# 一:Linux常用命令

1. pwd 获得当前路径
2. whoami 获得用户名
3. shutdown -s -t 0 延迟0秒关机
4. clear 清屏
5. cd 进入
6. mkdir 文件夹名 创建文件夹
7. touch 文件名.后缀名 创建文件
8. echo 内容>>文件名.后缀名 追加内容
9. echo 内容>文件名.后缀名 覆盖
10. rm 文件名.后缀名 删除文件
11. rmdir 文件夹 删除空文件夹
12. rm -rf 文件夹 删除文件夹以及文件
13. cd 文件夹\* 可自动补全
14. mv 文件名/文件夹名 文件名/文件夹名 复制文件/复制文件
15. cat 文件名 读文件
16. less more 读文件
17. cp 文件名 复制文件
18. head -2 文件名 显示前2行
19. tail -2 文件名 显示后2行
20. history 查看输入的历史记录

VI编辑器 vi 文件名(进入vi编辑器)

一:命令模式

1. yy p粘贴
2. q 退出
3. ZZ 保存并退出
4. u 撤销到上一步
5. i 切换到编辑模式
6. ESC 退出 i模式
7. 底层模式
8. :q 退出
9. :w保存
10. :wq保存并退出
11. :e!全部撤销返回上一次状态
12. :q!强制退出,不保存
13. :set nu 设置行号
14. 编辑模式
15. ESC 切换到命令模式
16. crxChrome.com 越岭 谷歌查阅

# 二git

## Git安装

**Window安装**

<http://git-scm.com/download/win>下载Git客户端软件，和普通软件安装方式一样。

**Linu安装**

CentOS发行版：sudo yum install git

Ubuntu发行版：sudo apt-get install git

**Mac安装**

打开Terminal直接输入git命令，会自动提示，按提示引导安装即可。

学习网站:

<http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000>

<http://backlogtool.com/git-guide/cn/>

<http://www.ruanyifeng.com/blog/2015/12/git-cheat-sheet.html>

<http://rogerdudler.github.io/git-guide/index.zh.html>

<https://git-scm.com/book/zh/v2>

## Git本地仓库

### Git基础

命令行方式：任意目录（建议开发目录）右键 > Git Bash Here

**1、配置用户**

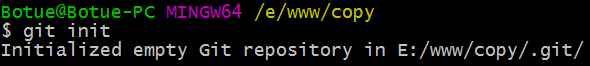
git config --global user.name "自已的名字"

git config --global user.email "自已的邮箱地址"

git config --list 查看当前用户配置

**2、初始化仓库**

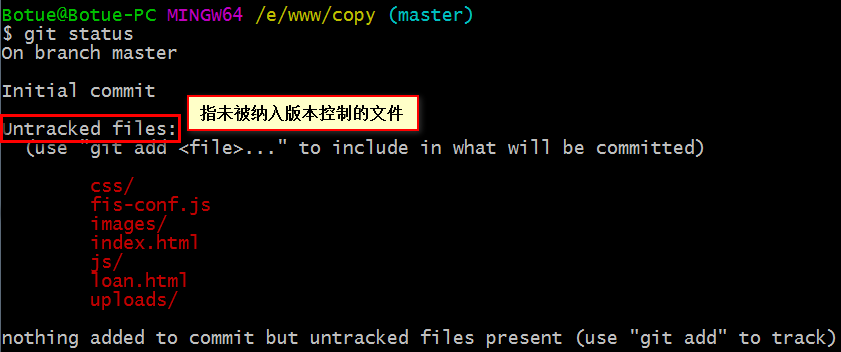
git init



git init只是创建了一个名为.git的隐藏目录，这个目录就是存储我们历史版本的仓库，ls -al 可以查看。

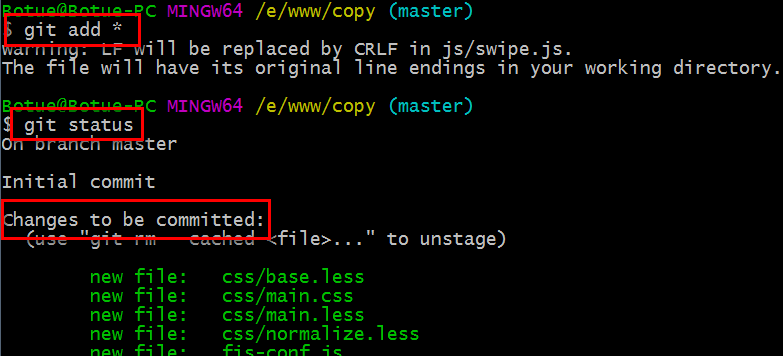
**3、查看文件状态**

git status可以检测当前仓库文件的状态



**4、添加文件到暂存区**

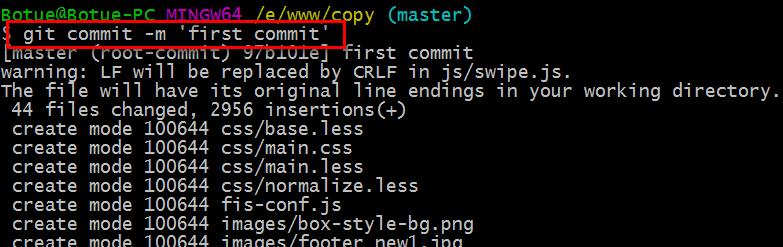
git add file “\*”或-A代表所有



放到暂存区的文件被标记成了绿色，等待提交。

**5、提交文件**

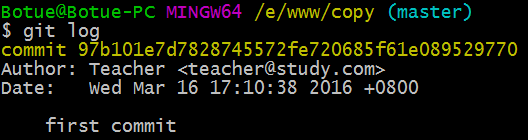
git commit -m '备注信息'



