editing.txt Per Vim version 7.4. Ultima modifica: 2016 Aug 06

VIM Manuale di Riferimento di Bram Moolenaar
Traduzione di questo testo: Marco Curreli e Antonio Colombo

Editare file *edit-files*

1. Introduzione |edit-intro|
2. Editare un file |edit-a-file|
3. La lista degli argomenti |argument-list|
4. Scrivere |writing|
5. Scrivere e uscire |write-quit|
6. Dialoghi |edit-dialogs|
7. La directory corrente |current-directory|
8. Editare file binari |edit-binary|
9. Cifratura |encryption|
10. Marcature orarie |timestamps|
11. Ricerca di file |file-searching|

==========

1. Introduzione *edit-intro*

Editare un file con Vim significa:

1. Leggere il file in un buffer
2. Modificare il buffer con comandi dell'editor
3. Scrivere il buffer su un file

current-file
Finché il relativo buffer non viene scritto, il file originale resta invariato. Se si inizia a modificare un file (leggendo il file in un buffer), il nome del file è ricordato come "nome del file corrente". Lo stesso nome è anche il nome del buffer corrente. Può essere usato indicandolo con "%" sulla riga dei comandi |:_%|.

alternate-file
Se già c'era un "nome del file corrente" allora il nome del buffer diventa il nome alternativo del file. Può essere usato indicandolo con "#" sulla riga dei comandi |:_#| e si può usare il comando |CTRL-^| per passare dal nome file corrente a quello alternativo e viceversa.
Tuttavia, il nome alternativo del file non viene cambiato se si usa |:keepalt|. Per ogni finestra è ricordato un nome alternativo del file.

:keepalt *:keepa*
:keepalt **{comando}** Esegue **{comando}**, mantenendo il nome alternativo corrente del file. Nota I comandi richiamati indirettamente (p.es., con una funzione) possono ugualmente impostare il nome alternativo del file. {non in Vi}

Tutti i nomi di file sono ricordati nella lista dei buffer. Quando si immette un nome di file, per modificarlo (p.es., con ":e filename") o per scriverlo (p.es., con ":w filename"), il nome di file è aggiunto alla lista. Si può usare la lista dei buffer per ricordare quali file si stanno editando e per passare velocemente da un file all'altro (p.es., per copiare del testo) con il comando |CTRL-^|. Occorre prima immettere il numero del file e poi battere **CTRL-^**.
{Vi: viene ricordato un solo nome alternativo del file}

CTRL-G o *CTRL-G* *:f* *:fi* *:file*
:f[ile] Stampa il nome del file corrente (come era stato specificato, a meno che non sia stato usato il comando ":cd"), la posizione del cursore (a meno che non sia attiva l'opzione 'ruler'), e lo stato del file (in sola lettura, modificato, errori in lettura, file nuovo). Vedere l'opzione 'shortmess' per come ridurre la lunghezza di questo messaggio.
{Vi non include il numero di colonna}

:f[ile]! Come |:file|, ma il nome file non viene troncato,

anche se richiesto dall'opzione **'shortmess'**.

{contatore}CTRL-G Come **CTRL-G**, ma stampa il nome del file corrente con il percorso completo. Se il contatore è maggiore di 1 viene anche stampato il numero di buffer corrente. {non in Vi}

g_CTRL-G *word-count* *byte-count*

g **CTRL-G** Stampa la posizione corrente del cursore, in cinque modi: Colonna, Riga, Parola, Carattere e Byte. Se il numero di caratteri e quello dei byte è lo stesso, il numero dei caratteri viene omesso.
Se ci sono caratteri sulla riga che occupano più di una posizione sullo schermo (**<Tab>** o caratteri speciali), sia la colonna "vera" che quella che appare sullo schermo sono indicate, separate da un trattino.
Vedere anche l'opzione **'ruler'** e la funzione |**wordcount()**|.
{non in Vi}

v_g_CTRL-G

{Visual}g **CTRL-G** Simile a "g **CTRL-G**", ma sono visualizzati i contatori di Parole, Caratteri, Righe, e Byte relativi alla regione selezionata in modo Visual.
In modo Blockwise, viene visualizzato anche il contatore di Colonna. (Per **{Visual}** vedere |**Visual-mode**|.)
{non in VI}

:file_f

:f[ile][!] **{nome}** Imposta il nome del file corrente a **{nome}**. Il ! opzionale evita di abbreviare il messaggio di risposta, come succederebbe con |**:file**|.
Se il buffer aveva un nome, quel nome diventa il nome alternativo del file |**alternate-file**|. Un buffer non visualizzato viene creato per depositarvi il nome precedente del file.

:0file

:0f[ile][!] Rimuove il nome del buffer corrente. Il ! opzionale evita di abbreviare il messaggio di risposta, come succederebbe con |**:file**|.
{non in Vi}

:buffers
:files
:ls Elenca tutti i nomi di file conosciuti in quel dato momento. Vedere 'windows.txt' |**:files**| |**:buffers**| |**:ls**|. {non in Vi}

Vim ricorda il percorso completo di ogni nome di file immesso. Nella maggior parte dei casi quando il nome del file è visualizzato viene mostrato solo il nome immesso, ma se in precedenza è stato usato il comando ":cd" |**:cd**| viene mostrato il percorso completo.

home-replace

Se la variabile d'ambiente $HOME è impostata, e il nome del file inizia con con tale stringa, il file viene spesso visualizzato con il valore di $HOME sostituito da "~". Ciò si fa per mantenere corti i nomi di file.
Quando si leggono o scrivono file, si usa in ogni caso il nome completo, "~" è usata solo nella visualizzazione dei nomi di file. Quando la sostituzione del nome file (completo) risulta essere solo "~", si usa invece "~/" (per evitare confusione fra opzioni impostate a $HOME, mentre **'backupext'** è impostato a "~").

Quando si scrive il buffer, il default è di usare il nome del file corrente. Quindi, se si dà il comando "ZZ" o ":wq", il file originale sarà sovrascritto. Se non si desidera ciò, il buffer può essere scritto con un altro nome di file, specificando un nome di file come argomento al comando ":write". Per esempio: >

```
vim testfile
[modificare il buffer con comandi dell'editor]
```

```
:w nuovofile
:q
```

In questo modo si crea un file "nuovofile", che è una copia modificata di "testfile". Il file "testfile" resterà non modificato. In ogni caso, se l'opzione **'backup'** è impostata, Vim rinomina o copia il file originale, prima di sovrascriverlo. Si può usare questo file se si scopre di aver bisogno del file originale. Vedere anche l'opzione **'patchmode'**. Il nome del file di backup è di solito uguale al nome del file originale, con aggiunto alla fine il valore dell'opzione **'backupext'**. Il valore di default "~" è un po' insolito per evitare di sovrascrivere accidentalmente file esistenti. Se si preferisce ".bak", si modifichi il valore dell'opzione **'backupext'**. Altri '.' presenti nel nome file sono rimpiazzati con '_' in sistemi MS-DOS, quando Vim determina che un file-system di tipo MS-DOS è in uso (p.es., messydos o crossdos) o quando l'opzione **'shortname'** è attiva. Il file di backup può essere scritto in un'altra directory impostando l'opzione **'backupdir'**.

auto-shortname

Dettagli: Amiga permette fino a 30 caratteri per un nome di file. Ma su un file-system compatibile con MS-DOS sono disponibili solo 8 più 3 caratteri. Vim tenta di determinare il tipo di file-system al momento della creazione del file .swp. Se viene deciso che il file-system è di tipo MS-DOS, viene impostato un flag che ha lo stesso effetto di attivare l'opzione **'shortname'**. Questo flag sarà reimpostato non appena si inizia a editare un nuovo file. Il flag sarà usato per formare il nome dei file ".swp" e ".~" per il file corrente. Ma se si sta modificando un file in un file-system normale e si scrive su un file-system di tipo MS-DOS il flag non può essere impostato. In tal caso, la creazione del file ".~" può fallire, e verrà inviato un messaggio di errore. In questo caso va usata l'opzione **'shortname'**.

Se si inizia a editare senza specificare un nome di file, nei messaggi viene visualizzato "Senza nome". Se si usa il comando ":write" con un nome di file come argomento, il nome del file corrente è impostato a quel nome. Ciò avviene solo se il flag 'F' è incluso in **'cpoptions'** (per default è incluso) |**cpo-F**|. Ciò è utile se si immette del testo in un buffer vuoto per poi scriverlo assegnando un nome al file da scrivere. Se **'cpoptions'** contiene il flag 'f' (per default NON è incluso) |**cpo-f**| il nome del file è impostato per essere usato da un successivo comando ":read nome-file". Ciò è utile quando si richiama Vim senza fornire un argomento e poi si dà il comando ":read nome-file" per iniziare a editare un file.
Quando il nome del file è stato impostato e l'opzione **'filetype'** non è stata assegnata (è vuota) verranno eseguiti gli autocomandi che cercano di determinare il tipo del file.

not-edited

Poiché il nome del file è stato impostato senza veramente iniziare a editare quel particolare file, è presente una protezione che cerca di evitare di lasciar sovrascrivere quel file. Ciò viene fatto attivando il flag "notedited". Si può vedere se questo flag è impostato coi comandi **CTRL-G** o ":file". Nel messaggio compare "[Non elaborato]" quando il flag "notedited" è impostato. Quando si forza la scrittura del buffer usando il nome del file corrente (con ":w!"), il flag "notedited" è disattivato.

abandon

Vim ricorda se il buffer è stato modificato. C'è una protezione per evitare di perdere le modifiche fatte. Se si prova a uscire dall'editor senza salvare, o se si cerca di modificare un altro file (senza aver prima salvato quello modificato), Vim rifiuterà di eseguire il comando.
Per forzare l'esecuzione del comando, basta aggiungere un '!' al comando.
A quel punto le modifiche (al primo file) andranno perse. Per esempio:
":q" non verrà eseguito se il buffer è stato modificato, ma ":q!" sì.
Per controllare se il buffer è stato modificato, si usi il comando "**CTRL-G**". Il messaggio fornito dal comando include la stringa "[Modificato]" se il buffer è stato cambiato.

Se si vogliono salvare automaticamente le modifiche, senza dover dare alcun comando, si attivi l'opzione **'autowriteall'**. **'autowrite'** è l'opzione simile a questa e compatibile con Vi, ma che non entra in funzione per tutti i comandi in cui sarebbe richiesta.

Se si vuole continuare ad avere a disposizione il buffer modificato, senza salvarlo, si può attivare l'opzione **'hidden'**. Vedere |**hidden-buffer**|. Alcuni comandi funzionano in questo modo anche se **'hidden'** non è stato attivato; si controlli la documentazione del comando.

