

Vim IDE per Php

Mario Santagiuliana

8 aprile 2010

Sommario

Se si programma codice php è importante avere un buon strumento che aiuti nella creazione e scrittura del codice.

Per scrivere applicazioni php basta anche un semplice editor di testo oppure si può utilizzare un Integrated Development Enviroment (**IDE**). Nel mondo open-source esistono moltissimi strumenti per scrivere in modo semplice e veloce i propri script o codici.

Io utilizzo Vi Improved (**Vim**), lo reputo molto potente e in grado di velocizzare di molto lo sviluppo e la scrittura di qualsiasi cosa, dopo aver compreso il suo funzionamento e dopo averlo opportunamente configurato ed eventualmente esteso (con appositi plugin) secondo le proprie necessità e gusti.

Voglio dunque presentare i passi che ho seguito per poter tramutare il mio editor di testo preferito in un potente editor per il linguaggio Php, fino a farlo diventare pressochè un IDE perfetto per Php che non ha nulla da invidiare ad altri strumenti di sviluppo.

Questo mini-howto si rivolge principalmente ad un utente che utilizza come sistema operativo fedora. Io infatti uso fedora (al momento in cui scrivo fedora 12 a 64bit). Ciò non significa che i passi proposti non possano essere seguiti su di un'altra distribuzione Linux o simil Unix, bisognerà solo fare attenzione ad alcune piccole differenze.

Ciò che presento è perfettamente funzionante sulla serie 7 di Vim installato su fedora 12 a 64 bit. Se esistono esperienze che confermino il buon funzionamento dei passaggi da me proposti anche in altre architetture e Sistema Operativo (**S.O.**) prego di farmi sapere.

Prima di continuare voglio far sapere che in rete si trovano moltissimi articoli e guide simili a ciò che ho scritto.

Questo articolo:

<http://www.koch.ro/blog/index.php?archives/63-VIM-an-a-PHP-IDE.html> mi ha dato dei buoni suggerimenti su come iniziare a fare la mia prima configurazione di Vim.



L'articolo è rilasciato sotto Licenza Creative Commons "Attribuzione-Non commerciale-Condividi allo stesso modo 2.5 Italia" reperibile al seguente link: <http://creativecommons.org/licenses/by-nc-sa/2.5/it/>.

Indice

1 Installare Vim

2

2	Configurazione base	2
3	ManPageView documentazione a portata di mano	3
4	Commentare il proprio codice Php	3
5	Abilitare l'auto-completamento in Vim	4
6	Ancora un po' di completamento automatico in Vim	4
7	Ricerca all'interno del codice con Cscope	4
8	Navigazione all'interno del codice con ctags	5
9	Source Code Browser con Taglist Plugin	6
10	Debug del codice Php con Xdebug	6
11	Estendere il % di Vim	8
12	Un piccolo aiuto con Closetag	9
13	Conclusioni	9

1 Installare Vim

Per incominciare bisogna installare Vim, aprendo un terminale basta digitare in Fedora:

```
$ su -c "yum install vim-minimal vim-common vim-enhanced"
```

Non esplico l'installazione e la configurazione di un server [LAMP](#).

2 Configurazione base

Fedora, nell'installazione standard di Vim, da già una configurazione di base di quest'ultimo. La sintassi colorata è già attivata per esempio. Si può leggere il file di configurazione di default di Vim in `/etc/vimrc`.

Ho però voluto personalizzare ulteriormente la configurazione di Vim aggiungendo alcune caratteristiche. Ho creato dunque un file di nome `.vimrc` nella home directory del mio utente e al suo interno ho aggiunto quanto segue:

```
set number "attivo la visualizzazione del numero delle righe
set mouse=a "attivo il supporto per l'uso del mouse
filetype on
autocmd FileType c,cpp,php set cindent shiftwidth=4 "vedi di seguito
set splitbelow "quando apro una nuova finestra in Vim voglio che si
  apra in basso e non in alto
```

Il comando che rende Vim già un ottimo editor per codice Php è il penultimo: `autocmd FileType c,cpp,php set cindent shiftwidth=4`. E' sempre importante creare una buona indentazione del proprio codice. Con il comando `filetype on` ho attivato in Vim la possibilità di riconoscere i tipi di file che sta caricando e con il penultimo comando

ho istruito Vim ad utilizzare l'indentazione standard del codice C per tutti i file con estensione c o cpp o php.