将暂存区被标记成绿色的文件，全部提交到仓库存储。

**6、查看提交历史**

git log

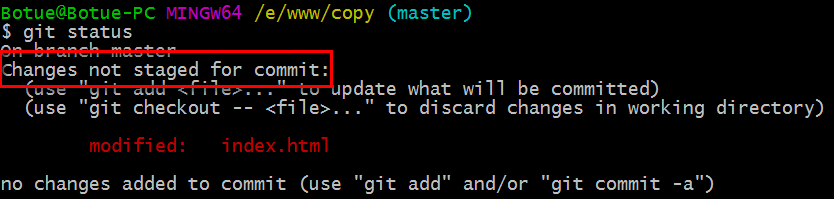


我们可以查看到一次提交记录，97b101e7d代表这次提交的唯一ID，一般称为SHA值。

这时我们对index.html文件做修改

**7、再次检测仓库文件状态**

git status

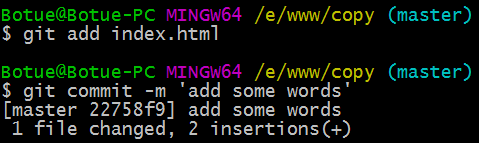


被修改过的文件被标记成了红色，等待重新添加到暂存区。

**8、重新添加暂存区然后提交**

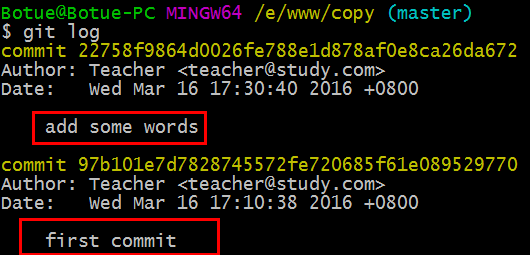
git add index.html

git commit -m 'add some words'



**9、再次查看历史**

git log 可查到所有提交历史



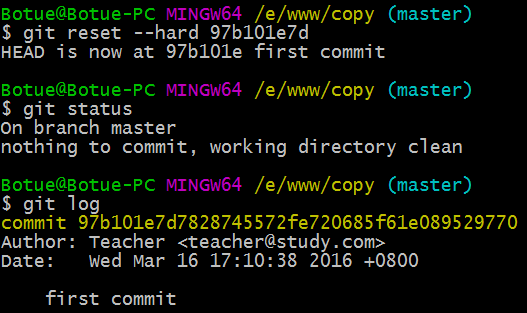
这时可以查看到两次提交历史。

这时关掉所有目录甚至关机！

**10、恢复上一次提交的状态**

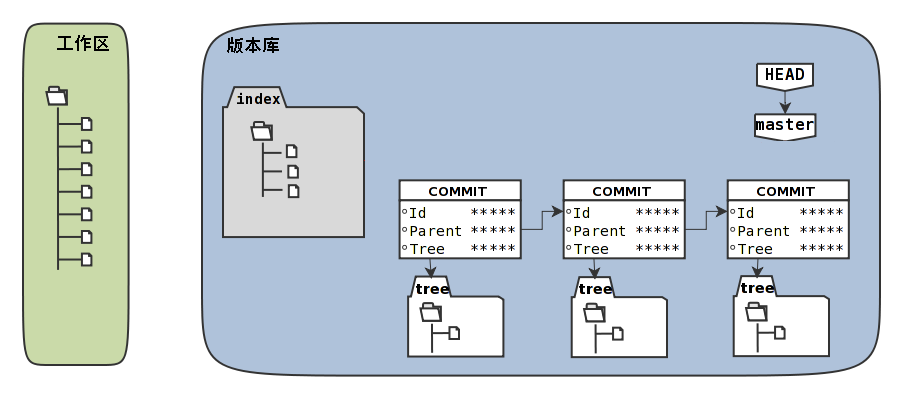
git reset --hard 97b101e7d 查看index.html发现回到了没有修改的状态

git log再次查看发现提交历史只有一个了



这时我们非常容易就回到了曾经的一个历史版本。

仓库示意图

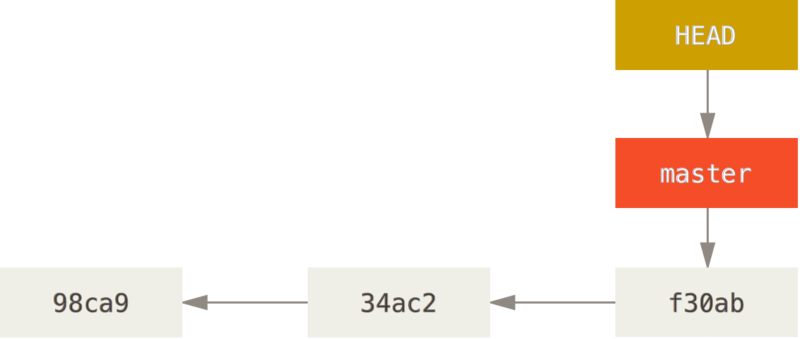


### Git分支

在我们的现实开发中，需求往往是五花八门的，同时开发个需求的情况十分常见，比如当你正在专注开发一个功能时，突然有一个紧急的BUG需要你来修复，这个时候我们当然是希望在能够保存当前任务进度，再去修改这个BUG，等这个BUG修复完成后再继续我们的任务。如何实现呢？

