

Git

本文件用于个人的Git知识点记录

一、什么是Git及相关概念

Git是一个分布式版本控制软件。

那什么是**版本控制**？

版本控制（Revision control）是一种在开发的过程中用于管理我们对文件、目录或工程等内容的修改历史，方便查看更改历史记录，备份以便恢复以前的版本的软件工程技术。

那什么又是**分布式**？

版本控制大体分为三类，分别是本地版本控制（RCS）、集中版本控制（SVN）、分布式版本控制（Git）。

简单来说

本地版本控制

记录你文件的每次更新并保存在本地，一般适用于个人。

集中版本控制

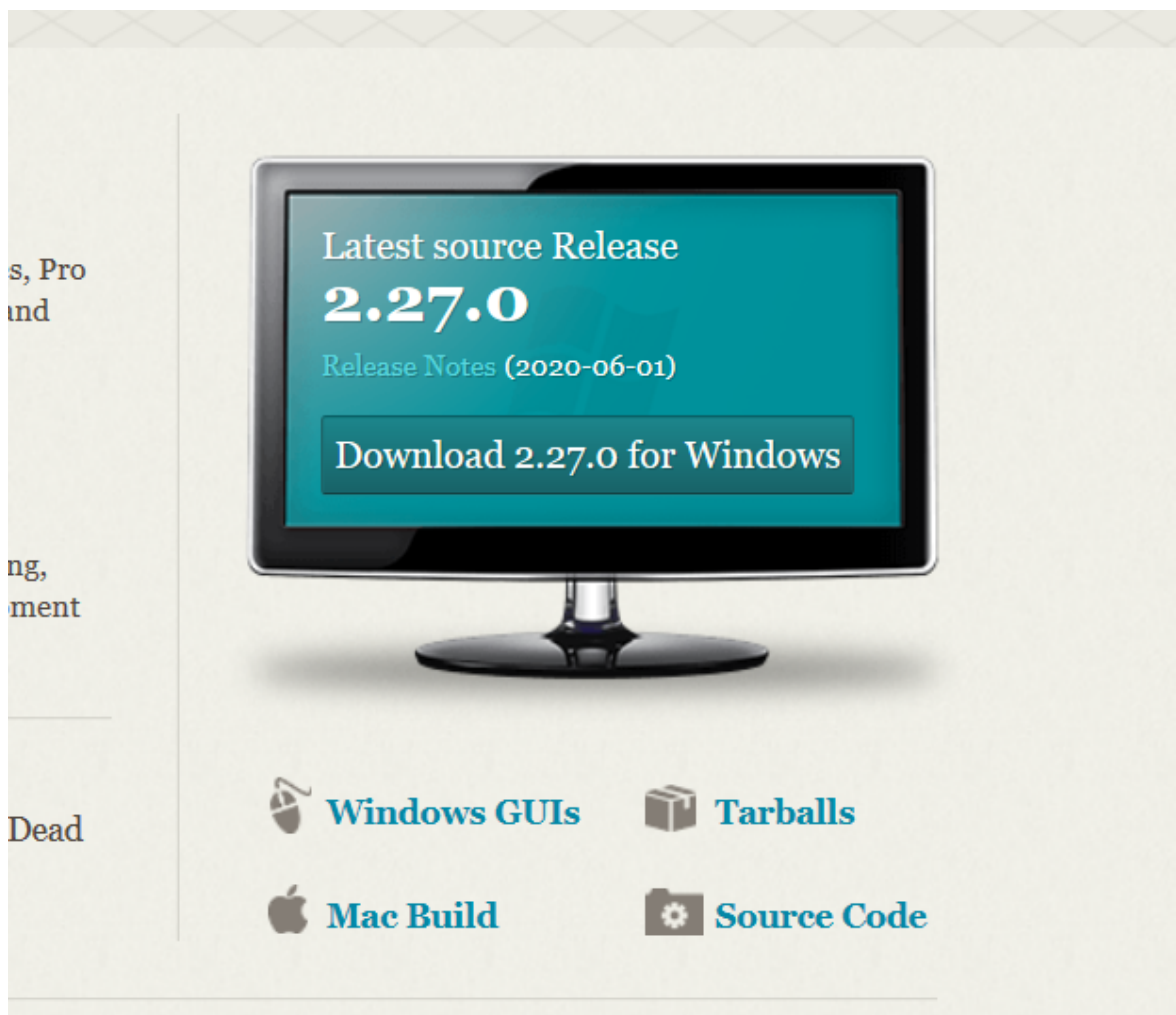
所有版本数据保存在一台服务器上，用户需要通过联网来查看历史版本及切换版本或上传自己的修改。但由于数据存储在单一的服务器上，服务器损坏则会丢失所有数据，存在安全隐患。