Il Php è abbastanza simile al C o al C++ come sintassi, quindi è una buona scelta usare l'indentazione del proprio codice come se fosse codice C con una indentazione di 4 spazi bianchi.

Ora però è venuto il momento di estendere Vim in modo tale da essere ancora più veloci e produttivi nella creazione dei propri script.

3 ManPageView documentazione a portata di mano

Una cosa importante è avere sempre a portata di mano la documentazione di Php. Doversi spostare in un browser web e navigare alla ricerca della documentazione di una determinata funzione è abbastanza noioso e fa perdere tempo. Io voglio poter leggere direttamente la documentazione di Php da Vim. Esiste un [plugin scritto da Charles Campbell](#) per far questo: [ManPageView](#).

L'installazione è davvero semplice, basta scaricare il pacchetto dalla pagina del progetto e come specificato nella documentazione, dopo essersi spostati nella directory del pacchetto scaricato procedere con:

```
vim manpageview.vba.gz
:so %
:q
```

Per utilizzare questo pacchetto è importante avere installato sul proprio sistema [elinks](#). Per installarlo in fedora basta digitare in un terminale:

```
$ su -c "yum install elinks"
```

L'uso di ManPageView è davvero molto semplice. Se c'è una connessione attiva a internet, all'interno della propria sessione di Vim in modalità normale (non inserimento) basta digitare:

```
:Man nome_funzione
```

(sostituire nome_funzione con il nome di una funzione Php). Se ad esempio si digita `:Man echo`, Vim aprirà una finestra contenente la documentazione della funzione echo presente nella documentazione ufficiale di Php a [questo indirizzo](#).

4 Commentare il proprio codice Php

Dato che si parla di documentazione e di buona scrittura del proprio codice Php...

E' sempre importante documentare il proprio codice. Esiste per Vim un [plugin scritto da Tobias Schlitt](#) che semplifica l'operazione: [phpDocumentator](#).

Anche qui, l'installazione di questo plugin è a dir poco semplice. Dopo aver scaricato il pacchetto basta copiare il file `php-doc.vim` nella directory `.vim/plugin` presente nella home directory dell'utente. Fatto questo bisogna aggiungere al proprio `.vimrc` le seguenti direttive:

```
inoremap <C-P> <ESC>:call PhpDocSingle()<CR>i
nnoremap <C-P> :call PhpDocSingle()<CR>
vnoremap <C-P> :call PhpDocRange()<CR>
```

Diventa così semplicissimo creare una struttura per commentare le proprie funzioni. Basta posizionarsi col cursore sulla dichiarazione della propria funzione o della classe e fare `Ctrl + p`.

5 Abilitare l'auto-completamento in Vim

E' una cosa davvero semplice abilitare l'auto-completamento in Vim per il codice Php. Dato che spesso quando si sviluppa qualche applicazione web si ha a che fare con i CSS ho aggiunto anche una direttiva per i CSS:

```
autocmd FileType css set omnifunc=csscomplete#CompleteCSS smartindent
      shiftwidth=4 "aggiungo l'indentazione per i css"
autocmd FileType php set omnifunc=phpcomplete#CompletePHP
```

Se per esempio sto scrivendo una funzione per connettermi al database MySQL ma voglio vedere quali sono le funzioni disponibili, in modalità inserimento di Vim digito direttamente `mysql` e poi pigio di seguito `Ctrl + x` e `Ctrl + o` e Vim presenterà una lista di tutte le funzioni Php che iniziano per `mysql`.