通过创建分支来解决实际开发中类似的问题。

如下图所示：



当我们在初始化仓库的时候，Git会默认帮我们创建了一个master的分支，并且HEAD默认指向了master末端。

我们也可以创建自已的分支

**1、创建分支**

git branch hotfix

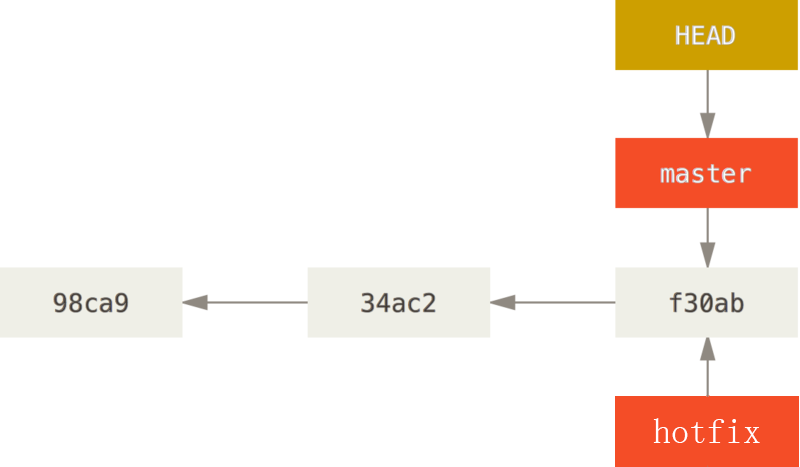


图4-1

**2、切换分支**

git checkout hotfix

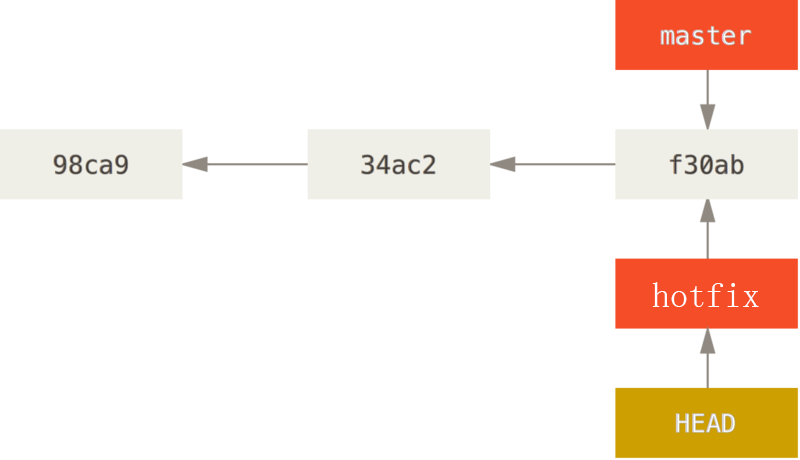


图4-2

我们发现HEAD现在又指向了developer的末端。

**3、再次提交操作**

git add -A

git commit -m 'add some code for hotfix'

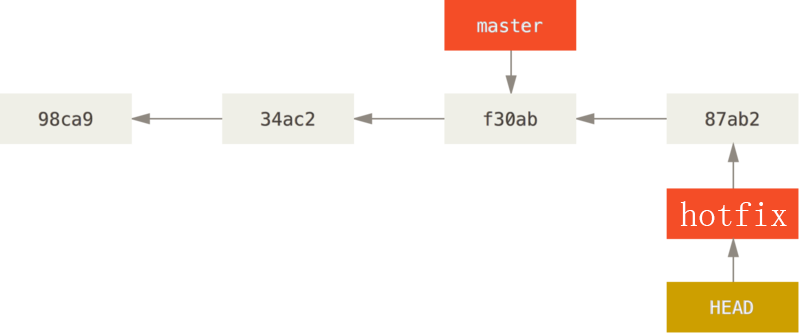


图4-3

这次的提交历史版本就会记录在developer这个分支上了，并且HEAD伴随developer在移动。

**4、当我们再次切回到master时**

git checkout master

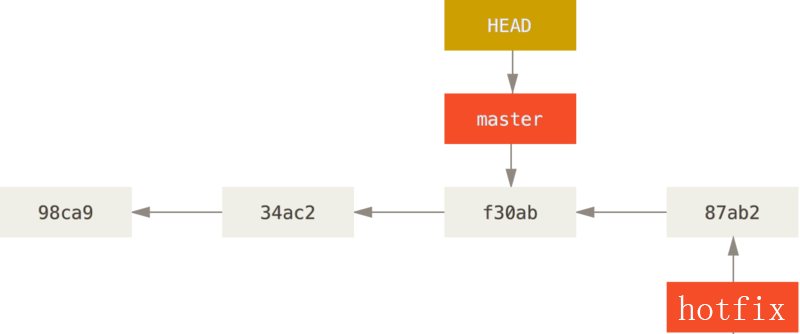


图4-4

当我们切换回master后，HEAD指向了master分支的末端。

**5、继续之前的开发**

git add -A

git commit -m 'add some code for master'