==========

2. Editare un file *edit-a-file*

:e *:edit* *reload*

:e[dit] **[++opt] [+cmd]** Edita il file corrente. Ciò è utile per entrare di nuovo in edit sul file corrente, se è stato modificato in maniera indipendente da Vim. Il comando non viene eseguito se sono state fatte modifiche al buffer corrente e **'autowriteall'** non è attiva o se il file non può essere scritto [solitamente per problemi di autorizzazione].
Vedere anche |**++opt**| e |**+cmd**|.
{Vi: non si può specificare ++opt}

:edit! *discard*

:e[dit]! **[++opt] [+cmd]**
Edita il file corrente incondizionatamente. Eventuali modifiche al buffer corrente saranno perse. Ciò è utile se si vuole iniziare l'edit da capo.
Vedere anche |**++opt**| e |**+cmd**|.
{Vi: non si può specificare ++opt}

:edit_f

:e[dit] **[++opt] [+cmd] {file}**
Edita **{file}**.
Questo comando non viene eseguito quando il buffer corrente risulti modificato, a meno che **'hidden'** sia impostato o **'autowriteall'** sia impostato, e quindi sia stato possibile riscrivere il file.
Vedere anche |**++opt**| e |**+cmd**|.
{Vi: non si può specificare ++opt}

:edit!_f

:e[dit]! **[++opt] [+cmd] {file}**
Edita comunque **{file}**. Scarta qualsiasi modifica sia stata fatta al buffer corrente.
Vedere anche |**++opt**| e |**+cmd**|.
{Vi: non si può specificare ++opt}

:e[dit] **[++opt] [+cmd]** #**[num]**
Edita il buffer numero **[num]** (come elencato dal comando |**:files**|).
Questo comando si comporta come **[num] CTRL-^**. Ma ":e #" non viene eseguito se il buffer indicato non ha un nome di file, mentre **CTRL-^** viene eseguito anche in quel caso.
Vedere anche |**++opt**| e |**+cmd**|.
{Vi: non si può specificare ++opt}

:ene *:enew*

:ene[w] Edita un nuovo buffer, che non ha un nome di file.
Questo comando non viene eseguito quando il buffer corrente risulti modificato, a meno che **'hidden'** sia impostato o **'autowriteall'** sia impostato, e quindi sia stato possibile riscrivere il file.
Se **'fileformats'** è impostato a un valore non nullo, il primo formato specificato verrà usato per il nuovo buffer. Se **'fileformats'** è vuoto, verrà usata l'opzione **'fileformat'** del buffer corrente.
{non in Vi}

:ene! *:enew!*

:ene[w]! Edita un nuovo buffer, che non ha un nome di file.
Le modifiche apportate al buffer corrente saranno scartate.
L'impostazione di **'fileformat'** è come in |**:enew**|.
{non in Vi}

:fin *:find*

:fin[d][!] **[++opt] [+cmd] {file}**
Trova il file **{file}** nella lista di directory specificata nell'opzione **'path'** e poi edita |:edit| il file stesso.
{non in Vi}
{non disponibile se compilato senza la funzionalità |+file_in_path|}

:**{contatore}**fin[d][!] **[++opt] [+cmd] {file}**
Come ":find", ma usa il **{contatore}** per trovare la corrispondenza in **'path'**. Quindi ":2find file" troverà il secondo file di nome "file" nel **'path'**. Se non ci sono sufficienti corrispondenze per il nome file in **'path'**, viene inviato un messaggio di errore.

:ex

:ex **[++opt] [+cmd] [file]**
Come |:edit|.

:vi *:visual*

:vi[sual][!] **[++opt] [+cmd] [file]**
Quando è usato in modo Ex: Lascia il modo Ex |Ex-mode|, e ritorna al modo Normal. Altrimenti è come |:edit|.

:vie *:view*

:vie[w][!] **[++opt] [+cmd]** file
Quando è usato in modo Ex: Lascia il modo Ex |Ex-mode|, e ritorna al modo Normal. Altrimenti è come |:edit|, ma imposta l'opzione **'readonly'** (sola lettura) per il buffer corrente. {non in Vi}

*CTRL-^* *CTRL-6*

CTRL-^
Edita il file alternativo. Di solito il file alternativo è il file editato subito prima. Questo è un modo veloce per passare da un file all'altro. È equivalente a ":e #", solo che viene eseguito anche quando non c'è un nome di file.

Se l'opzione **'autowrite'** o **'autowriteall'** è attiva e il buffer risulta modificato, il file viene riscritto. Per lo più il carattere ^ è posizionato sul tasto 6, e se si schiaccia CTRL e 6 si ottiene quel che qui è indicato come **CTRL-^**.
Su alcune tastiere non statunitensi **CTRL-^** è prodotto in un altro modo, ma **CTRL-6** funziona ugualmente.

{contatore}CTRL-^
Edita il file numero **[num]** nella lista dei buffer (equivale a a immettere ":e #**[num]**"). Questo è un modo veloce per alternare l'edit di diversi file. Vedere |CTRL-^| più sopra per ulteriori dettagli.
{non in Vi}

]f *[f*

[num]]f
[num][f
Come "gf". Deprecato.

gf *E446* *E447*

[num]gf
Edita il file il cui nome è sotto o dopo il cursore. Abbreviazione mnemonica di: "goto file" (vai al file). Usa l'opzione **'isfname'** per determinare quali caratteri possono essere usati in un nome di file. I segni d'interpunzione finali ".,:;!" vengono ignorati. Caratteri di spazio protetti da "\ " sono ridotti a uno spazio singolo.
Usa l'opzione **'path'** per la lista di nomi di directory nelle quali cercare il file. Vedere l'opzione **'path'** per dettagli sulle directory relative e sui metacaratteri ("wildcard").
Usa l'opzione **'suffixesadd'** per controllare nomi di file a cui vada aggiunto un suffisso.
Se non viene trovato alcun file, si usa **'includeexpr'**

per modificare il nome e fare un nuovo tentativo.
Se viene fornito il numero **[num]**, si va in edit sul file numero **[num]** trovato percorrendo le directory elencate nell'opzione **'path'**.
Questo comando non viene eseguito se Vim si rifiuta di abbandonare il file corrente (vedere |**abandon**|).
Se si vuole editare il file in una nuova finestra si usi |**CTRL-W_CTRL-F**|.
Se si vuole creare un file nuovo, si usi: >
:e <nuovo_file>
< Per far sì che gf funzioni in questo modo, la mappatura è: >
:map gf :e <nuovo_file><CR>
< Se il nome è un puntatore a un ipertesto, del tipo "tipo://macchina/percorso", si deve utilizzare il plugin |**netrw**|.
Per Unix il carattere '~' è espanso, come in "~utente/file". Anche le variabili d'ambiente sono espanse |**expand-env**|.
{non in Vi}
{non disponibile se compilato senza la funzionalità |**+file_in_path**|}

v_gf

{Visual}[num]gf Come "gf", ma il testo evidenziato è usato come nome del file da editare. **'isfname'** è ignorato.
Gli spazi che eventualmente precedono il nome sono ignorati, tutti gli altri spazi e caratteri speciali sono inclusi nel nome del file.
(Per **{Visual}** vedere |**Visual-mode**|.)
{non in VI}

gF

[num]gF Come "gf", ma se un numero di riga viene messo dopo il nome del file, il cursore viene inizialmente posizionato su quella particolare riga del file.
Il nome del file e il numero di riga devono essere separati da un carattere che non possa far parte di un nome di file (vedere **'isfname'**) e che non sia una cifra. Eventuali spazi bianchi fra il nome del file, il separatore e il numero di riga sono ignorati. Esempi:
eval.c:10 ~
eval.c @ 20 ~
eval.c (30) ~
eval.c 40 ~

v_gF

{Visual}[num]gF Come "v_gf".

Questi comandi sono usati per iniziare a editare un solo file. Ciò significa che il file è letto in un buffer e il nome del file corrente è impostato col suo nome. Il file che viene aperto dipende dalla directory corrente in cui ci si trova quando si immette il comando; vedere |**:cd**|.

Vedere |**read-messages**| per una spiegazione del messaggio inviato una volta che il file è stato letto.

Si può usare il comando ":e!" se è stato combinato qualche pasticcio all'interno del buffer, e si vuole ricominciare da capo. Il comando ":e" è utile solo se è stato cambiato il nome del file corrente.

:filename *{file}*

In aggiunta a ciò che è descritto qui, altri modi per specificare un nome di file sono spiegati in |**cmdline-special**|.

Nota Per sistemi diversi da Unix: Quando si usa un comando che accetta un unico nome di file (come ":edit file") degli spazi bianchi nel nome del file sono consentiti, e gli spazi alla fine del nome sono ignorati. Ciò è utile in sistemi che regolarmente includono degli spazi bianchi nei nomi di file (come MS-Windows e Amiga). Esempio: il comando
":e Nome lungo di file " andrà in edit sul file "Nome lungo di file".
Quando si usa un comando che accetta più nomi di file (come

":next file1 file2") eventuali spazi nei nomi di file vanno protetti con una barra inversa ("\").

wildcard *wildcards*
I metacaratteri ("wildcard") contenuti in **{file}** sono espansi, ma come con il completamento dei nomi di file, vengono applicate le opzioni **'wildignore'** e **'suffixes'**. Quali metacaratteri sono supportati dipende dal sistema. Questi sono quelli più comuni:

?	corrisponde a un carattere qualsiasi
*	corrisponde a qualsiasi stringa, compresa la stringa nulla
**	corrisponde a qualsiasi stringa, compresa la stringa nulla, e si applica ricorsivamente alle directory sottostanti
[abc]	corrisponde ai caratteri 'a', 'b' o 'c'

Per togliere il significato speciale ai metacaratteri, anteporre una barra inversa agli stessi. Tuttavia, in MS-Windows la barra inversa fa da separatore di percorso (indica l'inizio di un'altra directory) e quindi "percorso\[abc]" è ancora visto come un metacarattere quando il carattere "[" è incluso nell'opzione **'isfname'**. Un modo semplice per evitare ciò è di usare "percorso\[[]abc]". In questo modo il nome di file corrispondente è letteralmente "percorso\[abc]".

starstar-wildcard
L'espansione di "**" è possibile in Unix, Win32, Mac OS/X e alcuni altri sistemi. Questo permette una ricerca in un albero di directory. Si può arrivare fino a una profondità di 100.
Nota Ci sono alcuni comandi il cui funzionamento è leggermente differente, vedere |**file-searching**|.
Esempio: >

```
:n **/*.txt
```

Trova i file:

```
aaa.txt ~
sottodirectory/bbb.txt ~
a/b/c/d/ccc.txt ~
```

Quando nei criteri di ricerca sono inclusi caratteri che non sono metacaratteri subito prima o dopo "**", le corrispondenze relative sono cercate solo nella prima directory trovata. Non sono usati per le directory che si trovano più sotto nell'albero delle directory. Per esempio: >

```
:n /usr/inc**/types*.h
```

Trova i file:

```
/usr/include/types.h ~
/usr/include/sys/types.h ~
/usr/inc_old/types.h ~
```

Nota Il percorso con "/sys" è incluso perché non deve corrispondere a "/inc". Quindi si cercano corrispondenze per "/usr/inc*/*/*..." e non per "/usr/inc*/inc*/inc*".

backtick-expansion *`-expansion*
In Unix e in alcuni altri sistemi si possono usare anche richiami di comando (racchiusi fra un apice inverso ("`") iniziale e uno finale) per individuare i file che faranno da argomento, per esempio: >

```
:next `find . -name ver\\*.c -print`
:view `ls -t *.patch  \| head -n1`
```

Le barre inverse prima dell'asterisco sono necessarie per inibire l'espansione da parte della shell di "ver*.c" prima della chiamata del programma "find". La barra inversa prima del simbolo di pipe ("|") della shell serve per far sì che Vim lo consideri come un delimitatore di comando.
Ciò si applica anche a molti altri sistemi, con la restrizione che l'intero comando deve essere racchiuso fra apici inversi. Non è possibile avere del testo inserito direttamente davanti al primo o dopo il secondo apice inverso.

`=
È anche possibile che il testo fra apici inversi sia espanso come un'espressione di Vim, invece di essere un comando esterno. Per ottenere ciò occorre inserire un segno di "uguale" subito dopo il primo apice inverso, p.es.: >

```
:e `=tempname()`
```

Il contenuto dell'espressione può essere molto vario, e quindi questa notazione può essere usata per evitare il significato speciale dei caratteri '"', '|', '%' e '#'. Tuttavia, l'opzione **'wildignore'** è applicata come per tutti gli altri metacaratteri.