6 Ancora un po' di completamento automatico in Vim

Esiste un plugin per Vim molto interessante: [AutoComplPop](#) scritto da Takeshi NISHIDA.

Con questo plugin Vim aprirà in automatico dei menu a popup con dei suggerimenti per il completamento del testo. Il plugin non impedisce la scrittura normale del testo.

Per l'installazione basta scaricare il pacchetto, scompattarlo all'interno della propria cartella `.vim` presente nella home dell'utente. Fare attenzione a come si scompatta l'archivio, le sottodirectory devono corrispondere alle sottodirectory per la configurazione di Vim.

Una volta installato, il plugin funzionerà di suo.

Per avere la guida di aiuto per questo plugin è il caso di ricostruire prima l'indice delle guide di Vim. Per farlo basta aprire Vim e in modalità comandi digitare:

```
:helptags ~/.vim/doc
```

Aspettando un attimo sarà poi possibile richiamare la guida del plugin con:

```
:help acp
```

7 Ricerca all'interno del codice con Cscope

Spesso capita di aver la necessità, specialmente nei progetti che hanno molti file, di fare delle ricerche all'interno del codice.

Qui ci viene in aiuto cscope.

Esiste un [bellissimo tutorial in inglese](#) su come installare ed usare cscope con Vim.

I passi fondamentali da seguire per poter subito usufruire di cscope sono: installare cscope e con fedora basta:

```
$ su -c "yum install cscope"
```

poi scaricare all'interno della cartella `.vim/plugin` il plugin per Vim `cscope_maps.vim`, lo si dovrebbe fare agilmente con questo comando da bash:

```
$ wget -P ~/.vim/plugin http://cscope.sourceforge.net/cscope_maps.vim
```

Come usarlo?

Cscope viene normalmente usato per il codice scritto in C. Può però essere usato per il Php. Per farlo basta entrare nella directory del progetto in Php e creare un file che indichi a cscope quali file visionare per il suo lavoro (i file php), dopo di che lanciare cscope in modo che ricostruisca il suo database, i passaggi dunque sono:

```
find . -name '*.php' > ./cscope.files
cscope -b
rm cscope.files
```

Fatto questo possiamo passare all'uso di cscope. E' molto semplice.

Si può fare o utilizzando direttamente cscope, si entra nell'interfaccia di quest'ultimo e si fa una ricerca di una funzione, con le frecce ci si muove e con invio si può aprire in automatico Vi che ci farà vedere il codice sorgente. Con `Ctrl + d` si esce da cscope (ovviamente se si è dentro a Vi bisogna prima uscire da quest'ultimo). Per poter usare Vim bisogna impostare la variabile `EDITOR` in bash:

```
export EDITOR=vim
```

per non doverla impostare ogni volta si può inserire questa riga all'interno del proprio `bashrc` o `bash_profile`.

Il secondo sistema è utilizzare il plugin di Vim già installato. Basta lanciare Vim e usare cscope con il comando di Vim `:cscope`, si può vedere la guida con `:help cscope`.

Per la navigazione, aprire un file php con Vim e spostarsi su una funzione che viene ripetuta più volte. Digitare in modalità comandi `Ctrl + \` (`\` è il backslash) e poi subito `s`. Verrà presentata la lista delle righe in cui compare la funzione, inserire il numero che si vuole andare a vedere e premere invio. Per tornare indietro premere `Ctrl + t`.

Se invece di `Ctrl + \` si fa `Ctrl + barra_spaziatrice` e poi subito `s` e si seleziona poi la riga di interesse, Vim aprirà una nuova finestra che visualizzerà la scelta richiesta.

Questo per un uso basilare, per un uso più approfondito rimando al tutorial che ho citato prima e reperibile [qui](#).