分布式版本控制

所有版本信息仓库全部同步到本地的每个用户，这样就可以在本地查看所有版本历史，可以离线在本地提交，只需在连网时push到相应的服务器或其他用户那里。由于每个用户那里保存的都是所有的版本数据，只要有一个用户的设备没有问题就可以恢复所有的数据，但这增加了本地存储空间的占用。

二、Git安装

进入 Git官网 <https://git-scm.com/> 下载安装包



官网下载缓慢的同学，请移步淘宝镜像下载 <http://npm.taobao.org/mirrors/git-for-windows/>

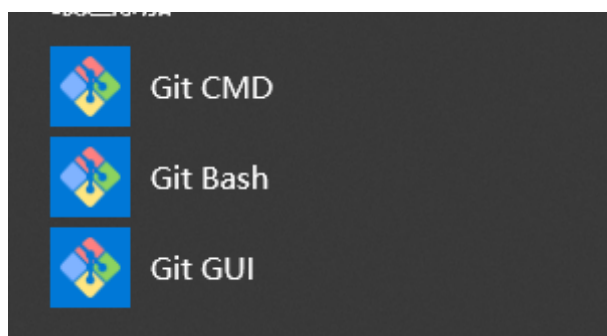
v2.27.0.windows.1/

2020-06-01T19:16:02Z

点进后根据操作系统来选择版本，windows系统的同学，下载64位以.exe结尾的文件即可

../		
Git-2.27.0-32-bit.exe	2020-06-01T19:15:07Z	47851216(45.63MB)
Git-2.27.0-32-bit.tar.bz2	2020-06-01T19:15:19Z	87635965(83.58MB)
Git-2.27.0-64-bit.exe	2020-06-01T19:15:05Z	48066136(45.84MB)
Git-2.27.0-64-bit.tar.bz2	2020-06-01T19:15:17Z	91718583(87.47MB)
MinGit-2.27.0-32-bit.zip	2020-06-01T19:15:12Z	24857196(23.71MB)
MinGit-2.27.0-64-bit.zip	2020-06-01T19:15:11Z	24709749(23.57MB)
MinGit-2.27.0-busybox-32-bit.zip	2020-06-01T19:15:14Z	20662592(19.71MB)
MinGit-2.27.0-busybox-64-bit.zip	2020-06-01T19:15:13Z	20541473(19.59MB)
pdb-for-git-32-bit-2.27.0.1.907ab1011d-1.zip	2020-06-01T19:15:21Z	20721761(19.76MB)
pdb-for-git-64-bit-2.27.0.1.907ab1011d-1.zip	2020-06-01T19:15:20Z	19197049(18.31MB)
PortableGit-2.27.0-32-bit.7z.exe	2020-06-01T19:15:10Z	43624104(41.6MB)
PortableGit-2.27.0-64-bit.7z.exe	2020-06-01T19:15:08Z	43803384(41.77MB)

打开安装包 选择下安装位置 无脑下一步就行



安装完成后 会有这三个东西 它们都是git不过是操作界面不同

Git CMD: windows命令行风格

Git Bash: Linux风格 (推荐使用 多熟悉下linux指令 以后用的到)

Git GUI: 图形界面

进入Git Bash 输入 `git --version` 正确返回版本号即安装成功

```
Alex FAN@C MINGW64 ~
$ git --version
git version 2.27.0.windows.1
```

