# Git教程

以windows系统为例进行说明：

GitHub操作流程 :

第一次提交 :

方案一 : 本地创建项目根目录, 然后与远程GitHub关联, 之后的操作一样;

-- 初始化Git仓库 :Git init ;

-- 提交改变到缓存 :git commit -m 'description' ;

-- 本地git仓库关联GitHub仓库 : git remote add origin git@github.com:han1202012/TabHost\_Test.git ;

-- 提交到GitHub中 : git push -u origin master ;

方案二 : 方案二就是不用关联GitHub仓库, 直接从GitHub冲克隆源码到本地, 项目根目录也不用创建;

-- 从GitHub上克隆项目到本地 :git clone git@github.com:han1202012/NDKHelloworld.git , 注意克隆的时候直接在仓库根目录即可, 不用再创建项目根目录 ;

-- 添加文件 :git add ./\* , 将目录中所有文件添加;

-- 提交缓存 :git commit -m '提交';

-- 提交到远程GitHub仓库 : git push -u origin master ;

之后修改提交 :

-- 与GitHub远程仓库同步 :git pull ;

-- 查看文件变更 : git status ;

-- 提交代码到本地缓存 : git commit -m 'description';

--提交代码到远程GitHub仓库 :git push ;

.gitignore用法 : 开放模式 注明忽略的文件 直接列出文件名, 保守模式 注明保留的文件 !文件名 ;

Git标签操作 : 轻量级标签, 带注释标签;

--查看标签 :git tag ;

--添加标签 : 轻量级标签git tag -a tagName -m 'description' ;

--删除标签 :git tag -d tagName ;

--提交标签到GitHub中 : git push origin --tags ;

Git分支操作: 创建分支后, 分支操作不会影响master分支, 但是master分支改变会影其它分支;

--列出分支 :git branch ;

--切换分支 :git checkout master ;

--提交分支 : git push origin branchName ;

--删除分支 : git branch -D branchName ;

--合并分支 : git merge branchName ;

Git详细教程

## 1 Git介绍：

分布式: Git版本控制系统是一个分布式的系统, 是用来保存工程源代码历史状态的命令行工具;

保存点: Git的保存点可以追踪源码中的文件, 并能得到某一个时间点上的整个工程项目额状态; 可以在该保存点将多人提交的源码合并, 也可以会退到某一个保存点上;

Git离线操作性:Git可以离线进行代码提交, 因此它称得上是完全的分布式处理, Git所有的操作不需要在线进行; 这意味着Git的速度要比SVN等工具快得多, 因为SVN等工具需要在线时才能操作, 如果网络环境不好, 提交代码会变得非常缓慢;

Git基于快照: SVN等老式版本控制工具是将提交点保存成补丁文件, Git提交是将提交点指向提交时的项目快照, 提交的东西包含一些元数据(作者, 日期, GPG等);

Git的分支和合并: 分支模型是Git最显著的特点, 因为这改变了开发者的开发模式, SVN等版本控制工具将每个分支都要放在不同的目录中, Git可以在同一个目录中切换不同的分支;

分支即时性: 创建和切换分支几乎是同时进行的, 用户可以上传一部分分支, 另外一部分分支可以隐藏在本地, 不必将所有的分支都上传到GitHub中去;

分支灵活性: 用户可以随时 创建 合并 删除分支, 多人实现不同的功能, 可以创建多个分支进行开发, 之后进行分支合并, 这种方式使开发变得快速, 简单, 安全;

## 2 下载与安装

Git客户端下载地址 :https://code.google.com/p/msysgit/downloads/list

安装安装顺序一步一步安装即可

## 3 配置GitHub

3.1 配置本地用户和邮箱

用户名邮箱作用 :

我们需要设置一个用户名和邮箱, 这是用来上传本地仓库到GitHub中, 在GitHub中显示代码上传者;

使用命令:

