

简明 Git 使用教程

本教程主要讲解 Git 基础用法，使用一个小例子演示 Git 的基础功能，包括：

1. 代码保存至本地仓库
2. 本地代码回退版本
3. 本地代码和远程仓库保持同步

1. Git 安装

安装步骤

- **Windows:**
 - i. 访问 [Git 官网](#)。
 - ii. 下载并安装。
 - iii. 验证安装： `git --version`。
- **macOS:**
 - i. 使用 Homebrew： `brew install git`。或下载安装包。
 - ii. 验证安装： `git --version`。
- **Linux (以 Ubuntu 为例):**
 - i. 安装命令： `sudo apt-get install git`。
 - ii. 验证安装： `git --version`。

2. 常用 Git 命令

基础命令

- 初始化仓库： `git init`
- 克隆仓库： `git clone [url]`
- 添加文件： `git add [file]`
- 提交更改： `git commit -m "[commit message]"`

分支管理

- 查看分支： `git branch`
- 创建分支： `git branch [new-branch]`

- 切换分支: `git checkout [branch-name]`
- 合并分支: `git merge [branch]`

远程仓库操作

- 推送更改: `git push [remote] [branch]`
- 拉取更新: `git pull [remote]`

3. 准备工作

Git 设置

```
git config --global user.name "jczhao1022" # 换成自己的
git config --global user.email "jczhao1022@yeah.net"
# 换成自己的
```

SSH Key 安装

SSH (Secure Shell Protocol) 的作用主要是用于安全地访问远程仓库。具体来说, SSH 的主要作用包括:

- 安全认证: SSH 提供了一种安全的方式来验证用户身份, 确保只有授权的用户能够访问远程仓库。
- 加密通信: SSH 对所有传输的数据进行加密, 保护数据在网络上的传输过程中不被截取或篡改。
- 简化操作: 使用 SSH 连接后, 用户可以无需每次访问远程仓库时都输入用户名和密码, 从而简化了操作流程。
- 多仓库管理: SSH 允许用户创建和使用多个 SSH 密钥, 方便在不同的项目或多个服务提供商之间管理和切换。

在 Git 中, 最常用的协议是 HTTPS 和 SSH 协议。两者都是为了数据传输安全, SSH 密钥是为了节省输入用户名与密码的过程, 又要同时保证传输安全, 不是必须设置, 但设置之后对 pull、push 代码可以不用输用户名密码。

创建 SSH Key

终端输入 (Windows: win+R 然后输入 cmd)

```
ssh-keygen -t rsa -b 4096 -C "jczhao1022@yeah.net"
```

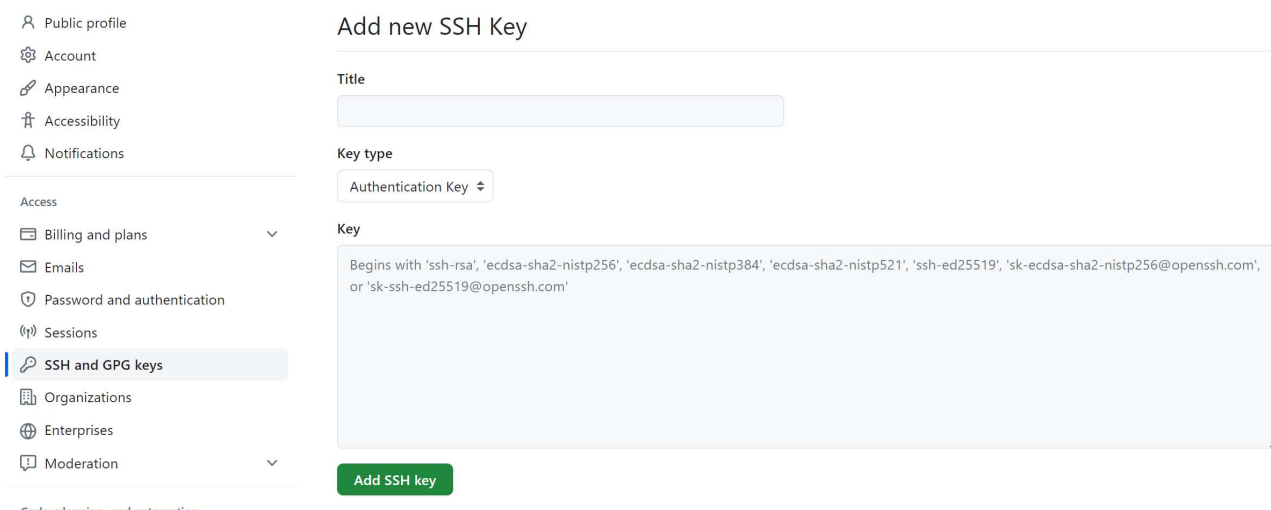
- 按照提示完成 key 的创建。(一直按回车)

