Handy .vimrc Things

2020-03-20 BY GENE

Here are some of the more handy things I have in my .vimrc file:

```
" netrw settings
let g:netrw_liststyle = 2
let g:netrw_banner = 0

" split window and open file explorer
map <leader>e :Sex<cr>
```

The above opens a file explorer as a split buffer in my ~ home directory, and changes the display from the first to the second screenshot:

The following super simple line exchanges the semi-colon to a colon in “normal mode”, because I tire of typing shift-semicolon **so** much:

```
" Alleviate shift: madnesss by remapping ; to :
nnoremap ; :
```

Here I add a couple essential commands to the excellent vcscommand plugin. One to git push my commits, and one to view the actual changes of each commit:

```
nmap <leader>cp :!git push<cr>
" Replace git log with -p:
nmap <leader>cl :!git log -p %<cr>
```

I use the simple but fabulous vim-gitgutter plugin, that displays colored git changes for a line, in the left margin. Here I use CTRL-G + CTRL-G to toggle the changes display, and CTRL-G + CTRL-N with CTRL-G + CTRL-P to move to the next and previous git changes respectively:

```
map <C-G><C-G> :GitGutterToggle<cr>
map <C-G><C-N> :GitGutterNextHunk<cr>
map <C-G><C-P> :GitGutterPrevHunk<cr>
```

The following will display either relative or absolute line numbers, along the left margin, when I double-tap CTRL + either the letter M or N:

```
map <C-M><C-M> :set relativenumber!<CR>
map <C-N><C-N> :set number!<CR>
```

This toggles word-wrapping on and off:

```
map <C-W><C-W> :set wrap!<cr>
```

And this is just extra-handy:

```
" Use the mouse!
set mouse=nv
```

I loves me a status line along the bottom of the screen:

```
" Have a status line
set laststatus=2
if has("statusline")
```

```
  set statusline =                      " clear
  set statusline+=%02n                  " leading zero 2 digit buffer numbe
  set statusline+=\ %<%f                " relative filename path
  set statusline+=[%{&ff}]              " [fileformat]
  set statusline+=%r                    " read only flag '[RO]'
  set statusline+=%m                    " modified flag '[+]' if modifiable
  set statusline+=%h                    " help flag '[Help]'
  set statusline+=%=                    " left/right separation point
  set statusline+=[%b                   " decimal byte
  set statusline+=\ x%02B]              " hex byte ' \x62'
  set statusline+=\ %{(line('.')-1)%16} " line
  set statusline+=:%{(line('.')-1)/16}  " block
  set statusline+=\/%{line('$')/16}     " max block
  set statusline+=\ %c                  " column number
  set statusline+=%V                    " virtual column '-{n}'
  set statusline+=:%l/%L                " line/lines
  set statusline+=\ %p%%                " percent of file
  set statusline+=%{&hlsearch?'+':'-'}
  set statusline+=%{&paste?'=':'\ '}
  set statusline+=%{&wrap?'<':'>'}
endif
```

My tab is set to insert 4 spaces, so I use this when I need to indent a line by 2:

```
" Indent 2 spaces
map <leader>. I  <esc>0<esc>
```

I use these constantly to comment/uncomment lines:

```
" Line head removal:
map <leader>1 0x
" Perl, shell, etc comment:
map <leader>3 0i#<esc>
```

Here are a number of “C.A.S.E.” type statements that I use regularly:

```
map <leader>;ul ouse lib '/Users/gene/sandbox//lib';<esc>bbba
map <leader>;db o$DB::single = 1;<esc>
map <leader>;dd ouse Data::Dumper;warn(__PACKAGE__,' ',__LINE__," MARK: ",D
map <leader>;ww owarn(__PACKAGE__,' ',__LINE__," MARK: ",,"\n");<esc>6hi
map <leader>;dt odone_testing();exit;<esc>
map <leader>;e o__END__<esc>
```

```
map <leader>;pp ggO#!/usr/bin/env perl<cr>use strict;<cr>use warnings;<cr>
map <leader>;iv Ashift \|\| die "Usage: perl $0 \n";<esc>F\i
```

I use Perl every day. So here are lines that I use frequently. Some execute perldoc in various ways; some check the syntax of the file; some run the file as perl:

```
map <leader>pc :!podchecker %<cr>              " Check the POD
map <leader>pd :!perldoc %<cr>                 " perldoc the file
map <leader>po :!perldoc <cword><cr>           " perldoc a module name unde
map <leader>pm :!vim $(perldoc -l <cword>)<cr> " Execute vim on the cursor
map <leader>pf :!perldoc -f <cword><cr>        " View the description of th
map <leader>pq :!perldoc -q <cword><cr>        " Search the FAQ for the wor
map <leader>pr :!perl -I. -Ilib %<cr>          " Run the file with perl
map <leader>ps :w !perl -I. -Ilib -c<cr>       " Check the perl syntax of t
map <leader>px :!carton exec -- perl -I. -Ilib -c %<cr>
map <leader>pu :!carton exec -- perl -I. -Ilib %<cr>
```

If you have a ~/bin/perldocv file with contents:

```
perldoc $1 | vim -R +"set ft=perl" -
```

Then you can replace the “<leader>po” line above with:

```
map <leader>po :!perldocv <cword><cr>
```

I like to see which line I am currently on with this:

```
" Highlight current line
set cursorline
highlight CursorLine cterm=NONE ctermbg=235 ctermfg=NONE
```

These lines allow me to open up perl and html files with syntax highlighting turned on appropriately:

```
autocmd BufNewFile,BufRead *.pl,*.pm,*.t set filetype=perl
autocmd BufNewFile,BufRead *.htm,*.html,*.tt set filetype=html
```

With a set of basic skeleton files saved, I can vim a new, unknown file with one of

the following extensions, and have it open the appropriate skeleton:

```
autocmd BufNewFile *.pm 0r ~/.vim/skeleton.pm
autocmd BufNewFile *.pl 0r ~/.vim/skeleton.pl
autocmd BufNewFile *.t 0r ~/.vim/skeleton.t
autocmd BufNewFile *.html 0r ~/.vim/skeleton.html
```

The shift-K lookup gets replaced with perldoc -f, with this line:

```
" Lookup with perldoc -f
noremap K :!perldoc <cword> <bar><bar> perldoc -f <cword><cr>
```

I override a couple colorizations (line numbers and comments) with ones that I prefer:

```
highlight LineNr ctermfg=darkgrey
highlight clear SignColumn

highlight Comment ctermfg=darkgrey
```

With the useful Perl::Tidy module installed, this allows me to visually select a range and apply perl tidy to it with the F2 key:

```
" Visual mode perl tidy
command -range=% -nargs=* Tidy <line1>,<line2>!perltidy
autocmd Filetype perl vmap <F2> :Tidy<CR>
```

And those are the highlights of my .vimrc! I *would* publish the entire file itself, but it is large, full of mundane/abstruse settings, and it keeps changing by me adding or tweaking this or that.

FILED UNDER: SYSTEMS
TAGGED WITH: PERL, VIM

^ Top