ATTENZIONE – Risoluzione di problemi

Quando lancio Vim mi viene fuori un messaggio di errore del tipo `e568`, ho risolto velocemente il bug modificando la riga 42 del plugin sostituendola con quanto segue:

```
42 if has("cscope")
43 if exists("./cscope.out")
44     cs add ./cscope.out
45 endif
```

8 Navigazione all'interno del codice con ctags

Cscope è davvero utile per ritrovare facilmente pezzi di codice all'interno del proprio progetto.

Ctags (anche noto come **Exuberant Ctags**) è un altro strumento molto utile per poter saltare e navigare all'interno del proprio codice.

Per prima cosa è necessario avere installato il programma ctags, se non è installato basta:

```
$ su -c "yum install ctags"
```

Ora bisogna creare la lista dei tag del proprio progetto Php con ctags, da bash dopo essere entrati nella directory di lavoro:

```
$ ctags -R
```

(Questo crea un file tags che contiene tutti i tag necessari alla navigazione).

Ora con Vim basta aprire un file php, posizionarsi sopra una funzione del proprio progetto e con la combinazione dei tasti **Ctrl +]** si verrà catapultati alla definizione della funzione. Per tornare indietro basta la combinazione dei tasti **Ctrl + t**.

Invece di usare la combinazione dei tasti si può, in modalità comandi di Vim, digitare **:ta nome_funzione** e si verrà rimandati alla dichiarazione della funzione ricercata.

Esistono anche altre funzioni di Vim per la navigazione nei tag, con **:ts** si può vedere la lista delle funzioni ricercate. Per altre informazioni rimando alla documentazione di Vim reperibile dall'editor con **:help tag**.

9 Source Code Browser con Taglist Plugin

Vim può diventare un ottimo browser del codice sorgente con il [plugin Taglist](#) ([pagina nel repository di vim](#)).

Ricordo che per usare questo plugin è necessario avere installato ctags, come ho spiegato precedentemente. Per installare questo plugin basta scaricarlo e scompattarlo all'interno della solita directory per vim (**.vim**) dell'utente. Dopo aver fatto questa operazione bisogna ricostruire l'indice della documentazione aprendo Vim e dando **:helptags ~/.vim/doc**, così si può vedere la documentazione completa di questo plugin con **:help taglist.txt**.

Non spiego approfonditamente l'uso di questo plugin, per un uso più approfondito rimando alla [documentazione ufficiale online](#) o da vim con **:help taglist-using**.

Per aprire la finestra con la lista dei tag basta digitare in modalità comandi di Vim **:TlistOpen**. Per chiudere la finestra basta usare il solito comando di Vim **:q**.

Per saltare da una voce della lista al codice si può usare o il mouse (se si è impostato Vim per usare il mouse come ho specificato all'inizio di questo articolo) o ci si posiziona col cursore sulla voce della lista e si da **Invio**. Se invece di **Invio** si preme la barra spaziatrice si può vedere in basso (dove si entra nella modalità di inserimento comandi di Vim) la dichiarazione della riga in cui si trova il tag.

10 Debug del codice Php con Xdebug

Nello sviluppo di un programma non può mancare la fase di debug. Xdebug è un'estensione per il Php molto utile per fare il debug di quest'ultimo.

Per installarlo in Fedora è molto semplice:

```
$ su -c "yum install php-pecl-xdebug"
```

Per fare il debug di una applicazione Php ci sono vari sistemi, quello che propongo e che trovo più semplice e utile è il debug remoto. La documentazione ufficiale e molto ben fatta, si trova a [questo indirizzo](#).

In questo articolo mi concentro su Vim, non sulle possibili differenti configurazioni di Xdebug e su come farlo funzionare correttamente, dunque rimando tutto alla documentazione ufficiale che è più che esauriente.