将 SSH Key 添加到 GitHub

首先找到本地的 SSH 公钥，我的电脑上的地址是
C:/Users/JCZhao/.ssh/id_rsa.pub

- 将 ~/.ssh/id_rsa.pub 文件中的内容添加到 GitHub 账户的 SSH keys 部分。

1. 登录 Github
2. 点击右上角头像，选择 SSH key
3. 选择 New SSH
4. 自定义 title，把 ~/.ssh/id_rsa.pub 中的内容复制进去



4. 演示项目

项目简介

- 使用一个简单的 Markdown 文件项目。

基本操作

新建文件夹，在文件夹中打开 Git 命令行 (Windows 中右键，选择 Git bash here)

```
git init # 这里初始化另一个 Git 仓库，是后续工作的基础
```

```
16345@Qinsy MINGW64 /d/gitwork
$ git init
Initialized empty Git repository in D:/gitWork/.git/
```

在文件夹中新建 qinsytest.txt 文件或任意文件

```
git add qinsytest.txt
#可以理解为把 qinsytest.txt 上传至“缓冲区”
#可以使用 git add . 来把当前目录下所有文件都上传至“缓冲区”
git commit -m "qinsytest.txt"
# 确认更改并上传，双引号的中的内容建议详细填写，这会方便后续版本控制(后面有对比)
```



```
16345@QinSy MINGW64 /d/gitWork (main)
$ git add qinsytest.txt

16345@QinSy MINGW64 /d/gitWork (main)
$ git commit -m qinsytest.txt
[main (root-commit) 08fb985] qinsytest.txt
1 file changed, 1 insertion(+)
create mode 100644 qinsytest.txt
```

```
# git commit -m "qinsytest.txt" 有两个作用
1.是将之前的“缓冲区”文件都推送到本地厂库
2. "qinsytest.txt"等于是注释
# 至此在 master/main 分支上添加了 qinsytest.txt 文件
```

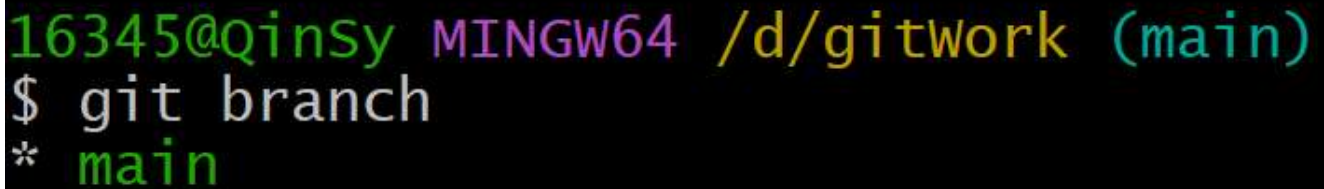
在 Git 中，分支的作用非常关键，它们允许多个开发者在不同的环境中同时工作，而不会干扰彼此。以下是 Git 分支的一些主要作用：

- 并行开发：分支允许团队成员同时进行不同的任务，比如新功能开发、错误修复或实验性探索，而不会影响到主代码库（通常是 master 或 main 分支）。
- 风险降低：通过在分支上进行开发和测试，可以降低直接对主分支所做更改带来的风险。这有助于保持主分支的稳定性和代码质量。
- 简化代码审查和合并：分支使得代码审查变得更加简单，因为可以专注于特定的更改。此外，通过合并请求（或合并命令），可以方便地将分支的更改合并回主分支。
- 版本控制和回滚：如果在分支上的更改出现问题，可以轻松地切换回先前的版本，或者完全放弃该分支的更改，而不会影响主分支。
- 实验和快速迭代：分支为实验性更改和快速迭代提供了空间，开发者可以在不影响主线开发的情况下自由探索新想法。

