目录

[Git 1](#_Toc137581501)

[<vim>操作 1](#_Toc137581502)

[<GitHub 入门> 2](#_Toc137581503)

[<Git版本控制管理> 4](#_Toc137581504)

[WebStorm 6](#_Toc137581505)

[PyCharm 7](#_Toc137581506)

[Eclipse IDE 7](#_Toc137581507)

# Git

设置代理

$ git config --global http.proxy 165.225.96.34:10015 // Or ENV HTTP\_PROXY

$git config --global --unset-all http.proxy 去除

## <vim>操作

1. 光標在屏幕文本中的移動既可以用箭頭鍵

2. 欲進入vim編輯器(從命令行提示符)，請輸入︰vim 文件名 <回車>

3. 欲退出vim編輯器，請輸入以下命令放棄所有修改︰

<ESC> :q! <回車>

或者輸入以下命令保存所有修改︰

<ESC> :wq <回車>

4. 在正常模式下刪除光標所在位置的字符，請按︰ x

5. 在正常模式下要在光標所在位置開始插入文本，請按︰

i 輸入必要文本 <ESC>

how to solve the slow access of github?

http://blog.csdn.net/wu\_\_di/article/details/50538916

access http://github.global.ssl.fastly.net.ipaddress.com/#ipinfo to get the fast IP for github.global.ssl.fastly.net, then search fast IP for github.com

$sudo gedit /etc/hosts

192.168.xx.xx github.com

185.31.17.xx github.global.ssl.fastly.net

then update DNS cache

$ sudo /etc/init.d/nscd restart

## <GitHub 入门>

<https://blog.csdn.net/qq_32452623/article/details/78355976>

1. Git配置

$ git //列出选项和最常用的子命令

$ git help --all

$ git config --global user.name "QiZhong Lin" //配置git用户名和邮箱

$ git config --global user.email "qizhong.liin@philips.com"

$ git config --unset --global user.email //移除设置

$ git config --global https.proxy http://185.46.212.34:10015

$ git config --global http.proxy http://185.46.212.34:10015

1. Remote Git

register username/password

new repository -> repository name -> create repository -> <https://github.com/qizhonglin/doc.git>

