificato uno o più buffer. Dovete scrivere i buffer modificati (con |:w|), oppure usare un comando per ignorare le modifiche. ad es., con ":qa!". Attenzione a non buttar via modifiche che invece intendevate tenere. Potreste essrvi dimenticati di un buffer, specialmente se è stata impostata l'opzione 'hidden' per nasconderlo.

E38 >
  Argomento nullo

Qualcosa è andato male all'interno di Vim, generando un puntatore invalido (NULL pointer). Se sapete come riprodurre l'errore, siete pregati di inviate un rapporto di errore. Si veda |bugs|.

E172 >
  Ammesso un solo nome file

Il comando ":edit" accetta un solo nome di file. Se volete specificare più di un file da trattare, usate ":next" |:next|.

E41 *E82* *E83* *E342* >
  Non c'è più memoria!
  Non c'è più memoria! (stavo allocando {numero} bytes)
  Non riesco ad allocare alcun buffer, esco...
  Non riesco ad allocare un buffer, uso l'altro...

Oh, oh. Stavate facendo qualcosa di complesso, oppure qualche altro programma sta consumendo la memoria del computer. Attemzione! Vim non è in grado di gestire TUTTE le situazioni di mancanza di memoria. Per prima cosa accisuratevi che le vostre modifiche siano state salvate. Solo dopo tentate di risolvere il problema di mancanza di memoria. Per andare sul sicuro, uscite da Vim, e ricominciate da capo. Si veda anche |msdos-limitations|.

E339 >

  Espressione troppo lunga

Càpita solo con sistemi che usano 16 bit per rappresentare i numeri interi: il modello di espressione regolare compilato supera i 65000 caratteri circa. Provate ad usare un'espressione regolare più breve.

E45 >
  file in sola lettura (aggiungi ! per eseguire comunque)

State tentanto di scrivere un file marcato come non modificabile, Per scrivere lo stesso il file, annullate l'opzione 'readonly' oppure aggiungete il carattere '!' subito dopo il comando che avete dato. Ad es.: >
        :w
va modificato in: >
        :w!
<

E294 *E295* *E301* >
  Errore leggendo swap file
  Errore di posizionamento durante lettura swap file
  Ahimè, lo swap file è perduto!!!

Vim ha provato a leggere del testo dallo |swap-file|, ma qualcosa non ha funzionato. Il testo nel buffer interessato può essersi corrotto! Controllate attentamente prima di riscrivere il buffer. Potreste preferire riscriverlo con un altro nome file, e controllare le differenze [rispetto al file di partenza].

E192 >
  Uso ricorsivo di :normal troppo esteso

Sate usando un comando ":normal", il cui arggomento usa un altro comando ":normal" in maniera ricursiva. La cosa è limitata a 'maxmapdepth' livelli. Il seguente esempio mostra come produrre questo messaggio: >
        :map gq :normal gq<CR>
Se battete "gq", verrà invocata la mappatura, che invoca nuovamente "gq".

E22 >
  Script troppo nidificati

File di comandi Vim si possono richiamare tramite l'argomento "-s" quando si manda in esecuzione Vim, e con il comando ":source". Questo file di comandi può a sua volta leggere un altro file di comandi. Questo può avvenire fino a un massimo di 14 livelli. Quando si supera tale soglia. Vim assume che si sia in presenza di un ciclo ricursivo [errato - NdT] da qualche parte. e si ferma con questo messaggio di errore.

E319 >
  Spiacente, comando non disponibile in questa versione

Avete dato un comando non presente nella versione di Vim che state usando. Durante la compilazione di Vim, è possibile abilitare o escludere molte diverse componenti. Questo dipende da quanto grande si sceglie di generare Vim, come pure dal Sistema Operativo. Si veda |+feature-list| per stabilire se questa componente è disponibile. Il comando |:version| mostra con quali componenti Vim è stato compilato.

E300 >
  Swap file already exists (symlink attack?)
  Lo swap file esiste già (un link simbolico?)

