GIT指令学习

MINGW64:/c/Users/杨毓栋

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~
$
```

MINGW64:/c/Users/杨毓栋/learngit

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~
$ git config --global user.name "yangyudong"

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~
$ git config --global user.email "1554366868@qq.com"

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~
$ mkdir learngit

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~
$ cd learngit

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit
$ pwd
/c/Users/杨毓栋/learngit

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit
$ git init
Initialized empty Git repository in C:/Users/杨毓栋/learngit/.git/

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ |
```

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ ls -ah
./  ../  .git/

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git add readme.txt

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git commit -m "wrote a readme file"
[master (root-commit) cb0780b] wrote a readme file
 1 file changed, 2 insertions(+)
 create mode 100644 readme.txt

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$
```

```
/***git add 增加根目录中某个文件到仓库中

*ls -ah是显示隐藏文件内容

*git commit 是将文件提交到仓库，-m后是本次提交的声明 //一般连贯输入 git commit -m“XXXX”

*代表着我声明将这个文件到仓库中并有记录存在

*1file changed代表一个文件发生改动，2  insertions表示插入了两行内容（readme.txt中）*/
```

answer:

Q：输入`git add readme.txt`，得到错误：`fatal: not a git repository (or any of the parent directories)`。

A：Git命令必须在Git仓库目录内执行（`git init`除外），在仓库目录外执行是没有意义的。

Q：输入`git add readme.txt`，得到错误`fatal: pathspec 'readme.txt' did not match any files`。

A：添加某个文件时，该文件必须在当前目录下存在，用`ls`或者`dir`命令查看当前目录的文件，看看文件是否存在，或者是否写错了文件名。

为什么Git添加文件需要`add`，`commit`一共两步呢？因为`commit`可以一次提交很多文件，所以你可以多次`add`不同的文件，比如：

```
$ git add file1.txt
$ git add file2.txt file3.txt
$ git commit -m "add 3 files."
```

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

git status 观察此时的git仓库情况

modified老单词了，修改修改

no change added to commit 的意思是修改的内容还没有提交

```
$ git diff
diff --git a/readme.txt b/readme.txt
index 0646d96..9247db6 100644
--- a/readme.txt
+++ b/readme.txt
@@ -1,2 +1,2 @@
-Git is a version control system.
-Git is free software0
+Git is a distributed version control system.
+Git is free software.
```

git diff的意思是你长时间没看仓库，不知道自己上一次修改修改了些什么，就会用到git diff

diff老单词了，difference

一般在观测到文件确实没有错误的时候再提交到仓库，做一个不提交bug的人

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git add readme.txt

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   readme.txt
```

提交后再看文件status，发现绿了！！！！！！！！！！

```
$ git commit -m "add distributed"
[master 6e50235] add distributed
 1 file changed, 2 insertions(+), 2 deletions(-)
```

老提交方法了，然后看最后一行后两个，发现有两行发生改变，两行以前的消失

```
$ git status
On branch master
nothing to commit, working tree clean
```

没有需要提交的修改，工作目录老干净了！

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git log
commit 176fd5ab7f3ad5cbc46ca3515aa7f6d44e523bf5 (HEAD -> master)
Author: yangyudong <1554366868@qq.com>
Date:   Mon Nov 2 19:37:36 2020 +0800

    append GPL

commit 6e50235a5b57d40620639782afa2ca8b1ac44476
Author: yangyudong <1554366868@qq.com>
Date:   Mon Nov 2 19:30:50 2020 +0800

    add distributed

commit cb0780bf90ebe5d213dcf82673d64809de420715
Author: yangyudong <1554366868@qq.com>
Date:   Mon Nov 2 19:09:08 2020 +0800

    wrote a readme file
```

git log用于查看近几次的修改历史纪录，可以追责哈啊哈哈

```
$ git log --pretty=oneline
176fd5ab7f3ad5cbc46ca3515aa7f6d44e523bf5 (HEAD -> master) append GPL
6e50235a5b57d40620639782afa2ca8b1ac44476 add distributed
cb0780bf90ebe5d213dcf82673d64809de420715 wrote a readme file
```

写为git log --pretty=oneline可以更简洁的看整个修改记录

前面那一行容易引人遐思的黄色字符是十六进制的版本号，为了防止修改冲突

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git reset --hard HEAD^
HEAD is now at 6e50235 add distributed
```