Le variabile d'ambiente contenute nell'espressione sono espanse durante
la valutazione della stessa; quindi si può scrivere: >
```
:e `=$HOME . '/.vimrc'`
```
Se invece si scrive come nella riga sotto, la cosa non funzione, in quanto
la variabile $HOME è all'interno di una stringa non modificabile, e quindi
viene usata letteralmente: >
```
:e `='$HOME' . '/.vimrc'`
```

Se l'espressione restituisce una stringa, i nomi in essa contenuti devono
essere separati da interruzioni di riga. Quando il risultato è una lista
|List| ogni elemento della lista è usato come nome. Anche le interruzioni
di riga sono usate come separatore di nomi.
Nota Espressioni di questo tipo sono permesse solo in posti in cui ci si
aspetta un nome di file come argomento a un comando Ex.

++opt *[++opt]*
L'argomento **[++opt]** può essere usato per forzare il valore di **'fileformat'**,
'fileencoding' o **'binary'** a un valore per un particolare comando, e per
specificare il comportamento da tenere se si incontrano caratteri non
validi. Il formato è: >
```
++{opt_nome}
```
O: >
```
++{opt_nome}={valore}
```

Dove {opt_nome} è uno fra: *++ff* *++enc* *++bin* *++nobin* *++edit*

ff	o	fileformat	prevale su **'fileformat'** esistente
enc	o	encoding	prevale su **'fileencoding'** esistente
bin	o	binary	imposta **'binary'**
nobin	o	nobinary	disattiva **'binary'**
bad			specifica comportamento per caratteri non validi
edit			solo per il comando |:read|: usa i valori delle opzioni come se si stesse editando un file

{valore} non può contenere spazi bianchi. Può essere qualsiasi valore
sia possibile specificabile per queste opzioni. Esempi: >
```
:e ++ff=unix
```
Così si edita ancora lo stesso file, con **'fileformat'** impostati a "unix". >

```
:w ++enc=latin1 nuovo_file
```
Così si scrive il buffer corrente in un file
di nome "nuovo_file" usando la codifica latin1.

Possono esserci più argomenti ++opt, separati da spazi bianchi. Devono
essere tutti posti prima di qualsiasi argomento |+cmd|.

++bad
L'argomento specificato con "++bad=" dice cosa fare se si trovano caratteri
che non possono essere convertiti e byte non regolari. Si può specificare una
delle seguenti tre alternative:

++bad=X	Un carattere di un singolo byte che rimpiazza il carattere sbagliato.
++bad=keep	Tieni i caratteri sbagliati senza convertirli. Nota Ciò può condurre ad avere byte non regolari nel testo!
++bad=drop	Rimuovi il carattere sbagliato.

Il default è "++bad=?": Rimpiazza ogni carattere sbagliato con un punto
interrogativo. In alcuni posti si usa il punto interrogativo rovesciato
(0xBF).

Nota Non tutti i comandi usano l'argomento ++bad, anche se non danno alcun
messaggio di errore se lo si aggiunge. P.es. |:write|.

Nota che in fase di lettura, le opzioni **'fileformat'** e **'fileencoding'**
saranno impostate in base al formato usato. In fase di scrittura ciò non
avviene, quindi una successiva scrittura userà il precedente valore
dell'opzione. Lo stesso vale per l'opzione **'binary'**.

+cmd *[+cmd]*
L'argomento **[+cmd]** si può usare per posizionare il cursore nel file
appena aperto, o per eseguire qualsiasi altro comando:

+	Posizionarsi all'ultima riga del file.

```
        +{num}          Posizionarsi alla riga {num}.
        +/{pat}         Posizionarsi alla prima riga che contiene {pat}.
        +{comando}      Eseguire {comando} dopo aver aperto il nuovo file.
                        {comando} è qualsiasi comando Ex.
```

Per includere spazi bianchi in **{pat}** o **{comando}**, va inserita una barra inversa davanti a ogni spazio bianco. Il numero di barre inverse va raddoppiato per inserire una singola barra inversa. >

```
        :edit  +/Il\ libro           file
        :edit  +/dir\ nome-dir\\     file
        :edit  +set\ dir=c:\\\\temp  file
```

Nota Nell'ultimo esempio il numero di barre inverse è dimezzato due volte: prima per l'argomento di "+cmd" e poi per il comando ":set".

file-formats

L'opzione **'fileformat'** imposta lo stile del delimitatore di riga **<EOL>** per un file:

'fileformat'	caratteri	nome	~
"dos"	**<CR><NL>** o **<NL>**	formato DOS	*DOS-format*
"unix"	**<NL>**	formato Unix	*Unix-format*
"mac"	**<CR>**	formato Mac	*Mac-format*

In passato, si poteva usare l'opzione **'textmode'**. Oggi è obsoleta.

Quando si legge un file, i caratteri di cui sopra sono interpretati come **<EOL>** (End Of Line - fine riga).
Nel formato DOS (default per MS-DOS, OS/2 e Win32), **<CR><NL>** e **<NL>** sono entrambi interpretati come **<EOL>**. Nota Quando si scrive il file in formato DOS, un carattere **<CR>** sarà aggiunto per ogni **<NL>** isolato. Vedere anche |**file-read**|.

Quando si scrive un file, i caratteri di cui sopra sono usati come **<EOL>**.
Per il formato DOS si usa **<CR><NL>**. Vedere anche |**DOS-format-write**|.

Si può leggere un file in formato DOS e riscriverlo in formato Unix. Ciò comporterà la sostituzione di tutte le coppie **<CR><NL>** con **<NL>** (presupponendo che l'opzione **'fileformats'** includa "dos"): >

```
        :e file
        :set fileformat=unix
        :w
```

Se si legge un file in formato Unix e lo si riscrive in formato DOS, tutti i caratteri **<NL>** saranno rimpiazzati con **<CR><NL>** (presupponendo che l'opzione **'fileformats'** includa "unix"): >

```
        :e file
        :set fileformat=dos
        :w
```

Se si inizia a editare un nuovo file e l'opzione **'fileformats'** non è impostata alla stringa nulla (non è questa l'impostazione di default), Vim tenta di determinare se le righe del file sono separate da uno dei formati che sono stati specificati. Quando **'fileformats'** è impostato a "unix,dos", Vim controlla se le righe terminano con un singolo **<NL>** (come in Unix e in Amiga) o con la coppia **<CR><NL>** (MS-DOS). Solo quando TUTTE le righe terminano con **<CR><NL>**, **'fileformat'** è impostato a "dos", altrimenti è impostato a "unix". Quando **'fileformats'** include "mac", e non sono stati trovati caratteri **<NL>**, **'fileformat'** è impostato a "mac".

Se l'opzione **'fileformat'** è impostata a "dos" su sistemi diversi da MS-DOS il messaggio "[DOS]" è visualizzato, per rammentare che c'è qualcosa di insolito nel file. Nei sistemi MS-DOS viene visualizzato il messaggio "[unix]" se **'fileformat'** è impostato a "unix". Su tutti i sistemi, eccettuato il Macintosh viene visualizzato il messaggio "[Mac]" se **'fileformat'** è impostato a "mac".

Se l'opzione **'fileformats'** è impostata alla stringa nulla e si sta usando il formato DOS, ma durante la lettura di un file qualche riga non termina con **<CR><NL>**, al messaggio del file verrà aggiunto "[manca CR]".
Se l'opzione **'fileformats'** è impostata alla stringa nulla e si sta usando il formato Mac, ma durante la lettura di un file si trova in qualche riga il carattere **<NL>**, al messaggio relativo al file verrà aggiunto "[manca NL]".

Se il nuovo file non esiste, viene usato il **'fileformat'** del buffer corrente se l'opzione **'fileformats'** è impostata alla stringa nulla.
Altrimenti si usa per il nuovo file il primo formato elencato in **'fileformats'**.

Prima di editare file binari, programmi eseguibili o file di script Vim, andrebbe impostata l'opzione 'binary'. Un modo semplice per farlo è di richiamare Vim con l'opzione "-b". In questo mondo non verrà usata l'opzione 'fileformat'. Se non si fa così, si rischia che dei caratteri <NL> singoli siano rimpiazzati inaspettatamente dalla coppia <CR><NL>.

Si possono cifrare i file in scrittura impostando l'opzione 'key'. Questo dà una certa sicurezza che altri non leggano quei file. |encryption|

==
3. La lista degli argomenti *argomento-list* *arglist*

Se si danno più nomi di file chiamando Vim, questa lista viene ricordata come "lista degli argomenti". Si può saltare a ogni file nella lista.

Questa lista non va confusa con la lista dei buffer, visualizzabile col comando |:buffers|. La lista degli argomenti era già presente in Vi, la lista dei buffer è stata introdotta da Vim. Ogni nome di file nella lista degli argomenti è anche presente nella lista dei buffer (se non è stata cancellata con |:bdel| o |:bwipe|). Ma è normale che nomi presenti nella lista dei buffer non siano presenti nella lista degli argomenti.

Questo argomento è illustrato nella sezione |07.2| del manuale dell'utente.

Esiste una lista globale degli argomenti, che è usata per default da tutte le finestre. È possibile creare una nuova lista degli argomenti per una particolare finestra, vedere |:arglocal|.

Si può usare la lista degli argomenti con i comandi seguenti, e con le espressioni di funzione |argc()| e |argv()|. Tutti utilizzano la lista degli argomenti della finestra corrente.

:ar *:args*
:ar[gs] Stampa la lista degli argomenti, con il nome del file corrente racchiuso fra parentesi quadre.

:ar[gs] [++opt] [+cmd] {arglist} *:args_f*
Definisce {arglist} come la nuova lista degli argomenti e inizia a editare il primo file specificato. Questo comando non viene eseguito quando sono state fatte delle modifiche e Vim richiede di salvarle prima di lasciare (|abandon|) il buffer corrente.
Vedere anche |++opt| e |+cmd|.
{Vi: non c'è ++opt}

:ar[gs]! [++opt] [+cmd] {arglist} *:args_f!*
Definisce {arglist} come la nuova lista degli argomenti e inizia a editare il primo file specificato. Scarta ogni modifica che sia stata apportata al buffer corrente.
Vedere anche |++opt| e |+cmd|.
{Vi: non c'è ++opt}

:[num]arge[dit][!] [++opt] [+cmd] {nome} *:arge* *:argedit*
Aggiunge {nome} alla lista degli argomenti e inizia a editarlo. Quando {nome} esiste già nella lista degli argomenti, inizia a editarlo.
Ciò equivale a usare |:argadd| e poi |:edit|.
Nota È consentito un solo nome di file, e possono esserci spazi nel nome del file, come con |:edit|.
[num] è usato come in |:argadd|.
[!] è richiesto se il file corrente ha degli aggiornamenti non salvati, e non può essere chiuso. Vedere |abandon|.
Vedere anche |++opt| e |+cmd|.
{non in Vi}

:[num]arga[dd] {nome} .. *:arga* *:argadd* *E479*
:[num]arga[dd]

Aggiunge **{nome}** (anche più di uno) alla lista degli argomenti. Quando **{nome}** è omesso, aggiunge il nome del buffer corrente alla lista degli argomenti.
Se **[num]** è omesso, il **{nome}** o i nomi sono aggiunti dopo l'elemento della lista degli argomenti che è attualmente in uso.
Altrimenti sono aggiunti dopo il file numero **[num]**.
Se la lista degli argomenti è "a b c", e "b" è l'argomento corrente, questo è quel che succede:

comando	**nuova lista degli argomenti** ~
:argadd x	a b x c
:0argadd x	x a b c
:1argadd x	a x b c
:$argadd x	a b c x

E dopo l'ultimo:

:+2argadd y	a b c x y

Non ci sono controlli per le duplicazioni, è possibile aggiungere un file alla lista degli argomenti più volte. Il file che si sta editando non è modificato.
{non in Vi} {non disponibile se compilato senza la funzionalità |**+listcmds**|}
Nota: Si può anche usare questo metodo: >
:args ## x
< In questo modo si aggiunge l'elemento "x" in fondo alla lista.