$ git config --global user.name

$ git config --global user.email

## 4 创建版本库

版本库(repository)也叫仓库，可以看做一个目录，这个目录里的所以文件都由Git进行管理，每个文件的修改、删除，Git都能跟踪。

### 4.1 择一个合适的地方，创建一个空目录：

cd /h #进入h盘

$ mkdir myGit #创建一个空目录

注意：如果使用Windows系统，要保证目录名不包含中文。

### 4.2 初始化仓库

初次使用时，需要初始化才能使用仓库，通过如下命令把这个目录变成Git可以管理的仓库：

$ git init

这样Git就把仓库建好啦，我们可以看到在当前目录下多了一个 .git的目录，这个目录是Git来跟踪管理版本库的。

默认状态下，.git目录是隐藏的。因为不能修改里面的文件，否则会出现损坏有关文件

### 4.3 把文件添加到版本库

我们在 learngit 目录下编写一个 readme.txt 文件，内容如下：

Git is a version control system.

Git is free software.

4.3.1 用 git add 命令，把文件添加到仓库：

$ git add read.txt

关于 LF will be replaced by CRLF 问题出现的原因以及解决方式:

https://blog.csdn.net/wowoniuzailushang/article/details/54317129

4.3.2 用 git commit 命令，把文件提交到仓库：

$ git commit -m "wrote a readme file" //-m 后面输入的是本次提交的说明，可以输入任意内容。

[master (root-commit) eaadf4e] wrote a readme file

1 file changed, 2 insertions(+) //1 file changed：1个文件被改动（新添加的readme.txt文件）；2 insertions：插入了两行内容（readme.txt内有两行内容）

create mode 100644 readme.txt

这样文件就提交到本地仓库了

## 5 修改文件

我们已经成功添加并提交了一个 readme.txt 文件，继续修改 readme.txt 文件，改成如下内容：

Git is a distributed version control system.

Git is free software.

运行 git status 命令：

$ git status //查看仓库当前的状态

On branch master

Changes not staged for commit: //没有文件将要被提交

(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")

上面的命令告诉我们，readme.txt 文件被修改过了，但还没有准备提交的修改。

如果我们想知道上次是怎么修改readme.txt 文件的，需要用 git diff 命令：

$ git diff readme.txt

diff --git a/readme.txt b/readme.txt

index 46d49bf..9247db6 100644

--- a/readme.txt

+++ b/readme.txt

@@ -1,2 +1,2 @@

-Git is a version control system. //这一句是被删掉的

+Git is a distributed version control system. //这一句是新添加的

Git is free software.

+

然后重复

5.1 git add

5.2 git commit

5.3 git status

$ git status

On branch master

nothing to commit, working tree clean //当前没有需要提交的修改，而且，工作目录是干净的。

注意：关于git bush 中不能复制黏贴的问题：

https://blog.csdn.net/nishiwodebocai21/article/details/72510341

## 6 版本回退

如果我们继续对 readme.txt 文件进行修改，改成如下内容：

Git is a distributed version control system.

Git is free software distributed under the GPL.

然后添加并提交：

$ git add readme.txt

$ git commit -m "append GPL"

[master 1094adb] append GPL

1 file changed, 1 insertion(+), 1 deletion(-)

到目前为止，readme.txt 文件一共有三个版本被提交到了 Git 仓库里，我们可以用 git log 命令进行查看：

$ git log //查看历史记录

commit 1094adb7b9b3807259d8cb349e7df1d4d6477073 (HEAD -> master)

Author: Michael Liao <askxuefeng@gmail.com>

Date: Fri May 18 21:06:15 2018 +0800

append GPL

commit e475afc93c209a690c39c13a46716e8fa000c366

Author: Michael Liao <askxuefeng@gmail.com>

Date: Fri May 18 21:03:36 2018 +0800

add distributed

commit eaadf4e385e865d25c48e7ca9c8395c3f7dfaef0

Author: Michael Liao <askxuefeng@gmail.com>

Date: Fri May 18 20:59:18 2018 +0800

wrote a readme file

我们还可以加上 --pretty=oneline 参数：

$ git log --pretty=oneline

1094adb7b9b3807259d8cb349e7df1d4d6477073 (HEAD -> master) append GPL

e475afc93c209a690c39c13a46716e8fa000c366 add distributed

eaadf4e385e865d25c48e7ca9c8395c3f7dfaef0 wrote a readme file //一大串数字是 commit id ，而且每个人的都不一样。