Se l'installazione è andata a termine correttamente si può passare al controllo della configurazione di xdebug. Il file di configurazione di xdebug in fedora si trova sotto `/etc/php.d/xdebug.ini`. Il mio file di configurazione è fatto così:

```
$ cat /etc/php.d/xdebug.ini
; Enable xdebug extension module
zend_extension=/usr/lib64/php/modules/xdebug.so
xdebug.remote_enable=1
xdebug.remote_log=/var/log/httpd/xdebug.log
xdebug.trace_output_dir=/tmp/xdebug
;
;xdebug.auto_trace = 0
;;
;xdebug.profiler_enable = 0
```

Dopo un riavvio di apache, nella pagina di informazioni del Php si potrà vedere che l'estensione xdebug è attiva sul proprio server.

Ora che il pacchetto Xdebug è installato e correttamente caricato (la pagina di informazioni di Php ce lo conferma) bisogna trasformare Vim in un client per il protocollo GDB. Per farlo si può usare il [plugin DBGp client](#) scritto da Sam Ghods il quale sul suo sito web ha fatto un'ottima guida su come installare e configurare xdebug ed il suo plugin. Questo plugin però è datato, Hadi Zefin ne ha scritto uno più aggiornato che risolve alcuni problemi ed è reperibile a [questo indirizzo](#).

Dunque prima di tutto bisogna scaricare il pacchetto del plugin da [qui](#).

Dopo averlo scaricato bisogna scompattare il suo contenuto (i due file `debugger.vim` e `debugger.py`) all'interno della directory `.vim/plugin` presente nella home directory del proprio utente. Se è andato tutto liscio si può passare al primo debug della propria applicazione.

Per il debug basta aprire un file qualsiasi del proprio progetto e pigiare F5. Si hanno da ora 10 secondi di tempo (il plugin precedente impostava 5 secondi massimi di attesa) per avviare una sessione di debug sul proprio server (o fare il refresh della pagina interessata sul browser nel caso si sia già avviata una sessione). Per farlo basta andare con il proprio browser web preferito alla pagina `index.php` (o alla pagina `php` di interesse) del proprio progetto avendo l'accortezza di richiamare la pagina aggiungendo in fondo all'URL `?XDEBUG_SESSION_START=1`. Per cui se ho il mio sito di nome `example.com` avvio la sessione di debug con l'URL: http://example.com/index.php?XDEBUG_SESSION_START=1.

Tornando su Vim si potrà notare che è stata catturata la sessione di debug, viene aperto un nuovo tab.

Se si usa firefox si può utilizzare un'estensione interessante per avviare la sessione di debug: [easy Xdebug](#).

La sessione dopo un tot di tempo scade, per riavviarla basta richiamare di nuovo la pagina `php` come detto pocanzi.

Si può anche fermare la sessione di debug richiamando la pagina stessa con:

```
?XDEBUG_SESSION_STOP
```

Prima di passare alla spiegazione su come usare il plugin in Vim per il debug faccio notare che la sessione di debug basta lanciarla solo una prima volta. Mi spiego: una volta che ho finito il debug della mia applicazione con Vim e voglio rifare il debug dei miei script non occorre che richiamo la sessione come con l'URL che ho citato prima (http://example.com/index.php?XDEBUG_SESSION_START=1), basta semplicemente che richiami la mia pagina normalmente (<http://example.com/index.php>) o faccia un refresh della pagina dopo aver pigiato F5 in Vim.

Sono costretto a richiamare l'url per l'avvio della sessione solo se il tempo di sessione del debug è scaduto o se ho chiuso la sessione di debug con XDEBUG_SESSION_STOP.

L'uso del plugin client per il debug in Vim è davvero molto molto semplice. Una volta che è stata catturata la sessione di debug per fare il debug "step by step" della pagina richiamata dal browser basta pigiare F2. Si scorrerà così il codice Php riga per riga.

Per saltare un ciclo o la chiamata di una funzione, senza dover entrare dentro ad ogni singolo annidamento si può usare al posto di F2 F3.

Importante è poi l'uso dei breakpoint. Per aggiungere un breakpoint basta aprire il file che ci interessa e col cursore sulla riga di interesse in modalità comandi di Vim dare :Bp, la riga si evidenzia in verde e indica che è stato inserito il breakpoint, per toglierlo basta usare nuovamente :Bp. Per saltare direttamente alla riga di breakpoint, dopo aver catturato la sessione di debug, basta pigiare F4.