总的来说，分支是 Git 中实现有效、高效协作和代码管理的重要工具。

```
git branch
```

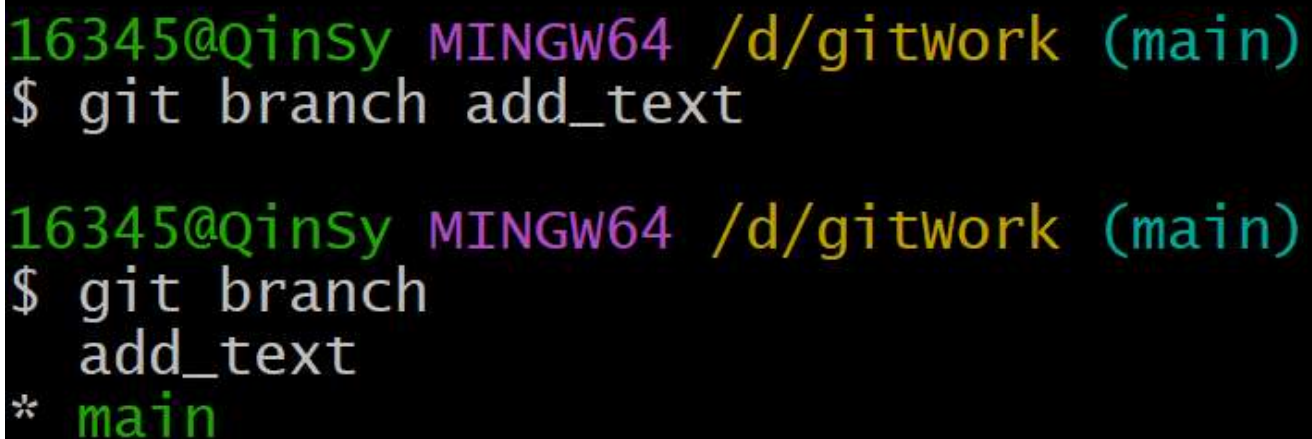
```
#先查看当前分支情况 只有一个main分支
```

A terminal window with a black background and colorful text. The prompt is '16345@Qinsy MINGW64 /d/gitWork (main)'. The command '\$ git branch' is entered, and the output is '* main', where 'main' is highlighted in green.

```
16345@Qinsy MINGW64 /d/gitWork (main)
$ git branch
* main
```

```
git branch add_text
```

```
# 新建分支 add_text , 可以修改分支名, 可以看到多了一个add_text分支
```

A terminal window with a black background and colorful text. The prompt is '16345@Qinsy MINGW64 /d/gitWork (main)'. The command '\$ git branch add_text' is entered. Below this, the prompt is repeated, followed by '\$ git branch', and the output is 'add_text' and '* main', where 'main' is highlighted in green.

```
16345@Qinsy MINGW64 /d/gitWork (main)
$ git branch add_text

16345@Qinsy MINGW64 /d/gitWork (main)
$ git branch
  add_text
* main
```

```
git checkout add_text
```

```
# 切换分支到 add_text 上
```

```
16345@QinSy MINGW64 /d/gitWork (main)
$ git checkout add_text
Switched to branch 'add_text'

16345@QinSy MINGW64 /d/gitWork (add_text)
$ git branch
* add_text
  main
```

在文件夹里添加一个 txt 文件。注意，这个改动是在 add_text 分支上的。假设新建的文件是 新分支添加测试.txt

```
git add 新分支添加测试.txt # 可以是中文名
git commit -m "在新分支上添加了 新分支添加测试.txt"
# 在 add_text 分支上添加了新文件
```

```
16345@QinSy MINGW64 /d/gitWork (add_text)
$ git add 新分支添加测试.txt

16345@QinSy MINGW64 /d/gitWork (add_text)
$ git commit -m "在新分支上添加了 新分支添加测试.txt"
[add_text e5553a1] 在新分支上添加了 新分支添加测试.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 "\\346\\226\\260\\345\\210\\206\\346\\224\\257\\346\\267\\273\\345\\212\\240\\346\\265\\213\\350\\257\\225.txt"
```

至此我们已经完成操作有：

1. 初始化 Git 仓库
2. 在 master（主分支）上上传了 qinsytest.txt
3. 在 add_text 分支上上传了 新分支添加测试.txt