Questo messaggio viene inviato quando vim prova ad aprire un file di swap che risulta esistere già, oppure che risulta essere un link simbolico. Questo non dovrebbe succedere, perché Vim ha già controllato che il file non esiste ancora. O qualcun altro ha aperto lo stesso file esattamente nello stesso momento (molto improbabile), oppure qualcuno sta tentando un attacco al vostro file tramite "symlink" (potrebbe capitare modificando un file nella directory /tmp oppure quando l'opzione 'directory' inizia con "/tmp", che è una cattiva scelta).

E432 >
  Tag file non ordinato alfabeticamente: {nome_file}

Vim (e Vi) si aspettano che i file di tag siano ordinati secondo la sequenza ASCII. Ciò permette di usare una ricerca binaria, molto più veloce che non quella lineare. Se i vostri file di tag non sono in questo ordine, reimpostate adeguatamente l'opzione |'tagbsearch'|.
Questo messaggio viene visualizzato solo se Vim rileva dei problemi mentre sta cercamdo una tag. Talora il messaggio non viene inviato, anche se il file di

tag non è correttamente ordinato.

E460 >
La 'fork' sulla risorsa verrebbe persa (aggiungi ! per eseguire comunque)

Nel Macintosh (classico), scrivendo un file, Vim tenta di mantenere ogni informazione relativa al file, compreso il suo "fork" sulla risorsa. Se ciò non è possibile, vedrete questo messaggio di errore. Agggiungete "!" al nome del comando per scrivere lo stesso (perdendo l'informazione).

E424 >
Troppi gruppi evidenziazione differenti in uso

Vim riesce a gestire solo circa 223 generi diversi di evidenziazione. Se superare questo limite, avete usato troppi comandi |:highlight| con diversi argomenti. I comandi ":highlight link" NON contribuiscono al totale.

E77 >
Troppi nomi file

Espandendo un nome file, si è trovata più di un nome che corrisponde. Un solo nome file è richiesto per il comando che era stato dato.

E303 >
Non riesco ad aprile lo swap file per "{nome_file}", recupero impossible

Vim non è riuscito a creare un file di swap. Potete ancora modificare il file, ma se Vim termina in maniera imprevista, le modifiche andranno perse. Vim [lavorando solo in memoria NdT] consumerà un mucchio di memoria se modificate un grosso file. Potreste modificare l'opzione 'directory' per evitare questo errore. Si veda |swap-file|.

E140 >
Usa ! per scrivere il buffer incompleto

Se si specifica un intervallo di linee per scrivere parte di un buffer, di solito non si intende ricoprire il file originale. Si tratta probabilmente di un errore (ad es. se la modalità Visuale era attiva e avete dato il comando ":w") e perciò Vim vi chiede di usare un ! dopo il comando, ad. es: ":3,10w!".
>

Warning: Cannot convert string "<Key>Escape,_Key_Cancel" to type VirtualBinding

Messaggi di questo tipo possono apparire quando si invoca Vim. Non si tratta di un problema di Vim, è la vostra configurazione X11 ad essere errata. Si possono trovare suggerimenti per risolvere di questo problema consultando: http://groups.yahoo.com/group/solarisonintel/message/12179.

W10 >
Attenzione: Modifica a un file in sola-lettura

Il file è in sola lettura e voi lo state cambiando lo stesso. Potete usare L'autocomando |FileChangedRO| per evitare questo messaggio (l'autocomando deve cambiare l'opzione 'readonly'). Si veda 'modifiable' per impedire completamente qualsiasi cambiamento a un file.

W13 >
Attenzione: Il file "{nome_file}" risulta creato dopo l'apertura

State modificando con Vim un file che non esisteva prima, ma che è rilevato esistere ora. Dovete decidere se volete tenere la versione in modifica con Vim o il file creato nel frattempo. Questo messaggio non viene inviato quando 'buftype' non è nullo.

W11 >
Attenzione: File "{nome_file}" modificato dopo l'apertura

Il file che avete cominciato a modificare è stato nel frattempo modificato ed il suo contenuto è cambiato (più precisamente: Se leggeste ancora il file con le impostazioni correnti di opzioni e di autocomandi, otterreste un testo differente). Questo probabilmente vuol dire che qualche altro programm ha modificato il file. A voi comprendere cosa è successo, e decidere quale versione del file volete conservare. Impostate l'opzione 'autoread' se volete che questo avvenga automaticamente. Questo messaggio non viene inviato quando 'buftype' non è nullo.

