# 2019.2.2

## Git的核心

Git有3大区域：

1. 工作区( Directory )
2. 暂存区( Index )
3. 仓库区( HEAD )

### 配置帐号

> git config --global user.name "FishC\_Service"

> git config --global user.email [fishc\_service@126.com](mailto:fishc_service@126.com)

#### 查看帐号信息

git config --list 命令可以查看信息是否写入成功

### 提交文件

第一步：Git add 文件名 // 文件放入暂存区

第二步：Git commit -m “改动注释说明” // 暂存区文件提交到仓库( commit 是“提交”的意思，-m参数是“写入注释说明” )

### 克隆github项目到本地

git clone 目标

例如： git clone

### 查看当前文件状态

Git status

### 撤销提交暂存区

git reset HEAD <file>

### 暂存区文件恢复到工作区

Git checkout 文件名

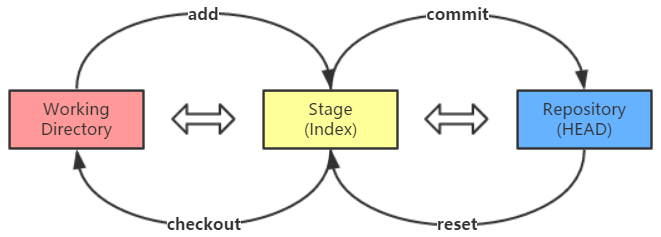
### 直接将全部修改的文件保存到仓库

git commit -am "change the license file" // 直接提交所有修改的文件到仓库

参数-a表示选中“工作区”全部文件存储到仓库

参数-m表示写入注释

## Git命令模型



git add 命令用于把工作目录的文件放入暂存区域

git commit 命令用于把暂存区域的文件提交到 Git 仓库

git reset 命令用于把 Git 仓库的文件还原到暂存区域

git checkout 命令用于把暂存区域的文件还原到工作目录

### 查看历史提交日志

Git log // 查看当前提交历史

Git reflog // 查看全部记录

## 关于git reset 恢复快照( 仓库区恢复到暂存区 )

#### Git reset可到未来和历史

Git reset ID号 文件路径 // 可恢复历史中部分文件

Git reset ID号 可以去历史与未来（ 文件将恢复到暂存区 ）

#### Git reset常用

Git reset HEAD~ // 恢复上一次提交到暂存区 ( ~为波浪线 )

Git reset HEAD~~ // 恢复上上一次提交到暂存区( 波浪线递增表示“上上…上” )

Git reset HEAD ~3 // 当然也可以使用数字代替波浪线

Git reset Hex号 //（“Hex号”可用git log获取）

例如：git reset bcab23

#### 关于git reset 参数 - -soft

Git reset –soft HEAD~ // 撤销上一次提交

#### 关于 git reset 参数 - - hard

Git reset -hard HEAD~ // 直接将指定仓库区恢复到工作区

因为默认git reset HEAD~ // 恢复到暂存区（ 原始态: git reset -mixed HEAD~ ）

# 2019.2.6

### Git diff 文件对照

1. 工作区与暂存区对比“ git diff ”
2. 仓库区对比“ git diff 仓库hex值 仓库hex值 ”
   1. 例：git diff 9d7a0e 82a3b7b

#### 关于git diff 显示小细节：

1. “ - ”表示旧文件
2. “ + ”表示新文件
3. Index 旧文件ID 新文件ID
4. @@ -1，20 +1，20@@
   1. @@开始/结束
   2. -1，20（ 旧文件从第一行显示到第20行 ）
   3. +1，20 ( 新文件从第一行显示到第20行 )

#### 关于diff颜色问题：

-红色： 表示旧文件独有

+绿色：表示新文件独有

灰色：表示二个文件共同拥有

### 修改仓库文件（提交最近一次仓库）

Git cmmit - -amend -m “提交说明”( 文件在暂存区 )