:argd[elete] **{pattern}** .. *:argd* *:argdelete* *E480*
Cancella file dalla lista degli argomenti che corrispondono a **{pattern}** (si possono specificare più **{pattern}**. **{pattern}** è usato come una stringa di ricerca di file, vedere |**file-pattern**|.
"%" può essere usato per cancellare l'elemento corrente.
Questo comando conserva il file su cui si sta scrivendo, anche se lo stesso viene cancellato dalla lista degli argomenti.
Esempio: >
:argdel *.obj
< {non in Vi} {non disponibile se compilato senza la funzionalità |**+listcmds**|}

:**[range]**argd[elete] Cancella i file nell'intervallo specificato da **{range}** dalla lista degli argomenti.
Esempio: >
:10,$argdel
< Cancella gli argomenti da 10 in poi, lasciando 1-9. >
:$argd
< Cancella solo l'ultimo argomento. >
:argd
:.argd
< Cancella l'argomento corrente. >
:%argd
< Cancella tutti i file dalla lista degli argomenti.
Quando l'ultimo numero in un intervallo è troppo elevato, vengono cancellati gli elementi fino all'ultimo disponibile.
{non in Vi} {non disponibile se compilato senza la funzionalità |**+listcmds**|}

:argu *:argomento*
:**[num]**argu[ment] **[num]** **[++opt]** **[+cmd]**
Edita il file numero **[num]** della lista degli argomenti.
Quando **[num]** è omesso viene usato l'elemento corrente.
Il comando non viene eseguito quando ci sono modifiche non salvate, perché Vim non lascia il buffer corrente (vedere |**abandon**|).
Vedere anche |**++opt**| e |**+cmd**|.
{non in Vi} {non disponibile se compilato senza la funzionalità |**+listcmds**|}

:**[num]**argu[ment]! **[num]** **[++opt]** **[+cmd]**
Edita il file numero **[num]** della lista degli

argomenti. Scarta senza salvarlo il buffer corrente. Quando **[num]** è omesso viene usato l'elemento corrente.
Vedere anche |**++opt**| e |**+cmd**|.
{non in Vi} {non disponibile se compilato senza la funzionalità |**+listcmds**|}

:**[num]**n[ext] **[++opt] [+cmd]** *:n* *:ne* *:next* *E165* *E163*
Edita il prossimo file numero **[num]**. Il comando non viene eseguito quando ci sono modifiche non salvate, perché Vim non lascia il buffer corrente (vedere |**abandon**|).
Vedere anche |**++opt**| e |**+cmd**|.
{Vi: non c'è contatore o ++opt}.

:**[num]**n[ext]! **[++opt] [+cmd]**
Edita il prossimo file numero **[num]** nella lista degli argomenti. Scarta senza salvarlo il buffer corrente.
Vedere anche |**++opt**| e |**+cmd**|.
{Vi: non c'è contatore o ++opt}.

:n[ext] **[++opt] [+cmd] {arglist}** *:next_f*
Come |**:args_f**|.

:n[ext]! **[++opt] [+cmd] {arglist}**
Come |**:args_f!**|.

:**[num]N**[ext] **[num] [++opt] [+cmd]** *:Next* *:N* *E164*
Edita il precedente file numero **[num]**. Il comando non viene eseguito quando ci sono modifiche non salvate, perché Vim non lascia il buffer corrente (vedere |**abandon**|).
Vedere anche |**++opt**| e |**+cmd**|.
{Vi: non c'è contatore o ++opt}.

:**[num]N**[ext]! **[num] [++opt] [+cmd]**
Edita il precedente file numero **[num]** nella lista degli argomenti. Scarta senza salvarlo il buffer corrente.
Vedere anche |**++opt**| e |**+cmd**|.
{Vi: non c'è contatore o ++opt}.

:**[num]**prev[ious] **[num] [++opt] [+cmd]** *:prev* *:previous*
Come :Next. Vedere anche |**++opt**| e |**+cmd**|.
{Vi: solo in alcune versioni}

:rew *:rewind*
:rew[ind] **[++opt] [+cmd]**
Inizia a editare il primo file della lista degli argomenti. Il comando non viene eseguito quando ci sono modifiche non salvate, perché Vim non lascia il buffer corrente (vedere |**abandon**|).
Vedere anche |**++opt**| e |**+cmd**|.
{Vi: non c'è ++opt}

:rew[ind]! **[++opt] [+cmd]**
Inizia a editare il primo file nella lista degli argomenti. Scarta senza salvarlo il buffer corrente.
Vedere anche |**++opt**| e |**+cmd**|.
{Vi: non c'è ++opt}

:fir *:first*
:fir[st][!] **[++opt] [+cmd]**
Nome alternativo per ":rewind". {non in Vi}

:la *:last*
:la[st] **[++opt] [+cmd]**
Inizia a editare l'ultimo file nella lista degli argomenti. Il comando non viene eseguito quando ci sono modifiche non salvate, perché Vim non lascia il buffer corrente (vedere |**abandon**|).
Vedere anche |**++opt**| e |**+cmd**|. {non in Vi}

:la[st]! **[++opt] [+cmd]**

Inizia a editare l'ultimo file nella lista degli
argomenti. Scarta senza salvarlo il buffer corrente.
Vedere anche |**++opt**| e |**+cmd**|.
{non in Vi}

:wn *:wnext*

:**[num]**wn[ext] **[++opt]**
Scrive il file corrente e inizia a editare il
prossimo file numero **[num]**.
Vedere anche |**++opt**| e |**+cmd**|. {non in Vi}

:**[num]**wn[ext] **[++opt]** **{file}**
Scrive il file corrente in **{file}** e inizia a editare
il prossimo file numero **[num]**, a meno che **{file}**
non esista già e l'opzione **'writeany'** sia off.
Vedere anche |**++opt**| e |**+cmd**|. {non in Vi}

:**[num]**wn[ext]! **[++opt]** **{file}**
Scrive il file corrente to **{file}** e inizia a editare
il prossimo file numero **[num]**.
Vedere anche |**++opt**| e |**+cmd**|. {non in Vi}

:**[num]**wN[ext][!] **[++opt]** **[file]** *:wN* *:wNext*
:**[num]**wp[revious][!] **[++opt]** **[file]** *:wp* *:wprevious*
Come :wnext, ma va al file precedente, invece che
al successivo. {non in Vi}

Il numero **[num]** nei comandi di cui sopra ha per default il valore uno. Per alcuni dei comandi è possibile specificare due numeri. L'ultimo di essi (quello più a destra) è quello che verrà usato.

Se non è specificato l'argomento **[+cmd]**, il cursore è posizionato all'ultima posizione conosciuta del cursore nel file. Se **'startofline'** è impostato, il cursore sarà posizionato al primo carattere visibile sulla riga, altrimenti sarà sull'ultima colonna in cui era [nella precedente sessione di edit]. Se l'ultima posizione del cursore non è disponibile, il cursore sarà sulla prima riga del file (sull'ultima riga se si è in Ex mode).

{arglist}

I metacaratteri nella lista degli argomenti vengono espansi e i nomi di file vengono ordinati. Quindi si può usare il comando "vim *.c" per editare tutti i file sorgenti di C. Da una sessione di Vim attiva, lo stesso si può fare con il comando ":n *.c".

Per separare i nomi di file viene usato uno spazio bianco.
Bisogna inserire una barra inversa prima di uno spazio o di un carattere **<TAB>** se un nome di file contiene tali caratteri. P.es., per editare un solo file di nome "pippo pluto": >

```
:next pippo\ pluto
```

In Unix e su alcuni altri sistemi si possono usare gli apici inversi, per esempio: >

```
:next `find . -name \\*.c -print`
```

Le barre inverse prima del carattere "*" sono necessarie per far sì che l'espressione "*.c" venga espansa dalla shell, prima dell'esecuzione del programma find.

arglist-position

Quando c'è una lista degli argomenti si può vedere quale file si sta editando nel titolo della finestra (se ce n'è uno, e se l'opzione **'title'** è attiva), e con il messaggio relativo al file, che si ottiene tramite il comando "**CTRL-G**". Si vedrà qualcosa del tipo

```
(file 4 of 11)
```

Se **'shortmess'** contiene 'f' lo stesso messaggio diventa

```
(4 of 11)
```

Se non si sta realmente editando il file alla posizione corrente della lista degli argomenti, il messaggio sarà

```
(file (4) of 11)
```

Questo significa che ci si trova nella posizione 4 nella lista degli argomenti, ma non si sta editando il quarto file della lista degli argomenti. Ciò capita quando si dà il comando ":e file".

LISTA LOCALE DEGLI ARGOMENTI

{non in Vi}
{non disponibile se compilato senza le funzionalità |+windows| o |+listcmds|}

:arglocal
:argl[ocal] Fa una copia locale della lista globale degli argomenti. Non inizia a editare un altro file.

:argl[ocal][!] **[++opt] [+cmd] {arglist}**
Definisce una nuova lista degli argomenti, che è locale alla finestra corrente. Per gli altri aspetti, funziona come |:args_f|.

:argglobal
:argg[lobal] Usa la lista globale degli argomenti per la finestra corrente. Non inizia a editare un altro file.

:argg[lobal][!] **[++opt] [+cmd] {arglist}**
Usa la lista globale degli argomenti per la finestra corrente. Definisce una nuova lista globale degli argomenti come |:args_f|.
Tutte le finestre che usano la lista globale degli argomenti useranno questa nuova lista.

Ci possono essere più liste degli argomenti. Possono essere condivise fra diverse finestre. Quando sono condivise, modificando la lista degli argomenti in una finestra il cambiamento vale anche per le altre finestre.

Quando si divide una finestra la nuova finestra eredita la lista degli argomenti dalla finestra corrente. Le due finestre continueranno a condividere la stessa lista, finché una delle due non imposta l'opzione |:arglocal| o |:argglobal| per usare un'altra lista degli argomenti.

USARE LA LISTA DEGLI ARGOMENTI

:argdo
:**[range]**argdo[!] **{comando}** Esegue **{comando}** per ogni file della lista degli argomenti o, se l'intervallo **[range]** è specificato, solo per gli argomenti in quell'intervallo. Funziona come se si scrivesse: >
```
:rewind
:{comando}
:next
:{comando}
etc.
```
< Quando il file corrente non può essere lasciato (|abandon|) e il [!] non è presente, il comando non viene eseguito.
Quando viene individuato un errore in un file, i rimanenti file nella lista degli argomenti non saranno visitati.
L'ultimo file nella lista degli argomenti (o quello dove si è riscontrato un errore) diventa il file corrente.
{comando} può contenere '|' per concatenare più comandi.
{comando} non deve cambiare la lista degli argomenti.
Nota: Mentre questo comando è in esecuzione, l'autocomando relativo all'evento Syntax è disabilitato, aggiungendolo alla lista contenuta nell'opzione **'eventignore'**. Questo accelera notevolmente l'edizione di ciascun file.
{non in Vi} {non disponibile se compilato senza la funzionalità |+listcmds|}
Vedere anche |:windo|, |:tabdo|, |:bufdo|, |:cdo|, |:ldo|, |:cfdo| e |:lfdo|

Esempio: >
```
:args *.c
```

```
        :argdo set ff=unix | update
```
Questo imposta l'opzione 'fileformat' a "unix" e riscrive il file se risulta cambiato. Ciò è fatto per tutti i file individuati dall'espressione *.c.