时光穿梭机，git reset --hard HEAD^

在git中，上一个版本是HEAD^,上上个版本是HEAD^^。以此类推，在往上咱肯定不会写^了，太多了，所以咱就用HEAD~100（表示向上一百个版本，但我希望未来的自己不会用到这个指令）

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ cat readme.txt
Git is a distributed version control system.
Git is free software.
```

棒棒，果然被还原了

```
$ git log
commit 6e50235a5b57d40620639782afa2ca8b1ac44476 (HEAD -> master)
Author: yangyudong <1554366868@qq.com>
Date:   Mon Nov 2 19:30:50 2020 +0800

    add distributed

commit cb0780bf90ebe5d213dcf82673d64809de420715
Author: yangyudong <1554366868@qq.com>
Date:   Mon Nov 2 19:09:08 2020 +0800

    wrote a readme file
```

上个版本已经看不到了（我似乎找到了不被追责的把办法）

办法其实还是有的，只要上面的命令行窗口还没有被关掉，你就可以顺着往上找啊找啊，找到那个`append GPL`的`commit id`是`1094adb...`，于是就可以指定回到未来的某个版本：

！！！！！！（注意消除数据库啊啊啊啊啊啊啊啊啊啊啊啊）

```
$ git reset --hard 176fd
HEAD is now at 176fd5a append GPL
```

恢复辽。。。。。。（版本号不必要写全）

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ cat readme.txt
Git is a distributed version control system.
Git is free software distributed under the GPL.
```

又回来了（小心被追责）

Git的版本回退速度非常快，因为Git在内部有个指向当前版本的`HEAD`指针，当你回退版本的时候，Git仅仅是把HEAD从指向`append GPL`：

```
HEAD
  |
  |__> o append GPL
       |
       o add distributed
       |
       o wrote a readme file
```

改为指向`add distributed`：

```
HEAD
  |
  |    o append GPL
  |    |
  |__> o add distributed
       |
       o wrote a readme file
```

熟悉的指针，指向某个版本

在Git中，总是有后悔药可以吃的。当你用`$ git reset --hard HEAD^`回退到`add distributed`版本时，再想恢复到`append GPL`，就必须找到`append GPL`的commit id。Git提供了一个命令`git reflog`用来记录你的每一次命令：

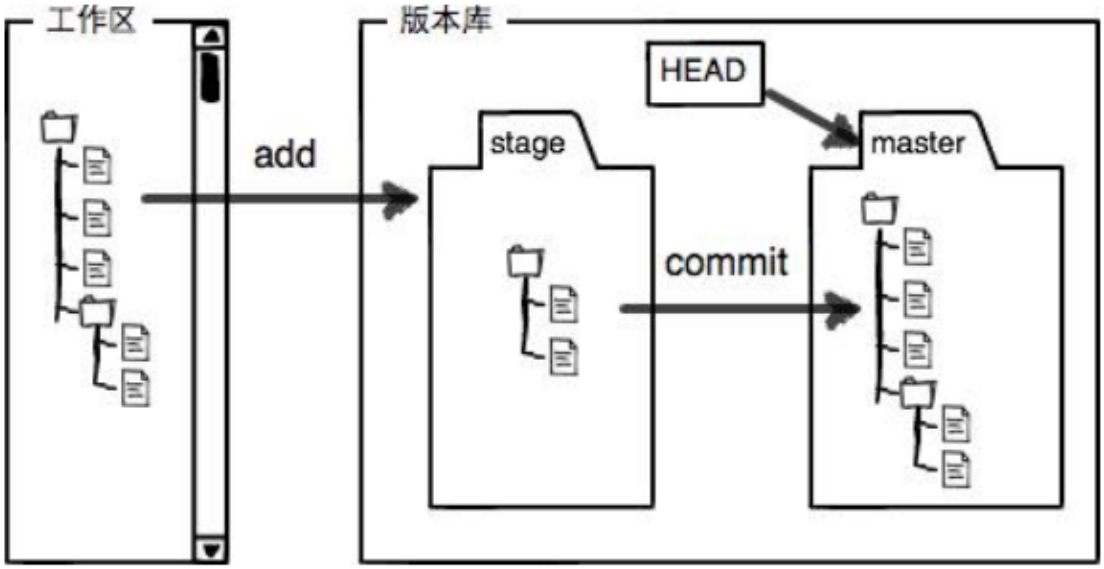
我不是很想要被追责的后悔药

```
$ git reflog
176fd5a (HEAD -> master) HEAD@{0}: reset: moving to 176fd
6e50235 HEAD@{1}: reset: moving to HEAD^
176fd5a (HEAD -> master) HEAD@{2}: commit: append GPL
6e50235 HEAD@{3}: commit: add distributed
cb0780b HEAD@{4}: commit (initial): wrote a readme file
```