### 关于git删除文件的小细节

Git rm -f 文件名 // 删除 暂存区，工作区 文件

Git rm - -cached 文件名 // 只删除暂存区文件

如果想要删除的文件已经在仓库，只能撤销最近的提交仓库

Git reset - -soft HEAD~ // 撤销最近一次提交的仓库

### 使用git文件修改名称

Git mv 原文件名称 新文件名称 // 工作区以及暂存区文件名称将会修改

无关紧要的补充：Ren 文件名 文件名 // 为dos修改名称

### 使Git忽略文件，不进行跟踪

比如文件test.temp

Echo \*.temp > \*gitignore // \*.temp符号正则表达式规则

### Git分支使用规范



分支详细说明链接：<https://fishc.com.cn/thread-73806-1-1.html>

#### 各分支关系标准逻辑（ 按python3语法辨析逻辑关系 ）

Master: 主要分支

Hotfix: 修补BUG分支

Release: 调试分支

Develop: 成长分支

Feature1: 功能分支

Feature2: 功能分支

# 2019.2.7

## 创建分支，以及查看分支

Git branch 分支名 // 创建分支

Git checkout -b 分支名 // 快速创建分支,并进入分支

Git log - -oneline // 简洁版查看已有分支

q - -graph - - oneline // 分支图简洁版显示

参数：

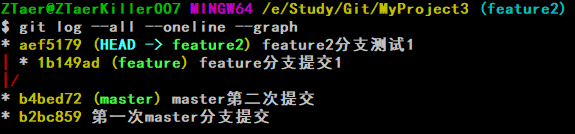
1. - -all 显示全部仓库
2. - -graph 绘制分支图
3. - -oneline 显示简洁版

#### 切换分支

Git checkout 分支名

注意：

切换分支所提交的仓库属于当前分支



#### 查看全部分支

Git log –all –oneline --graph

#### 合并分支

Git merge 分支名称

关于Fast-forward合并方式：

如果合并的分支逻辑关系与之“连接在上”，则会进行Fast-forward进行合并 // 速度很快

关于“第三方合并”方式：

合并的二个分支，逻辑关系并没有连接

强制进行第三方合并：

Git merge –no-ff 分支名称

小细节：及时删除分支，也会留下被删除分支的痕迹，而Fast-forward不会留下痕迹

#### 撤销合并分支

感受git reset 的强大：

Git reset 目标HEX 配合 git reflog // 可回到过去，可前进未来

#### 删除分支

Git branch -d 分支名 // 删除已合并分支

Git branch -D 分支名 // 删除未合并分支

# 2019.2.8

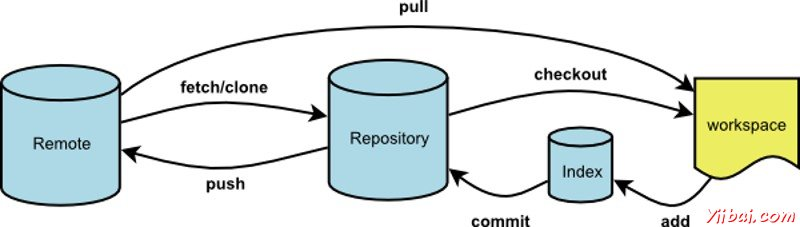
## 关于为什么用git checkout 切换分支，而不用 git reset 切换分支

Git checkout 分支// 它并不会删除快照，只是换个指针位置

而 git reset 分支 // 它会变动快照信息

# 2019.2.12

## Git远程操控



### 要关联一个远程库

git remote add origin SSH链接

关联后，使用命令

git push -u origin master // 第一次推送master分支的所有内容

此后，每次本地提交后

git push origin master // 推送最新修改；

### 强制将本地文件提交到仓库

Git push -f orgin master

# 2019.2.17

### 配置ssh到github

1. ssh-keygen -t rsa -C “[ztaerkille@gmail.com](mailto:ztaerkille@gmail.com)” // 配置创建本地sshKey，( 一路回车，默认生成里面包括id\_rsa（私密钥）、id\_rsa.pub（公密钥）文件 )
2. 复制id\_rsa.pub文件内容(一大堆看不懂的码) -> 到github 的“NEW SSH KEY”（ 标题注释主机名,能回忆即可 ）
3. Win-Id\_rsa.pub位置：C:\Users\用户名\.ssh
4. Linux-id\_rsa.pub位置: ~/.ssh
5. Ssh -T [git@github.com](mailto:git@github.com) // 测试验证是否链接成功

### Git pull（ 注意: 在pull之前先commit一下，防止出错 ）

git pull <远程主机名> <远程分支名>:<本地分支名>

Git pull origin development // development为远程分支，合并远程分支到当前分支( 防止GITHUB因文件不同出现错误 )