In un caso potete ricevere questo messaggio anche se non ci sono problemi: Se salvate il file in Windows nel momento in cui comincia l'ora legale. La cosa si può ovviare in uno dei modi seguenti:
- aggiungete questa linea al vostro autoexec.bat: >
```
SET TZ=-1
```
< Modificate il "-1" a seconda del vostro fuso orario.
- Disabilitate "Passa automaticamente all'ora legale".
- Riscrivete ancora il file il giorno successivo. Oppure impostate la data del PC al giorno dopo, scrivete il file due volte, e tornate indietro con la data del PC.

W12 >
Attenzione: File "{nome}" modificato su disco ed anche nel buffer di Vim

Come il messaggio precedente, e il buffer per il file è modificato anche in Vim. Dovete decidere se tenere buona la versione in modifica con Vim o quella su disco. Questo messaggio non viene inviato quando 'buftype' non è nullo.

W16 >
Attenzione: Modo File "{nome_file}" modificato dopo l'apertura

Se c'è stata una modifica che riguarda il modo (i permessi) del file, ma non il suo contenuto. Succede quando si richiede di modificare un file usando un software di controllo versione sorgenti, che reimposta il bit di "sola lettura". Non ci dovrebbero essere problemi a ricaricare il file. Impostate 'autoread' se volete che il file sia ricaricato automaticamente.

E211 >
Attenzione: Il file "{nome_file}" non esiste più

Il file che avete cominciato a modificare è scomparso, o non è più accessibile. Scrivete il buffer altrove per evitare di perdere le modifiche. Questo messaggio non viene inviato quando 'buftype' non è nullo.

W14 >
Attenzione: Superato limite della lista dei nomi di file

State usando veramente tanti buffer. È ora possibile che due buffer abbiano lo stesso numero, il che genera vari problemi. Sarebbe consigliabile uscire da Vim e ricominciare da capo.

E296 *E297* >
Errore di posizionamento scrivendo swap file
Errore scrivendo swap file

Questo succede per lo più quando non c'è spazio su disco. Vim non è riuscito a scrivere del testo nello |swap-file|. Questo non è pericoloso in sé, ma se Vim termina in maniera imprevista, una parte del testo può andare perduta, senza possibilità di ricuperarla. Vim potrebbe anche esaurire la memoria disponbile al programma, laddove questo problema si ripresenti più volte.

connection-refused >
Xlib: connection to "<machine-name:0.0" refused by server

Questo succede quando Vim prova a connettersi al server X, ma il server X non permette la connessione. La connessione al server X è necessaria per poter modificare il titolo della sessioe e per supportare la "clipboard" del terminale X (xterm). Sfortunatamente questo messaggio d'errore non è eliminabile, senza inibire le componenti |+xterm_clipboard| e |+X11|.

E10 >
\\ dovrebbe essere seguito da /, ? oppure &

Una linea comando inizia con un "\" o l'intervallo di linee di un comando ha un "\" in un punto sbagliato. Questo succede spesso se la continuazione di linee comando è disabilitata. Togliete la flag 'C' dall'opzione 'cpoptions' per abilitarla.

E471 >
Argomento necessario

Succede dopo aver eseguito un comando di Ex che richiede uno o più argomenti, ma non ne era stato specificato nessuno.

E474 *E475* >
Argomento non valido

È stato eseguito un comando di Ex, al quale era stato specificato un argomento

non valido.

E488 >
  Caratteri in più a fine comando

Un argomento è stato specificato per un comando Ex che non ne ammette.

E477 *E478* >
  ! non consentito
  Non lasciarti prendere dal panico!

Avete aggiunto un "!" ad un comando Ex che non ne ammette.

E481 >
  Nessun intervallo consentito

Un intervallo è stato specificato per un comando Ex che non ne ammette.
Si veda |cmdline-ranges|.

E482 *E483* >
  Non riesco a creare il file {nome_file}
  Non riesco ad ottenere nome file 'temp'

Vim non riesce a creare un file temporaneo.