三、基础Linux命令

因为Git Bash 涉及到linux指令操作, 所以这里简单列出一些常用命令, 不做详细介绍, 有需要的自己再去深入学习

- 1)、`cd`: 改变目录。
- 2)、`cd ..` 回退到上一个目录, 直接`cd`进入默认目录
- 3)、`pwd`: 显示当前所在的目录路径。
- 4)、`ls(l)`: 都是列出当前目录中的所有文件, 只不过`ll`(两个`l`)列出的内容更为详细。
- 5)、`touch`: 新建一个文件 如 `touch index.js` 就会在当前目录下新建一个`index.js`文件。
- 6)、`rm`: 删除一个文件, `rm index.js` 就会把`index.js`文件删除。
- 7)、`mkdir`: 新建一个目录,就是新建一个文件夹。
- 8)、`rm -r`: 删除一个文件夹, `rm -r src` 删除`src`目录
- 9)、`mv` 移动文件, `mv index.html src index.html` 是我们移动的文件, `src` 是目标文件夹,当然, 这样写, 必须保证文件和目标文件夹在同一目录下。
- 10)、`reset` 重新初始化终端/清屏。
- 11)、`clear` 清屏。
- 12)、`history` 查看命令历史。
- 13)、`help` 帮助。
- 14)、`exit` 退出。
- 15)、`#`表示注释

注意: `rm -rf` 无提示强制递归删除文件 **请务必谨慎使用**

四、Git的相关配置

查看Git的系统配置

```
MINGW64:/c/Users/Alex FAN
Alex FAN@C MINGW64 ~
$ git config -l
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=F:/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
credential.helper=manager
pull.rebase=false
user.name=binggao
user.email=alexlong511@foxmail.com

Alex FAN@C MINGW64 ~
$
```

配置 也可以分级别来查看（用户、系统）

```
Alex FAN@C MINGW64 ~
$ git config --system -l

Alex FAN@C MINGW64 ~
$ git config --global -l
```

这里需要注意的是，我们需要设置自己的用户名和email地址用于Git提交时被添加进去

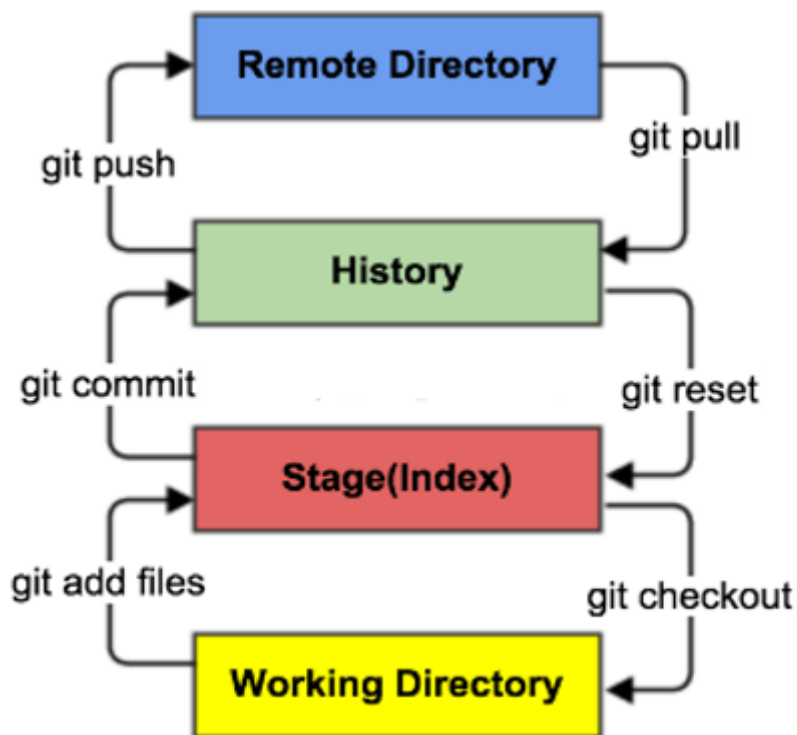
```
Alex FAN@C MINGW64 ~
$ git config --global user.name "fanfuni"

Alex FAN@C MINGW64 ~
$ git config --global user.email "2205436513@qq.com"

Alex FAN@C MINGW64 ~
$ git config --global -l
user.name=fanfuni
user.email=2205436513@qq.com
```

五、Git相关理论及原理

Git本地有三个工作区域：工作目录（Working Directory）、暂存区(Stage/Index)、资源库(Repository或Git Directory)。如果在加上远程的git仓库(Remote Directory)就可以分为四个工作区域。文件在这四个区域之间的转换关系如下