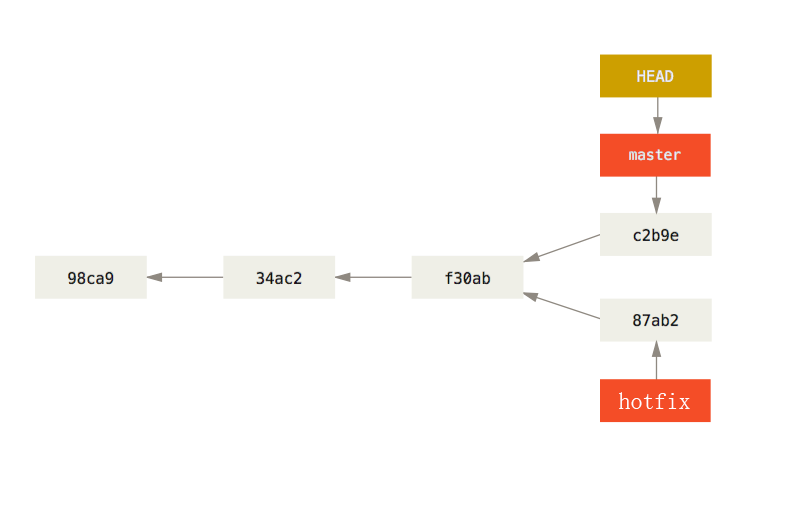


图4-5

总结：当我们git checkout branchname时，HEAD会自动指向对应分支的末端，工作目录中的源码也会 随之发生改变。

这个时候我们就在hotfix这个分支上修复了这个BUG，而我们原来在master分支上的操作并未受到影响。

思考一个问题：

现在master这个分支上是否包含了hotfix的修复呢？

实际上从图4-5可以看出这时的master分支并没有包含有hotfix的修复。

**6、合并（融合）分支**

git checkout master

git merge hotfix

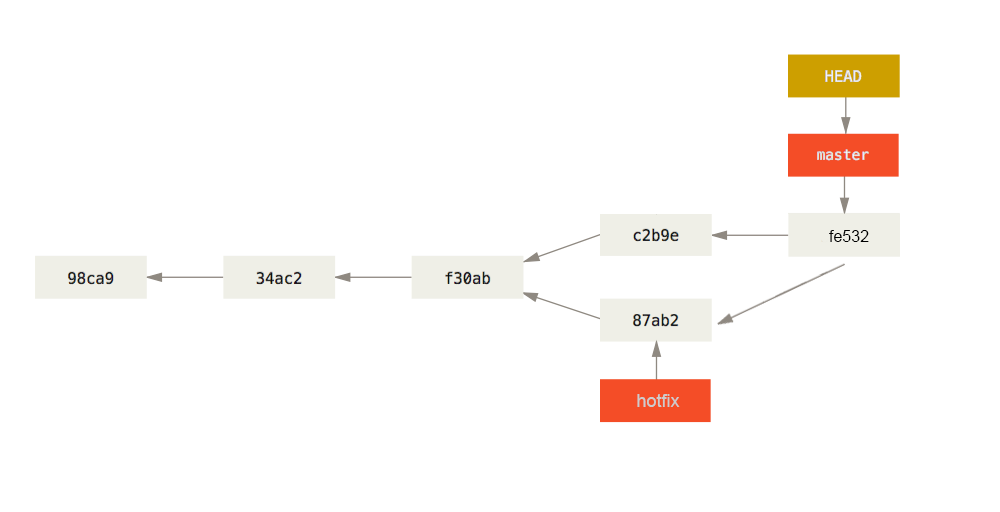


图4-6

这时master会有两个父结点了，master便包含了hotfix里的修复了

**7、删除分支**

git branch -d hotfix

这时用来修复BUG创建的hotfix分支已经没有用处了，我们可以将它删除。