Esempio: >
```
        :args *.[ch]
        :argdo %s/\<pippo_mio\>/Pippo_Mio/ge | update
```
Questo cambia le parole "pippo_mio" a "Pippo_Mio" in tutti i file *.c e *.h file. Il flag "e" è usato per il comando ":substitute" per evitare una segnalazione di errore nei file in cui "pippo_mio" non è usato. ":update" riscrive il file solo nel caso sia stato modificato.

==========

4. Scrivere *writing* *save-file*

Nota: Quando l'opzione 'write' è off, non si può scrivere alcun file.

:w *:write*
E502 *E503* *E504* *E505*
E512 *E514* *E667* *E796*

:w[rite] [++opt] Scrive l'intero buffer sul file corrente. Questo è il modo normale per salvare le modifiche fatte a un file. Non viene eseguito quando l'opzione 'readonly' è impostata o quando per qualche altra ragione il file non può essere scritto.
Per ++opt vedere |++opt|, ma solo le opzioni ++bin, ++nobin, ++ff e ++enc sono valide qui.

:w[rite]! [++opt] Come ":write", ma forza la riscrittura se l'opzione 'readonly' è impostata, o se per qualche altra ragione la riscrittura era stata precedentemente rifiutata.
Nota: Ciò può cambiare i permessi di accesso e il proprietario del file, e invalidare i collegamenti (simbolici).
Si aggiunga il flag 'W' a 'cpoptions' per evitarlo.

:[range]w[rite][!] [++opt]
Scrive le righe specificate nell'intervallo [range] sul file corrente. Questo di solito non succede, perché il file scritto non conterrà tutte le righe presenti nel buffer in memoria.

:w_f *:write_f*

:[range]w[rite] [++opt] {file}
Scrive le righe specificate su {file}, a meno che il file non sia già esistente e l'opzione 'writeany' non sia impostata a off.

:w!

:[range]w[rite]! [++opt] {file}
Scrive le righe specificate su {file}. Sovrascrive un file già esistente.

:w_a *:write_a* *E494*

:[range]w[rite][!] [++opt] >>
Aggiunge le righe specificate in fondo al file corrente.

:[range]w[rite][!] [++opt] >> {file}
Aggiunge le righe specificate in fondo al file {file}. '!' forza la scrittura, anche nel caso in cui il file ancora non esista.

:w_c *:write_c*

:[range]w[rite] [++opt] !{comando}
Esegue {comando} usando le righe dell'intervallo [range] come standard input
(si noti lo spazio PRIMA dell'opzione '!'). {comando} è eseguito come ":!{comando}" e ogni '!' è sostituito con il comando precedente |:!|.

L'intervallo [range] di default per il comando ":w" è l'intero buffer (1,$).

Se si scrive l'intero buffer, questo non viene più considerato come modificato. Quando si scrive su un file differente da quello corrente, con ":w nomefile" l'impostazione del flag di modifica dipende dal flag "+" dell'opzione **'cpoptions'**. Quando questo flag è presente in **'cpoptions'**, il comando di scrittura annullerà il flag **'modified'**, anche se il buffer corrente può ancora essere differente dal suo file di partenza.

Se col comando ":w" viene indicato un nome di file, questo diventa il nome di file alternativo. Questo si può usare, per esempio, quando la scrittura non riesce e si vuole riprovare più tardi con ":w #". Questo comportamento non si verifica se si toglie il flag 'A' dall'opzione **'cpoptions'**.

:sav *:saveas*

:sav[eas][!] **[++opt]** **{file}**
Salva il buffer corrente col nome **{file}** e imposta il nome-file del buffer corrente a **{file}**. Il nome precedente è usato come nome alternativo del file. Il [!] è necessario per sovrascrivere un file già esistente. Quando **'filetype'** è impostata alla stringa nulla si effettua il tentativo di determinare il tipo di file associato al nuovo nome, prima di riscrivere il file.
Quando la scrittura riesce, il flag **'readonly'** è annullato.
{non in Vi}

:up *:update*

:**[range]**up[date][!] **[++opt]** [>>] **[file]**
Come ":write", ma scrive solo se il buffer è stato modificato. {non in Vi}

SCRIVERE PIÙ DI UN BUFFER *buffer-write*

:wa *:wall*

:wa[ll]
Scrive tutti i buffer modificati. I buffer a cui non è associato un nome di file, o che sono in sola lettura, non vengono scritti. {non in Vi}

:wa[ll]!
Scrive tutti i buffer modificati, anche quelli che sono in sola lettura. I buffer a cui non è associato un nome di file non vengono scritti. {non in Vi}

Vim avvisa se si sta cercando di riscrivere un file che nel frattempo è stato modificato da qualche altro programma. Vedere |**timestamp**|.

backup *E207* *E506* *E507* *E508* *E509* *E510*

Se si scrive su un file esistente (ma non se si aggiunge qualcosa in fondo al file) mentre è attiva l'opzione **'backup'**, **'writebackup'** o **'patchmode'**, viene creato un file di backup del file originale. Il file originale è o copiato o rinominato (vedere **'backupcopy'**). Dopo che il file è stato scritto con successo, e quando l'opzione **'writebackup'** è attiva e l'opzione **'backup'** è inattiva, il file di backup viene cancellato. Quando l'opzione **'patchmode'** è attiva il file di backup può essere rinominato.

backup-table

'backup'	**'writebackup'**	**azione** ~
off	off	nessun backup effettuato
off	on	backup del file corrente, poi cancellato (default)
on	off	cancella precedente backup, backup del file corrente
on	on	cancella precedente backup, backup del file corrente

Quando il nome di file da scrivere corrisponde al criterio specificato con l'opzione **'backupskip'** non si crea alcun file di backup. In questo caso i valori di **'backup'** e **'writebackup'** sono ignorati.

Quando l'opzione **'backup'** è attiva, il precedente file di backup (con lo stesso nome del nuovo file di backup) verrà cancellato. Se **'backup'** non è attivo, ma **'writebackup'** è attivo, il file di backup eventualmente esistente non sarà cancellato. Al file di backup creato mentre il file viene scritto sarà assegnato un nome differente.

In qualche file-system è possibile che in seguito a una caduta di sistema si finisca per perdere sia il backup che il file appena scritto (talora quest'ultimo è presente, ma il contenuto non è corretto). In tal caso, si può provare di recuperare il contenuto dal file di swap, perché la scrittura su disco del file di swap è forzata (tramite sync) e quindi il file potrebbe ancora essere presente nello swap. |**:recover**|

Le directory specificate con l'opzione **'backupdir'** sono usate per scriverci il file di backup. (default: la stessa directory in cui il file viene scritto).

Se il file di backup è un nuovo file, che è una copia del file originale, o se è il file originale rinominato, dipende dall'opzione **'backupcopy'**. Vedere la relativa documentazione che spiega quando vien fatta una copia e quando il file viene rinominato.

Se la creazione di un file di backup non riesce, la riscrittura del file non viene effettuata. Se si vuole scrivere comunque, aggiungere '!' al comando.

write-permissions
Quando si scrive un file nuovo i permessi sono quelli di lettura e scrittura. In Unix la maschera d'autorizzazione è 0666, e si applicano le modifiche previste col comando umask. Quando si scrive un file che era stato letto in precedenza Vim conserva i permessi originali, ma azzera il bit "s". [Vedere manuale di chmod per dettagli.]

write-readonly
Quando l'opzione **'cpoptions'** contiene 'W', Vim si rifiuta di riscrivere un file in sola lettura. Quando 'W' non è presente, ":w!" riscriverà un file in sola lettura, se il sistema lo consente (si deve essere autorizzati a scrivere sulla directory in cui si trova il file).

write-fail
Se la scrittura del nuovo file non riesce, si deve stare attenti a non perdere sia il nuovo file che il file originale. Se non c'è file di backup e la scrittura del nuovo file non è riuscita, il file originale è già perso! NON USCIRE DA VIM PRIMA DI RIUSCIRE A SCRIVERE IL FILE! Se un backup è stato fatto, viene messo al posto del file originale (se possibile). Se si esce da Vim e si perdono le modifiche, il file originale dovrebbe essere ancora disponibile. Se il ripristino del file originale non riesce, un messaggio di errore informa riguardo alla perdita del file originale.

DOS-format-write
Se l'opzione **'fileformat'** è "dos", come **<EOL>** viene usato **<CR> <NL>**. Questo è il default per MS-DOS, Win32 e OS/2. Su altri sistemi il messaggio "[DOS]" è visualizzato per segnalare che si sta usando un delimitatore di riga (**<EOL>**) insolito.

Unix-format-write
Se l'opzione **'fileformat'** è "unix", come **<EOL>** viene usato **<NL>**. In MS-DOS, Win32 e OS/2 è visualizzato il messaggio "[unix]".

Mac-format-write
Se l'opzione **'fileformat'** è "mac", come **<EOL>** viene usato **<CR>**. Su sistemi non-Mac viene visualizzato il messaggio "[Mac]".

Vedere anche |**file-formats**| e le opzioni **'fileformat'** e **'fileformats'**.

ACL
ACL è un acronimo per Access Control List. Indica una maniera avanzata per controllare i diritti d'accesso a un file. È usato sui sistemi MS-Windows e Unix più recenti, ma solo quando il file-system lo supporta.
Vim tenta di conservare le informazioni ACL nello scrivere un file. Il file di backup avrà le informazioni ACL del file originale.
Le informazioni ACL sono anche usate per controllare se un file è in sola lettura (all'apertura del file).

read-only-share
Quando MS-Windows condivide un disco sulla rete, il disco può essere contrassegnato come di sola lettura. Ciò significa che anche se per il file l'attributo di sola lettura è assente, e le impostazioni ACL sulla rete NT consentirebbero la scrittura del file, risulta lo stesso impossibile scrivere sul file. Vim, su piattaforme Win32, controlla se ci si trova su un disco

in sola lettura, e in quel caso marca il file come di sola lettura. Non sarà possibile superare tale condizione tramite il comando |**:write**|.

write-device
Quando il nome del file è in realtà un nome di dispositivo, Vim non fa un backup (in quel caso, sarebbe impossibile). Occorre usare "!", per scriverci sopra, perché il dispositivo esiste già.
Esempio per Unix: >
```
:w! /dev/lpt0
```
e per MS-DOS o MS-Windows: >
```
:w! lpt0
```
Per Unix un dispositivo viene individuato quando il nome non designa un file normale o una directory. Una pipe fifo o "named pipe" è vista anch'essa come un dispositivo da Vim.
Per MS-DOS e MS-Windows il dispositivo è individuato dal nome che ha:
```
AUX
CON
CLOCK$
NUL
PRN
COMn    n=1,2,3... etc.
LPTn    n=1,2,3... etc.
```
I nomi possono essere sia in maiuscolo che in minuscolo.