好了，现在如果我们想把 readme.txt 文件退回到上一个版本，就可以使用 git reset 命令：

$ git reset --hard HEAD^ //HEAD表示当前版本，则HEAD^表示上一个版本，那么上上版本就是HEAD^^

HEAD is now at e475afc add distributed

这时候用 cat 命令查看一下 readme.txt 的内容：

$ cat readme.txt //查看 readme.txt 文件的内容

Git is a distributed version control system.

Git is free software.

果然 readme.txt 文件返回到了上一个版本。

我们现在想要回到最新的版本，还是使用 git reset 命令：

$ git reset --hard 1094a //这里不能用HEAD而必须使用 commit id ，因为最新版本在之前返回时已经被删除了，1094a就是最新版本的 commit id，可以在之前的代码中查到

HEAD is now at 83b0afe append GPL

这时再查看一下 readme.txt 文件内容：

$ cat readme.txt

Git is a distributed version control system.

Git is free software distributed under the GPL.

如果想要查看所有的提交历史记录，可以用git feflog命令：

$ git reflog

4d8abe0 (HEAD -> master) HEAD@{0}: reset: moving to HEAD

4d8abe0 (HEAD -> master) HEAD@{1}: reset: moving to HEAD^

c908bac HEAD@{2}: commit: this commit changed something behind the second line

4d8abe0 (HEAD -> master) HEAD@{3}: commit: add a world: distrubute in the first line

d8231de HEAD@{4}: commit (initial): my first git commit

## 7 工作区和暂存区

学习git，理解工作区和暂存区是非常重要的。

工作区（Working Directory）:myGit文件夹就是一个工作区。

版本库（Repository）:工作区有个隐藏目录 .git ，这个不算工作区，而是 Git 的版本库。

版本库里面的 index(stage) 文件叫暂存区，还有Git为我们自动创建的第一个分支 master ，以及指向 master 的一个指针叫做 HEAD。(刚刚通过HEAD^,实现了git 版本的回退与还原)



前面我们提到过，如果我们想把文件添加到Git里面时，需要分两步：

第一步是用 git add 把文件添加进去，实际上就是把文件修改添加到暂存区。

第二步是用 git commit 提交更改，实际上就是把暂存区的所有内容提交到当前分支。（我们现在只有唯一一个分支 master，所以现在就是往 master 分支上提交更改）

我们可以实践一下：

在 readme.txt 文件中加上一行内容：

Git is a distributed version control system.

Git is free software distributed under the GPL.

Git has a mutable index called stage.

然后在工作区新建一个 LICENSE 文本文档（任意内容）

使用两次 git add 命令分别把 readme.txt 和 LICENSE 都添加后，可以用 git status 命令查看一下：

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: LICENSE

modified: readme.txt

现在，暂存区的状态就变成这样了：



再使用 git commit 命令把暂存区的所有修改提交到分支：

$ git commit -m "understand how stage works"

[master e43a48b] understand how stage works

2 files changed, 2 insertions(+)

create mode 100644 LICENSE

这时候的工作区就是干净的：

$ git status

On branch master

nothing to commit, working tree clean

这时候版本库就变成了这样：



## 8 管理修改

Git 如此的优秀是因为，Git 跟踪并管理的不是文件，而是修改。

我们对 readme.txt 文件进行修改：

$ 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.

然后，添加：

$ git add readme.txt

$ git status

# On branch master

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

# modified: readme.txt

Git tracks changes.

然后再修改 readme.txt 文件：

$ 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.