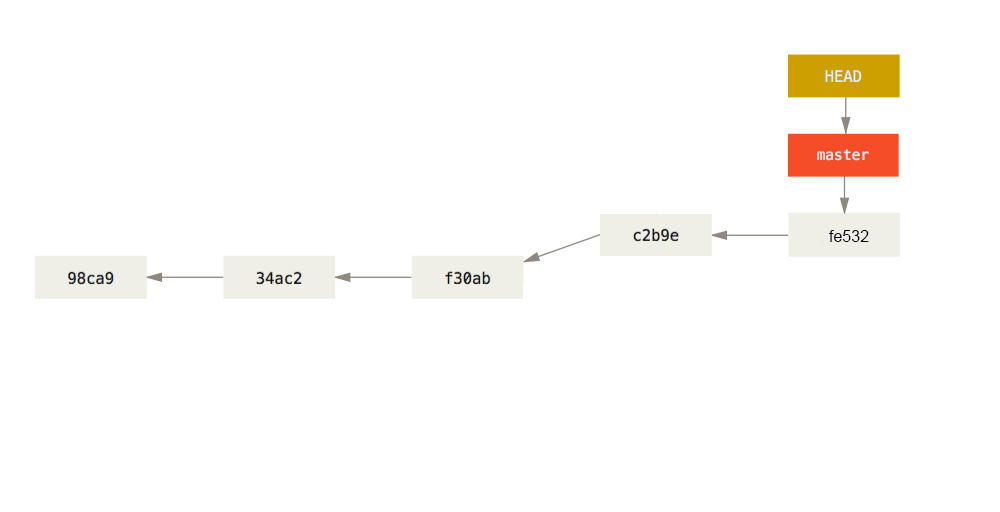


图4-7

## Git远程仓库

通过上面学习我们可以很好的管理本地版本控制了，可是如果我们下班回到家里突然来了灵感觉得有部分代码可以优化，如果能接着公司电脑上的代码继续写该有多好呀！另一种情形，假设项目比较大，不同的功能模块由不同的开发人员完成，不同模块儿之间又难免会依赖关系，这时如果我们的代码互相合并（融合）该有多好呀！所有模块开发完毕后，需要整合到一起，要能做到准确无误该有多好呀！

借助一个远程仓库，大家可以共享代码、历史版本等数据，便可以解决以上遇到的所有问题，在学习远程仓库前我们先来学习。

git clone path这个命令。

## gitHub和gitLab

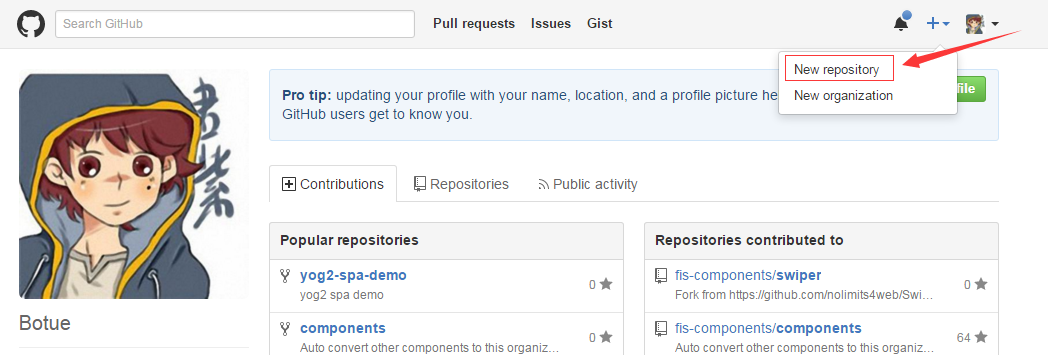
如果我们熟悉服务器的话，我们完全可以将上述的步骤在我们的远程服务器上进行操作，然后再做一些登录权限的设置，就可非常完美的搭建一个共享服务器了，但是为了可以更好的管理我们的仓库，一些第三方机构开发出了Web版仓库管理程序，通过Web界面形式管理仓库。