1. local Git

// clone, by default, origin == https://github.com/qizhonglin/doc.git

GitHub $ git clone https://github.com/qizhonglin/doc.git

GitHub $ cd doc

CRUD files inside GitHub/doc

GitHub/doc (master) $ git status

GitHub/doc (master) $ **git add .**

GitHub/doc (master) $ **git commit -m “update…”**

GitHub/doc (master) $ git log // history log before current status, 如果很长，按q退出

GitHub/doc (master) $ git show 9da... //查看特定提交的详细信息,若无提交码，只显示最近一次提交的详细信息

// check in, git push -u origin master

GitHub/doc (master) $ **git push**

1. branch

GitHub/doc (master) $ git branch // list local branch

GitHub/doc (master) $ git branch -r // list remote branch

GitHub/doc (master) $ git branch -a // list local and remote branch

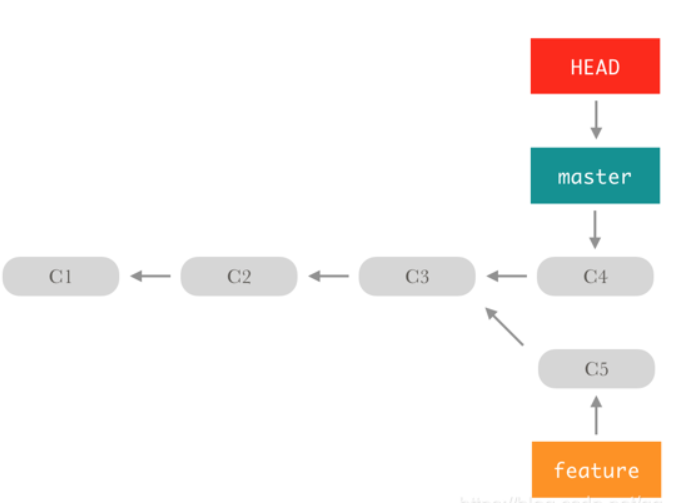
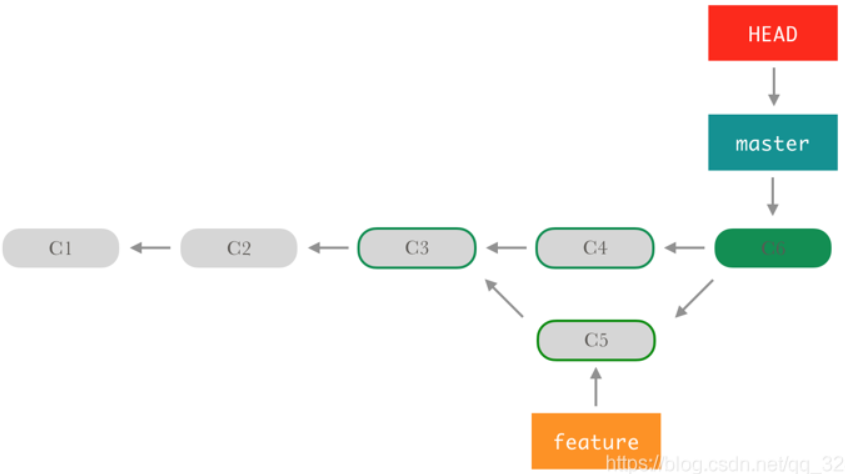
//create new branch and switch to new branch in local repo

GitHub/doc (master) $ git branch feature

GitHub/doc (master) $ **git checkout feature**

GitHub/doc (feature) $ git reflog // action log of current repository

# 合并分支

 =>

GitHub/doc (feature) $ git checkout master # switch to master branch

GitHub/doc (master) $ git merge feature // merge

如果branch feature and master 有冲突（即可能修改同一个文件），git无法自动合并，需要手动处理冲突，按i编辑提交信息，:wq!保存并退出页面

GitHub/doc (master) $ git status # 查看冲突信息

查看造成冲突的文件，修改冲突部分，比如\*.txt

GitHub/doc (master) $ git add\*.txt # 对修改后冲突文件

GitHub/doc (master) $ git commit -m ‘fix conflict’ # 创建一个修改冲突的提交。

实际上master一般作为一个保护分支，是不允许被直接提交和推送的，这种情况，我们可以会去到对应的git管理页面（比如gitlab或者github），通过创建MR的方式，来创建合并请求，并在通过人工审核后进行合并。这样就少了需要在本地进行本地分支合并的过程。

# 分支回滚

GitHub/doc (master) $ git reset --hard fd0cbf0 // reset to selected version, fd0... is commit id

# 分支删除

GitHub/doc (master) $ git branch -d feature # 提醒删除

GitHub/doc (master) $ git branch -D feature # 强制删除

GitHub/doc (master) $ git push origin --delete feature # 远端删除

// check out, git pull origin master

GitHub/doc (master) $ git pull

1. 回退指定版本

GitHub/doc (master) $ git tag -l #检查历史版本

GitHub/doc (master) $ git checkout tags/v2.7.1 #回退2.7.1版本

…

GitHub/doc (master) $ git branch #查看本地版本 估计含最新版本

GitHub/doc (master) $ git checkout develop #选择最新版本

1. delete repository in Github

进入指定repository -> Settings -> delete repository

1. push existing Git project to Github

在Github先创建repository: https://github.com/qizhonglin/ionic-blank-ex0.git

$git remote add origin https://github.com/qizhonglin/ionic-blank-ex0.git

$git push origin master

1. Git pull 强制覆盖本地文件

git fetch --all

git reset --hard origin/master

1. ignore files, only submit selected files

# create file .gitignore

\_\_pycache\_\_/ ignore any directory called \_\_pycache\_\_/ in any subdirectory

build/

1. if the repository already exists, do the following:

git rm -r --cached .

