从零搭建和配置 OSX 开发环境

Apr 3rd, 2015 10:52 pm

对于每一名开发者来说，更换系统或者更换电脑的时候，都免不了花上不短的时间去折腾开 发环境的问题。我本人也是三番两次，深知这个过程的繁琐。所有，根据我自己以往的经验， 以及参考一下他人的意见，整理一下关于在 Mac 下做各种开发的配置，包含 Java, Ruby, Vim, git, 数据库等等。欢迎补充与修正。

Terminal 篇

这篇文章包含配置控制台环境，包括包管理工具, zsh, Vim, git 的安装配置。

Homebrew, 你不能错过的包管理工具

包管理工具已经成为现在操作系统中不可缺少的一个重要工具了，它能大大减轻软件安装的 负担，节约我们的时间。Linux 中常用的有 `yum` 和 `apt-get` 工具，甚至 Windows 平台也 有 `Chocolatey` 这样优秀的工具，OSX 平台自然有它独有的工具。

在 OSX 中，有两款大家常用的管理工具：`Homebrew` 或者 `MacPorts`。这两款工具都是为了解决同样的问题——为 OSX 安装常用库和工具。Homebrew 与 MacPorts 的主要区别是 Homebrew 不会破坏 OSX 原生的环境，这也是我推荐 Homebrew 的主要原因。同时它安装的所有文件都是在用户独立空间内 的，这让你安装的所有软件对于该用户来说都是可以访问的，不需要使用 `sudo` 命令。

> 在安装 Homebrew 前，你需要前往 AppStore 下载并安装 Xcode.

安装方式：

```
# OSX 系统基本上都自带 Ruby1.9
# 所以无需先安装 Ruby，但是之后我们需要管理 Ruby
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)
```

`Homebrew` 常用命令：

```
brew list                 # 查看已经安装的包
brew update               # 更新 Homebrew 自身
brew doctor               # 诊断关于 Homebrew 的问题(Homebrew 有问题时请用它)
brew cleanup              # 清理老版本软件包或者无用的文件
brew show ${formula}      # 查看包信息
brew search ${formula}    # 按名称搜索
brew upgrade ${formula}   # 升级软件包
brew install ${formula}   # 按名称安装
brew uninstall ${formula} # 按名称卸载
brew pin/unpin ${formula} # 锁定或者解锁软件包版本，防止误升级
```

zsh，好用的 shell

Shell 程序就是 Linux/UNIX 系统中的一层外壳，几乎所有的应用程序都可以运行在 Shell 环境 下，常用的有 bash, csh, zcsh 等。在`/etc/shells` 文件中可以查看系统中的各种 shell。

```
cat /etc/shells

# List of acceptable shells for chpass(1).
# Ftpd will not allow users to connect who are not using
# one of these shells.

/bin/bash
/bin/csh
/bin/ksh
/bin/sh
/bin/tcsh
/bin/zsh
```

而 zsh 是 OSX 系统原生的 shell 之一，其功能强大，语法相对于 bash 更加友好和强大，所以推荐 使用 zsh 作为默认的 shell。

```
# 切换 zsh 为默认 shell
chsh -s $(which zsh)
```

如果你想使用最新的 zsh，你可以使用 Homebrew，此方法也会保留原生的 zsh，防止你在某个 时刻需要它。

```
# 查看最新 zsh 信息
brew info zsh

# 安装 zsh
brew install --disable-etcdir zsh

# 添加 shell 路径至/etc/shells 文件中
# 将 /usr/local/bin/zsh 添加到下面文件中
sudo vim /etc/shells

# 更换默认 shell
chsh -s /usr/local/bin/zsh
```

下面贴上我的 zsh 配置以供参考

我的 zsh 配置 (zshrc) download

```
# modify the prompt to contain git branch name if applicable
git_prompt_info() {
  ref=$(git symbolic-ref HEAD 2> /dev/null)
  if [[ -n $ref ]]; then
    echo " %{$fg_bold[green]%}${ref#refs/heads/}%{$reset_color%}"
  fi
}
setopt promptsubst
export PS1='${SSH_CONNECTION+"%{$fg_bold[green]%}%n@%m:"}%{$fg_bold[blue]%}%c%{$reset_c

# load our own completion functions
fpath=(~/.zsh/completion $fpath)

# completion