gitHub关于它的名气与意义，大家可以自行查阅，我们这里介绍它的使用

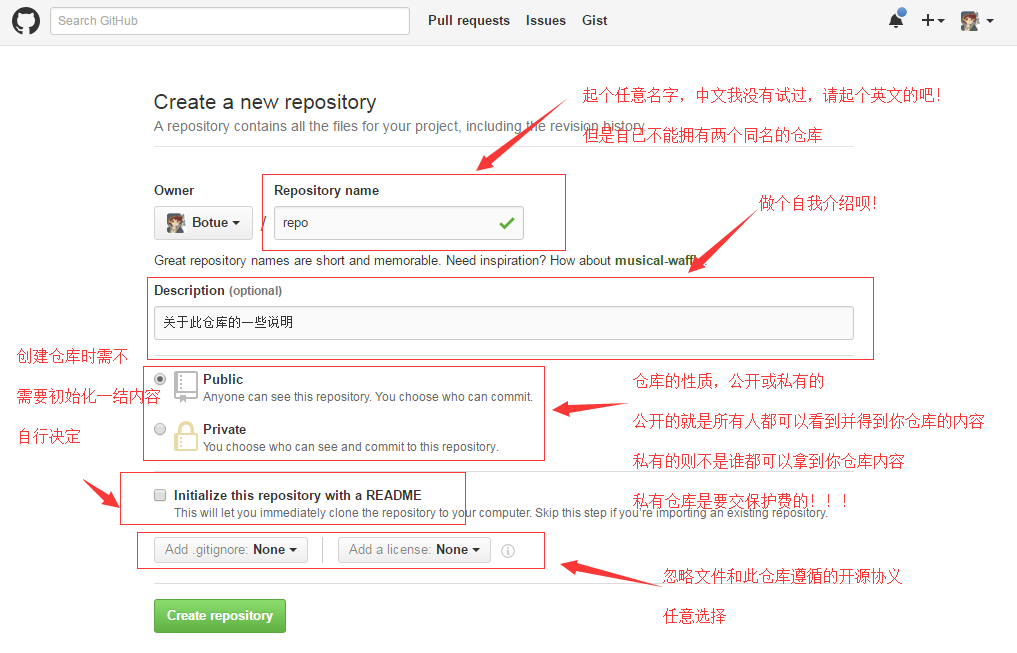
**1、注册账号并完善资料**

自行注册略过

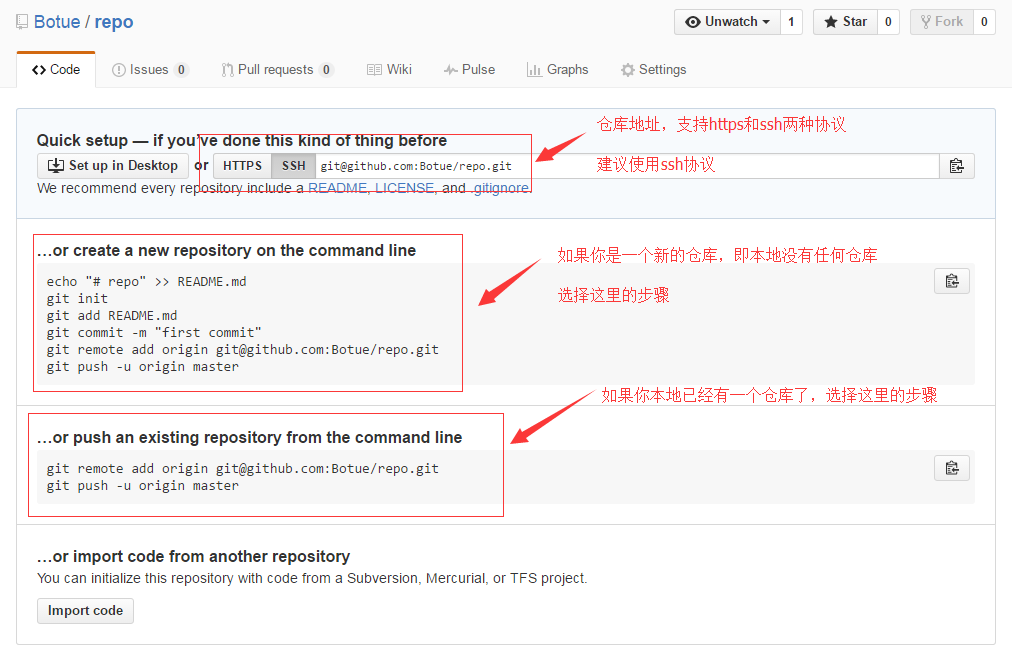
**2、创建共享仓库**



**3、填写仓库资料**



**4、共享仓库**

****

远程地址特别长，我们可以给他起一个别名

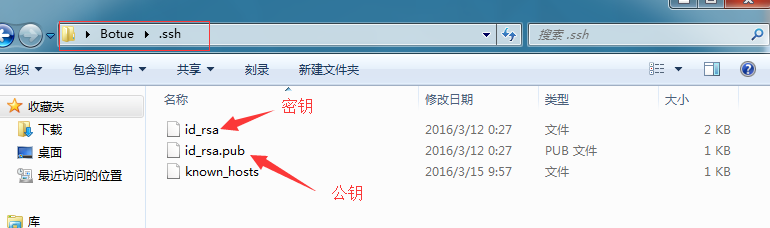
git remote add origin git@github.com:Botue/repo.git

这样origin 就代表 git@github.com:Botue/repo.git

当我们通过git clone 从共享仓库获内容时，会自动帮我们添加origin到对应的仓库地址，例如：git clone git@github.com:Botue/repo.git 会自动添加origin 对应 git@github.com:Botue/repo.git