小知识

工作区有一个隐藏目录`.git`，这个不算工作区，而是Git的版本库。

Git的版本库里存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自动创建的第一个分支`master`，以及指向`master`的一个指针叫`HEAD`。



master是一个分支，以及指向这个分支的HEAD（指针）

提交GIT版本库两步走

第一步：git add添加文件，将文件修改添加到暂存区

第二步：git commit 提交更改，将暂存区的内容提交到分支去

（PS：我们在创建git版本库时，git就自动为我们创建了唯一一个master分支，所以，git commit就是在分支master上的修改）

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   readme.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        LICENSE.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

新增了一个名为licenes的文件，并且将readme,txt进行修改

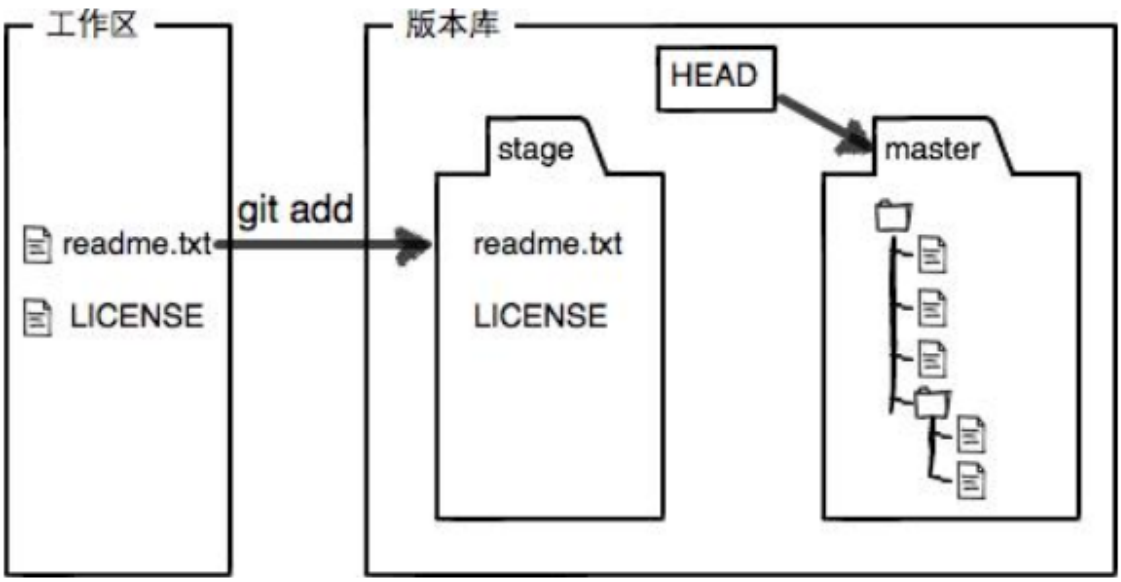
git大大在告诉我们，readme被填写了但是我们没有提交，而licenes被放在子目录了我们还什么也没干

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master
$ git add readme.txt

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master
$ git add LICENSE.txt