### Git PUSH

Git push origin 本地分支名:远程分支名 // 推送本地分支到远程分支

# 2019.4.27

### Git全屏显示快捷键:

**ALT + ENTER / SPACE**

# **2019.5.10**

### 删除git中已备份文件

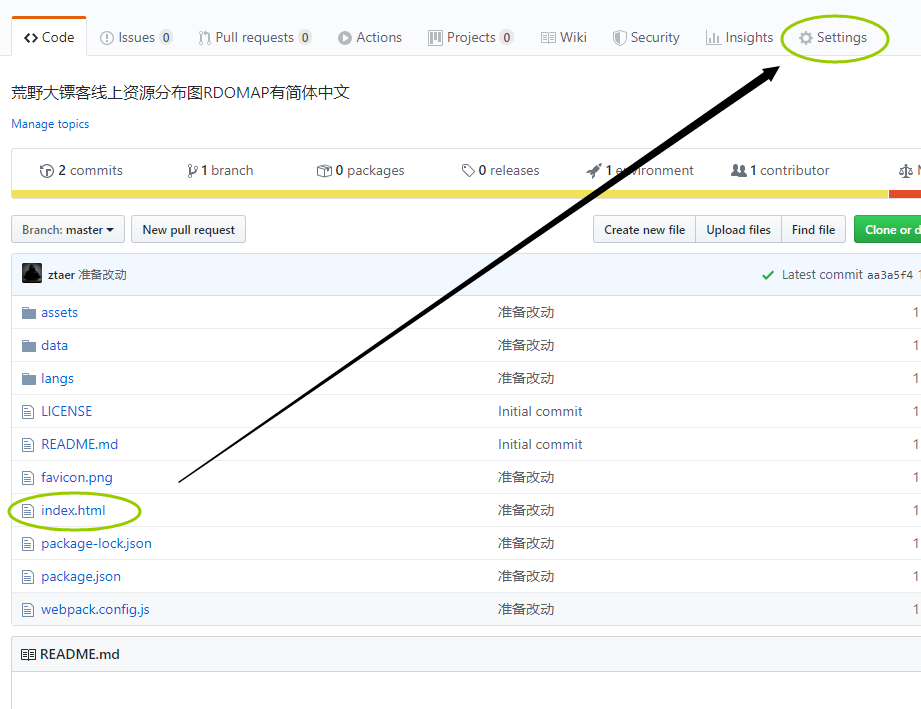