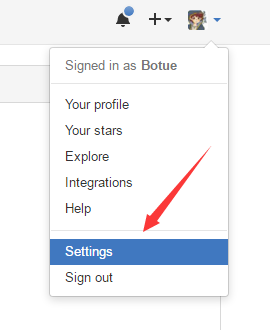
**5、生成密钥**

ssh-keygen -t rsa 然后一路回车，这里会在当前用户生成了一个.ssh的文件夹

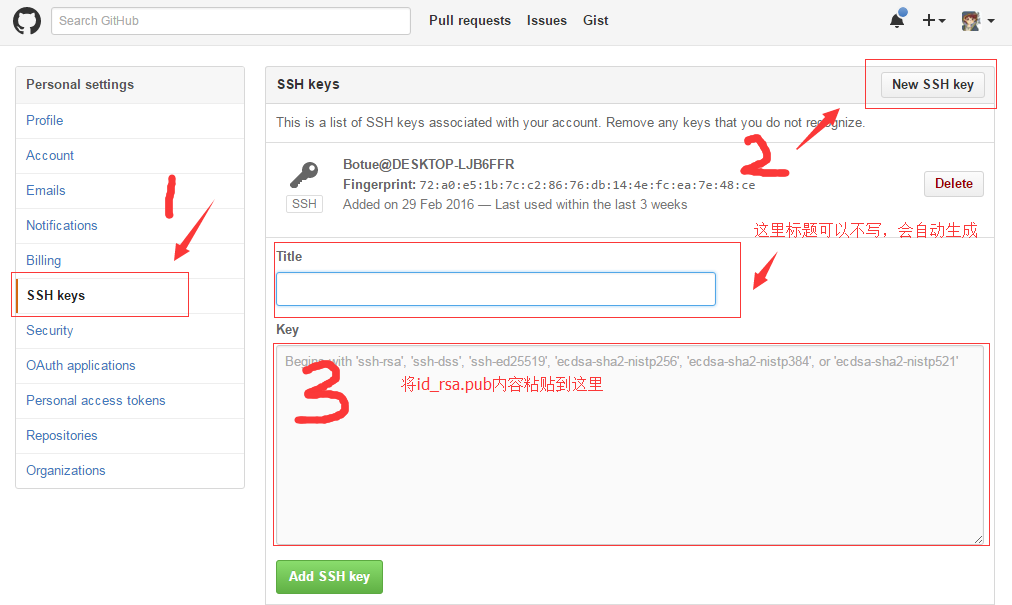


将id\_rsa.pub公钥的内容复制

打开gitHub的个人中心



打到SSH keys



到此我们便可以通过gitHub 提供的Web界面来管理我们的仓库了。

我们发现通过gitHub管理仓库实在是太方便了，可是只能免费使用公开仓库，自已公司的代码当然不能公开了，可是私有仓库又是需要交“保护费”的，无耐国人还是比较喜欢免费的，网络界总是有很多雷峰的，比如gitLab!!!

gitLab也是一个可以通过Web界面管理仓库的网站程序，我们可以把它架设到公司自已的服务器上，实现仓库私有化，这也是大部分公司通常采用的方法，其使用方法与gitHub十分相似。

我将闲置电脑配置成了一台服务器，上面架设了gitLab程序，我们接下来的练习全部会在gitLab上进行演示。

省略很多内容.....

## 冲突解决

假如两个开发同时改到同一文件的同一段内容会发生什么事情呢？

这时就会就会产生冲突了

## 命令汇总

git config配置本地仓库

常用git config --global user.name、git config --global user.email

git config --list查看配置详情

git init 初始一个仓库，添加--bare可以初始化一个共享仓库

git clone [仓储地址] 从某个地址复制一个仓库到本地

git clone [仓储地址] --depth 获取共享仓库但是只获取最后一次分支

git status 可以查看当前仓库的状态

git add“文件” 将工作区中的文件添加到暂存区中，其中file可是一个单独的文件，也可以是一个目录、“\*”、-A

git commit -m '备注信息' 将暂存区的文件，提交到本地仓库

git log 可以查看本地仓库的提交历史

git log –graph 图形化查看日志

git log --oneline 简短显示日志

git reflog 命令查看历史操作来进行回退

git reset --hard 版本号 版本回退到版本号的位置

git branch查看分支

git branch“分支名称” 创建一个新的分支

git checkout“分支名称” 切换分支

git checkout -b“分支名称” 创建分支并且切换分支

git merge“分支名称” 合并分支

git branch -d “分支名称” 删除分支

git clone “仓库地址”获取已有仓库的副本，会默认创建master分支

git push “仓库地址” “本地分支名称:远程分支名称”将本地分支推送至远程仓库，本地仓库分支名称和远程仓库分支名称一样的情况下可以简写成一个，即git push “仓库地址” “分支名称”，如果远程仓库没有对应分支，将会自动创建