git add .

git commit -m "..."

git push

Tool: GitHub Desktop

a. Install GitHub Desktop

username and password

b. Autenticate GitHub

if behind a proxy server,

git config --global http.proxy <http://ip:port>

## <Git版本控制管理>

开发以及维护开发出来的代码，方便读取代码的历史版本、记录所有的修改。

分布式开发：允许并行开发，各人可以在自己的版本库中独立且同时地开发，而不需要与一个中心版本库时刻同步（因为这样会造成开发瓶颈）

CVS给予每个开发人员对于自己的私有版本写的权限，因此，不同开发人员的改动可以自动合并，除非两个开发人员尝试修改同一行。**如果出现修改同一行的情况，那这一行将会作为“冲突”被标记出来，由开发人员手动去解决。**

Git Bash/Gui

Git是作为一套简单的、独特的、独立的命令提供的，并按照“UNIX"工具包的哲学来开发的：打造小的、可互操作的工具。

GNU风格：git commit -m "..." 等价于 git commit --message="..."

建立Git版本库：

1. 从头开始创建，用现有的内容填充它。
2. clone已有的版本库

$ mkdir public\_html && cd public\_html

‘’’

为了显示目录是一个Git版本库，git init命令创建了一个隐藏目录，在项目的顶层目录，名为.git

而CVS和SVN则将修订版信息放在项目的每一个目录下的CVS和.svn子目录里，Git把所有修订信息都放在这唯一的顶层.git目录里。

‘’’

public\_html$ git init //将目录转化为Git版本库

$ git add . //把当前目录及子目录中的文件都添加到版本库里。

$ git commit -m "..." //add and commit步聚分开，是为了避免频繁变化

$ git show-branch --more=10 //显示当前开发分支简洁的单行摘要

$ git diff

// 删除文件和， 和添加操作一样，删除操作也需要两步：

1. git rm表示你想要删除这个文件的意图并暂存这个变更，
2. git commit在版本库里实现这个变更。

$ git rm poem.html

$ git commit -m "Remove a poem"

// 文件改名

$ git mv foo.html bar.html

$ git commit -m "Moved foo to bar"

版本库的配置文件： .git/config

用户特定的配置文件: ~/.gitconfig 可用--global选项修改

$ git config -l //列出所有的配置参数

$ cat .git/config //查看版本库的配置文件内容

$ find .git/objects //查找指定目录下的文件

Git版本库只是一个简单的数据库，其中包含所有用来维护与管理项目的修订版本和历史信息。Git在每个版本库里维护一组配置值。版本库的用户和email地址。Git对每个网站、每个用户和每个版本库的配置和设置信息都进行管理和检查。在版本库中，Git维护两个主要的数据结构：**对象库(boject store)和索引(index)**.所有这些版本库数据存放在工作目录根目录下一个名为.git的隐藏子目录中。

对象库在复制操作的时候能进行有效复制，这也是用来支持完全分布式VCS的一种技术。对象库是Git版本库实现的心脏。它包含你的原始数据文件和所有日志消息、作者信息、日期，以及其化用来重建项目任意版本或分支的信息。

索引是暂时的信息，对版本库来说是私有的，并且可以在需要的时候按需求进行创建和修改。

在互联网上，文件或者任意大小的blob都可以通过仅比较它们的SHA1标识符来判断是否相同

**Git仅根据文件内容来计算每一个文件的散列码**，如果文件有相同的SHA1值，它们的内容就是相同的，然后将这个blob对象放到对象库里，并以SHA1值作为索引。如果这些文件中的一个发生了变化，Git会为它计算一个新SHA1值，识别出它现在是一个不同的blob对象，然后把这个新的blob加到对象库里。原来的blob在对象库里保持不变，为没有变化的文件所使用。

Git把文件名视为一段区别于文件内容的数据。Git仅仅记录每个路径名，并且确保能通过它的内容精确地重建文件和目录，这些是由散列值来索引的

Git使用打包文件(pack file)的更有效的存储机制。要创建一个打包文件，Git首先定位内容非常相似的全部文件，然后为它们之一存储整个内容。之后计算相似文件之间的差异并且**只存储差异**。