Git rm -r - -cached 文件名

# 2019.11.29( CloudFlare )

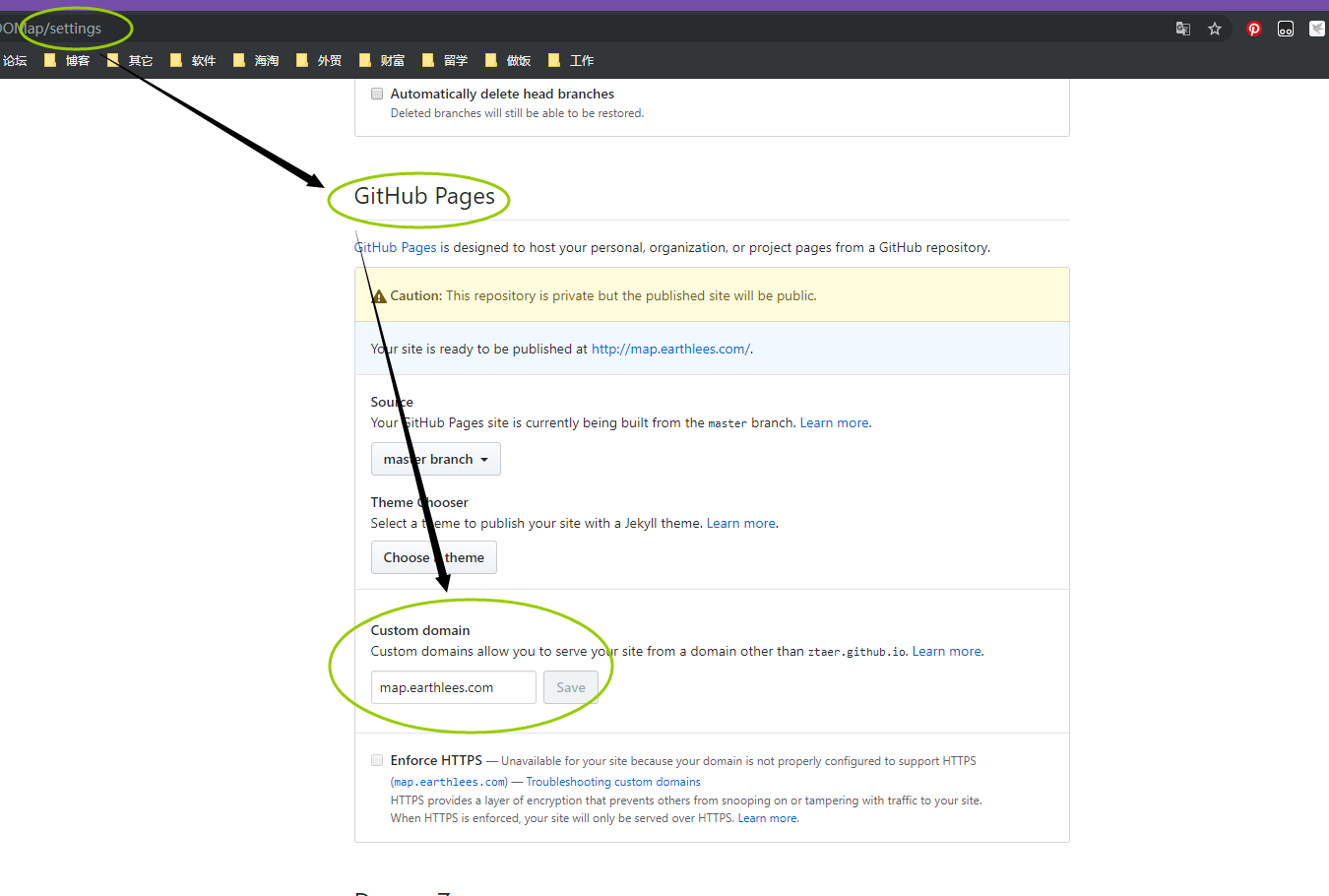
## 小型不死鸟搭建：GitHub Pages + CloudFlare + 免费的强制SSL

