Introduzione a Vim

Fabio Pozzi

Un po' di storia

Vim è un text editor (ma non un sistema operativo :) scritto da Bram Moolenaar e rilasciato nel 1991.
Scritto originariamente per Amiga, vim ora funziona su:

- Windows,
- Linux,
- unix,
- MacOSX,
- OS/2,
- QNX,
- BSD,

quello-che-vuoi in pratica.

Un po' di storia

E' free software, è open source, è anche charityware,

ovvero se ti piace vim e lo usi spesso (magari per lavoro),
sei incoraggiato a donare soldi ad una fondazione
che si occupa di bambini in Uganda (fatelo)

un altro modo è quello di acquistare un libro tramite i link che trovate su vim.org, una piccola percentuale verrà girata alla fondazione.

Meglio di così è difficile :)

Piano della lezione

Ci occuperemo di: una overview dell'organizzazione di vim:

- comandi di base

comandi di movimento, editing di base

- gestione files

buffer, tab, finestre

- comandi più avanzati

ricerca, macro, find-replace

- espansioni

Overview

Lanciando vim la prima cosa che noterete è che se provate ad iniziare a scrivere non otterrete quello che desiderate.
Questo perchè Vim è un editor - *modale* - ovvero ha diverse modalità d'uso:

- la modalità di *inserimento*
- la modalità *normale* per l'esecuzione di comandi
- la modalità *paste* per incollare testo
- la modalità *visuale*

Overview

quando lanciamo vim ci troviamo inizialmente nella modalità *normale*

In questa modalità possiamo eseguire comandi, spostarci nel testo, alterarlo,
tagliare o incollare ma senza inserirne di nuovo

Comandi di base

Lo schema dei tasti di navigazione di vim deriva da vi, l'editor di cui vim è il successore.

Questi a sua volta li deriva dal fatto che il suo autore, Bill Joy, sviluppò l'editor da un terminale Lear Siegler ADM3A in cui non erano presenti i tasti freccia ma tale funzione veniva svolta dai tasti h,j,k,l

Comandi di base

In ogni caso se apriamo un file di testo con vim possiamo spostarci nelle quattro direzioni usando i tasti *hjkl*, che sono proprio uno accanto all'altro e sulla fila di tasti dove si dovrebbero trovare le nostre dita nella posizione a riposo.

Quindi partendo da sinistra *h* ci permette di spostarci di una lettera a *sinistra*, *j* e *k* rispettivamente *giù* e *su*, mentre *l* a *destra*.
Ci vuole un po' per abituarcisi ma proprio in virtù della loro posizione rendono i brevi spostamenti del cursore più veloci, soprattutto se abbinate alle altre combinazioni di navigazione nel testo.

Comandi di base

I tasti di navigazione alrettanto importanti sono *b,w,e*.
b e B ci permettono di spostarci indietro di una "parola".

Vi chiederete quale sia la differenza tra *b* e *B*.
La differenza è nella definizione di "parola".
Come scopriremo diversi comandi in vim hanno una sottile differenza tra minuscola e maiuscola, spesso sono delle variazioni di un unico comando.

b ad esempio ci porta alla parola precedente intesa come sequenza di caratteri alfabetici o simboli.

Comandi di base

B invece ci porta alla precedente sequenza di caratteri separata da uno spazio, saltando l'eventuale punteggiatura nel caso ci sia.

Allo stesso modo *w* e *W* ci porteranno alla prossima sequenza, mentre *e* ed *E* ci porteranno al termine della sequenza corrente.

Comandi di base

Gli ultimi tasti utili per lo spostamento:

- *0* (zero) per spostarsi al primo carattere della riga
- ^ per spostarsi al primo carattere *non-whitespace* della riga
- $ per spostarsi all'ultimo carattere della riga

Inserimento

Ora finalmente passiamo a scrivere premendo *i*.
A questo punto vedremo che siamo passati nella modalità di inserimento da cui possiamo uscire premendo *Esc*, tornando alla modalità *normale*.
Alla modalità di inserimento possiamo arrivarci anche premendo:

- *A* per appendere del testo a fine riga
- *o* che ci permette di inserire alla riga successiva
- *O* che ci permette di inserire alla riga precedente

Cancellazione

d usato in combinazione con gli altri modificatori cancella del testo (es. *de*, *dw*)

D cancella a partire dalla posizione del cursore corrente fino a fine riga

dd agisce sull'intera riga, cancellandola

Taglia, copia, incolla

d in realtà non cancella definitivamente, ma pone il testo nella clipboard *principale* di vim (perchè ne esistono altre, sono chiamate *registri*) così che quindi non esista il comando di taglia, si cancella e poi si incolla.

y (yank) copia, *yy* copia tutta la riga corrente

p incolla *dopo* il cursore, *P* incolla *prima* del cursore

Undo e redo

u per fare undo, *Ctrl-r* per fare redo.
Ci sarebbe un mega discorso da fare sugli undo, sui livelli di undo perchè vim mantiene l'intero *albero*[0] delle modifiche e c'è un plugin[1] per gestirlo e navigarlo.

