usr_07.txt   Per Vim version 6.3. Ultima modifica: 2004 Mar 12

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.

|07.1| Elaborare un altro file
|07.2| Una lista di file
|07.3| Saltare da file a file
|07.4| File di backup
|07.5| Copiare testo tra più file
|07.6| Visualizzare un file
|07.7| Rinominare un file

  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 at libero.it