提交：

$ git commit -m "git tracks changes"

[master 519219b] git tracks changes

1 file changed, 1 insertion(+)

这时候我们查看一下状态：

$ 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")

我们可以发现，第二次修改并没有被提交。

因为在工作区的第一次修改被放入暂存区，准备提交；而在工作区的第二次修改并没有被放入暂存区，所以， git commit 命令只负责把暂存区的修改提交了。

提交后，我们可以用 git diff HEAD -- readme.txt 命令去查看工作区和版本库里面最新版本的区别：

$ git diff HEAD -- readme.txt

diff --git a/readme.txt b/readme.txt

index 76d770f..a9c5755 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 GPL.

Git has a mutable index called stage.

-Git tracks changes.

+Git tracks changes of files.

## 9 撤销修改

假如说你在 readme.txt 文件中添加了一行内容如下：

$ 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.

9.1 没有 git add 之前

可以手动删除最后一行，手动把文件恢复到上一个版本的状态。然后再用 git checkout -- file 命令丢弃工作区的修改：

$ git checkout -- readme.txt //把readme.txt文件在工作区的修改全部撤销。

现在看一下 readme.txt 文件内容：

$ 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.

果然复原了。

9.2 git add了，但没有git commit

这时候的修改添加到了暂存区，但没有提交到分支，用 git status 查看一下：

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

modified: readme.txt

这时候我们可以使用 git reset HEAD file 命令把把暂存区的修改撤销掉，重新放回工作区：

$ git reset HEAD 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，HEAD表示最新版本。

Unstaged changes after reset:

M readme.txt

现在再用 git status 查看一下：

$ 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 //暂存区是干净的，工作区有修改。

这时候再丢弃工作区的修改就OK了：

$ git checkout -- readme.txt //丢弃工作区的修改。

$ git status

On branch master

nothing to commit, working tree clean

现在看一下 readme.txt 文件内容：

$ 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.

果然复原了。终于大功告成了。

(3) 既 git add 了，也 git commit 了

可以回退到上一个版本，见回退版本内容。这个时候应该是没有提交到远程。不然只能祈求boss没有看到咯。

## 10 删除文件

在工作区即myGit文件夹下新建一个test.txt 文件，并添加和提交到Git：

$ git add test.txt

$ git commit -m "add test.txt"

[master b84166e] add test.txt

1 file changed, 1 insertion(+)

create mode 100644 test.txt

这时候可用 rm 命令删除：

$ rm test.txt

此时工作区和版本库就不一样了。

现在又分两种情况：

10.1确实要从版本库中删除该文件，那就用 git rm 命令删除，并且 git commit：

$ git rm test.txt

rm 'test.txt'

$ git commit -m "remove test.txt"

[master d46f35e] remove test.txt

1 file changed, 1 deletion(-)