git pull“仓库地址” “本地分支名称:远程分支名称”将获取远程仓库分支并合并该分支

git remote add “主机名称” “远程仓库地址”添加远程主机，即给远程主机起个别名，方便使用

git remote rm“主机名称” 命令用于删除远程主机。

git remote 可以查看已添加的远程主机

git remote show “主机名称”可以查看远程主机的信息

ssh-keygen -t rsa 创建rsa密钥

## Git高级

熟悉掌握以上操作，基本上是可以满足日常开的需要的，但是在解决一些特殊问题时，就又需要我们能够掌握更多的命令。

### gitignore忽略文件

在项目根目录下创建一个.gitignore文件，可以将不希望提交的罗列在这个文件里，如项目的配置文件、node\_modules等

### 比较差异

当内容被修改，我们无法确定修改哪些内容时，可以通过git diff来进行差异比较。

git diff 比较的是工作区和暂存的差异

git diff “提交ID”比较与特定提交的差异

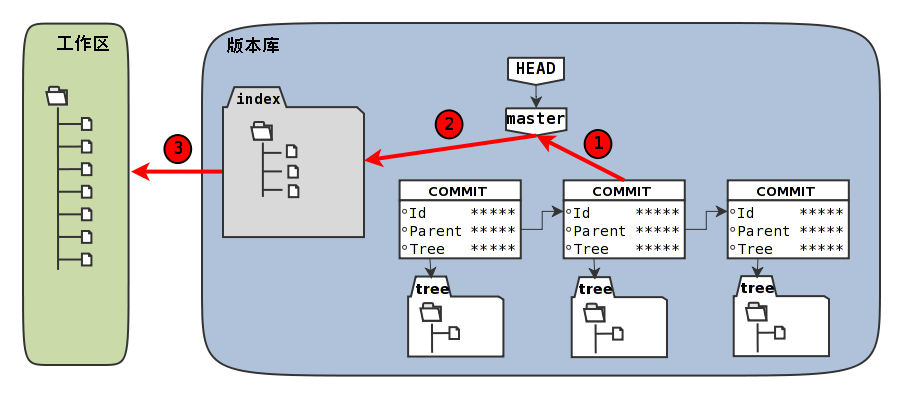
git diff 提交ID 提交ID比较某两次提交的差异

git diff 分支名称 比较与某个分支的差异

git diff --cached 比较暂存区文件与上一次提交的文件的差异

git difftool 详细比较

**2、回滚（撤销）操作**



HEAD 默认指向当前分支的“末端”，即最后的一次提交，但是我们通过git reset 可以改变HEAD的指向。

git reset

git reset --hard Head^^ 表示回到最近往前第二次的提交

git reset --hard Head~1 回退到最近一次提交的前1次提交.

### 更新仓库

在项目开发过程中，经常性的会遇到远程（共享）仓库和本地仓库不一致，我们可以通过git fetch 命令来更新本地仓库，使本地仓库和远程（共享）仓库保持一致。

git fetch “远程主机” “分支名称”

我们要注意的是，利用git fetch 获取的更新会保存在本地仓库中，但是并没有体现到我们的工作目录中，需要我们再次利用git merge来将对应的分支合并（融合）到特定分支。

## 其它

git stash

git stash pop

git push origin --delete 分支名称

git push origin :分支名称

### Git操作

git --help 帮助

git config --list 查看配置

git config --global user.name 设置用户名

git config --global user.email 设置用户邮箱

git --version 查看版本

get init 初始化

git add ./文件名 添加到缓存区

git status 提交状态

git commit -m “备注” 提交到仓库并且添加备注

get log 显示所选时间点之前的提交日志

get log --oneline 简单显示所有日志

get log --graph 图形化提交记录

git reflog 命令查看历史操作来进行回退

git reset --hard 哈希值 恢复对应时间点的版本

git branch 分支名 创建分支

git branch 查看所有分支

git checkout 分支 切换分支

git merge 分支 合并分支 如果有冲突,删除

## 三 bower

## 如何使用Bower

安装Bower:npm install -g bower

创建包bower.json:bower init

安装各种包文件:bower install < package>,如下载jquery: bower install jquery

卸载包文件:bower uninstall < package>

包搜索：bower search <package>

包版本：bower info <package>

指定的版本:bower install <package>#版本号,如下载jquery: bower install

Jquery#2.2.0

修改Bower安装位置：在项目根目录下创建.bowerrc然后添加以下内容

{

"directory": "vendor"//这里是指定包安装路径

}

## 四 gulp

### Gulp基础

Gulp是基于Nodejs开发的一个构建工具。

**1、全局安装**

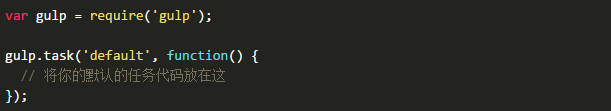
npm install -g gulp

**2、本地安装（做为项目依赖）**

在项目根目录下执行

npm install gulp --save-dev

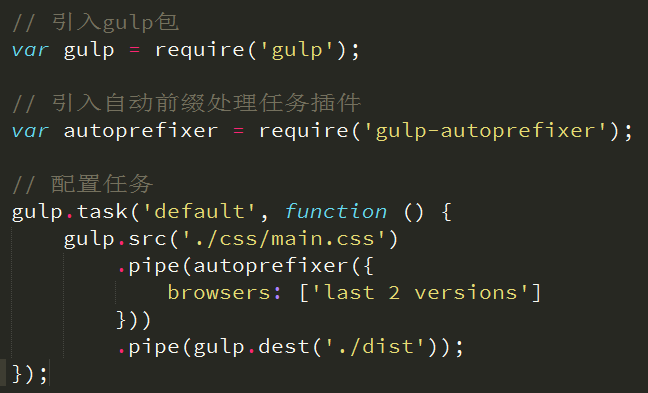
**3、创建任务清单gulpfile.js**



**4、定义任务**

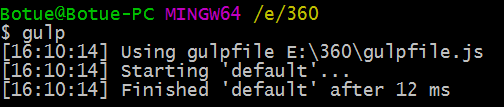
比如自动添加CSS私有前缀

本地安装gulp-autoprefixer --save-dev



**5、执行任务**

输入命令 gulp



这样我们的CSS文件便会被自动的添加了浏览器私有前缀了

### Gulp API

### 常用Gulp插件

gulp-less(编译 Less)https://www.npmjs.com/package/gulp-less

gulp-concat(合并文件)https://www.npmjs.com/package/gulp-concat

gulp-uglify(最小化 js 文件)https://www.npmjs.com/package/gulp-uglify

gulp-rename(重命名文件)https://www.npmjs.com/package/gulp-rename

gulp-minify-css(最小化 css 文件)https://www.npmjs.com/package/gulp-minify-css

gulp-minify-html(压缩html文件)https://www.npmjs.com/package/gulp-minify-html

gulp-imagemin(最小化图像)https://www.npmjs.com/package/gulp-imagemin

gulp插件库：

http://gulpjs.com/plugins/