Lo butto lì tanto per dare un'idea delle potenzialità dei plugin, che vedremo dopo

[0]http://vim.wikia.com/wiki/Using_undo_branches
[1]http://sjl.bitbucket.org/gundo.vim/

Modalità incolla

Questa modalità, che possiamo abilitare con il comando *:set paste*
(che possiamo assegnare ad un keybinding, nel mio caso l'ho assegnato al tasto F10)
ci permette di incollare del testo all'interno di vim senza che intervengano i plugin che gestiscono l'indentazione o l'autocompletamento rovinando tutto.
Lo possiamo poi disabilitare usando il comando
:set nopaste

Modalità visuale

La modalità visuale ci permette di selezionare testo e di vedere l'area selezionata o di usare il mouse per selezionare il testo.
Ad esempio usando *V* abbiamo la possibilità di attivare la selezione delle righe, se ora ci muoviamo all'interno del documento selezioneremo
tutte le righe in cui passiamo, che potremo poi usare per copiarle, tagliarle, cancellarle e così via.
v invece ci permette di selezionare lettera per lettera, quindi è da combinare con i tasti di movimento che abbiamo già visto

Buffer

I buffer sono un modo di caricare il file in memoria e poterci lavorare sopra, possiamo aprire un file in un buffer senza nemmeno vederlo sullo schermo, per poi modificarlo successivamente
:badd file.txt

Possiamo scorrere tra i buffer usando i comandi

- *:bnext*
- *:bprevious*
- *:bfirst*
- *:blast*

e così via

Tab

Tra l'altro la logica dei comandi sui buffer è la stessa anche per i *tab*, che descriveremo ora.

Un *tab* è un concetto familiare, è come un tab di firefox, ne possiamo aprire di nuovi, switchare tra gli esistenti, etc.
Una volta creato un nuovo tab lo vedremo nella parte alta della schermata.
I comandi per gestirli saranno quindi
:tabnext :tabprevious :tabfirst e così via

Splits

Gli *split* (che ho ribattezzato finestre) invece ci permettono di dividere una schermata in più parti, in orizzontale o in verticale.

Ctrl-w v (ovvero Ctrl-w e poi v) divide la schermata attuale in due, verticalmente.

Il comando equivalente è *:vsplit*

Per dividere orizzontalmente potete usare *:split* o
Ctrl-w s

Splits

E' possibile navigare tra gli split usando la combinazione
Ctrl-w w

Per chiudere uno split possiamo usare
Ctrl+w q

Plugins

I plugin sono, secondo me, la feature più importante di vim.

Permettono infatti di espandere ulteriormente le possibilità di vim e di rendere alcune funzionalità più immediate o complete.

Plugins

I plugin possono essere scritti usando il linguaggio di scripting nativo di vim, *vimscript*.

Dato che *vimscript* non è proprio un linguaggio che tutti conoscono sono stati aggiunti dei *bindings* verso altri linguaggi.

In questo modo non è necessario implementare un plugin tutto in vimscript, ma è possibile implementare le funzioni in *ruby*, *python* o altro e poi richiamare tali funzioni dall'interno del plugin.

Plugins

C'è una buona comunità di utenti e anche di scrittori di plugin che è ancora attiva, anzi si potrebbe dire che la community vim è in un momento di buona vitalità

(ad esempio il recente sviluppo dei famosi plugin rails.vim, command-t, fugitive, la nascita di vimcasts)

Plugin che uso

nerdcommenter permette di commentare blocchi di codice automagicamente usando il giusto simbolo a seconda del linguaggio di programmazione

syntastic controlla la sintassi del codice che scrivi e ti segnala se c'è qualcosa che non va

tagbar permette di visualizzare in uno split un elenco navigabile di variabili globali, costanti e funzioni dichiarate nel file aperto

Plugin che uso

supertab permette di avere completamento automatico usando tab sia nell'inserimento che nella modalità *normale*

solarized è un tema colori per vim che uso anche per konsole e sembra ben pensato

command-t è un plugin per la ricerca intelligente di file per nome nelle sottocartelle figlie di quella corrente

fugitive permette di lanciare comandi git e vederne l'output direttamente da dentro vim