delete mode 100644 test.txt

这时候文件就从版本库被删除了。

10.2文件被删错了。因为版本库里有，所以很好恢复：

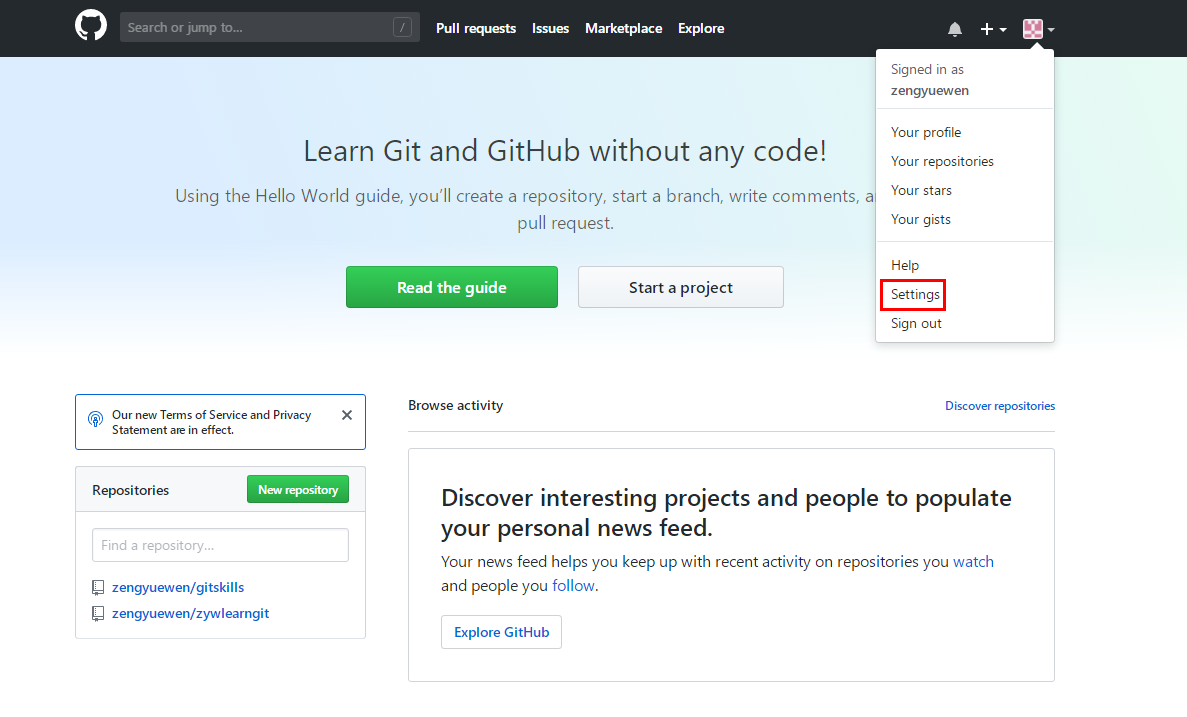
$ git checkout -- test.txt //用版本库里的版本替换工作区的版本。

## 11 远程仓储准备工作

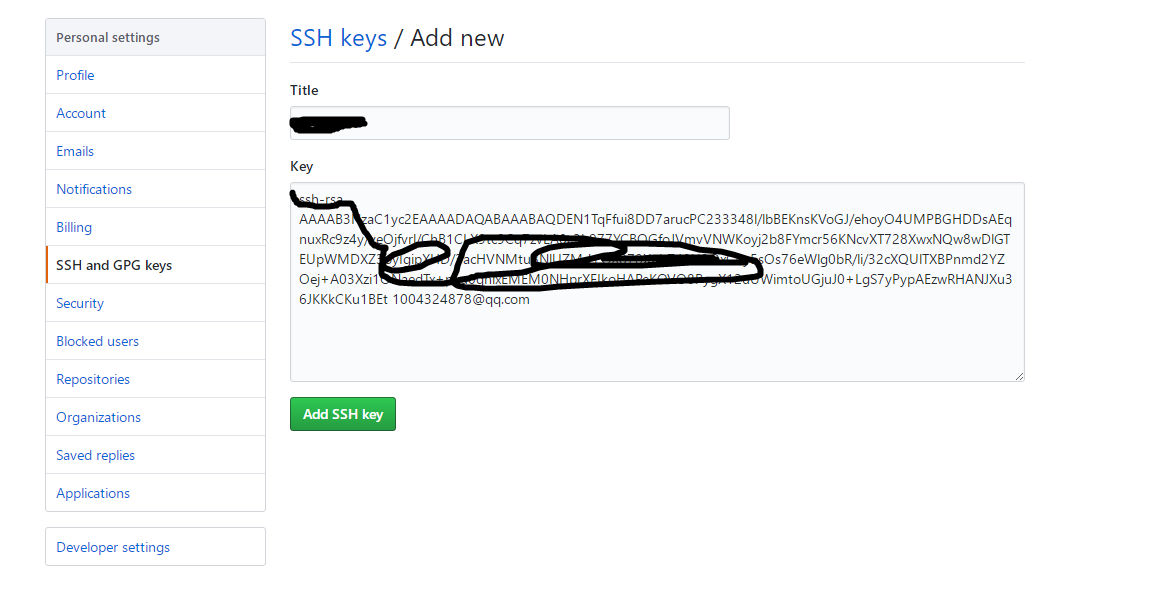
在开始这部分之前，我们需要自行注册GitHub账号。而且，因为你的本地Git仓库和GitHub仓库之间的传输是通过SSH加密的，所以需要设置：

