usr_43.txt Per **Vim version 8.0.** Ultima modifica: 2015 Oct 23

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

Utilizzo dei tipi di file

Quando state scrivendo un file di un certo tipo, per esempio un programma C o uno script di shell, spesso usate lo stessa scelta di impostazioni e di mappature. Rapidamente diverrete stanchi di impostarli manualmente ogni volta. Questo capitolo spiega come farlo automaticamente.

|**43.1**| Plugin per un tipo di file
|**43.2**| Aggiunta di un tipo di file

Capitolo seguente: |**usr_44.txt**| Evidenziazione della vostra sintassi
Capitolo precedente: |**usr_42.txt**| Aggiungere nuovi menù
Indice: |**usr_toc.txt**|

==========

43.1 Plugin per un tipo di file *filetype-plugin*

Come iniziare ad usare i plugin per i tipi di file è già stato discusso qui: |**add-filetype-plugin**|. Ma probabilmente non siete soddisfatti dalle impostazioni di default, perché sono state ridotte al minimo. Supponiamo che per i file in C vogliate impostare l'opzione 'softtabstop' a 4 e definire una mappatura per inserire un commento di tre linee. Potete farlo con due sole operazioni:

your-runtime-dir

1. Creare una vostra runtime directory. Su Unix questa è abitualmente "~/.vim". In questa directory create la directory "ftplugin": >

```
mkdir ~/.vim
mkdir ~/.vim/ftplugin
```

<
   Se non siete su Unix, controllate il valore dell'opzione 'runtimepath' per vedere dove Vim cercherà la directory "ftplugin": >

```
set runtimepath
```

<
   Usare normalmente il primo nome di directory (quello che precede la prima virgola). Oppure potete premettere un nome di directory all'opzione 'runtimepath' nel vostro file |**vimrc**| se non vi piace il valore di default.

2. Creare il file "~/.vim/ftplugin/c.vim", con il contenuto: >

```
setlocal softtabstop=4
noremap <buffer> <LocalLeader>c o/**************<CR><CR>/<Esc>
let b:undo_ftplugin = "setl softtabstop< | unmap <buffer> <LocalLeader>c"
```

Provate ora ad editare un file C. Potreste accorgervi che l'opzione 'softtabstop' è impostata a 4. Ma quando editate un altro file è riportata al valore di default che è zero. Questo succede perché si è usato il comando ":setlocal". Questo imposta l'opzione 'softtabstop' solo localmente al buffer. Appena aprite un altro buffer, viene impostata al valore proprio di quel buffer. Per un nuovo buffer prenderà il valore di default o quello impostato dall'ultimo comando ":set".

Analogamente, la mappatura per "\c" sparirà quando si lavora in un altro buffer. Il comando ":map<buffer>" crea una mappatura locale al buffer corrente. Questo funziona con ogni comando di mappatura: ":map!", ":vmap", ecc. Il |**<LocalLeader>**| è sostituito con il valore della variabile "maplocalleader".

La riga che imposta b:undo_ftplugin serve quando il tipo del file è impostato a un altro valore. In quel caso si vorrà annullare quelle preferenze. La variabile b:undo_ftplugin è eseguita come un comando. Occorre prestare attenzione ai caratteri con significato speciale all'interno di una stringa, come la barra inversa.

Trovate degli esempi di plugin per i tipi di file in questa directory: >

```
        $VIMRUNTIME/ftplugin/
```

Ulteriori approfondimenti su come scrivere plugin per i tipi di file si trovano qui: |**write-plugin**|.

==

43.2 Aggiunta di un tipo di file

Se usate un tipo di file che non viene riconosciuto da Vim, ecco come si può farglielo riconoscere. Avete bisogno di una vostra runtime directory. Vedere sopra |**your-runtime-dir**|.

Creare un file "filetype.vim" che contenga un autocomando per il vostro tipo di file. (Gli autocomandi sono stati spiegati nella sezione |**40.3**|). Esempio: >

```
        augroup filetypedetect
        au BufNewFile,BufRead *.xyz     setf xyz
        augroup END
```

Questo riconoscerà tutti i file che terminano in ".xyz" come tipo di file "xyz". Il comando ":augroup" pone questo autocomando nel gruppo "filetypedetect". Questo permette di rimuovere tutti gli autocomandi di riconoscimento del tipo di file facendo ":filetypeoff". Il comando "setf" imposta l'opzione 'filetype' al suo argomento, a meno che non fosse già impostata. Questo evita che 'filetype' sia impostato due volte.