```

```
16 autoload -U compinit
17 compinit
18
19 # load custom executable functions
20 for function in ~/.zsh/functions/*; do
21   source $function
22 done
23
24 # makes color constants available
25 autoload -U colors
26 colors
27
28 # enable colored output from ls, etc
29 export CLICOLOR=1
30
31 # history settings
32 setopt hist_ignore_all_dups inc_append_history
33 HISTFILE=~/.zhistory
34 HISTSIZE=4096
35 SAVEHIST=4096
36
37 # awesome cd movements from zshkit
38 setopt autocd autopushd pushdminus pushdsilent pushdtohome cdablevars
39 DIRSTACKSIZE=5
40
41 # Enable extended globbing
42 setopt extendedglob
43
44 # Allow [ or ] whereever you want
45 unsetopt nomatch
46
47 # vi mode
48 bindkey -v
49 bindkey "^F" vi-cmd-mode
50 bindkey jj vi-cmd-mode
51
52 # handy keybindings
53 bindkey "^A" beginning-of-line
54 bindkey "^E" end-of-line
55 bindkey "^R" history-incremental-search-backward
56 bindkey "^P" history-search-backward
57 bindkey "^Y" accept-and-hold
58 bindkey "^N" insert-last-word
59 bindkey -s "^T" "^[Isudo ^[A" # "t" for "toughguy"
60
61 # use vim as the visual editor
62 export VISUAL=vim
63 export EDITOR=$VISUAL
64
65 # load rbenv if available
66 if which rbenv &>/dev/null ; then
67   eval "$(rbenv init - --no-rehash)"
68 fi
69
70 # load thoughtbot/dotfiles scripts
71 export PATH="$HOME/.bin:$PATH"
72
73 # mkdir .git/safe in the root of repositories you trust
74 export PATH=".git/safe/../../bin:$PATH"
75
76 # aliases
77 [[ -f ~/.aliases ]] && source ~/.aliases
78
```

```
# Local config
[[ -f ~/.zshrc.local ]] && source ~/.zshrc.local
```

好用的编辑器 Vim

对于 Vim，无需溢美之词，作为与 emacs 并列的两大编辑器，早已经被无数人奉为经典。而它却 又以超长的学习曲线，使得很多人望而却步。长久以来，虽然拥有大量的插件，却缺少一个 确之有效的插件管理器。所幸，Vundle 的出现解决了这个问题。

Vundle 可以让你在配置文件中管理插件，并且非常方便的查找、安装、更新或者删除插件。 还可以帮你自动配置插件的执行路径和生成帮助文件。相对于另外一个管理工具 pathogen， 可以说有着巨大的优势。

```
# vundle 安装和配置
git clone https://github.com/gmarik/Vundle.vim.git ~/.vim/bundle/Vundle.vim
```

```
" 将下面配置文件加入到.vimrc 文件中
set nocompatible " 必须
filetype off     " 必须

" 将 Vundle 加入运行时路径中(Runtime path)
set rtp+=~/.vim/bundle/Vundle.vim
call vundle#begin()

" 使用 Vundle 管理插件，必须
Plugin 'gmarik/Vundle.vim'

"
" 其他插件
"

call vundle#end() " 必须
filetype plugin indent on " 必须
```

最后，你只需要执行安装命令，即可以安装好所需的插件。

```
# 在 vim 中
:PluginInstall

# 在终端
vim +PluginInstall +qall
```

下面列出我的 Vim 插件和配置

Vim 插件 (vimrc.bundles) download

```
if &compatible
  set nocompatible
end

filetype off
set rtp+=~/.vim/bundle/vundle/
call vundle#rc()

" Let Vundle manage Vundle
Bundle 'gmarik/vundle'

```

```
" Define bundles via Github repos
Bundle 'christoomey/vim-run-interactive'
Bundle 'croaky/vim-colors-github'
Bundle 'danro/rename.vim'
Bundle 'kchmck/vim-coffee-script'
Bundle 'kien/ctrlp.vim'
Bundle 'pbrisbin/vim-mkdir'
Bundle 'scrooloose/syntastic'
Bundle 'slim-template/vim-slim'
Bundle 'thoughtbot/vim-rspec'
Bundle 'tpope/vim-bundler'
Bundle 'tpope/vim-endwise'
Bundle 'tpope/vim-fugitive'
Bundle 'tpope/vim-rails'
Bundle 'tpope/vim-surround'
Bundle 'vim-ruby/vim-ruby'
Bundle 'vim-scripts/ctags.vim'
Bundle 'vim-scripts/matchit.zip'
Bundle 'vim-scripts/tComment'
Bundle "mattn/emmet-vim"
Bundle "scrooloose/nerdtree"
Bundle "Lokaltog/vim-powerline"
Bundle "godlygeek/tabular"
Bundle "msanders/snipmate.vim"
Bundle "jelera/vim-javascript-syntax"
Bundle "altercation/vim-colors-solarized"
Bundle "othree/html5.vim"
Bundle "xsbeats/vim-blade"
Bundle "Raimondi/delimitMate"
Bundle "groenewege/vim-less"
Bundle "evanmiller/nginx-vim-syntax"
Bundle "Lokaltog/vim-easymotion"
Bundle "tomasr/molokai"

if filereadable(expand("~/.vimrc.bundles.local"))
  source ~/.vimrc.bundles.local
endif

filetype on
```