1) 创建SSH Key。在用户主目录下， 看看有没有.ssh 目录，如果有的话，看此目录下有没有 id\_rsa 和 id\_rsa.pub 这两个文件，如果有，直接跳到下一步。如果没有，打开Git Bash，创建SSH

(2) 登陆GitHub，打开"Account settings"



然后点击"Add SSH and GPG Keys"，再点击"New SSH Key"进行SSH Key 的创建，填上任意 Title ，把 id\_rsa.pub 中的内容复制到Key文本框内：



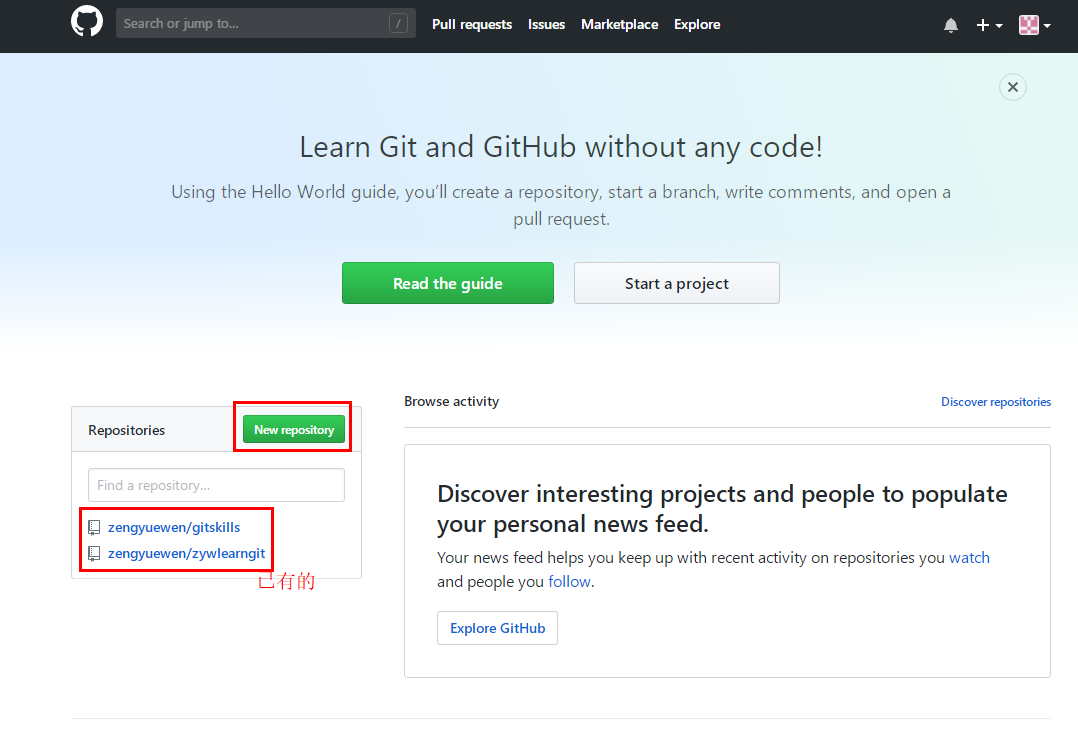