Workspace: 工作区, 就是你平时存放项目代码的地方

Index / Stage: 暂存区, 用于临时存放你的改动, 事实上它只是一个文件, 保存即将提交到文件列表信息

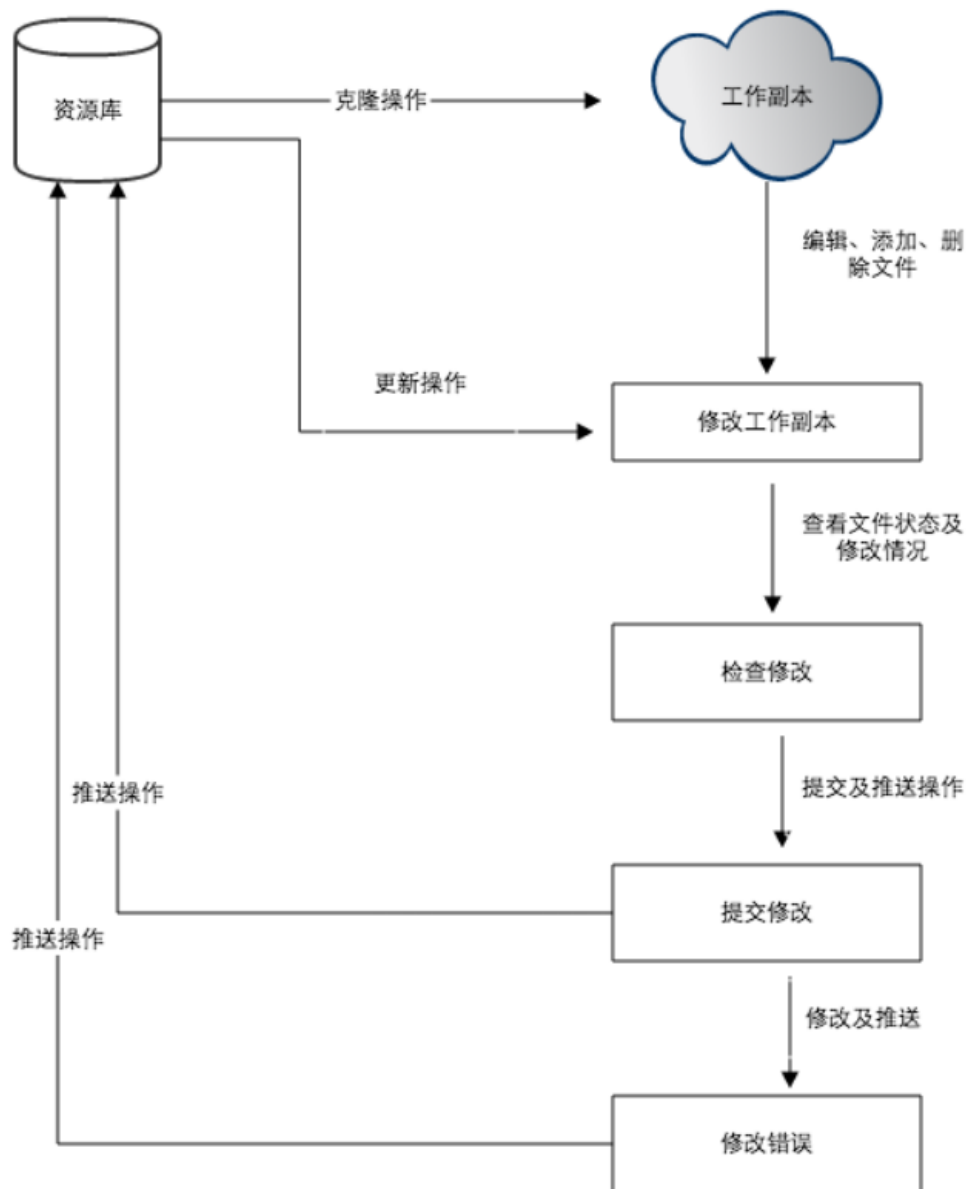
Repository: 仓库区 (或本地仓库), 就是安全存放数据的位置, 这里面有你提交到所有版本的数据。其中HEAD指向新放入仓库的版本