这样做的目的有：

1. 熟悉将文件添加到 Git 仓库的命令：add 和 commit
2. 切换分支的命令：branch 和 checkout

回退版本

在 Git 中，回退版本的功能是为了恢复到先前的代码状态，这在多种情况下非常有用：

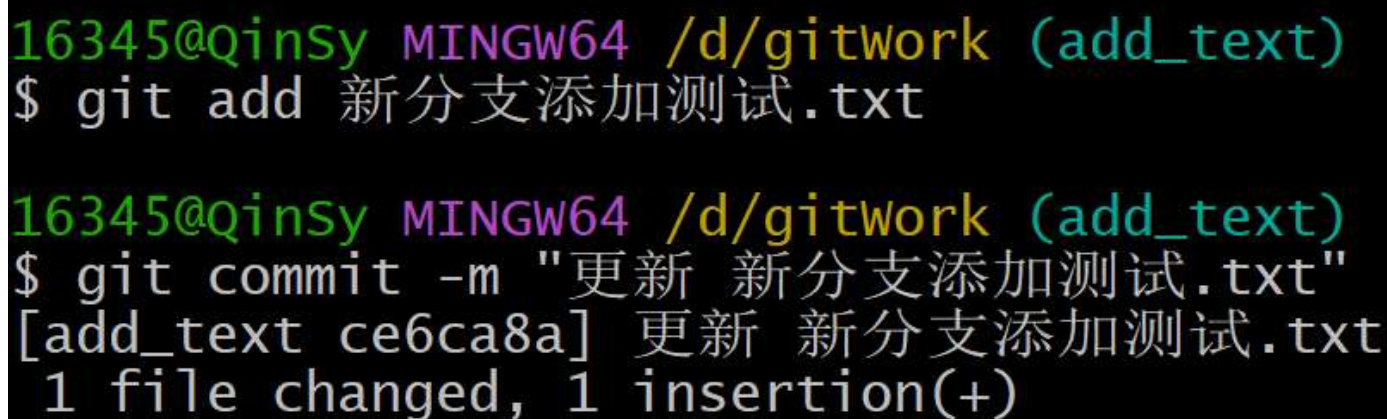
- 撤销错误：如果最近的提交包含了错误或不希望的更改，回退可以快速恢复到之前正确的状态。
- 代码审查和测试：在代码审查或测试过程中，如果发现新引入的更改不适合或有问题，可以回退到更稳定或更合适的版本。
- 实验和探索：在尝试新功能或进行实验性开发时，如果实验结果不满意，可以方便地回退到实验前的状态。
- 避免影响主分支：在开发过程中，回退可以帮助保持主分支的稳定性，特别是在发现合并错误或不当更改时。

通过回退版本，Git 提供了一种灵活的方式来管理代码和减少错误的长期影响。

```
git log # 查看历史信息
```

此时，分支被切换为 `add_text`，此时对文件做的所有修改都被看作是在 `add_text` 上的。
对 新分支添加测试.txt 做修改，比如添加一行文本。

```
git add 新分支添加测试.txt
git commit -m "更新 新分支添加测试.txt"
# 新分支添加测试.txt 将上传至仓库
```



```
16345@QinSy MINGW64 /d/gitwork (add_text)
$ git add 新分支添加测试.txt

16345@QinSy MINGW64 /d/gitwork (add_text)
$ git commit -m "更新 新分支添加测试.txt"
[add_text ce6ca8a] 更新 新分支添加测试.txt
1 file changed, 1 insertion(+)
```

```
git log # 查看历史信息
```

```
#这里也可以看到为什么要好好写 git commit -m "注释" 双引号中的内容了，因为方便以后查看日志log（相当于注释）
```

```
16345@QinSy MINGW64 /d/gitwork (add_text)
$ git log
commit ce6ca8afff135ad9154ab19941eb790ae5128403 (HEAD -> add_text)
Author: qinsy <qinsy131212@gmail.com>
Date: Wed Nov 22 22:32:13 2023 +0800

    更新 新分支添加测试.txt

commit e5553a1148e11e46482efe22b165ea8598c67780 (origin/main, main)
Author: qinsy <qinsy131212@gmail.com>
Date: Wed Nov 22 19:53:31 2023 +0800

    在新分支上添加了 新分支添加测试.txt

commit 08fb985219450d7d97bc4b693f5eba34ef94bc72
Author: qinsy <qinsy131212@gmail.com>
Date: Wed Nov 22 19:40:57 2023 +0800

    qinsytest.txt
```

