

前端必备-协同开发git

前端必备-协同开发git

1. 入门

2. gitignore 设置忽略文件

3. git 代理导致无法push远程仓库

问题描述

原因分析

解决方法

4. github 添加ssh

问题描述:

5. git 规范提交 Husky规范

网上的推荐方式是

我推荐的方式是

提交消息格式

修改类型type

Git 常用命令速查表

master :默认开发分支 Head :默认开发分支
origin :默认远程版本库 Head^ :Head 的父提交

创建版本库

```
$ git clone <url>            #克隆远程版本库  
$ git init                    #初始化本地版本库
```

修改和提交

```
$ git status                #查看状态  
$ git diff                  #查看变更内容  
$ git add .                 #跟踪所有改动过的文件  
$ git add <file>            #跟踪指定的文件  
$ git mv <old> <new>        #文件改名  
$ git rm <file>             #删除文件  
$ git rm --cached <file>    #停止跟踪文件但不删除  
$ git commit -m "commit message"    #提交所有更新过的文件  
$ git commit --amend        #修改最后一次提交
```

查看提交历史

```
$ git log                    #查看提交历史  
$ git log -p <file>         #查看指定文件的提交历史  
$ git blame <file>          #以列表方式查看指定文件的提交历史
```

撤销

```
$ git reset --hard HEAD      #撤销工作目录中所有未提交文件的修改内容  
$ git checkout HEAD <file>    #撤销指定的未提交文件的修改内容  
$ git revert <commit>        #撤销指定的提交
```

分支与标签

```
$ git branch                #显示所有本地分支  
$ git checkout <branch/tag>    #切换到指定分支或标签  
$ git branch <new-branch>    #创建新分支  
$ git branch -d <branch>      #删除本地分支  
$ git tag                    #列出所有本地标签  
$ git tag <tagname>          #基于最新提交创建标签  
$ git tag -d <tagname>        #删除标签
```

合并与衍合

```
$ git merge <branch>        #合并指定分支到当前分支  
$ git rebase <branch>        #衍合指定分支到当前分支
```

远程操作

```
$ git remote -v              #查看远程版本库信息  
$ git remote show <remote>    #查看指定远程版本库信息  
$ git remote add <remote> <url>    #添加远程版本库  
$ git fetch <remote>          #从远程库获取代码  
$ git pull <remote> <branch>    #下载代码及快速合并  
$ git push <remote> <branch>    #上传代码及快速合并  
$ git push <remote> :<branch/tag-name>    #删除远程分支或标签  
$ git push --tags            #上传所有标签
```

Git Cheat Sheet <CN> (Version 0.1) # 2012/10/26 -- by @riku < riku@gitcafe.com / http://riku.wowubuntu.com >

CSDN @搜捕马了

这个有什么好处呢？学了这个，你把写的所有代码上传到github上，加上vscode这样子你到任何地方都能编码学习。就算你不想凡在远端，在本地也有好处，就是版本管理，你昨天写的提交了，你今天改了提交了，你又觉得还是昨天的好，那么你可以回退到昨天的内容。这就是版本管理。

和他人协同开发的乐趣妙不可言！！

1. 入门

git学习可以先vscode的可视化操作学起走，知道要做的过程，再去学命令操作。最后是理解更深层的原理。 一句话先用再学原理

来学吧！

[【GeekHour】一小时Git教程](#)

2.gitignore 设置忽略文件

一、设置ignore文件

有些时候我们创建了一个项目，但是项目中有些文件不想被Git跟踪、提交。例如maven项目中的target目录，日志目录，idea 或者 eclipse 在加载项目后自动生成的一些本地化文件或者目录等。怎么办呢？这就需要我们为项目设置ignore文件。

步骤：

1. 在项目根目录下创建 .gitignore 文件，一定要是根目录下；
2. 编辑 .gitignore 文件，按照如下规则过滤需要忽略的文件或者文件夹

```
1  # 注释 - 以井号(#)开头的行为注释
2  # 忽略单个文件
3  filename.txt
4  # 忽略文件类型（例如所有的txt文件）
5  *.txt
6  # 忽略目录（例如一个名为"logs"的目录）
7  /logs/
8  # 忽略特定目录下的所有文件和子目录（例如一个名为"temp"的目录）
9  /temp/*
10 # 忽略特定目录及其子目录中的特定文件（例如忽略"logs"目录下的所有.log文件）
11
```

- 1 3. 保存并关闭 .gitignore 文件；
- 2 4. 将.gitignore文件添加到Git仓库中并提交更改

```
1 git add .gitignore
2 git commit -m "Add .gitignore file, xxxx"
```

从此以后，Git将忽略.gitignore文件中指定的文件和目录，并且它们不会出现在git status、git add和git commit等命令的结果中。

3.git 代理导致无法push远程仓库

问题描述

从本地提交代码到 GitHub 远程仓库，由于 DNS 污染的问题，国内提交速度很慢，有时候还报错。笔者自己花钱买了一个梯子，但开启梯子的代理后仍然没有解决问题，不过 Google 等倒是可以访问了。

原因分析

虽然开启了代理，但可能 git push 并没有走代理，因为需要在 git 里面进行配置。

解决方法

方法一：

配置 git push 直接走[网络代理](#)：

```
1 #全局代理
2 git config --global http.proxy socks5://127.0.0.1:7890
3 git config --global https.proxy socks5://127.0.0.1:7890
4 12
```

其中 1080 是 SOCKS 代理的端口，一般默认 1080，可以在代理工具的设置中查看

方法二：

```
1 #只对github代理
2 #使用socks5代理（推荐）
3 git config --global http.https://github.com.proxy socks5://127.0.0.1:51837
4 #使用http代理（不推荐）
5 git config --global http.https://github.com.proxy http://127.0.0.1:58591
```