Oltre a vedere la sequenza su come viene eseguito il codice Php è importante anche vedere che valori assumono le variabili in gioco. Per farlo basta usare o F11 o F12.

Il primo fa vedere, nel riquadro in alto di Vim, tutte le variabili presenti fino al punto in cui si è eseguito il codice.

Il secondo permette invece, dopo essersi posizionati col cursore sopra la variabile di interesse, di vedere il valore di questa sempre nel riquadro in alto.

Se si è in debug e in modalità normale (o comandi) di Vim si può anche usare la combinazione ,e per poter avere la possibilità di scrivere la variabile di interesse.

Quando si sta eseguendo il debug capita che lo stack del Php si riempia di file che vengono richiamati. Con questo plugin è facile muoversi nello stack per recuperare la posizione da dove un file è stato richiamato per esempio. Per farlo basta semplicemente usare uno dei due comandi :Up :Dn per salire o scendere nello stack. Lo stack è visualizzato nel riquadro in basso a destra di Vim.

Nel riquadro centrale a destra c'è una piccola finestra che ricorda le funzioni che ho descritto in questa sezione del mio articolo.

11 Estendere il % di Vim

Trovo molto utile la funzione % di Vim per muoversi in un file di testo. Lo trovo ancor più utile utilizzando un plugin che ne estende le funzioni: [Matchit](#) creato da Benji Fisher.

È stato costruito principalmente per essere usato in codice HTML e altri linguaggi. Quando si lavora in un progetto Php si ha spesso a che fare con codice HTML.

Anche qui l'installazione è molto semplice, si scarica il pacchetto dalla pagina del plugin e lo si scompatta all'interno della solita cartella di Vim: .vim presente nella home dell'utente. Poi basta ricostruire l'indice dell'"help" di Vim con:

```
:helptags ~/.vim/doc
```

In fedora questo plugin in realtà è già presente all'interno del pacchetto vim-common. Per usarlo senza dover scaricare il file basta:

```
$ cp /usr/share/vim/vim72/macros/matchit.txt ~/.vim/doc/  
$ cp /usr/share/vim/vim72/macros/matchit.vim ~/.vim/plugin/
```

E poi procedere con la ricostruzione dei tag della guida di Vim:

```
:helptags ~/.vim/doc
```

Se la direttiva `:filetype plugin on` è già stata data all'interno del file di configurazione di Vim si potrà usare `%` esteso.

12 Un piccolo aiuto con Closetag

Come già detto, quando si sviluppa un applicativo in Php è facile aver a che fare con codice HTML, questo perchè il Php è stato appositamente pensato e creato per la costruzione di applicativi web.

Esiste un plugin utile per chiudere in modo facile e veloce i tag HTML aperti senza doverli scrivere a mano: [Closetag](#) creato da Steven Mueller.

L'installazione si esegue come al solito: si scarica il pacchetto e si posiziona direttamente all'interno di `.vim/plugin` presente nella home dell'utente.

Il plugin dovrebbe già essere installato e funzionante. Lo si può testare provando ad aprire un file html (o anche xml), scrivere un tag html qualsiasi e poi provare a chiuderlo con `Ctrl + _`.

13 Conclusioni

Con questo articolo ho spiegato velocemente come configurare ed estendere [Vim](#) per poterlo tramutare in un [IDE](#) per la scrittura di codice Php.

Esistono molti altri plugin per Vim, reperibili dal sito web di Vim, che non ho avuto modo di studiare ed adoperare.

Se esiste un plugin utile che non ho menzionato prego di farmi sapere così vedrò di estendere ulteriormente questo articolo.

Acronimi

IDE	Integrated Development Enviroment
Vim	Vi Improved
S.O.	Sistema Operativo
LAMP	Linux Apache MySql Php
GDB	The GNU Project Debugger