### 项目上传到github，开启github pages，并使用自定义域名

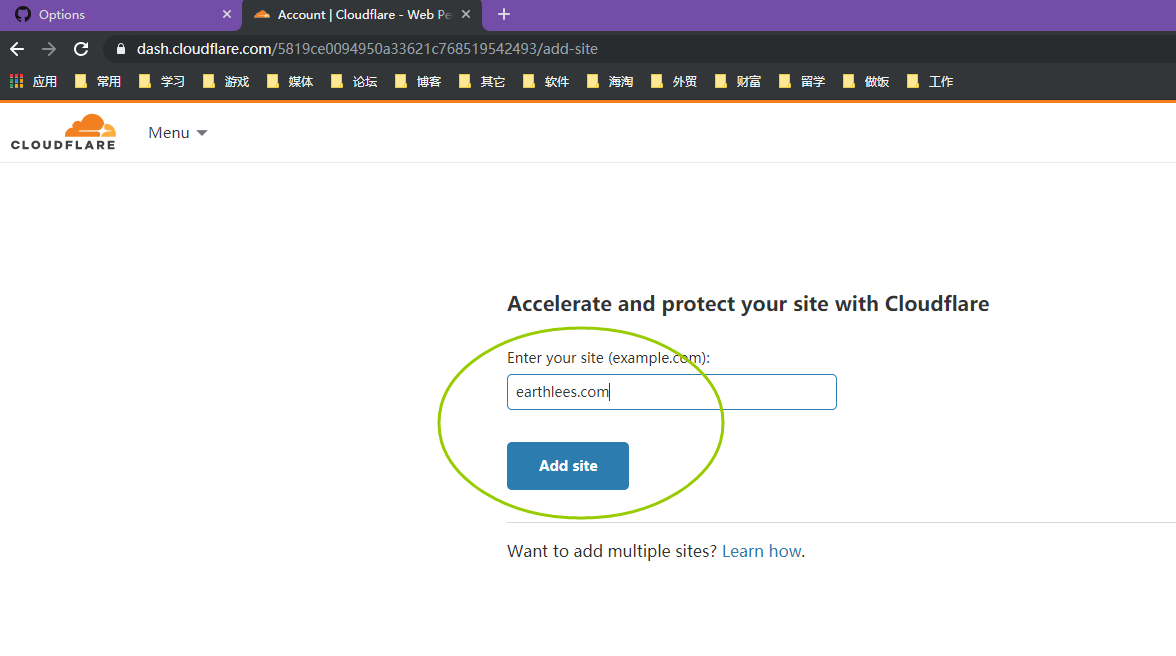
#### 保证项目根目录有index.html文件

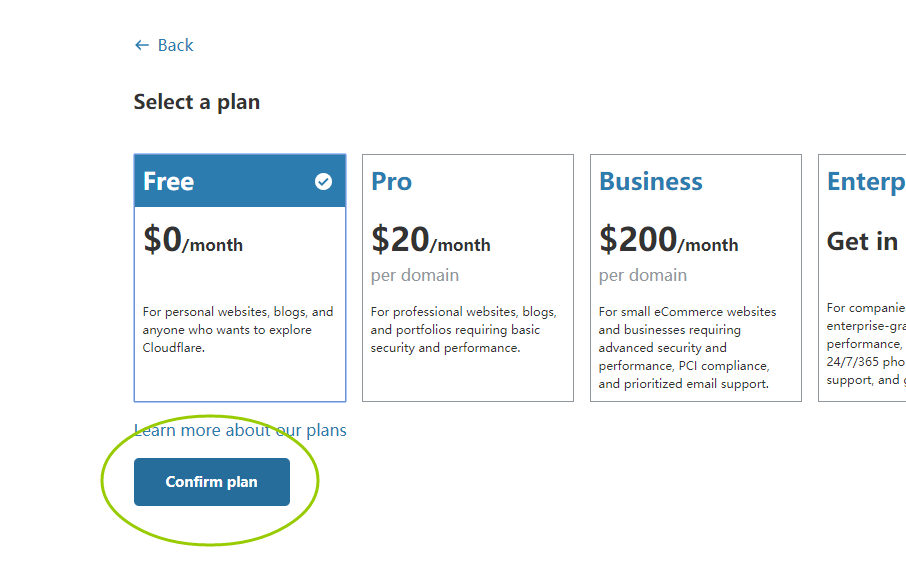


#### 在项目的‘设置’下的githubPages输入自定义域名