`git reset --hard commit ID`

[commit ID] 部分替换为“在 new-branch 上新建了文本文件”这一步的版本号

```
16345@QinSy MINGW64 /d/gitwork (add_text)
$ git log
commit ce6ca8afff135ad9154ab19941eb790ae5128403 (HEAD -> add_text)
Author: qinsy <qinsy131212@gmail.com>
Date: Wed Nov 22 22:32:13 2023 +0800

    更新 新分支添加测试.txt

commit e5553a1148e11e46482efe22b165ea8598c67780 (origin/main, main)
Author: qinsy <qinsy131212@gmail.com>
Date: Wed Nov 22 19:53:31 2023 +0800

    在新分支上添加了 新分支添加测试.txt

commit 08fb985219450d7d97bc4b693f5eba34ef94bc72
Author: qinsy <qinsy131212@gmail.com>
Date: Wed Nov 22 19:40:57 2023 +0800

    qinsytest.txt

16345@QinSy MINGW64 /d/gitwork (add_text)
$ git reset --hard e5553a1148e11e46482efe22b165ea8598c67780
HEAD is now at e5553a1 在新分支上添加了 新分支添加测试.txt

16345@QinSy MINGW64 /d/gitwork (add_text)
$
```


- # 版本号就是 `git log` 命令显示出 `commit` 字段后面跟的一串文本
- # 这是最简单的回退方式，但一般是不可撤销的
- # 相对复杂的回退方式可以学习参考资料中的内容

此时会发现 新分支添加测试.txt 回退到了新建它的时候。

合并分支

通常情况下，使用 Git 做版本控制时需要先新建分支，在分支上制作新版本之后再合并到主分支上。

- `git checkout master` # 切换回主分支
- `git merge add_text` # 把 `add_text` 合并进主分支
- # 解决冲突（如果有的话）

```
16345@QinSy MINGW64 /d/gitwork (add_text)
$ git checkout main
Switched to branch 'main'

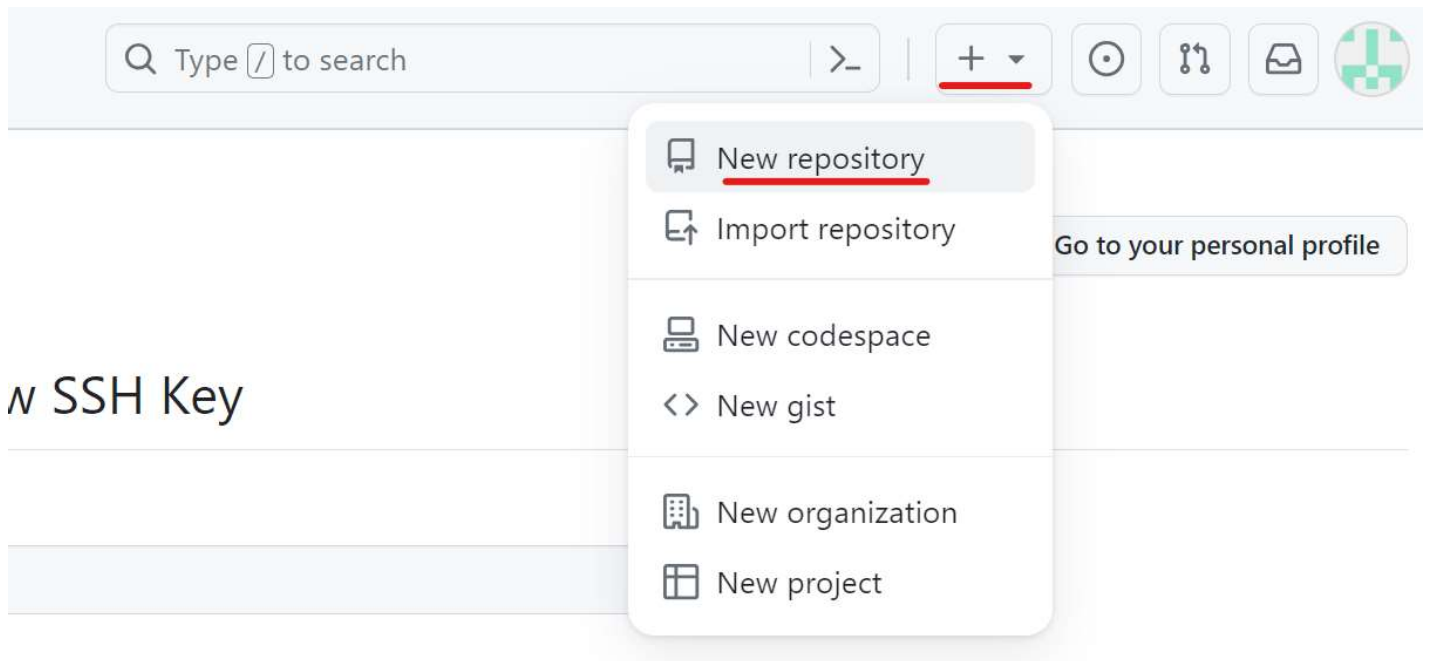
16345@QinSy MINGW64 /d/gitwork (main)
$ git merge add_text
Updating 08fb985..e5553a1
Fast-forward
...0\206\346\224\257\346\267\273\345\212\240\346\265\213\350\257\225.txt" | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 "...346\226\260\345\210\206\346\224\257\346\267\273\345\212\240\346\265\213\350\257\225.txt"
```