SHA1：安全散列加密算法，3b1..., 160位，20个字节 = 40个字节的十六进制来显示

散列函数：在数学意义上是一个真正的函数，对于一个给定的输入，它总产生相同的输出。这样的散列函数有时也称为摘要，用来强调它就像散列对象的摘要一样。应用：提供了一种有效的方法来比较两个对象，甚至是两个非常大而复杂的数据结构，而且并不需要完全传输。

SourceTree使用

# WebStorm

Search everywhere 

Search for project elements

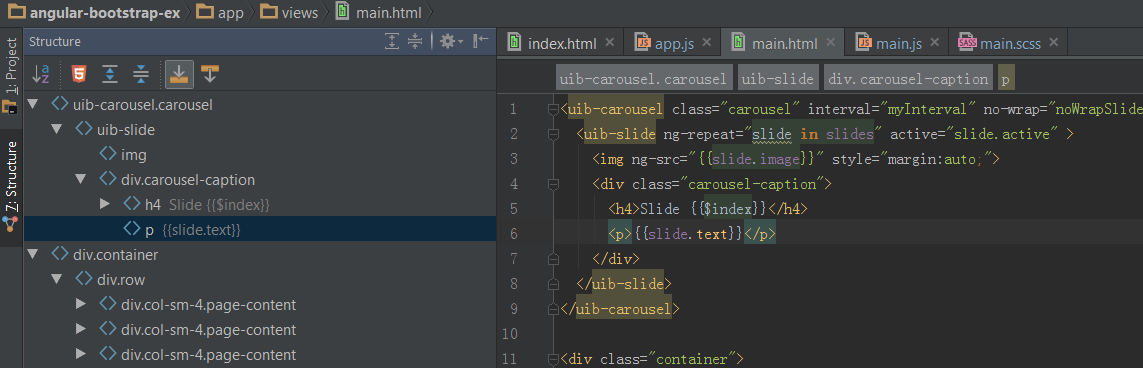
Ctrl + n for class

Ctrl + shift + n for a file

Ctrl + shift + alt + n for a symbol

Search for code elements ctrl + f

To view file structure view Alt + 7



Navigate across the IDE ctrl + tab

Edit and write code

Code completion ctrl + space



Auto-create a new method that you have not yet declared

Intellij idea

How to add junit4 test?

<https://www.jetbrains.com/help/idea/2016.2/creating-tests.html>

将鼠标放到editor里的类名，按Alt+Enter会弹出Create Test, Create subclass

# PyCharm

File -> settings -> Project -> Project Interpreter

默认是系统默认的python 即/usr/bin/python 一般是python2

也可以改为/usr/bin/python3

甚至可以改为virutalenv下的env/bin/python

甚至是远端的python执行环境

这样就可以debug

# Eclipse IDE

1. eclipse中,把java函数代码折叠/展开

首先在eclipse 中开启设置代码折叠功能

a. windows->perferences->General->Editors->Structured Text Editors

可以看到Enable folding选项，打上勾就可以使用代码折叠功能，但还要在具体的语言中设置。

b.windows->perferences->Java->Editors->Folding

可以看到Enable folding选项，打上勾就可以使用代码折叠功能。

其次使用快捷键

下面你就可以用如下快捷键在你的java class 中 折叠或者展开你的代码了.

代码折叠的快捷键，默认是：

Ctrl+Shift+Numpad\_Divede(小键盘的/号)

Ctrl+Shift+Numpad\_Multiply(小键盘的\*号)

笔记本没小键盘，于是改成：

Ctrl+Shift+-

Ctrl+Shift+=

eclipse中，把python函数代码折叠/展开

Ctrl + 9/0

2. override父类API

右击 -> Source -> override/implement methods

3. java注释快捷键

ctrl + shift + /

ctrl + shift + \