git:如何生成git ssh key（windows下）：

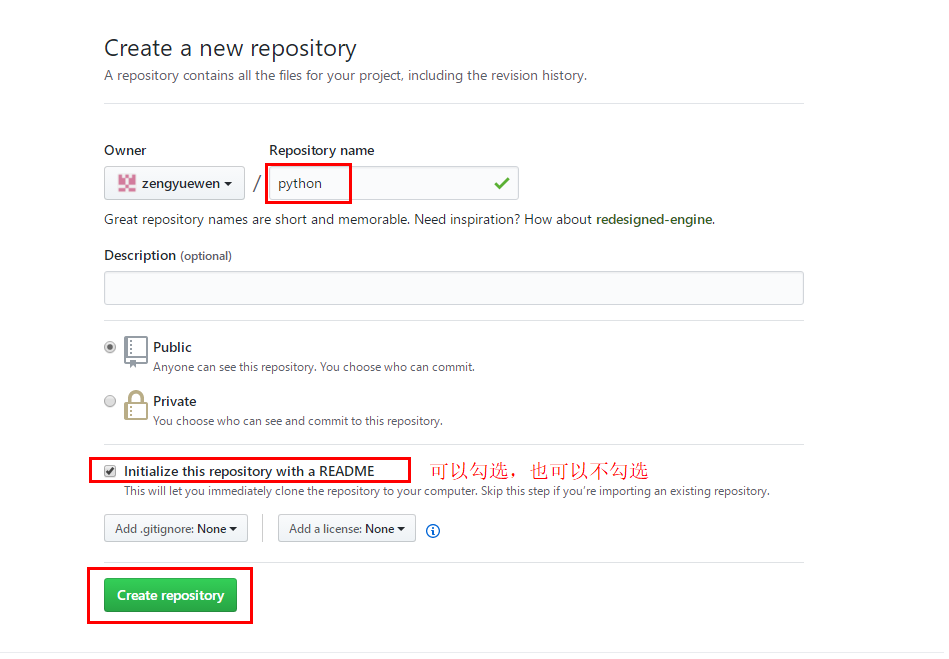
https://blog.csdn.net/lkt\_anhua/article/details/78835226

12 添加远程库

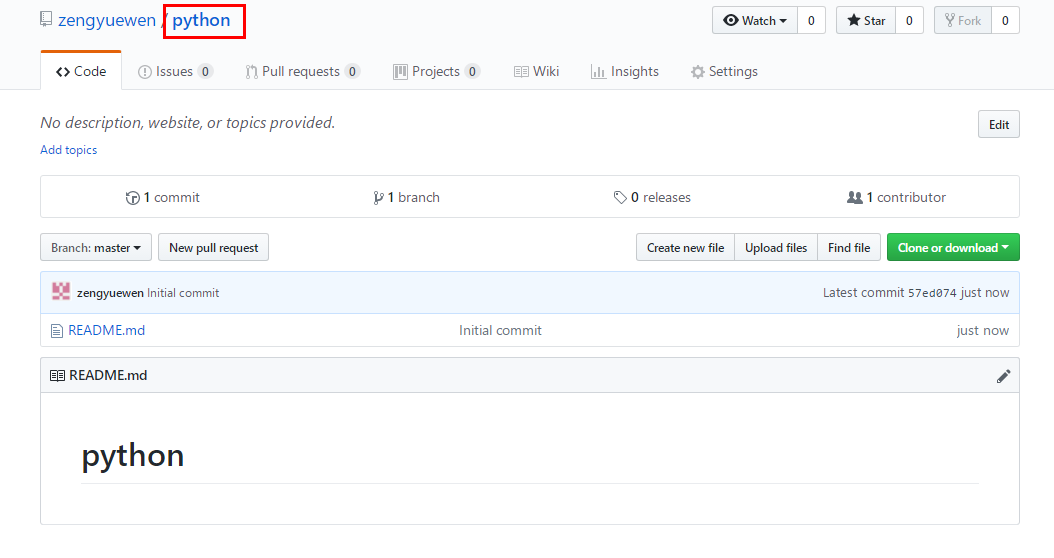
现在我们已经在本地创建了一个Git仓库了，又想在GitHub上创建一个Git仓库，然后让这两个仓库进行远程同步，这样，GitHub上的仓库既可以作为备份，又可以让其他人通过该仓库来协作。那么我们应该怎么做呢？