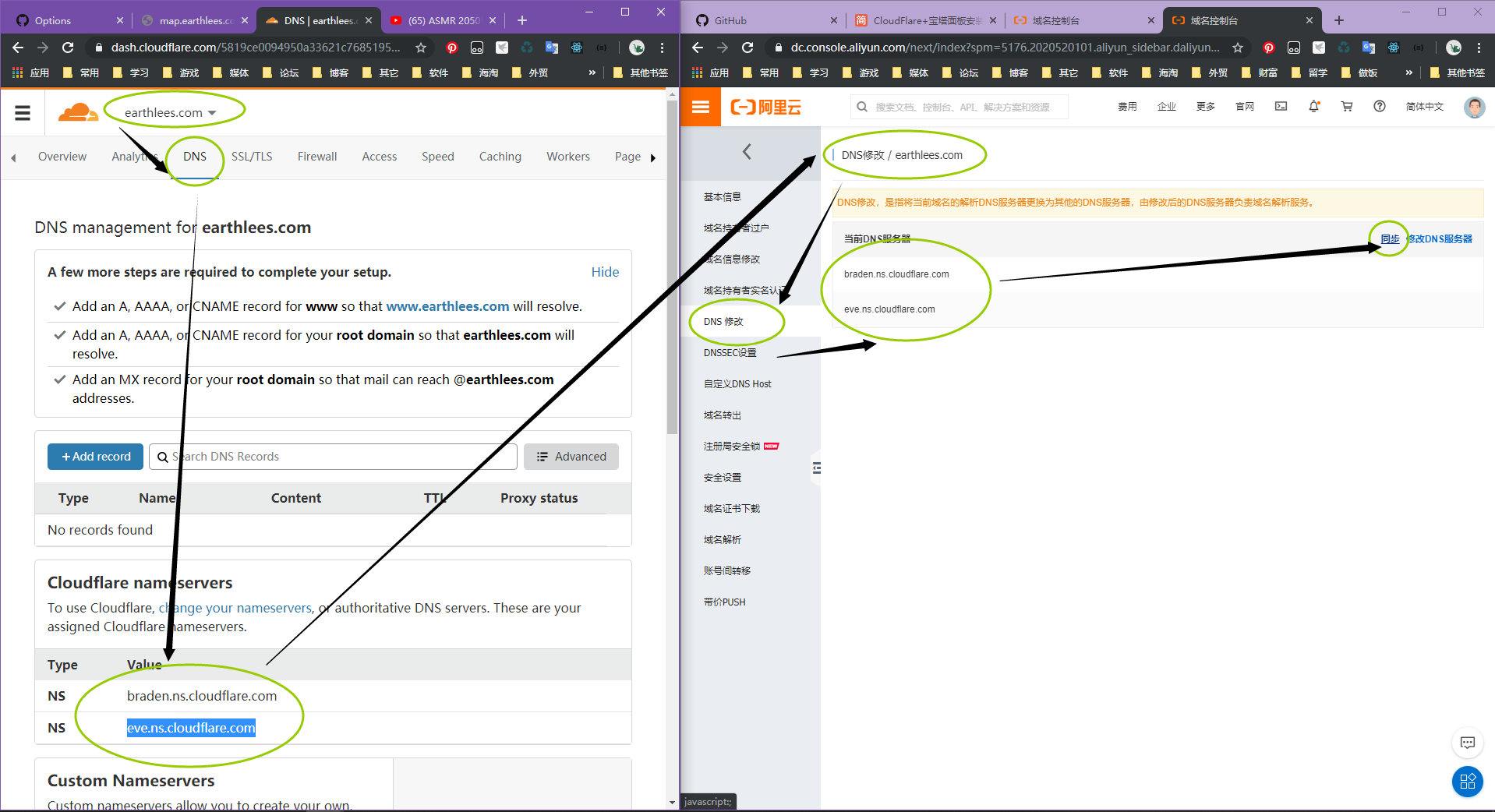


### 将域名解析到cloudflare

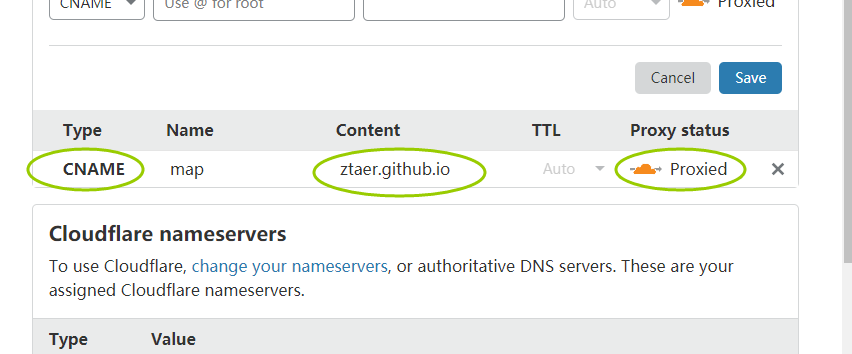




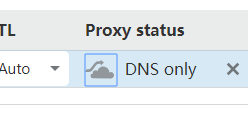
#### 将阿里的域名DNS修改成cloudfare的DNS即可交由cloudfare来管理域名