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   LICENSE.txt
        modified:   readme.txt
```

添加莫忘后缀.txt

git明确告诉我们，一个新的文件，一次修改

现在，暂存区的状态就变成这样了：



所以，`git add`命令实际上就是把要提交的所有修改放到暂存区（Stage），然后，执行`git commit`就可以一次性把暂存区的所有修改提交到分支。

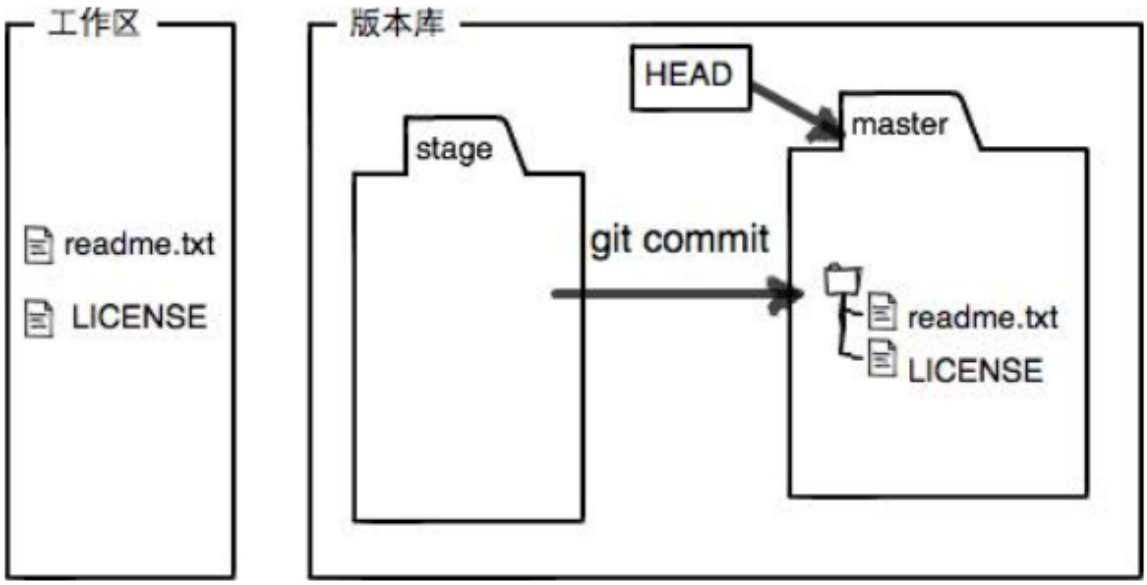
在所有修改后，git commit 可以一次性将所有修改提交到git库中

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (mas
$ git commit -m "understand how stage works"
[master 461819a] understand how stage works
 2 files changed, 3 insertions(+)
 create mode 100644 LICENSE.txt
```

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (ma
$ git status
On branch master
nothing to commit, working tree clean
```

okey，文件很干净

现在版本库变成了这样，暂存区就没有任何内容了：




```
$ cat readme.txt
Git is a distributed version control system.
Git is free software distributed under the GPL.
Git has a mutable index called stage.
```

cat 文件名 的意思是我们将txt文本内容拿出来看看，里面有些什么

第一次修改 -> `git add` -> 第二次修改 -> `git commit`

你看，我们前面讲了，Git管理的是修改，当你用`git add`命令后，在工作区的第一次修改被放入暂存区，准备提交，但是，在工作区的第二次修改并没有放入暂存区，所以，`git commit`只负责把暂存区的修改提交了，也就是第一次的修改被提交了，第二次的修改不会被提交。

第一次修改时没有commit而是git add ，第二次修改则是git commit

那么很明显，我们第一次的修改可能将文件放在了暂存区；

第二次修改的内容没有往暂存区放，所以git commit可能只把放在暂存区的第一次修改提交

```
$ git diff HEAD -- readme.txt
diff --git a/readme.txt b/readme.txt
index db28b2c..2b73578 100644
--- a/readme.txt
+++ b/readme.txt
@@ -1,4 +1,4 @@
 Git is a distributed version control system.
 Git is free software distributed under the GP
 Git has a mutable index called stage.
-Git tracks changes.
\ No newline at end of file
+Git tracks changes of files
\ No newline at end of file
```

Git diff HEAD --readme.txt　　查看不同 并将指针指向readme.txt文件

```
@@ -1,4 +1,4 @@
 Git is a distributed version control syst
 Git is free software distributed under th
 Git has a mutable index called stage.
-Git tracks changes.
\ No newline at end of file
+Git tracks changes of files
\ No newline at end of file

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit
$ git add readme.txt

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit
$ git commit -m"git changes two "
[master 85aa612] git changes two
 1 file changed, 1 insertion(+), 1 deletio

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit
$ git status
On branch master
nothing to commit, working tree clean
```

每一次修改文件后，注意先add添加到暂存区，再由暂存区提交到库中

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git checkout -- readme.txt
```

git checkout -- readme.txt的意思是把文件在工作区的修改全部撤销

因为readme还没有提交，没有add到暂存区，更没有commit，所以撤销掉会恢复上一个版本，没有什么影响

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ cat readme.txt
Git is a distributed version control system.
Git is free software distributed under the GPL.
Git has a mutable index called stage.
Git tracks changes of files
```

啊，不用被炒鱿鱼了

`git checkout -- file`命令中的`--`很重要，没有`--`，就变成了“切换到另一个分支”的命令，我们在后面的分支管理中会再次遇到`git checkout`命令。

另一种方案撤销假设：

现在假定是凌晨3点，你不但写了一些胡话，还`git add`到暂存区了：

（第一想法：完犊子了）

Git同样告诉我们，用命令`git reset HEAD <file>`可以把暂存区的修改撤销掉（unstage），重新放回工作区：