E484 *E485* >
  Non riesco ad aprire il file {nome_file}
  Non riesco a leggere il file {nome_file}

Vim non riesce a leggere un file temporaneo.

E464 >
  Uso ambiguo di comando definito dall'utente

Esistono [almeno NdT] due comandi definiti dall'utente con un prefisso comune,
ed avete usato il completamento linea-Comando per eseguirne uno.
|user-cmd-ambiguous|
Esempio: >
        :command MioComando1 echo "uno"
        :command MioComando2 echo "due"
        :MioComando
<
E492 >
  Non è un comando dell'editor

Avete tentato di eseguire un comando che non è un comando Ex e neppure un
comando definito dall'utente.

==
3. Messaggi *messages*

Questa è una panoramica (incompleta) di vari messaggi inviati da Vim:

hit-enter *press-enter* *hit-return* *press-return* >

  Batti INVIO o un comando per proseguire

Questo messaggio è inviato quando c'è qualcosa sullo schermo che dovreste
leggere, prima che lo schermo venga ridisegnato:
- Dopo aver eseguito un comando esterno (ad es., ":!ls" e "=").
- Sulla linea di stato va visualizzato un messaggio più lungo della linea
  medesima, o che "invade" le aree contenenti messaggi 'showcmd' o 'ruler'.

-> Battete <Invio> o <Spazio> per ridisegnare lo schermo e proseguire, senza
   che il tasto battuto venga usato in altro modo.
-> Battete ":" o qualunque comando in modalità Normale, per cominciare ad
   eseguire il comando stesso.
-> Battete <C-Y> per copiare (yank - acciuffare) una selezione senza modalità
   nel registro di "clipboard".
-> Usate un menu. I caratteri definiti per la modalità Linea-Comando sono
   usati.
-> Quando l'opzione 'mouse' contiene la flag 'r', cliccare il bottone di
   sinistra del mouse equivale a battere <Spazio>. Questo rende peraltro
   impossibile selezionare del testo.
-> Per la GUI cliccare il bottone di sinistra del mouse sull'ultima linea è
   equivalente a premere <Spazio>.
{Vi: solo i comandi ":" sono interpretati}

Per ridurre il numero di richieste "Batti INVIO":
- Impostate 'cmdheight' a 2 o più.
- Aggiungete flag a 'shortmess'.
- Annullate opzioni 'showcmd' e/o 'ruler'.

Si veda anche 'mouse'. Il messaggio "Batti INVIO" è evidenziato col gruppo sintattico |hl-Question|.

more-prompt *pager* >

```
-- Ancora --
-- Ancora -- (RET: linea, SPAZIO: pagina, d: mezza pagina, q: esci)
-- Ancora -- (RET/BS: linea, SPAZIO/b: pagina, d/u: mezza pagina, q: esci)
```

Questo messaggio viene inviato se lo schermo è pieno di messaggi. È inviato solo quando l'opzione 'more' è attivata. È evidenziato col gruppo |hl-MoreMsg|.

Battete	effetto ~
<CR> o <NL> o j o <Giu>	una linea avanti
<BS> o k o <Su>	una linea indietro (*)
<Spazio> or <PageDown>	pagina seguente
b o <PageUp>	pagina precedente (*)
d	mezza pagina in giù (down)
u	mezza pagina in su (up) (*)
q, <Esc> o CTRL-C	basta con la lista
:	basta con la lista ed immetti linea comando
<C-Y>	yank (copia) una selezione senza modalità alla "clipboard" (registri "* e "+)
{scelta-menu}	quel che il menu definisce, in modalità Linea-Comando
<LeftMouse> (**)	pagina seguente

Ogni altro tasto provoca la visualizzazione del messaggio con le possibili scelte.

(*) Tornare indietro in una lista è supportato solo per questi comandi: >
    :clist
(**) Cliccare il bottone di sinistra del mouse funziona solo:
     - Per la GUI: nell'ultima linea dello schermo.
     - Se la flag 'r' è inclusa nell'opzione 'mouse' (ma in questo caso la selezione di testo non funziona).

Note: Il tasto battuto è ottenuto direttamente dal terminale, non è mappato ed eventuali altri caratteri immessi vengono ignorati.

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

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