Remote: 远程仓库, 托管代码的服务器, 可以简单的认为是你项目组中的一台电脑用于远程数据交换

Git的工作流程

一般来说工作流程是这样的

- 1、克隆项目/新建项目
- 2、修改工作目录中的文件
- 3、将修改目录提交到暂存区
- 4、将暂存区文件提交到git仓库



六、Git项目搭建

Git创建本地仓库一般有两种方式，一种是自己创建一个全新的目录，另一种是clone别人的仓库

搭建本地仓库

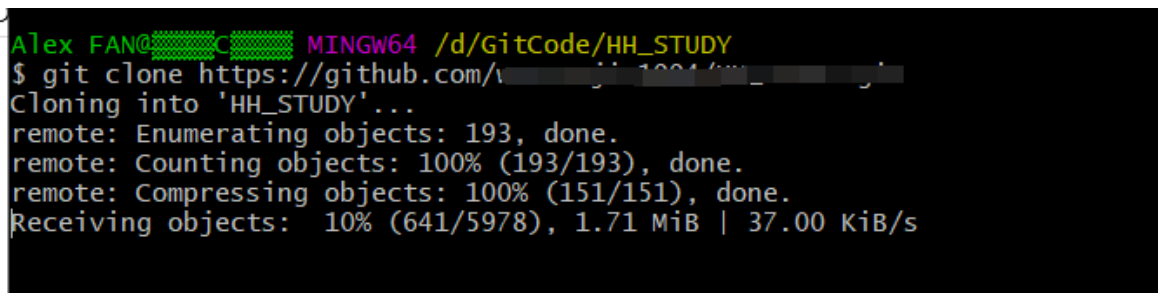
首先创建一个空的文件夹，并在文件夹中右击打开Git Bash



输入 `Git init` 运行之后 Git项目初始化成功



克隆远程仓库

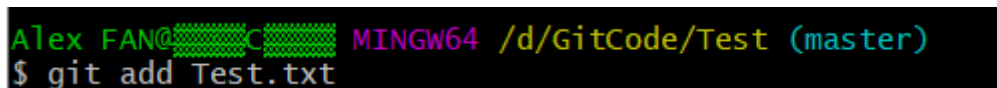


接下来我们从零开始自己创建一个仓库来并推送到github上来熟悉相关操作

刚才我们搭建了自己的本地仓库，接下来我们随便写一个test文件

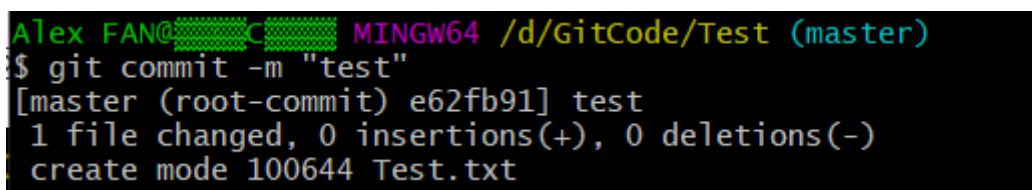
.git	2020/7/4 0:02	文件夹	
Test.txt	2020/7/4 22:40	文本文档	0 KB

使用git add命令来把文件添加到暂存区



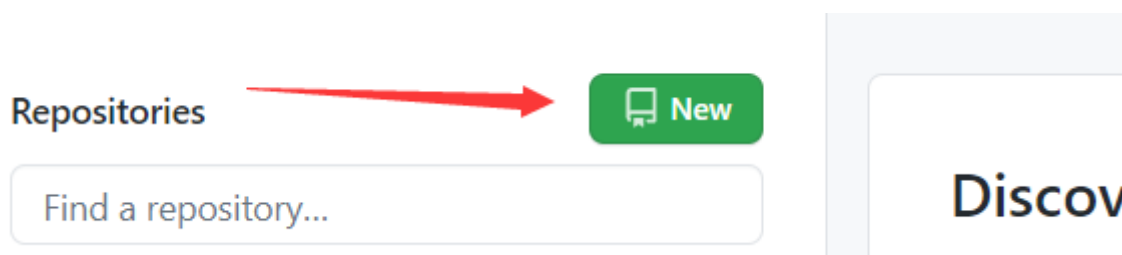
最后使用git commit将文件提交到仓库

这里我们最好使用git commit -m "xxx说明" 这样有了说明会方便你以后从历史版本中找到自己想要的版本





ok 现在已经到了本地仓库 如果提交到远程仓库呢?