```
$ git reset HEAD readme.txt
Unstaged changes after reset:
M   readme.txt
```

`git reset`命令既可以回退版本，也可以把暂存区的修改回退到工作区。当我们用`HEAD`时，表示最新的版本。

再用`git status`查看一下，现在暂存区是干净的，工作区有修改：

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
directory)

    modified:   readme.txt
```

还记得如何丢弃工作区的修改吗？

```
$ git checkout -- readme.txt

$ git status
On branch master
nothing to commit, working tree clean
```

整个世界终于清静了！

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   readme.txt


杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ cat readme.txt
Git is a distributed version control system.
Git is free software distributed under the GPL.
Git has a mutable index called stage.
Git tracks changes of files
My stupid boss still prefers SVN.
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git reset HEAD readme.txt
Unstaged changes after reset:
M       readme.txt
```

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git checkout -- readme.txt

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git status
On branch master
nothing to commit, working tree clean

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$
```

删除大法

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ rm test.txt

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    test.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

名称	修改日期	类型	大小
.git	2020/11/2 23:10	文件夹	
LICENSE.txt	2020/11/2 20:15	文本文档	1 KB
readme.txt	2020/11/2 20:39	文本文档	1 KB

我放入的test文件被删除了 用rm test.txt 此时此刻，我似乎知道了，为什么会有rm rf这个梗了

打开status会显示你删除了这个文件

删除的两种情况

1.误删

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git checkout -- test.txt

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git status
On branch master
nothing to commit, working tree clean

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ cat test.txt
this is text files.
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$
```

git checkout -- 文件名 可以将此时误删，但还存在于版本库中的文件恢复

2.逃避追责

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git rm test.txt
rm 'test.txt'

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git commit -m"remove test.txt"
[master fb21149] remove test.txt
 1 file changed, 1 deletion(-)
 delete mode 100644 test.txt
```

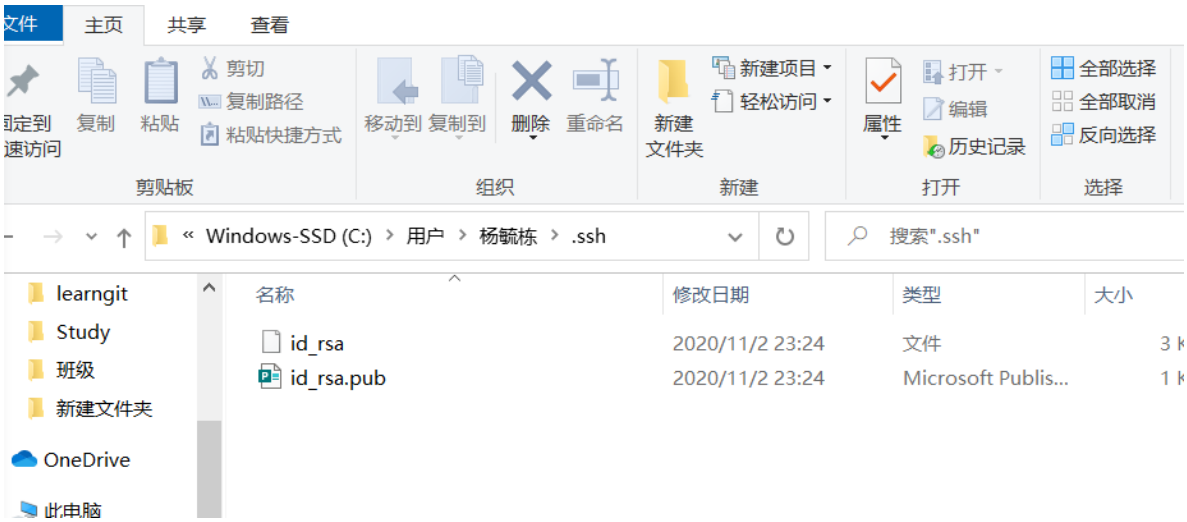
先删除，然后再提交这个删除信息

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git status
On branch master
nothing to commit, working tree clean

杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ cat test.txt
cat: test.txt: No such file or directory
```

不见了！！！！！！！！！！！哦耶！

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ ssh-keygen -t rsa -C"1554366868@qq.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/杨毓栋/.ssh/id_rsa):
Created directory '/c/Users/杨毓栋/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/杨毓栋/.ssh/id_rsa.
Your public key has been saved in /c/Users/杨毓栋/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:Ja+HvjTprjRYZBpOllUGrz217Ru6CygwA7XM10pg1gA 1554366868@qq.com
The key's randomart image is:
+---[RSA 3072]----+
|E.*o  ooo        |
| * o.+ o         |
|. + B + o o      |
| . * * o = o     |
|  + + o S o .    |
|   + o . = .     |
|    o + B . o    |
|     o = + . o   |
|      .o=.+o.    |
+----[SHA256]-----+
```

```
$ ssh-keygen -t rsa -C"1554366868@qq.com"
```



如果一切顺利的话，可以在用户主目录里找到`.ssh`目录，里面有`id_rsa`和`id_rsa.pub`两个文件，这两个就是SSH Key的秘钥对，`id_rsa`是私钥，不能泄露出去，`id_rsa.pub`是公钥，可以放心地告诉任何人。

SSH keys

New SSH key

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

yangyudong01
SHA256:Ja+HvjTprjRYZBpOllUGrz217Ru6CygwA7XM10pg1gA
Added on 2 Nov 2020
Never used — Read/write
SSH
Delete

Check out our guide to generating SSH keys or troubleshoot common SSH Problems.

```
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 8 threads
Compressing objects: 100% (19/19), done.
Writing objects: 100% (23/23), 1.91 KiB | 651.00 KiB/s, done.
Total 23 (delta 6), reused 0 (delta 0)
remote: Resolving deltas: 100% (6/6), done.
To github.com:yangyudong2020/learngit.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

可能是因为我打开了密钥，变成了无法上传，不过现在好了

现在，我们根据GitHub的提示，在本地的`learngit`仓库下运行命令：

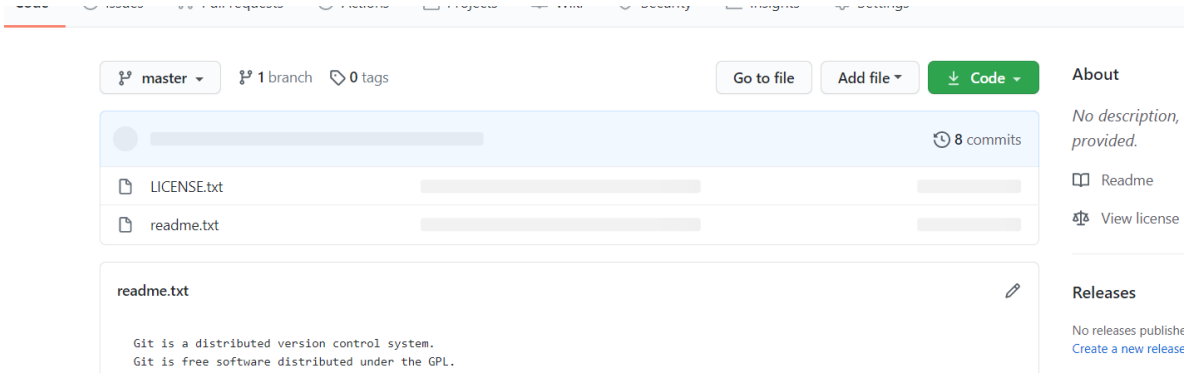
```
$ git remote add origin git@github.com:michaelliao/learngit.git
```

请千万注意，把上面的`michaelliao`替换成你自己的GitHub账户名，否则，你在本地关联的就是我的远程库，关联没有问题，但是你以后推送是推不上去的，因为你的SSH Key公钥不在我的账户列表中。

添加后，远程库的名字就是`origin`，这是Git默认的叫法，也可以改成别的，但是`origin`这个名字一看就知道是远程库。

下一步，就可以把本地库的所有内容推送到远程库上：

```
$ git push -u origin master
Counting objects: 20, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (15/15), done.
Writing objects: 100% (20/20), 1.64 KiB | 560.00 KiB/s, done.
Total 20 (delta 5), reused 0 (delta 0)
remote: Resolving deltas: 100% (5/5), done.
To github.com:michaelliao/learngit.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```



上传成功！！！！！

```
杨毓栋@LAPTOP-TBS99QJD MINGW64 ~/learngit (master)
$ git push origin master
Everything up-to-date
```

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：

吓我一跳，原来是这样啊

tips！！！！！

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。