首先，登陆GitHub，在右上角找到“Create a new repo”按钮，创建一个新的仓库：





在Repository name填入python，勾选initialize thisrepository with a README点击“Create repository”按钮，就成功地创建了一个新的Git仓库：



创建完以后应该是一个空的仓库。这时候我们可以在本地python仓库下运行命令：

$ git remote add origin [git@github.com:zengyuewen/python.git](mailto:git@github.com:zengyuewen/python.git)

//“zengyuewen”是我的GitHub的账户名，你需要填写你自己的账户名

添加后，远程库的名字就是 origin ，这是Git默认的叫法

然后，我们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:RFHzhj/learngit.git

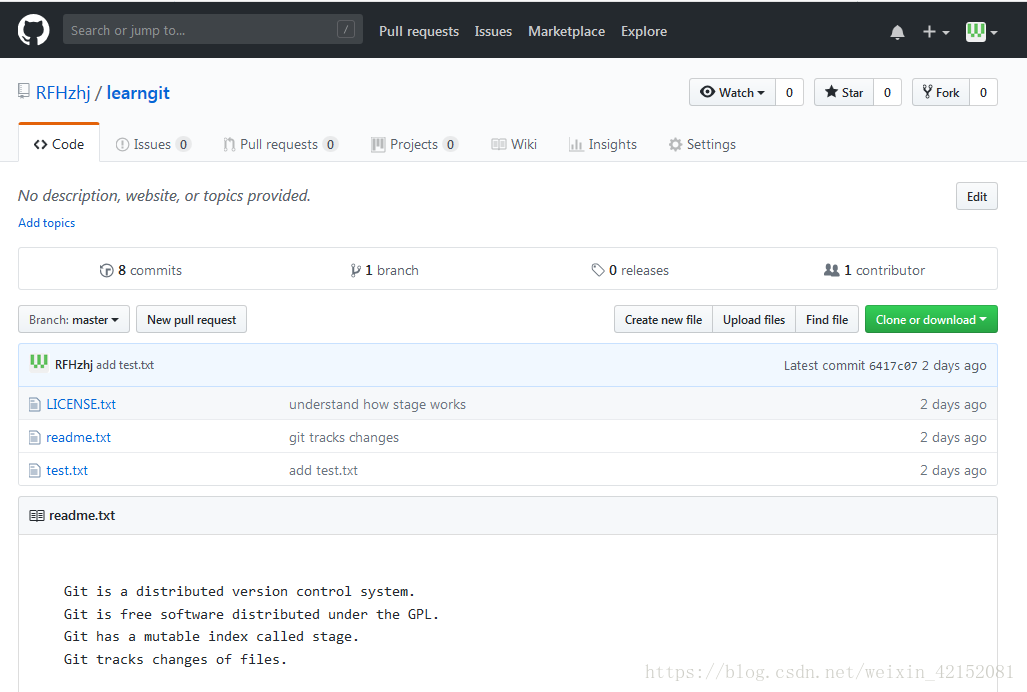
\* [new branch] master -> master

Branch 'master' set up to track remote branch 'master' from 'origin'.

使用 git push 命令，就是把当前分支 master 推送到远程。

因为远程库是空的，所以我们在第一次推送 master 分支时，要加上 -u 参数，Git不但会把本地的 master 分支内容推送的远程新的 master 分支，还会把本地的 master 分支和远程的 master 分支关联起来，在以后的推送或者拉取时就可以简化命令。

推送成功后，可以立刻在GitHub页面中看到远程库的内容已经和本地一模一样：



从现在起，只要本地作了提交，就可以通过命令：

$ git push origin master

把本地 master 分支的最新修改推送至GitHub。现在，我们拥有了真正的分布式版本库。

## 13 从远程库克隆

我们的学习过程是先有了本地库，后有了远程库，然后再对远程库进行关联

其实，我们大部分时候是先创建远程库，然后再从远程库克隆。