Vim 配置 (vimrc) download

```
  " Use Vim settings, rather then Vi settings. This setting must be as early as
  " possible, as it has side effects.
  set nocompatible

  " Highlight current line
  au WinLeave * set nocursorline nocursorcolumn
  au WinEnter * set cursorline cursorcolumn
  set cursorline cursorcolumn

 " Leader
 let mapleader = ","

 set backspace=2   " Backspace deletes like most programs in insert mode
 set nobackup
 set nowritebackup
 set noswapfile    " http://robots.thoughtbot.com/post/18739402579/global-gitignore#com
 set history=50
 set ruler         " show the cursor position all the time
 set showcmd       " display incomplete commands
 set incsearch     " do incremental searching
 set laststatus=2  " Always display the status line
```

```
set autowrite     " Automatically :write before running commands
set confirm       " Need confrimation while exit
set fileencodings=utf-8,gb18030,gbk,big5

" Switch syntax highlighting on, when the terminal has colors
" Also switch on highlighting the last used search pattern.
if (&t_Co > 2 || has("gui_running")) && !exists("syntax_on")
  syntax on
endif

if filereadable(expand("~/.vimrc.bundles"))
  source ~/.vimrc.bundles
endif

filetype plugin indent on

augroup vimrcEx
  autocmd!

  " When editing a file, always jump to the last known cursor position.
  " Don't do it for commit messages, when the position is invalid, or when
  " inside an event handler (happens when dropping a file on gvim).
  autocmd BufReadPost *
    \ if &ft != 'gitcommit' && line("'\"") > 0 && line("'\"") <= line("$") |
    \   exe "normal g`\"" |
    \ endif

  " Cucumber navigation commands
  autocmd User Rails Rnavcommand step features/step_definitions -glob=**/* -suffix=_st
  autocmd User Rails Rnavcommand config config -glob=**/* -suffix=.rb -default=routes

  " Set syntax highlighting for specific file types
  autocmd BufRead,BufNewFile Appraisals set filetype=ruby
  autocmd BufRead,BufNewFile *.md set filetype=markdown

  " Enable spellchecking for Markdown
  autocmd FileType markdown setlocal spell

  " Automatically wrap at 80 characters for Markdown
  autocmd BufRead,BufNewFile *.md setlocal textwidth=80
augroup END

" Softtabs, 2 spaces
set tabstop=2
set shiftwidth=2
set shiftround
set expandtab

" Display extra whitespace
set list listchars=tab:»·,trail:·

" Use The Silver Searcher https://github.com/ggreer/the_silver_searcher
if executable('ag')
  " Use Ag over Grep
  set grepprg=ag\ --nogroup\ --nocolor

  " Use ag in CtrlP for listing files. Lightning fast and respects .gitignore
  let g:ctrlp_user_command = 'ag %s -l --nocolor -g ""'

  " ag is fast enough that CtrlP doesn't need to cache
  let g:ctrlp_use_caching = 0
```

```
endif

" Color scheme
colorscheme molokai
highlight NonText guibg=#060606
highlight Folded  guibg=#0A0A0A guifg=#9090D0

" Make it obvious where 80 characters is
set textwidth=80
set colorcolumn=+1

" Numbers
set number
set numberwidth=5

" Tab completion
" will insert tab at beginning of line,
" will use completion if not at beginning