==========

5. Scrivere e uscire *write-quit*

:q *:quit*

:q[uit] Esce dalla finestra corrente. Esce da Vim se si è nell'ultima finestra rimasta. Il comando non viene eseguito quando ci sono state delle modifiche, e Vim rifiuta di uscire senza salvare (|**abandon**|) il buffer corrente, e quando l'ultimo file nella lista degli argomenti non è stato ancora editato.
Se ci sono delle linguette residue e si sta lasciando l'ultima delle linguette nella pagina di linguette corrente, la pagina di linguette corrente viene chiusa |**tab-page**|.
Viene innescato l'evento di autocomando |**QuitPre**|.

:conf[irm] q[uit] Esce, ma chiede conferma se ci sono dells modifiche non salvate, o se l'ultimo file nella lista degli argomenti non è stato ancora editato.
Vedere |**:confirm**| e **'confirm'**. {non in Vi}

:q[uit]! Esce senza riscrivere, anche quando il buffer corrente ha delle modifiche non salvate. Il buffer è scaricato, anche quando è attiva l'opzione **'hidden'**.
Se quella corrente è l'ultima finestra, e se c'è un buffer nascosto modificato, il buffer corrente è scartato e il primo buffer nascosto modificato diventa il buffer corrente.
Usare ":qall!" per uscire incondizionatamente.

:cq[uit] Esce sempre, senza scrivere, e restituisce un codice di errore. Vedere |**:cq**|. Usato per il modo QuickFix del compilatore Amiga Manx (vedere |**quickfix**|).
{non in Vi}

:wq

:wq **[++opt]** Scrive il file corrente ed esce. La scrittura non riesce quando il file è in sola lettura o il buffer non ha un nome. L'uscita non riesce quando l'ultimo file nella lista degli argomenti non è stato ancora editato.

:wq! **[++opt]** Scrive il file corrente ed esce. La scrittura non riesce quando il buffer corrente non ha un name.

:wq **[++opt]** **{file}** Scrive su **{file}** ed esce. L'uscita non riesce quando l'ultimo file nella lista degli argomenti non è stato ancora editato.

:wq! [++opt] {file} Scrive su {file} ed esce.

:[range]wq[!] [++opt] [file]
Come sopra, ma scrive solo le righe specificate con [range].

:x *:xit*
:[range]x[it][!] [++opt] [file]
Come ":wq", ma scrive solo quando sono state fatte delle modifiche.
Quando 'hidden' è impostato e ci sono ancora finestre, il buffer corrente diviene nascosto, dopo aver scritto il file.

:exi *:exit*
:[range]exi[t][!] [++opt] [file]
Come :xit.

ZZ
ZZ Scrive il file corrente, se modificato, ed esce (come ":x"). (Nota: Se ci sono più finestre per il file corrente, il file è scritto se era stato modificato e la finestra è chiusa).

ZQ
ZQ Esce ignorando eventuali modifiche (come ":q!"). {non in Vi}

FINESTRE E BUFFER MULTIPLI *window-exit*

:qa *:qall*
:qa[ll] Esce da Vim, se non ci sono buffer modificati. (Usare ":bmod" per passare al successivo buffer modificato). Quando l'opzione 'autowriteall' è impostata tutti i buffer modificati verranno scritti, come con |:wqall|. {non in Vi}

:conf[irm] qa[ll]
Esce da Vim. Chiede cosa fare quando qualche buffer è stato modificato. Vedere |:confirm|. {non in Vi}

:qa[ll]! Esce da Vim. Eventuali modifiche ai buffer sono perse. {non in Vi}
Vedere anche |:cquit|, che fa la stessa cosa, ma esce con un codice di ritorno diverso da zero.

:quita *:quitall*
:quita[ll][!] Come ":qall". {non in Vi}

:wqa[ll] [++opt] *:wqa* *:wqall* *:xa* *:xall*
:xa[ll] Scrive tutti i buffer modificati ed esce da Vim. Se ci sono buffer senza un nome di file, che sono in sola lettura o che non possono essere scritti per qualche altra ragione, Vim non uscirà. {non in Vi}

:conf[irm] wqa[ll] [++opt]
:conf[irm] xa[ll]
Scrive tutti i buffer modificati ed esce da Vim. Chiede all'utente quando incontra buffer in sola lettura o che non possono essere scritti per qualche altra ragione.
Vedere |:confirm|. {non in Vi}

:wqa[ll]! [++opt]
:xa[ll]! Scrive tutti i buffer modificati, anche quelli in sola lettura, ed esce da Vim. Se ci sono buffer senza un nome di file o che non possono essere scritti per qualche altra ragione, Vim non uscirà.
{non in Vi}

==========

6. Dialoghi *edit-dialogs*

:confirm *:conf*

:conf[irm] **{comando}** Esegue **{comando}**, e chiede all'utente quando un'operazione richiede conferma. Si può usare con i comandi |:q|, |:qa| e |:w| (in quest'ultimo caso per riscrivere un file che era stato aperto in sola lettura), e con ogni altro comando che può non essere eseguito in modo simile, come |:only|, |:buffer|, |:bdelete|, etc.

Esempi: >
```
  :confirm w pippo
<       Chiederà conferma quando "pippo" esiste già. >
  :confirm q
<       Chiederà conferma quando ci sono modifiche non salvate. >
  :confirm qa
<       Se c'è qualche buffer modificato e non ancora salvato, per ognuno
        sarà chiesto se si vuole salvarlo o scartare le modifiche.
        È possibile anche scegliere di salvare tutti i buffer ("Salva tutto")
        o nessun buffer ("Scarta tutto")
```

Se si vuole sempre usare ":confirm", va impostata l'opzione **'confirm'**.

:browse *:bro* *E338* *E614* *E615* *E616*

:bro[wse] **{comando}** Apre una finestra di dialogo che permette di scegliere un file che faccia da argomento a **{comando}**. Al momento i comandi riconosciuti sono |:e|, |:w|, |:wall|, |:wq|, |:wqall|, |:x|, |:xall|, |:exit|, |:view|, |:sview|, |:r|, |:saveas|, |:sp|, |:mkexrc|, |:mkvimrc|, |:mksession|, |:mkview|, |:split|, |:vsplit|, |:tabe|, |:tabnew|, |:cfile|, |:cgetfile|, |:caddfile|, |:lfile|, |:lgetfile|, |:laddfile|, |:diffsplit|, |:diffpatch|, |:open|, |:pedit|, |:redir|, |:source|, |:update|, |:visual|, |:vsplit|, e |:qall| se **'confirm'** è impostato.
{solo nelle GUI Win32, Athena, Motif, GTK e Mac}
Quando ":browse" non è possibile, viene emesso un messaggio di errore. Se la funzionalità |+browse| non è disponibile, o se **{comando}** non è in grado di usarla, il **{comando}** è eseguito senza il dialogo di scelta.
":browse set" equivale a |:options|.
Vedere anche |:oldfiles| per ":browse oldfiles".

La sintassi è chiarita meglio con qualche esempio: >
```
        :browse e $vim/pippo
<               Apre la finestra di selezione file nella directory $vim/pippo,
                ed edita il file scelto. >
        :browse e
<               Apre la finestra di selezione file nella directory specificata
                con l'opzione 'browsedir', ed edita il file scelto. >
        :browse w
<               Apre la finestra di selezione file nella directory del buffer
                corrente, avendo come nome di default quello del buffer
                corrente, e salva il buffer con il nome scelto. >
        :browse w C:/pluto
<               Apre la finestra di selezione file nella directory C:/pluto,
                avendo come nome di default quello del buffer corrente, e
                salva il buffer con il nome scelto.
```
Vedere anche l'opzione |'browsedir'|.
Per le versioni di Vim nelle quali non è prevista la possibilità di scelta dei file, il comando è eseguito senza modifiche.

browsefilter

Per MS-Windows e GTK, si possono modificare i filtri da usare nella finestra di dialogo per la selezione dei file. Impostando le variabili g:browsefilter o b:browsefilter, si possono cambiare i filtri a livello globale, oppure a livello locale, per ogni singolo buffer. La variabile è costituita da una stringa nel formato
"{nome del filtro}\t**{criterio-di-ricerca}**;**{criterio-di-ricerca}**\n"
dove {nome del filtro} è il testo che appare nel riquadro "Files of Type", e {criterio di ricerca} è il criterio in base al quale filtrare i nomi dei file. Più criteri di ricerca possono essere specificati, separati da ';'.

Per Motif si usa lo stesso formato, ma solo il primo criterio di ricerca specificato è effettivamente usato (Motif consente un solo criterio, ma si può adattare alle necessità modificandolo all'occasione).

Per esempio, per avere solo file relativi a Vim nel dialogo di scelta file, si può usare il seguente comando: >

```
let g:browsefilter = "Vim Scripts\t*.vim\nVim Startup file\t*vimrc\n"
```

Si possono usare differenti filtri, ognuno riferito a un particolare buffer, impostando la variabile b:browsefilter. Un posto appropriato per impostare b:browsefilter è in un plugin a livello di tipo file, in modo che il dialogo per la scelta di ulteriori file contenga elementi correlati al tipo di file correntemente in edit. Inconveniente: ciò rende difficile iniziare a editare un file di tipo differente. Per evitare questo, si potrebbe aggiungere "Tutti i file\t*.*\n" come ultimo filtro, in modo che l'utente possa ancora accedere a qualsiasi tipo di file desidera.

Per evitare di impostare browsefilter quando Vim in realtà non lo supporta, si può usare has("browsefilter"): >

```
if has("browsefilter")
   let g:browsefilter = "quel-che-si-vuole"
endif
```

==========

7. La directory corrente *current-directory*

Si possono usare i comandi |:cd| e |:lcd| per portarsi su un'altra directory, in modo da non dover scrivere il nome della directory in questione prima dei nomi di file che si desidera editare. La cosa ha importanza anche per eventuali comandi esterni che si vogliano eseguire, p.es., ":!ls".

Il cambio di directory non riesce quando il buffer corrente è modificato, il flag '.' è presente in 'cpoptions' e "!" non è usato nel comando.

:cd *E747* *E472*

:cd[!] Su sistemi non-Unix: stampa il nome della directory corrente. Su sistemi Unix: Cambia la directory corrente alla home directory. Si usi |:pwd| per stampare il nome della directory corrente in tutti i sistemi.

:cd[!] {percorso} Cambia la directory corrente a {percorso}. Se {percorso} è in formato relativo, viene ricercato nelle directory elencate in |'cdpath'|. Questo non cambia il nome di un file già aperto, perché il nome completo del percorso del file è ricordato. Tuttavia i file contenuti nella lista degli argomenti |arglist| possono ancora cambiare! In MS-DOS ciò può cambiare anche il disco su cui si sta lavorando. Per passare alla directory del file corrente: >

```
:cd %:h
```

<

:cd- *E186*

:cd[!] - Ritorna alla precedente directory corrente (prima dell'ultimo comando ":cd {percorso}"). {non in Vi}

:chd *:chdir*

:chd[ir][!] [path] Come |:cd|.

:lc *:lcd*

:lc[d][!] {percorso} Come |:cd|, ma imposta la directory corrente solo per la finestra corrente. La directory corrente nelle altre finestre non è cambiata. {non in Vi}

:lch *:lchdir*

:lch[dir][!] Come |:lcd|. {non in Vi}

:pw *:pwd* *E187*

:pw[d] Stampa il nome della directory corrente. {Vi: non c'è pwd}

Vedere anche |**getcwd()**|.

Finché non è stato usato alcun comando |**:lcd**|, tutte le finestre hanno la stessa directory corrente. Se si usa un comando che passa a un'altra finestra, la directory corrente continua a rimanere la stessa.
Quando si usa un comando |**:lcd**| per una finestra, la directory specificata diventa la directory corrente per quella finestra. Le finestre in cui il comando |**:lcd**| non è stato usato continueranno ad avere la directory corrente globale. Se si salta da una finestra all'altra, la directory corrente sarà l'ultima directory corrente locale specificata per quella finestra. Se non ne è stata specificata nessuna, sarà usata la directory corrente globale.
Quando si usa un comando |**:cd**|, la finestra corrente perde la sua directory corrente locale e da lì in poi userà la directory corrente globale.