此时，我们将 `add_text` 上的更改合并到主分支上了。

设置远程仓库

使用 Git 的最显著优势就是可以利用远程仓库保管我们的代码。

先在 Github 上新建一个仓库，然后查看其 URL，比如 `git@github.com:QinSy77/test.git`



```
git remote add origin git@github.com:QinSy77/test.git
# 链接到远程仓库
git push -u origin master
# 如果主分支叫 main 就修改为 git push -u origin main
```

```
16345@QinSy MINGW64 /d/gitwork (main)
$ git remote add origin git@github.com:QinSy77/test.git

16345@QinSy MINGW64 /d/gitwork (main)
$ git push -u origin main
The authenticity of host 'github.com (20.205.243.166)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (6/6), 554 bytes | 554.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:QinSy77/test.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

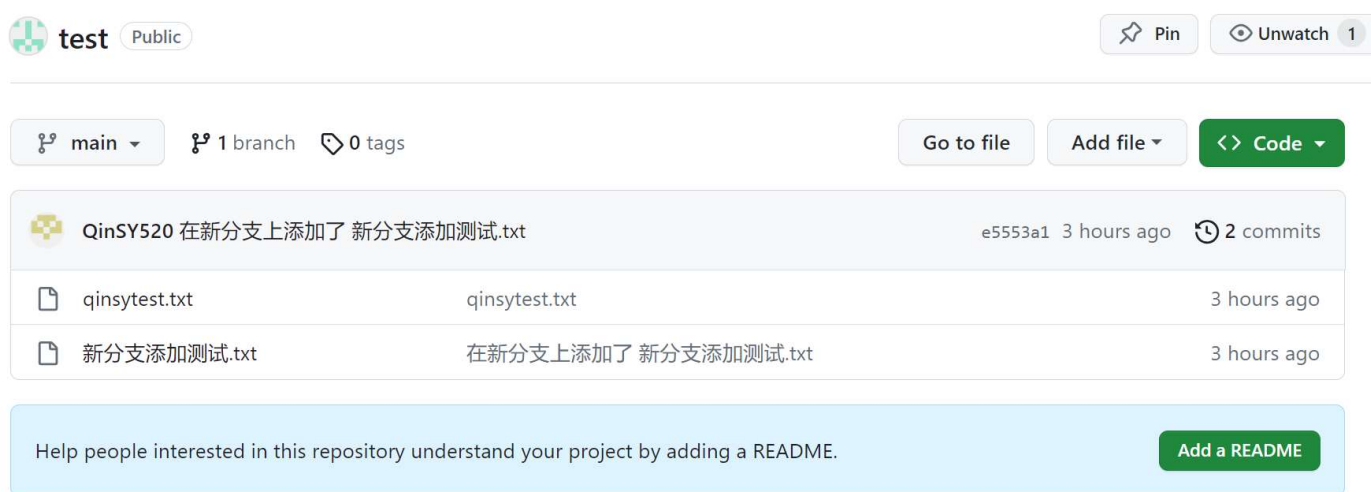
从远程仓库拉取和推送

对本地文件做更改，然后将其全部上传至仓库 (使用之前讲过的 add 和 commit 命令)