Potete usare molti schemi differenti per individuare il nome del vostro file. Si possono includere anche i nomi delle directory. Vedere |**autocmd-patterns**|. Per esempio che i file in "/usr/share/scripts/" siano tutti file "ruby", ma non abbiano l'estensione attesa. Basta aggiungere una riga all'esempio di sopra: >

```
        augroup filetypedetect
        au BufNewFile,BufRead *.xyz                     setf xyz
        au BufNewFile,BufRead /usr/share/scripts/*      setf ruby
        augroup END
```

Comunque, se ora aprite il file /usr/share/scripts/README.txt, non è un file di Ruby. Il pericolo di uno schema che finisce in "*" è che individua immediatamente troppi tipi di file. Per evitare problemi con ciò, mettete il file filetype.vim in un'altra directory, una che sia alla fine di 'runtimepath'. Per Unix, ad esempio, potreste usare "~/.vim/ultimo/filetype.vim".
   Ora mettete il riconoscimento dei file di testo in ~/.vim/filetype.vim: >

```
        augroup filetypedetect
        au BufNewFile,BufRead *.txt                     setf text
        augroup END
```

Questo file viene trovato nel 'runtimepath' per primo. Poi scrivete questo in ~/.vim/ultimo/filetype.vim, che viene trovato per ultimo: >

```
        augroup filetypedetect
        au BufNewFile,BufRead /usr/share/scripts/*      setf ruby
        augroup END
```

Quello che succede ora è che Vim cerca i file "filetype.vim" in ogni directory del 'runtimepath'. Per primo viene trovato ~/.vim/filetype.vim. L'autocomando per individuare i file *.txt è definito lì. Poi Vim trova il file filetype.vim in $VIMRUNTIME, che è a metà strada nel 'runtimepath'. Infine viene trovato ~/.vim/ultimo/filetype.vim e viene aggiunto l'autocomando per individuare i file ruby in /usr/share/scripts.
   Ora, quando editate /usr/share/scripts/README.txt, gli autocomandi vengono provati nell'ordine in cui sono stati trovati. Lo schema *.txt è soddisfatto, e così si esegue "setf text" e il tipo di file è impostato a "text". Anche lo schema per ruby è soddisfatto, e anche "setf ruby" viene eseguito. Ma siccome 'filetype' è stato già impostato a "text", questa seconda volta non succede nulla.
   Quando invece si apre /usr/share/scripts/foobar vengono eseguiti gli stessi autocomandi. Ma solo quello per ruby è soddisfatto, e "setf ruby" imposta 'filetype' a ruby.

RICONOSCIMENTO PER CONTENUTO

Se il vostro file non può essere riconosciuto semplicemente dal nome,
potreste essere in grado di riconoscerlo dal suo contenuto. Per esempio,
molti file di script iniziano con una linea del tipo:

```
#!/bin/xyz ~
```

Per riconoscere questo file di comandi create un file "scripts.vim" nella
vostra runtime directory (lo stesso posto in cui va filetype.vim). Potrebbe
apparire così: >

```
if did_filetype()
  finish
endif
if getline(1) =~ '^#!.*[/\\]xyz\>'
  setf xyz
endif
```

Il primo controllo con did_filetype() serve ad evitare di controllare i
contenuti dei file per i quali filetype era già stato riconosciuto dal nome.
Questo evita di sprecare del tempo a controllare il file quando il comando
"setf" non farà nulla.
Il file scripts.vim viene letto da un autocomando nel default file
filetype.vim. Quindi l'ordine di verifica è:

1. i file filetype.vim prima di $VIMRUNTIME in 'runtimepath'
2. la prima parte di $VIMRUNTIME/filetype.vim
3. tutti i file scripts.vim in 'runtimepath'
4. il resto di $VIMRUNTIME/filetype.vim
5. i file filetype.vim dopo $VIMRUNTIME in 'runtimepath'

Se questo non vi basta, aggiungete un autocomando che sia soddisfatto da ogni
file, e che legga uno script od esegua una funzione che controlli il contenuto
del file.

==

Capitolo seguente: |**usr_44.txt**| Evidenziazione della vostra sintassi

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