首先我们要去我们的github上创建一个对应的仓库




Owner * Repository name *


 git2205436513 / Test 

Great repository names are short and memorable. Need inspiration? How about **shiny-spoon**?

Description (optional)

测试本地仓库推送到远程仓库

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer.

请注意：这里请不要初始化README，因为本地文件中没有README 会导致本地仓库与远程仓库的版本不一致 具体原因请见下链接

<https://blog.csdn.net/aaqingying/article/details/96165215>

...or push an existing repository from the command line

```
git remote add origin https://github.com/git2205436513/Test.git
git push -u origin master
```

这里github提示我们 使用本地仓库需要进行的操作

```
Alex FAN@MINGW64 /d/GitCode/Test (master)
$ git remote add origin git@github.com:git2205436513/Test.git
```

```
Alex FAN@MINGW64 /d/GitCode/Test/.git (GIT_DIR!)
$ git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 201 bytes | 201.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:git2205436513/Test.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

注意

SSH警告

当你第一次使用Git的 **clone** 或者 **push** 命令连接GitHub时，会得到一个警告：

```
The authenticity of host 'github.com (xx.xx.xx.xx)' can't be established.
RSA key fingerprint is xx.xx.xx.xx.xx.
Are you sure you want to continue connecting (yes/no)?
```

这是因为Git使用SSH连接，而SSH连接在第一次验证GitHub服务器的Key时，需要你确认GitHub的Key的指纹信息是否真的来自GitHub的服务器，输入 **yes** 回车即可。

Git会输出一个警告，告诉你已经把GitHub的Key添加到本机的一个信任列表里了：