```
# 本地更改推送远程
git push origin master
```

```
16345@QinSy MINGW64 /d/gitwork (main)
$ git remote add origin git@github.com:QinSy77/test.git

16345@QinSy MINGW64 /d/gitwork (main)
$ git push origin main
Everything up-to-date
```



此时刷新 Github 页面，就会发现已经同步了。

之后，使用网页版的 Github 对现有的仓库做一些改动，比如上传一些新文件。然后在命令行输入

```
# 远程更改推送本地
git pull origin master
```

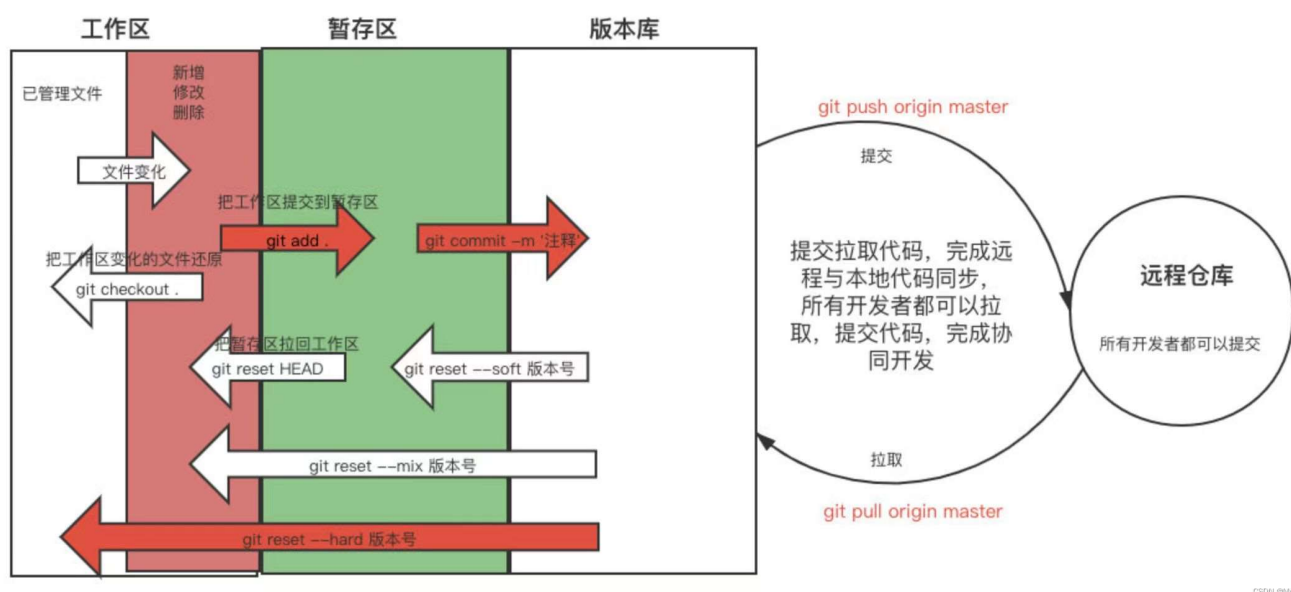
此时会发现本地文件夹和远程仓库保持一致了。

5. 总结

我们学习了：

1. Git 安装，Git 初始设置，生成并添加 SSH key

2. 新建本地 Git 仓库，学习了添加文件到本地仓库的命令 `add` 和 `commit`
3. 新建(`branch`)、切换(`checkout`)、合并(`merge`)分支
4. 回退版本： `log` 查看版本， `reset --hard [版本号]` 回退版本
5. 本地仓库和远程仓库保持同步： `push` 和 `pull`



6. 参考资料

Git 还有更加丰富的内容，比如多端同步，多人协作，相关内容可以参考：

1. SSH key: https://blog.csdn.net/weixin_42310154/article/details/118340458
2. Git 官网: <https://git-scm.com/>
3. Git 资料: <https://zhuanlan.zhihu.com/p/30044692> 和 <https://zhuanlan.zhihu.com/p/653979004>

错误和建议可以联系赵吉辰