Dopo aver dato |**:cd**| verrà usato il nome di percorso completo per leggere e scrivere i file. Questo può causare dei problemi su file-system che risiedono in rete. La conseguenza dell'uso del nome di percorso completo è che i nomi di file correntemente in uso continueranno a essere riferiti agli stessi file. Esempio: Se si ha un file a:test e una directory a:vim i comandi ":e test" ":cd vim" ":w" riscriveranno il file a:test invece di scrivere il file a:vim/test. Ma se il comando dato è ":w test" verrà scritto il file a:vim/test, perché è stato dato un nuovo nome di file, e non si è fatto riferimento al nome di file che era presente prima di dare il comando ":cd".

==========

8. Editare file binari *edit-binary*

Sebbene Vim sia stato fatto per editare file di testo, è possibile editare file binari. L'argomento di Vim |**-b**| (b per binario) fa sì che Vim esegua le operazioni di I/O sui file in modalità binaria, e imposta alcune opzioni per editare file binari (**'binary'** on, **'textwidth'** a 0, **'modeline'** off, **'expandtab'** off). Impostare l'opzione **'binary'** ha lo stesso effetto.
Non dimenticare di farlo PRIMA di leggere il file.

Ci sono alcune cose da tener presenti editando file binari:
- Quando si editano file eseguibili il numero di caratteri non deve cambiare. Si usino solo i comandi "R" o "r" per modificare il testo. Non si devono cancellare caratteri con "x" o usando il tasto di cancellazione all'indietro.
- Impostare l'opzione **'textwidth'** a 0. Altrimenti le righe potrebbero essere inaspettatamente divise in due.
- Se non ci sono molti caratteri **<EOL>**, le righe saranno molto lunghe. Se si vuole editare una riga che non è interamente contenuta in un schermata, si metta a off l'opzione **'wrap'**.
  In questo modo lo scorrimento usato è quello orizzontale. Se una riga diventa troppo lunga (più di circa 32767 caratteri nell'Amiga, molto di più in sistemi e 32 bit, vedere |**limits**|) non si può editare quella riga. La riga verrà spezzata in due in fase di lettura del file. È anche possibile che si abbia un errore di "memoria esaurita" in fase di lettura del file.
- Ci si deve assicurare che l'opzione **'binary'** sia impostata PRIMA di caricare il file. Altrimenti sia **<CR><NL>** che **<NL>** sono visti come caratteri di fine riga, e quando il file è riscritto i caratteri **<NL>** saranno rimpiazzati da **<CR><NL>**.
- I caratteri **<Nul>** sono visualizzati sullo schermo come ^@.
  Si possono immettere digitando "**CTRL-V CTRL-@**" o "**CTRL-V** 000"
  {Vi non è in grado di gestire caratteri <Nul> in un file}
- Per inserire un carattere **<NL>** nel file basta dividere in due una riga. Quando quel buffer verrà riscritto, il carattere **<NL>** sarà scritto per marcare la fine della riga (**<EOL>**).
- Vim normalmente aggiunge un carattere **<EOL>** a fine file, se non è già presente. Impostando l'opzione **'binary'** questo non viene fatto. Se si vuole aggiungere a fine file un **<EOL>**, si imposti l'opzione **'endofline'**. Si può anche leggere il valore di quest'opzione per controllare se c'era oppure no un **<EOL>** alla fine dell'ultima riga (nella schermata non è possibile visualizzarlo nel testo).

==========

9. Cifratura *encryption*

Vim è in grado di scrivere file cifrati, e di rileggerli. Il testo cifrato

cifrato non può essere letto senza fornire la chiave corretta.
{disponibile solo se compilato con la funzionalità |+cryptv|} *E833*

Sia il testo nel file di swap che quello nel file undo sono cifrati. *E843*
Tuttavia, la cifratura è fatta a blocchi, e questo può ridurre il tempo
necessario per tentare di scoprire la chiave utilizzata.
Si può disabilitare il file di swap, ma in quel caso una caduta di sistema
potrebbe determinare la perdita del lavoro fatto. Il file di undo può
essere disabilitato senza troppi inconvenienti. >
```
:set noundofile
:noswapfile edit cose-segrete
```

Nota: Il testo in memoria non è cifrato. Un amministratore di sistema è in
grado di vedere il testo mentre lo si sta editando. Anche quando si
applicano dei filtri al testo, con il comando ":!filter" o usando
":w !comando", il testo non è cifrato, e quindi potrebbe essere
visto da altri. Il file 'viminfo' non è cifrato.

Per editare un testo da mantenere molto segreto, si potrebbe fare così: >
```
:set noundofile viminfo=
:noswapfile edit cose-segrete.txt
```
Va tenuto presente che senza un file di swap si rischia di perdere il
lavoro fatto in caso di caduta del sistema o di mancanza di corrente
elettrica.

ATTENZIONE: Se si sbaglia a immettere la chiave e in seguito si riscrive
il file e si esce da Vim, il testo andrà perduto!

Il modo normale di lavorare con del testo cifrato, è di usare il comando
":X", che chiede di immettere una chiave. Ogni successivo comando di
scrittura userà quella chiave per cifrare il file. Se in seguito si vuole
editare di nuovo quel file, Vim richiederà l'immissione di una chiave.
Se si immette la stessa chiave usata in fase di scrittura, il testo sarà
nuovamente leggibile. Se si fornisce una chiave errata, il testo sarà
completamente incomprensibile.

:X
:X	Richiede una chiave di cifratura. La digitazione è fatta senza
	visualizzare il testo immesso, in modo che se qualcuno sta guardando
	la schermata, non vedrà la chiave immessa.
	La chiave immessa è mantenuta nell'opzione 'key', che è usata per
	cifrare il file al momento della scrittura. Il file (su disco) resta
	invariato finché non viene scritto. Vedere anche |-x|.

Quando il testo viene scritto viene usato il valore dell'opzione 'key'. Quando
l'opzione non è nulla, il file scritto verrà cifrato, usando il valore come
chiave di cifratura. Un numero magico è messo all'inizio del file, in modo
che Vim possa riconoscere che il file è stato cifrato.

Per disabilitare la cifratura, si assegni un valore nullo all'opzione 'key': >
```
:set key=
```

Si può usare l'opzione 'cryptmethod' per scegliere il tipo di cifratura, tra
uno di questi: >
```
:setlocal cm=zip        " metodo debole, tenuto per compatibilità
:setlocal cm=blowfish   " metodo con problemi
:setlocal cm=blowfish2  " metodo di forza media
```

Questo va fatto prima di scrivere il file. Quando si legge un file cifrato
l'opzione sarà automaticamente impostata al metodo usato quando il file era
stato scritto. Si può cambiare 'cryptmethod' prima di riscrivere quel
file, per farlo scrivere con un metodo di cifratura diverso.

Per impostare il metodo di default da usare per file nuovi, si scriva nel
file |vimrc|: >
```
set cm=blowfish2
```
L'uso di "blowfish2" è altamente raccomandato. Si usi un altro metodo solo
se si ha a disposizione una versione datata di Vim che non lo supporta.

Il messaggio dato quando si legge o scrive un file conterrà "[cifrato]" se
si usa zip, "[blowfish]" se si usa blowfish, etc.

Quando si scrive un file di undo, la stessa chiave e lo stesso metodo

saranno usati per il testo contenuto nel file di undo. |**persistent-undo**|.

Per controllare se è disponibile il supporto blowfish si possono usare le condizioni seguenti: >
```
        has('crypt-blowfish')
        has('crypt-blowfish2')
```
Questo controllo funziona a partire da Vim 7.4.1099 mentre il supporto per blowfish era stato aggiunto in precedenza.
Quindi se il controllo ha esito negativo, non significa che blowfish non sia disponibile. Si può controllare la disponibilità di blowfish con: >
```
        v:version >= 703
```
E per blowfish2 con: >
```
        v:version > 704 || (v:version == 704 && has('patch401'))
```
Se si è sicuri che Vim include la patch 7.4.237 un controllo più semplice è: >
```
        has('patch-7.4.401')
```
<

E817 *E818* *E819* *E820*

Se la cifratura non funziona correttamente, si potrebbe non riuscire più a rileggere un file, dopo averlo scritto. Per questo motivo, viene effettuato un controllo per stabilire se la cifratura funziona come dovrebbe. Se si riceve uno di questi errori, non si deve scrivere il file in formato cifrato! Invece occorre ricompilare il codice binario di Vim per essere in grado di proseguire correttamente.

E831 Questo è un errore interno, "non può succedere". Se è possibile riprodurlo, ci si dovrebbe mettere in contatto con gli sviluppatori di Vim.

Quando si legge un file che è stato cifrato e l'opzione **'key'** non è nulla, la chiave indicata sarà usata per decifrare. Se il valore è invece impostato alla stringa nulla, la chiave da usare verrà richiesta all'utente.
Se non si fornisce alcuna chiave, o se si fornisce la chiave sbagliata, il file viene editato senza essere decifrato. Non viene inviato alcun avviso per dire che si sta usando una chiave sbagliata (ciò è fatto per creare difficoltà a chi voglia usare il metodo della forza bruta per trovare la chiave).

Se si desidera iniziare a leggere un file che usa una chiave differente, si imposti l'opzione **'key'** alla stringa nulla, in modo che Vim richieda la nuova chiave all'utente. Non si usi il comando ":set" per immettere la nuova chiave, altri alle vostre spalle potrebbero leggere il comando che si sta immettendo.

Poiché il valore dell'opzione **'key'** dovrebbe essere segreto, questo valore non può mai essere visualizzato. Quest'opzione non dovrebbe essere impostata in un file vimrc.

Un file cifrato si può riconoscere dal comando "file", se si aggiungono le righe seguenti al file "/etc/magic", "/usr/share/misc/magic" o nel posto dove risiede il file "magic" nel sistema in uso: >
```
     0  string  VimCrypt~       Vim encrypted file
     >9 string  01      - "zip" cryptmethod
     >9 string  02      - "blowfish" cryptmethod
     >9 string  03      - "blowfish2" cryptmethod
```

Note:
- La cifratura non è possibile quando si effettuano delle conversioni con **'charconvert'**.
- Il testo che si copia o cancella va a uno dei registri numerati. I registri possono venir salvati nel file .viminfo, dove qualcuno potrebbe leggerli. Modificare la propria opzione **'viminfo'** per sicurezza.
- Qualcuno può immettere comandi in Vim in caso di assenza momentanea dalla postazione di lavoro, e non dovrebbe essere in grado di procurarsi la chiave di cifratura.
- Se si fa un errore di battitura immettendo la chiave, si potrebbe non essere più un grado di rileggere il testo cifrato!
- Se si immette la chiave con un comando ":set key=valore", questo può essere presente nella cronologia dei comandi, mostrando in chiaro a chi legga il file viminfo il valore dell'opzione **'key'**.
- Una sicurezza al 100% non ci sarà mai. La cifratura in Vim non è stata testata per verificarne la robustezza.
- La chiave usata con il metodo di cifratura (**'cryptmethod'**) "zip" può essere facilmente scoperta. Una chiave di 4 caratteri in un'ora circa,

una chiave di 6 caratteri in un giorno (su un PC Pentium 133).
È necessario conoscere qualche parte di testo che è contenuta
nel file. Una persona esperta in materia è in grado di scoprire qualsiasi
chiave sia stata immessa. Quando il testo è stato decifrato, la chiave
usata può essere resa disponibile, e altri file cifrati con la stessa
chiave possono essere decifrati.
- Pkzip usa la stessa cifratura usata da Vim se **'cryptmethod'** è "zip", e
il governo degli Stati Uniti non ha obiezioni a esportare file cifrati
con questo metodo. Il file pubblico di pkzip APPNOTE.TXT descrive
l'algoritmo in maniera dettagliata.
- L'implementazione di Vim con **'cryptmethod'** "blowfish" ha un
difetto. È possibile decifrare i primi 64 byte di un file, e in alcuni
casi anche altre parti del file. Il suo uso non è raccomandato, ma è
ancora il metodo più forte disponibile nelle versioni di Vim 7.3 e 7.4.
Il metodo "zip" è ancora più debole.
- Vim proviene dall'Olanda. È da qui che provengono i file sorgenti.
Quindi il codice di cifratura non è esportato dagli Stati Uniti d'America.