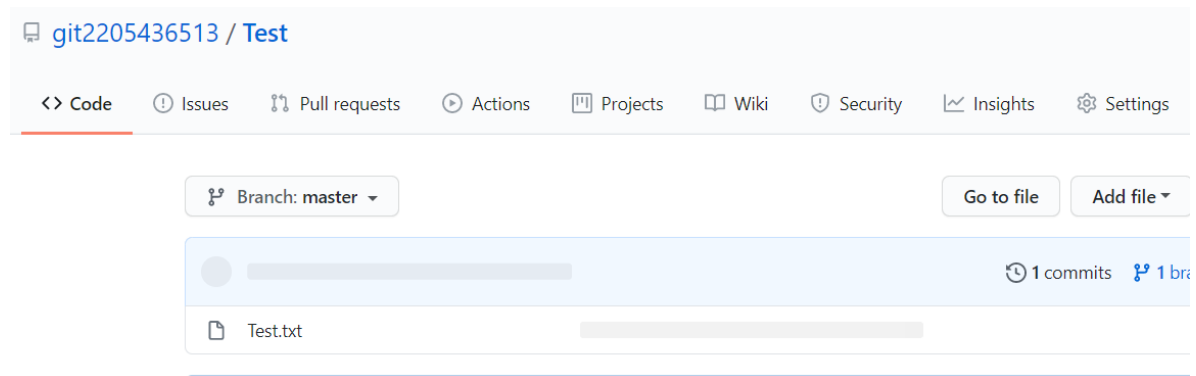
```
Warning: Permanently added 'github.com' (RSA) to the list of known hosts.
```

这个警告只会出现一次，后面的操作就不会有任何警告了。

如果你实在担心有人冒充GitHub服务器，输入 **yes** 前可以对照GitHub的RSA Key的指纹信息是否与SSH连接给出的一致。

图示操作后遇到问题 请参考以下链接 https://blog.csdn.net/weixin_44394753/article/details/91410463

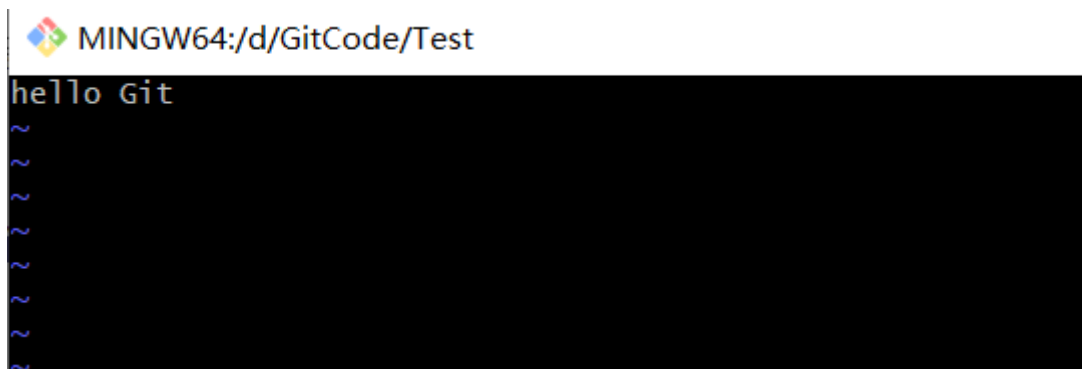
操作完后在github界面进行刷新 我们可以看到推送已经完成



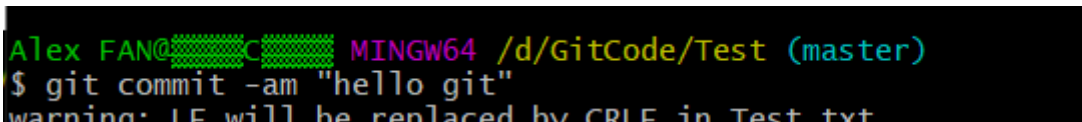
七、版本回滚

我们在上一节中学会了怎么提交新的版本，那么这一节我们来探讨如何回退版本。

首先修改项目中的Test

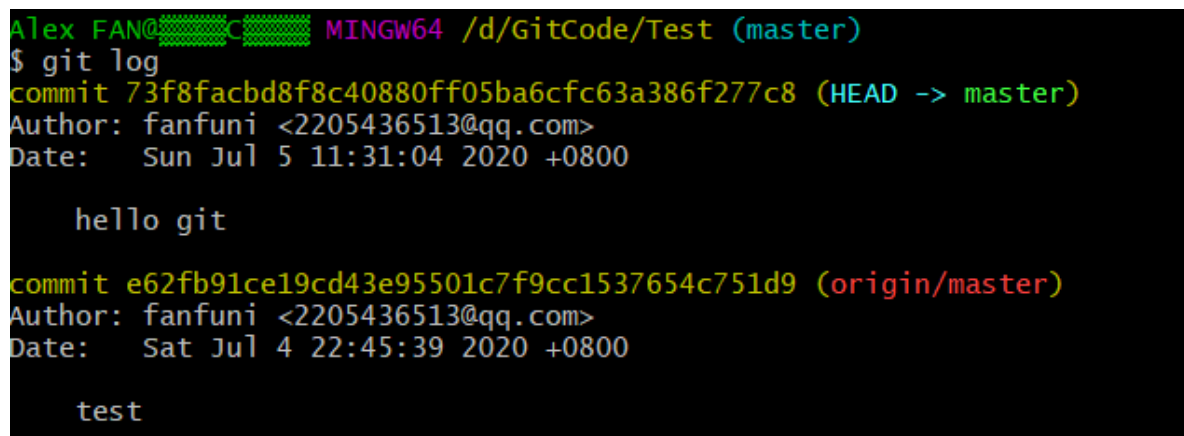


然后进行提交



这里可以看到我使用的是git commit -a 指令 它的作用是提交跟踪过的文件 上文中我们已经add 过Test文件 所以它已经被跟踪，如果不使用这个命令 你还是需要add 之后才能 commit哦

使用 git log来查看我们提交过的版本信息



接下来我们来回退到之前的版本 使用 git reset --hard commit地址

```
Alex FAN@C MINGW64 /d/GitCode/Test (master)
$ git reset --hard e62fb91
HEAD is now at e62fb91 test
```

```
Alex FAN@C MINGW64 /d/GitCode/Test (master)
$ cat Test.txt
```

```
Alex FAN@C MINGW64 /d/GitCode/Test (master)
```

```
Alex FAN@C MINGW64 /d/GitCode/Test (master)
$ git log
commit e62fb91ce19cd43e95501c7f9cc1537654c751d9 (HEAD -> master, origin/master)
Author: fanfuni <2205436513@qq.com>
Date: Sat Jul 4 22:45:39 2020 +0800

    test
```

可以看到Test中已经没有内容，git log 中相关信息已消失

那么问题来了 回退是回到过去 那么回退之后怎么穿越未来呢？

git log中虽然没有了相关信息 但是之前我们git log中是存在的呀 所以我上翻到那一栏 再输入对应的commit号就行啦