#### 设定CNAME解析到GitHub（设定完成后 – 等待一些时间才能正常访问）

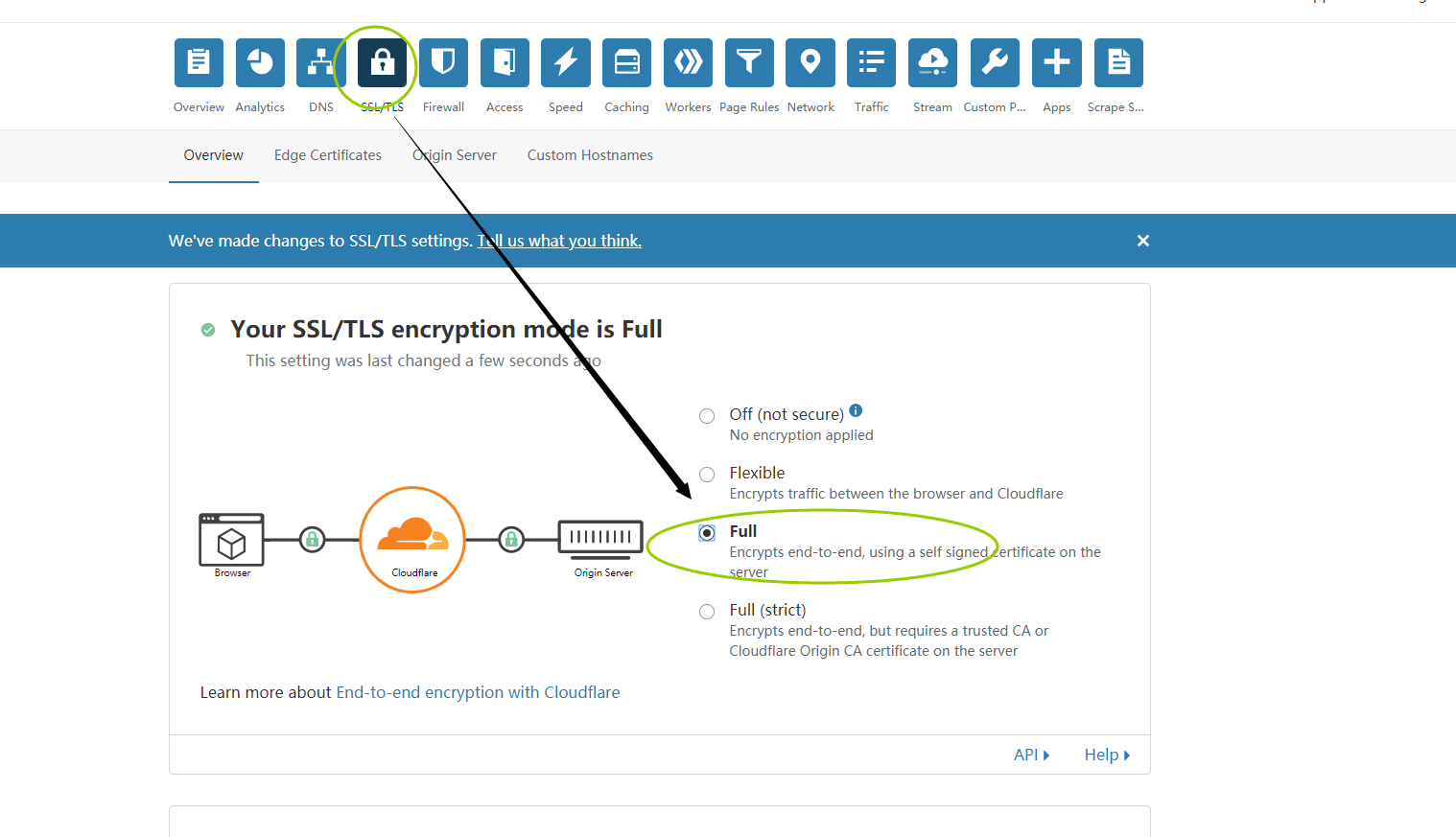


#### 查看是否开启CDN

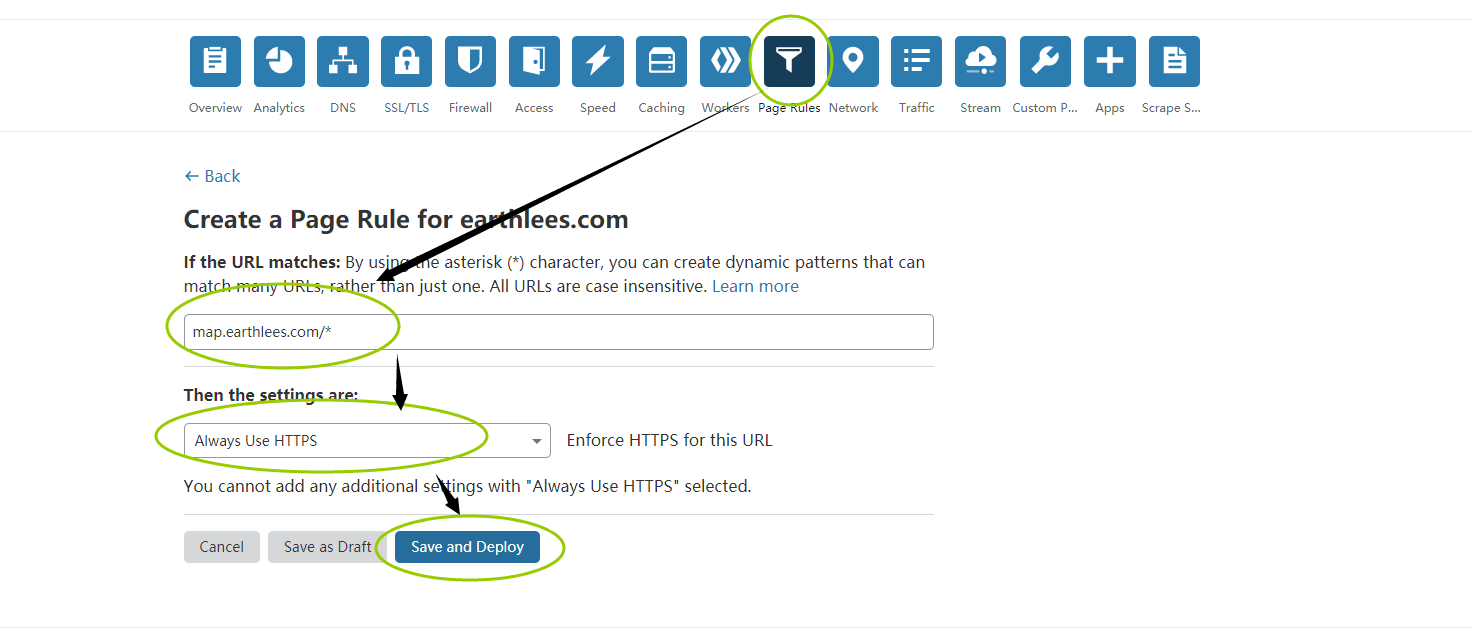
有颜色代表开启CDN加速，无颜色则代表不使用CDN加速

### 使用CouldFlare免费的SSL并且设定强制https

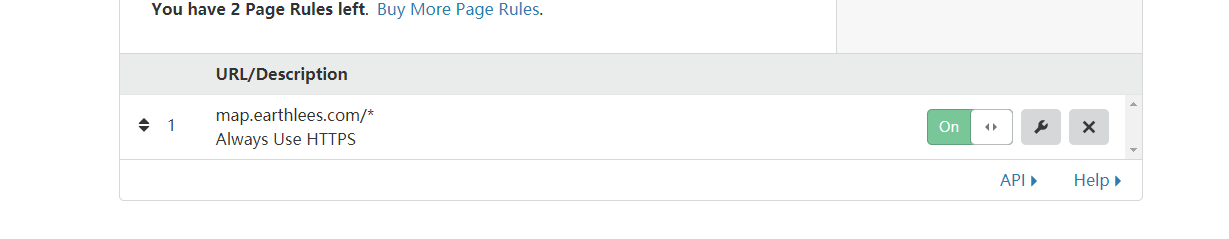
#### 设定加密模式为full



#### 开启https



#### 强制https开启成功



### CouldFlare细节 - 优化 – 免费SSL证书：

* 1. 具体参考：<https://www.jianshu.com/p/69d9c843a3e6>
  2. Speed选项：
     1. 开启压缩js/css/html
     2. 开启AMP移动设备加速
  3. Caching选项：
     1. 确认开启“永久在线”

### 创建免费SSL证书给宝塔

## cloudflare配置主域名跳转子域名/301跳转/302跳转

### 如：sosee.org跳转到map.sosee.org

#### 首先配置DNS



#### 接着配置页面规则



# 2019.12.1

指定不上传文件： .gitignore

# 2019.12.15

## Fork同步更新

1. 查看当前远程链接
   1. git remote -v
2. 给分支upstream配置源
   1. git remote add upstream 目标源
3. 更新fork内容到分支upstream
   1. git fetch upstream
4. 切换到本地主分支
   1. git checkout master
