usr_21.txt   Per Vim version 7.0. Ultima modifica: 2006 Giu 21

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.

==
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 finché 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. Poiché 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. Ad es. 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
```

<
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": **>**

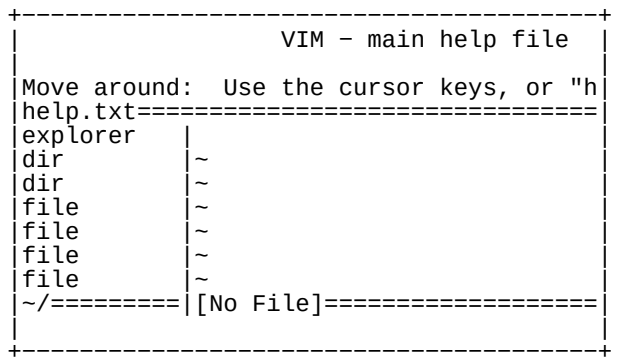
```
:wall
:mksession! ~/.vim/segreto.vim
:source ~/.vim/noioso.vim
```

La prima usa ":wall" per scrivere tutti i file 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:



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 clic 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, perché 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

Segnalare refusi a Bartolomeo Ravera - E-mail: barrav at libero.it
oppure ad Antonio Colombo - E-mail: azc100 at gmail.com