```
Alex FAN@C MINGW64 /d/GitCode/Test (master)
$ git log
commit 73f8facbd8f8c40880ff05ba6cfc63a386f277c8 (HEAD -> master)
Author: fanfuni <2205436513@qq.com>
Date: Sun Jul 5 11:31:04 2020 +0800

    hello git

commit e62fb91ce19cd43e95501c7f9cc1537654c751d9 (origin/master)
Author: fanfuni <2205436513@qq.com>
Date: Sat Jul 4 22:45:39 2020 +0800


    test
```

```
Alex FAN@C MINGW64 /d/GitCode/Test (master)
$ git reset --hard 73f8facb
HEAD is now at 73f8facb hello git

Alex FAN@C MINGW64 /d/GitCode/Test (master)
$ cat Test.txt
hello Git
```

返回成功

我们可以通过版本库来撤销修改，那么如果文件不在版本库呢？

 MINGW64:/d/GitCode/Test

```
hello Git
wuhu!!!! qifei!!!!!!
~
~
```

比如有一天你开始乱写了一通，那肯定不能随便上传。例如我们发现的即使，当然可以删除掉这一行，手动回复。但是我们也可以借助git restore <file...>来纠正

```
Alex FAN@C MINGW64 /d/GitCode/Test (master)
$ git checkout -- Test.txt

Alex FAN@C MINGW64 /d/GitCode/Test (master)
$ cat Test.txt
hello Git
```

这个指令 会把Test.txt工作区的修改全部撤销

哦，那如果你已经提交到暂存区了呢？

那我们可以使用 `git restore --staged <file..>` 来把暂存区的修改撤销掉

八、分支操作

1、查看现有分支

```
Alex FAN@C MINGW64 /d/GitCode/HH_STUDY (master)
$ git branch
* master
```

2、创建新分支并跳转

```
Alex FAN@C MINGW64 /d/GitCode/HH_STUDY (master)
$ git checkout -b fanjuncheng2355
Switched to a new branch 'fanjuncheng2355'
```

```
$ git branch
* fanjuncheng2355
  master
```

3、将新分支推送到git仓库

```
Alex FAN@C MINGW64 /d/GitCode/HH_STUDY (fanjuncheng2355)
$ git push
fatal: The current branch fanjuncheng2355 has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin fanjuncheng2355
```

```
Alex FAN@C MINGW64 /d/GitCode/HH_STUDY (fanjuncheng2355)
$ git push --set-upstream origin fanjuncheng2355
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'fanjuncheng2355' on GitHub by visiting:
remote:   https://github.com/wenrenjie1994/HH_STUDY/pull/new/fanjuncheng2355
remote:
To https://github.com/wenrenjie1994/HH_STUDY.git
 * [new branch]      fanjuncheng2355 -> fanjuncheng2355
Branch 'fanjuncheng2355' set up to track remote branch 'fanjuncheng2355' from 'origin'.
```

这样在你的github项目中就可以查看到这一分支啦

我们又如何将分支修改并推送到远程仓库呢？

```
Alex FAN@C MINGW64 /d/GitCode/HH_STUDY (fanjuncheng2355)
$ git commit -m "Git Note"
[fanjuncheng2355 b512115] Git Note
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 Note_Git.pdf

Alex FAN@C MINGW64 /d/GitCode/HH_STUDY (fanjuncheng2355)
$ git remote -v
origin https://github.com/wenrenjie1994/HH_STUDY.git (fetch)
origin https://github.com/wenrenjie1994/HH_STUDY.git (push)

Alex FAN@C MINGW64 /d/GitCode/HH_STUDY (fanjuncheng2355)
$ git push origin fanjuncheng2355
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 878.24 KiB | 11.41 MiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/wenrenjie1994/HH_STUDY.git
fd109bc..b512115 fanjuncheng2355 -> fanjuncheng2355
```

[^]: by fanfuni