取消代理

当你不需要使用代理时，可以取消之前设置的代理。

```
1 git config --global --unset http.proxy git config --global --unset
https.proxy
```

4.github 添加ssh

问题描述：

copy一些私人的项目，需要输密码，不想每次都输密码，就配置ssh；当有虚拟机或者服务器的时候，我们需要远程访问需要密码，配置ssh可以免密登录

在github上添加SSH key[看这里](#)

5.git 规范提交 [Husky](#)规范

为什么要规范提交，每个人按照自己想法写的太难看懂了，规范就是给他一个类型，让我们知道它是个啥

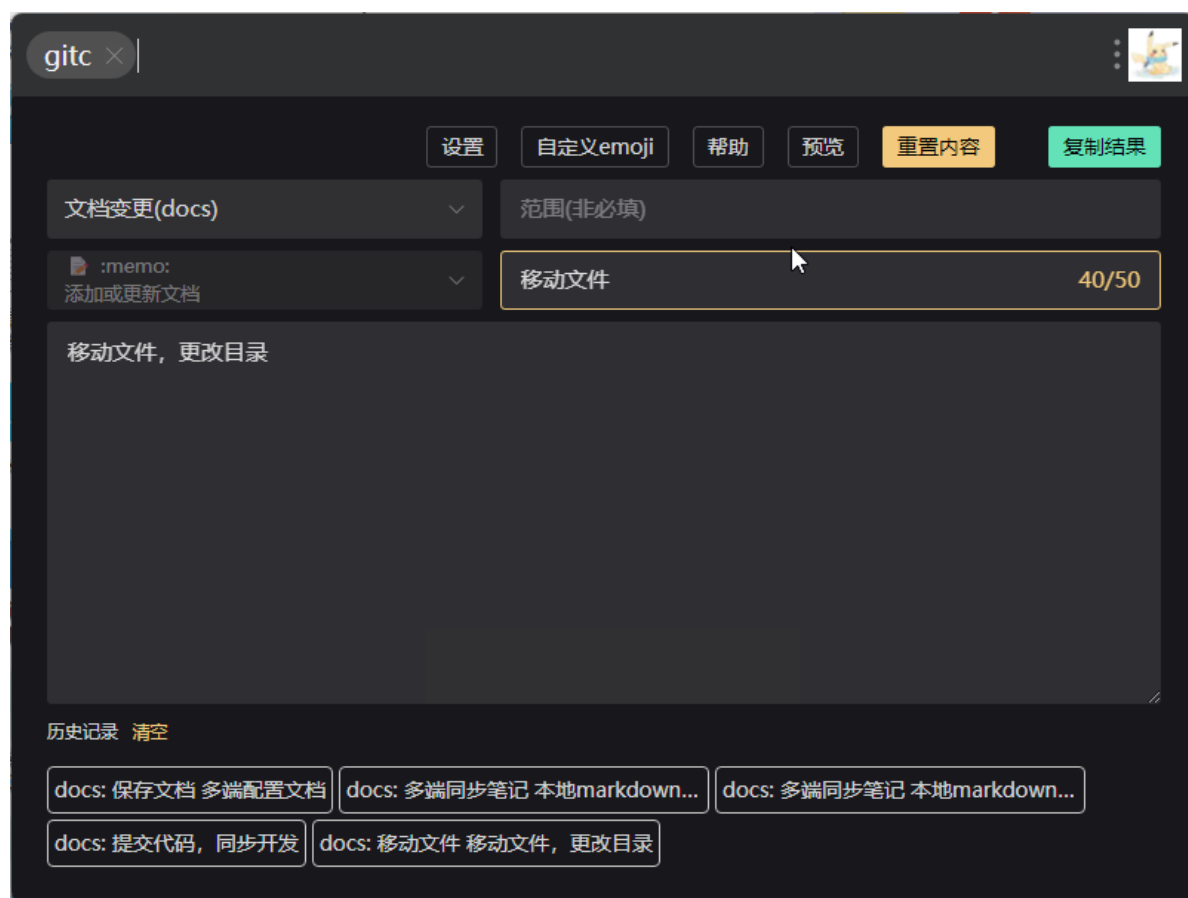
网上的推荐方式是

[git commit 最佳实践](#), [commitizen + husky + commitlint 规范化校验](#)

[Webpack githooks husky系统精讲](#) 这个是husky的原理，讲的很棒

我推荐的方式是

下载uools，安装gitc插件，==commitlint +husky进行代码提交前的检查



提交消息格式

每个提交消息都由一个标题、一个正文和一个页脚组成。而标题又具有特殊格式，包括修改类型、影响范围和内容主题：

```
1 | 修改类型(影响范围)： 标题
2 | <--空行-->
3 | [正文]
4 | <--空行-->
5 | [页脚]
```

标题是**强制性的**，但标题的**范围是可选的**。

提交消息的任何一行都不能超过100个字符！这是为了让消息在GitHub以及各种Git工具中都更容易阅读。

修改类型type

每个类型值都表示了不同的含义，类型值必须是以下的其中一个：

- **feat**：提交新功能
- **fix**：修复了bug
- **docs**：只修改了文档
- **style**：调整代码格式，未修改代码逻辑（比如修改空格、格式化、缺少分号等）
- **refactor**：代码重构，既没修复bug也没有添加新功能
- **perf**：性能优化，提高性能的代码更改
- **test**：添加或修改代码测试
- **chore**：对构建流程或辅助工具和依赖库（如文档生成等）的更改

```
1 | <type>(<scope>)： <subject>
2 | // 注意冒号 ： 后有空格
3 | // 如 feat(miniprogram)：增加了小程序模板消息相关功能
```