5. 将upstream合并到 master 上
   1. git merge upstream/master
6. 更新到 GitHub
   1. git push origin master

## 关于pull与fetch的区别

Pull = fetch+merge，

pull的话，下拉远程分支并与本地分支合并。

fetch只是下拉远程分支，怎么合并，可以自己再做选择。一般在需要merge配合

# 2020.5.22( 搭建V2Ray )

ROOT权限：sudo -i

查香时间：date -R

校正本地时间：cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

# 2021.01.20( 常用功能 )

## 正常开发Git流程: ( Gitlab下 – 如果是Github注意精确分支名称 )

1. git clone xxx | clone下项目
2. git checkout -b xxx | 创建分支开发
3. git checkout xxx | 切换到指定分支开发
4. git pull | 保持当前分支代码为最新
5. 正常开发……
6. git status | 查看当前分支是否为，指定开发的分支
7. git add . | 保存代码到，暂存区
8. git commit -m “ 代码备注 ” | 将代码提交至本地仓库
9. git pull | 预防万一，拉下当前开发分支最新代码
10. git push | 将代码推送到云端仓库

## 正常开发回滚Git流程:

1. git log | 查看回滚目标hex
2. git reset 目标仓库hex | 将该仓库代码回复至暂存区
3. git diff | 查看当前代码，与暂存区代码差异
4. git checkout | 指定文件回复至工作区
5. git checkout . | 全部回复至工作区