set wildmode=list:longest,list:full
function! InsertTabWrapper()
    let col = col('.') - 1
    if !col || getline('.')[col - 1] !~ '\k'
        return "\<tab>"
    else
        return "\<c-p>"
    endif
endfunction
inoremap <Tab> <c-r>=InsertTabWrapper()<cr>
inoremap <S-Tab> <c-n>

" Exclude Javascript files in :Rtags via rails.vim due to warnings when parsing
let g:Tlist_Ctags_Cmd="ctags --exclude='*.js'"

" Index ctags from any project, including those outside Rails
map <Leader>ct :!ctags -R .<CR>

" Switch between the last two files
nnoremap <leader><leader> <c-^>

" Get off my lawn
nnoremap <Left> :echoe "Use h"<CR>
nnoremap <Right> :echoe "Use l"<CR>
nnoremap <Up> :echoe "Use k"<CR>
nnoremap <Down> :echoe "Use j"<CR>

" vim-rspec mappings
nnoremap <Leader>t :call RunCurrentSpecFile()<CR>
nnoremap <Leader>s :call RunNearestSpec()<CR>
nnoremap <Leader>l :call RunLastSpec()<CR>

" Run commands that require an interactive shell
nnoremap <Leader>r :RunInInteractiveShell<space>

" Treat <li> and <p> tags like the block tags they are
let g:html_indent_tags = 'li\|p'

" Open new split panes to right and bottom, which feels more natural
set splitbelow
set splitright

" Quicker window movement
nnoremap <C-j> <C-w>j
nnoremap <C-k> <C-w>k
```

```
    nnoremap <C-h> <C-w>h
148 nnoremap <C-l> <C-w>l
149
150 " configure syntastic syntax checking to check on open as well as save
151 let g:syntastic_check_on_open=1
152 let g:syntastic_html_tidy_ignore_errors=[" proprietary attribute \"ng-"]
153
154 autocmd Syntax javascript set syntax=jquery " JQuery syntax support
155
156 set matchpairs+=<:>
157 set statusline+=%{fugitive#statusline()} "  Git Hotness
158
159 " Nerd Tree
160 let NERDChristmasTree=0
161 let NERDTreeWinSize=40
162 let NERDTreeChDirMode=2
163 let NERDTreeIgnore=['\~$', '\.pyc$', '\.swp$']
164 let NERDTreeShowBookmarks=1
165 let NERDTreeWinPos="right"
166 autocmd vimenter * if !argc() | NERDTree | endif " Automatically open a NERDTree if no
167 autocmd bufenter * if (winnr("$") == 1 && exists("b:NERDTreeType") && b:NERDTreeType =
168 nmap <F5> :NERDTreeToggle<cr>
169
170 " Emmet
171 let g:user_emmet_mode='i' " enable for insert mode
172
173 " Search results high light
174 set hlsearch
175
176 " nohlsearch shortcut
177 nmap -hl :nohlsearch<cr>
178 nmap +hl :set hlsearch<cr>
179
180 " Javascript syntax hightlight
181 syntax enable
182
183 " ctrap
184 set wildignore+=*/tmp/*,*.so,*.swp,*.zip     " MacOSX/Linux"
185 let g:ctrlp_custom_ignore = '\v[\/]\.(git|hg|svn)$'
186
187 nnoremap <leader>w :w<CR>
188 nnoremap <leader>q :q<CR>
189
190 " RSpec.vim mappings
191 map <Leader>t :call RunCurrentSpecFile()<CR>
192 map <Leader>s :call RunNearestSpec()<CR>
193 map <Leader>l :call RunLastSpec()<CR>
194 map <Leader>a :call RunAllSpecs()<CR>
195
196 " Vim-instant-markdown doesn't work in zsh
197 set shell=bash\ -i
198
199 " Snippets author
200 let g:snips_author = 'Yuez'
201
202 " Local config
203 if filereadable($HOME . "/.vimrc.local")
204   source ~/.vimrc.local
    endif
```

