UNIX and Linux

Vim Tutorial

A quick Guide to VIM

Contents

Modal Editing

vim is a *modal* editor. In other words, it uses different modes to execute editing commands, insert text, and select text. There are three modes:

- *Insert mode*: this mode is used to enter text into the file. It is also possible to move the cursor around in insert mode, however, command mode is usually better for this.
- *Command mode*: this mode is used to move the cursor, and perform commands on the text (like "delete this line"
- *Replace mode*: this mode is used to type over existing text.
- *Visual mode*: This is used to select text for operations like cut, copy and paste. Note that the traditional version of vi does not have visual mode.

The advantages of modal editing

While the concept of modal editing seems awkward at first (indeed, modal editors are hard to use iniotially), it is also very powerful. Most non modal editors can not conveniently carry out commands such as "delete lines 3-5, and insert a date stamp in the second line of the file" in the space of a few keystrokes. Or delete all html tags in the space of a few keystrokes. Command mode makes this very simple. One

might be tempted to cite emacs as a non modal editor. However, emacs does have a command mode of its own , which one can enter by pressing <ESC>X. In fact, my contention is that some kind of modal editing is a must have for an editor to be powerful.

The disadvantages of modal editing

Modal editing comes with the minor drawback that you have to change modes to go from one task to another. This is inconvenient in some circumstances, for example: suppose you are editing a buffer containing one lione. Then the power of the modal editor is underutilised, and the inconvenience of mode switching outwieghs the power of modality. This example is not as silly as it might sound: you edit a one line buffer every time you type a unix command (unless you are a way-out power user and you use zsh all the time ...). It is possible to use vi like keybindings for ocommand line editing, however, most people choose the non-modal emacs style keybindings.

Modal editing commands

Here are the commands used to switch modes.

`i`	Enter insert mode and proceed to insert text *before* the current cursor position	Command mode
`a`	Enter insert mode and proceed to insert text *after* the current cursor position (a is short for "append")	Command mode
`I`	Position the cursor at the start of the line and then enter insert mode.	Command mode
`A`	Append text to current line (ie go to end of line and then enter "insert mode".)	Command mode
`v`	Enter visual mode	visual or insert mode
<ESC>	Enter command mode	visual or insert mode
`R`	Enter replace mode	command mode

Moving Around

start of line	`0`
end of line	`$`
left	`h`
right	`l`
down	`j`
up	`k`
back one word	`b`
forward one word	`w`
back one paragraph	`{`
forward one paragraph	`}`
go to line 33	33G

Cut/Copy/Paste

Basically all that's involved in cutting and copying is the following :

1. highlight a region by pressing "v" and moving the cursor.
2. use either `x` to cut or `y` to copy

Use *P* to *insert* the text "before" the cursor, and `p` to put the text at the point after the cursor

Searching for strings

To search for a string, while in command mode , hit

```
/
```

Followed by the search expression. Note that the search string is actually a pattern with support for wildcards, etc. Try `:help pattern` for details on how patterns work.

Examples

/foo	Search for the next instance of foo
/foo.*bar	search for any line containing "foo" followed by "bar" (possibly with some stuff in between)
`/[0-9]\{3}[ -][0-9]\{4}`	Search for a phone number (any 7 digit phone number)

Replacing Text

To search and replace, use the `s` command. The format is this:

```
[address]s/search_pattern/replacement string/[falgs]
```

where the address can be line numbers, or a comma-seperated range of line numbers.

Patterns are involved little beasts, and a discussion of them here would take us too far afield. Use `:help pattern` for more information on patterns. If this info looks ike Greek to you , then assuming that you aren't Greek, read my Grep tutorial which contains a step by step tutorial on how patterns work. Grep patterns are slightly different from vim patterns in the syntax (ie the symbols used are slightly different) , but conceptually, it's exactly the same. To put a delimiter (ie '/') in the search or replace string, you 'escape' it with a backslash '\'. The backslash is used to escape all special characters.

Some types of strings (like directory lists) use several backslashes, so in these cases, it makes sense to use a different delimiter. The delimiters usually used are :;/ but several other delimiters are allowed including these *%@ Some examples are given below. There are also special ways of denoting line numbers , for example:

Conventions for denoting line numbers

.	current line
.+3	3 lines below current line
.-3	3 lines before the current line
$	The last line
%	All lines from beginning to end

Some flags worth knowing

g Substitute all matches on specified line (the default only substitutes for the first match)

i use case insensitive matching

c prompt for confirmation prior to each substitution

Examples of substitution commands

`.s/foo/bar/g`	Substitute all occurences of `foo` on the current line with `bar`
`%s/foo/bar/gi`	Subsitute all matches of the word foo in the file with bar. Ignore case in searches.
`1,8s/<[^>]*>//g`	Remove all html tags (ie strings enclosed in <>) in lines 1 to 8.
`%s/foo/bar/gi`	Subsitute all matches of the word foo in the file with bar. Ignore case in searches.
`.+3,$-1s:\(\(--\=\\|+\)[a-zA-Z-]\):<EM>\1<EM>:g`	Put an emphasis html tag around all unix command line options between three lines from the current line, and the second last line of the file
`%s;http://www.microsoft.com/;http://www.linux.org;g`	Subsitute all matches of the URL http://www.microsoft.com with the URL http://www.linux.org.

Including Files

To embed a file in your document, type

:r filename

Powerful Stuff: Unix at Your fingertips

There are two cool ways you can make use of the UNIX commands while you use

vim. Note that you need some form of UNIX to do this. (if you have a PC, get a $2- copy of linux, the freeware UNIX) ...

Inserting the output of a command into the file.

This is done as follows :

```
:r! command
```

Example

```
:r! date
```

inserts a date stamp in the file

Fitering a region through a command

Filtering: what is it ?

Filtering a block of text through a command means executing the command, using the selected text as input. This can be quite useful. We give some examples to demonstrate.

To filter a region through a command, use 'v' to highlight that region, then do the following : type

```
!command
```

and this will filter through command.

Example

```
!sort
```

will sort the selected text alphabetically.