==========

10. Marcature orarie *timestamp* *timestamps*

Vim conserva la data e ora dell'ultima modifica, le autorizzazioni e la
dimensione di un file quando si è iniziato a editarlo. Questo serve
a evitare di avere due differenti versioni dello stesso file (senza saperlo).

Dopo che si è eseguito (internamente a Vim) un comando della shell (|**:!cmd**|
|**suspend**| |**:read!**| |**K**|), vengono controllate data e ora dell'ultima modifica,
autorizzazione e dimensione dei file contenuti nei buffer di una finestra.
Vim eseguirà ogni autocomando associato all'evento |**FileChangedShell**| o
visualizzerà un messaggio di avvertimento per ogni file che è
stato modificato. Nella GUI ciò viene fatto quando Vim ottiene di nuovo
il focus di input.

E321 *E462*
Se si vuole ricaricare automaticamente un file che sia stato cambiato
esternamente a Vim, impostare l'opzione **'autoread'**. L'opzione non è
tuttavia attiva quando il file è stato modificato da Vim durante la
sessione di edit.

Nota Se è stato definito un autocomando per l'evento fileChangedShell
non verrà emesso alcun messaggio o richiesta da parte di Vim. Si
suppone che sia l'autocomando a gestire la situazione.

Non ci sono avvertimenti riguardo alle modifiche a una directory (p.es.,
mentre si usa |**netrw-browse**|). Ma un avvertimento viene dato se si
inizia a editare un file nuovo, e successivamente viene creata una directory
con quello stesso nome (esternamente a Vim).

Quando Vim si accorge che la data e ora di ultima modifica di un file è
cambiata, e il file è in corso di modifica, ma non è ancora stato
modificato, Vim controlla se il contenuto del file non è cambiato.
Questo viene fatto leggendo ancora il file (in un buffer nascosto, che è
immediatamente eliminato subito dopo) e confrontando il testo. Se il testo
è uguale, non viene inviato alcun messaggio di avvertimento.

Se non si viene avvertiti con sufficiente frequenza, si può usare il comando
seguente.

:checkt *:checktime*

:checkt[ime] Controlla se qualche buffer sia stato cambiato
esternamente a Vim.
Questo controllo serve a generare un avvertimento
se si rischiasse di avere due versioni differenti
di uno stesso file.
Se il comando è richiamato da un autocomando, da un
comando ":global" o non è immesso direttamente,
l'effettivo controllo è rinviato fino al momento
in cui gli effetti collaterali (caricando di nuovo
il file) siano innocui.
Per ogni buffer caricato si controlla se il relativo
file associato sia stato cambiato. Se il file
risulta cambiato, Vim agisce di conseguenza.
Se non ci sono modifiche nel buffer di Vim e l'opzione

'autoread' è attiva, il buffer è nuovamente caricato.
Se invece ci sono modifiche, è offerta all'utente la
scelta se ricaricare il file oppure no. Se
il file è stato nel frattempo cancellato, Vim invia
un messaggio di errore. Se il file non esisteva in
precedenza, ma è stato creato nel frattempo, Vim
invia un messaggio di avvertimento al riguardo.
Dopo che il file è stato controllato, la data e ora
di ultima modifica sono aggiornate, e nessun
ulteriore messaggio di avvertimento viene inviato.

:[N]checkt[ime] {filename}
:[N]checkt[ime] [N]
Controlla la data e ora di ultima modifica di un
particolare buffer. Il buffer può essere indicato
per nome, per numero, o con una stringa di ricerca.

E813 *E814*
Vim caricherà di nuovo il buffer se lo si chiede. Se c'è una finestra
visibile che contiene quel buffer, il caricamento avverrà in quella
finestra. Se no si usa una finestra speciale, in modo
che la maggior parte degli autocomandi sia attiva. Questa finestra non
si può chiudere. Viene applicatalcune qualche altra restrizione. La cosa
migliore è di assicurarsi che non succeda nulla fuori dal buffer corrente.
P.es., impostare delle opzioni locali per una finestra potrebbe aver
effetto sulla finestra sbagliata. Spezzare in due la finestra (":split"),
fare qualcosa nella nuova finestra e chiuderla non dovrebbe dare problemi.
Chiudere finestre e buffer differenti da quelli correnti potrà creare
problemi.

Prima di scrivere un file viene controllata la data e l'ora dell'ultimo
aggiornamento. Se questa è cambiata, Vim chiederà se si vuole veramente
sovrascrivere il file:

```
AVVISO: File modificato dopo essere stato letto!!!
Vuoi davvero riscriverlo (y/n)?
```

Se si risponde 'y' Vim riscriverà il file. Se si risponde 'n' l'operazione
di scrittura è annullata. Se è stato usato ":wq" o "ZZ" Vim non
esce subito, ma offre prima di uscire un'altra possibilità di scrivere
il file.

Il messaggio visto sopra normalmente indica che qualcuno ha scritto
il file dopo che la sessione di modifica è iniziata. Potrebbe trattarsi di
un'altra persona, e in quel caso si dovrebbe controllare se le proprie
modifiche e quelle dell'altra persona dovrebbero essere
entrambe applicate. Per far questo, scrivere il file con un altro nome e
controllare quali sono le differenze (il programma "diff" può essere usato
a questo scopo).

È anche possibile che la modifica del file sia stata fatta dall'utente
stesso, da un'altra sessione di edit o con un altro comando (p.es., un
comando di filtro). A quel punto si dovrebbe essere in grado di decidere
quale versione del file si desidera tenere.

Esiste una situazione in cui il messaggio è inviato anche se non c'è nulla
di sbagliato. In un sistema Win32, nel giorno in cui inizia l'ora legale.
C'è qualcosa nelle librerie Win32 che lascia Vim confuso riguardo alla
differenza di tempo di un'ora (dovuta all'ora legale). Il problema scompare
da solo il giorno dopo.

==========
11. Ricerca di file *file-searching*

{non disponibile se compilato senza la funzionalità |+path_extra|}

La ricerca di file è correntemente usata per le opzioni 'path', 'cdpath' e
'tags', e per le funzioni |finddir()| e |findfile()|. Altri comandi usano
metacaratteri |wildcard|, il che funziona in modo leggermente differente.

Ci sono tre differenti tipi di ricerca:

1) Ricerca all'ingiù: *starstar*
   La ricerca in giù usa i metacaratteri '*', '**' ed eventualmente
   altri, supportati dal sistema operativo in uso. '*' e '**' sono
   gestiti internamente da Vim, e quindi funzionano con qualsiasi sistema
   operativo. Nota "**" funziona come metacarattere solo se si trova
   scritto all'inizio di un nome.

   L'uso di '*' è abbastanza semplice: Corrisponde a 0 o più caratteri.
   In un'espressione di ricerca si scrive ".*".   Nota Il "." non è usato
   per la ricerca file.

   '**' è più sofisticato:
      - Cerca solo directory.
      - Trova corrispondenze fino a 30 directory di profondità per default,
        e quindi si può usare per ricercare un intero albero di directory.
      - Il numero massimo di livelli ai quali scendere per cercare
        corrispondenze può essere specificando aggiungendo un numero a '**'.
        Quindi '/usr/**2' potrebbe corrispondere a: >
                /usr
                /usr/include
                /usr/include/sys
                /usr/include/g++
                /usr/lib
                /usr/lib/X11
                ....
<   La directory '/usr/include/g++/std' non viene individuata, perché ciò
        implicherebbe di discendere per tre livelli.
        L'intervallo che è possibile specificare va da 0 ('**0' è rimosso)
        fino a 100.
        Se il numero dato è minore di 0 viene preso il default di 30, se il
        numero è maggiore di 100, si usa 100. Il sistema ha a sua volta un
        limite sulla lunghezza massima di un percorso, normalmente 256 o
        1024 byte.
      - '**' può solo essere alla fine del percorso, o essere seguito da un
        separatore di percorso, oppure da un numero e da un separatore di
        percorso.

   Si possono combinare '*' e '**' in un qualsiasi ordine: >
        /usr/**/sys/*
        /usr/*tory/sys/**
        /usr/**2/sys/*

2) Ricerca all'insù:
   Qui si può dare un nome di directory e da qui cercare nell'albero delle
   directory verso l'alto, alla ricerca di un file. Si può indicare una
   directory in cui fermarsi, per limitare la ricerca. Le directory alla
   quale fermarsi sono messe in fondo al percorso (per l'opzione 'path') o
   al nome di file (per l'opzione 'tags') con un ';'. Se si desiderano
   più directory nelle quali fermarsi, vanno separate fra loro con un ';'.
   Se non si desidera limitare la ricerca ("ricerca in su fino alla
   directory radice) basta mettere un ';'. >
        /usr/include/sys;/usr
< cercherà in: >
           /usr/include/sys
           /usr/include
           /usr
<
   Se si usa un percorso relativo, la ricerca in su inizia dalla directory
   corrente di Vim o nella directory del file corrente (se il percorso
   relativo inizia con './' e 'd' non è incluso nell'opzione 'cpoptions').

   Se il percorso corrente di Vim è /u/user_x/work/release e si digita >
        :set path=include;/u/user_x
< e poi si cerca un file con il comando |gf| il file è ricercato in: >
        /u/user_x/work/release/include
        /u/user_x/work/include
        /u/user_x/include

3) Ricerca mista in su e in giù:
   Se il percorso corrente di Vim è /u/user_x/work/release e si digita >
        set path=**;/u/user_x
< e poi si cerca un file con il comando |gf| il file è ricercato in: >
        /u/user_x/work/release/**

```
        /u/user_x/work/**
        /u/user_x/**
<
```
   ATTENZIONE! Questo comando potrebbe richiedere molto tempo
   in fase di esecuzione, poiché la ricerca di
   '/u/user_x/**' include '/u/user_x/work/**' e
   '/u/user_x/work/release/**'. Quindi '/u/user_x/work/release/**' è
   ricercato tre volte e '/u/user_x/work/**' è carcato due volte.

   Nell'esempio precedente, si potrebbe impostare il percorso a: >
```
        :set path=**,/u/user_x/**
<  Ciò ricerca in:
        /u/user_x/work/release/** ~
        /u/user_x/** ~
```
   Le ricerche riguardano le stesse directory, ma in ordine differente.

   Nota Il completamento per i comandi ":find", ":sfind", e ":tabfind" non
   funzionano, attualmente, con elementi in 'path' che contengono un url o
   che usano il doppio asterisco con le notazioni delimitatore di profondità
   (/usr/**2) o per la ricerca in su (;).

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