新世代的版本管理工具 git

Git 是一个分散式版本控制软件。最初的目的是为了更好的管理 Linux 内核开发而设计。与 CVS、Subversion 等集中式版本控制软件不同，Git 不需要服务器端软件就可以发挥版本控制的作用。 使得代码的维护和发布变得非常方便。

Git 库目录结构

- hooks : 存储钩子文件夹
- logs : 存储日志文件夹
- refs : 存储指向各个分支指针的(SHA-1)的文件夹
- objects : 存储 git 对象
- config : 存储配置文件
- HEAD : 指向当前分支的指针文件路径

```
1 # 安装 git
2 brew install git
```

Git 安装完毕后，只需要使用 `git config` 简单配置下用户名和邮箱就可以使用了。

- Git 中文简易指南
- Git 官网帮助

为了使 Git 更好用，对 Git 做一些配置，`.gitconfig` 文件中可以设置自定义命令等，`.gitignore` 文件是默认被忽略版本管理的文件。

(gitconfig) download

```
1  [push]
2    default = current
3  [color]
4    ui = auto
5  [alias]
6    aa = add --all
7    ap = add --patch
8    ca = commit --amend
9    ci = commit -v
10   co = checkout
11   br = branch
12   create-branch = !sh -c 'git push origin HEAD:refs/heads/$1 && git fetch origin && git
13   delete-branch = !sh -c 'git push origin :refs/heads/$1 && git branch -D $1' -
14   merge-branch = !git checkout master && git merge @{-1}
15   pr = !hub pull-request
16   st = status
17   up = !git fetch origin && git rebase origin/master
18 [core]
19   excludesfile = ~/.gitignore
20   autocrlf = input
21 [merge]
22   ff = only
23 [include]
24   path = .gitconfig.local
25 [commit]
26   template = ~/.gitmessage
27
28
```

```
  [fetch]
    prune = true
[user]
    name = zgs225
    email = zgs225@gmail.com
[credential]
    helper = osxkeychain
[github]
    user = zgs225
```

(gitignore) download

```
*.DS_Store
*.sw[nop]
.bundle
.env
db/*.sqlite3
log/*.log
rerun.txt
tags
tmp/**/*
!tmp/cache/.keep
```

自动集成 ternimal 环境

感谢 thoughtbot 组织发布的开源项目，可 以轻松的完成上述配置。这是我 fork 项目的地址 (https://github.com/zgs225/dotfiles)， 欢迎 fork 并完善成属于你自己的配置。

安装步骤：

```
# 更改为 zsh, 详细参考上面 zsh 部分
chsh -s $(which zsh)

# clone 源码
git clone https://github.com/zgs225/dotfiles.git

# 安装 rcm
brew tap thoughtbot/formulae
brew install rcm

# 安装上述环境并且完成配置
rcup -d dotfiles -x README.md -x LICENSE -x Brewfile
```

语言篇

编程语言五花八门，它们各自的版本也是万别千差。各种语言之间或多或少都存在着向前， 或者向后的不兼容。因为版本不兼容导致的 bug 也是格外的招人烦。所以，在语言篇这篇，也 是侧重与到编程语言版本管理已经环境控制。

简洁优美的类 Lisp 语言 Ruby

以 Ruby 作为语言篇的开篇，足以看得出来我对 Ruby 的喜爱。虽然它有着这样或者那样令人诟 病的缺点，不过作为让我体会到 Web 世界美妙的第一门语言，我对 Ruby 一直有着别样的感情。

目前，Ruby 的常用版本是 1.9，2.1 和最新的 2.2。最新版本并不是完全向后兼容，所以如果你 的电脑中存在着老版本的 Ruby 项目，这时候又想切换到新版本中来，那可就头疼了。好在， 有像 `rvm` 和 `rbenv` 这样的 Ruby 版本管理软件。它们各有优劣，而我喜欢更为自动化的 rvm。

一个完整的 Ruby 环境包括 Ruby 解释器、安装的 RubyGems 以及它们的文档。rvm 用 `gemsets` 的概念，为每一个版本的 Ruby 提供一个独立的 RubyGems 环境。可以很方便的在不同的 Ruby 环境 中切换而不相互影响。

```
# 安装 rvm
# 设置 mpapis 公钥
gpg --keyserver hkp://keys.gnupg.net --recv-keys 409B6B1796C275462A1703113804BB82D39DC0B

# 安装稳定版 rvm
\curl -sSL https://get.rvm.io | bash -s stable
```

由于网络原因，可以将 rvm 的 Ruby 安装源修改为国内淘宝的 Ruby 镜像服务器。 该镜像服务器 15 分钟以此更新，尽量保证与官方同步。这样能提高安装速度。

```
# 出自 http://ruby.taobao.org
sed -i .bak 's!cache.ruby-lang.org/pub/ruby!ruby.taobao.org/mirrors/ruby!' $rvm_path/con
```

推荐一篇关于 rvm 的文章: https://ruby-china.org/wiki/rvm-guide

同样，由于网络原因，需要将 RubyGems 的安装源修改到镜像服务器上。

```
 # 切换源
 gem sources --remove https://rubygems.org/
 gem sources -a https://ruby.taobao.org/

 # 查看源列表
 gem sources -l

 *** CURRENT SOURCES ***

https://rubygems.org/
```

以上，你就拥有了一个相对舒适的 Ruby 开发环境，不用为版本和网络问题发愁。啊！天空都 清净了。