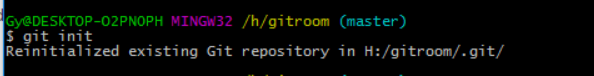
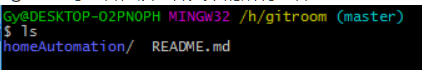
首先在已经安装配置了git环境下鼠标右键快捷键打开 git bash here,然后紧接着就在git控制台输入命令：

$ git init 创建一个本地的仓库



$ git  ls 可查看在该文件夹下的所有文件



$ git status -s  查看项目的当前状态

$ git add test.html（单独添加该目录下的某个文件）或者 $ git add . （添加该目录下的所有文件写入缓存区，这里特别注意点号后加空格，否则会报错）

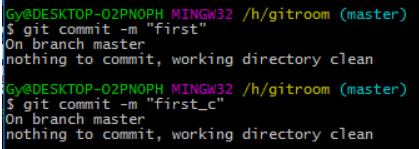


$ git diff 查看执行git  status 的结果的详细信息



到这里已经将想要的快照的内容写入了缓存区，接下来用

$ git commit -m "这里写一些文字作为注释"  将缓存区的内容添加到本地仓库



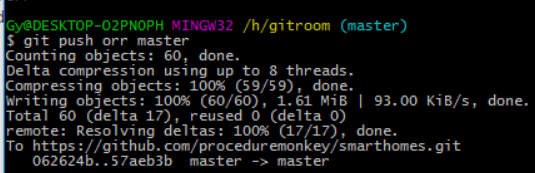
接下来

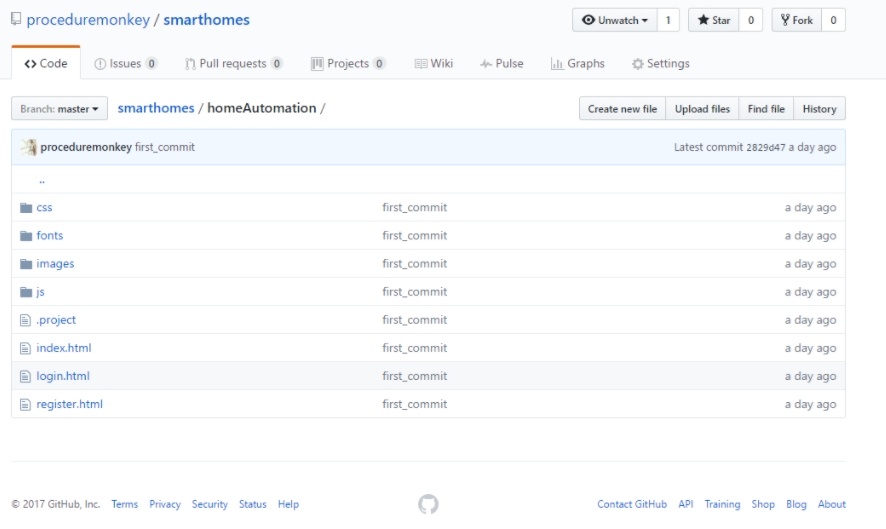
在github官网上新建一个远程仓库，然后将远程仓库的链接url copy下来

$ git remote add origin [url]  连接远程仓库



$ git push origin master 将本地仓库提交到github上面。提交前会让你输入你的github账号和密码。搞定





### **git reset HEAD**

git reset HEAD 命令用于取消已缓存的内容。

我们先改动文件 README 文件，内容如下：

# Runoob Git 测试# 菜鸟教程

hello.php 文件修改为：

<?php

echo '菜鸟教程：www.runoob.com';

echo '菜鸟教程：www.runoob.com';

echo '菜鸟教程：www.runoob.com';?>

现在两个文件修改后，都提交到了缓存区，我们现在要取消其中一个的缓存，操作如下：

$ git status -s

M README

M hello.php

$ git add .

$ git status -s

M README

M hello.pp

$ git reset HEAD -- hello.php Unstaged changes after reset:

M    hello.php

$ git status -s

M README

M hello.php

现在你执行 git commit，只会将 README 文件的改动提交，而 hello.php 是没有的。

$ git commit -m '修改'[master f50cfda] 修改

1 file changed, 1 insertion(+)

$ git status -s

M hello.php

可以看到 hello.php 文件的修改并为提交。

这时我们可以使用以下命令将 hello.php 的修改提交：

$ git commit -am '修改 hello.php 文件'[master 760f74d] 修改 hello.php 文件

1 file changed, 1 insertion(+)

$ git statusOn branch master

nothing to commit, working directory clean

简而言之，执行 git reset HEAD 以取消之前 git add 添加，但不希望包含在下一提交快照中的缓存。

### **git rm**

如果只是简单地从工作目录中手工删除文件，运行 **git status** 时就会在 **Changes not staged for commit** 的提示。

要从 Git 中移除某个文件，就必须要从已跟踪文件清单中移除，然后提交。可以用以下命令完成此项工作

git rm <file>

如果删除之前修改过并且已经放到暂存区域的话，则必须要用强制删除选项 **-f**

git rm -f <file>

如果把文件从暂存区域移除，但仍然希望保留在当前工作目录中，换句话说，仅是从跟踪清单中删除，使用 **--cached** 选项即可

git rm --cached <file>

如我们删除 hello.php文件：

$ git rm hello.php

rm 'hello.php'

$ ls

README

不从工作区中删除文件：

$ git rm --cached README

rm 'README'

$ ls

README

可以递归删除，即如果后面跟的是一个目录做为参数，则会递归删除整个目录中的所有子目录和文件：

git rm –r \*

进入某个目录中，执行此语句，会删除该目录下的所有文件和子目录。

### **git mv**

git mv 命令用于移动或重命名一个文件、目录、软连接。

我们先把刚移除的 README 添加回来：

$ git add README

然后对其重名:

$ git mv README README.md

$ ls

README.md

# Git 分支管理

几乎每一种版本控制系统都以某种形式支持分支。使用分支意味着你可以从开发主线上分离开来，然后在不影响主线的同时继续工作。

有人把 Git 的分支模型称为"必杀技特性"，而正是因为它，将 Git 从版本控制系统家族里区分出来。

创建分支命令：

git branch (branchname)

切换分支命令:

git checkout (branchname)

当你切换分支的时候，Git 会用该分支的最后提交的快照替换你的工作目录的内容， 所以多个分支不需要多个目录。

合并分支命令:

git merge

你可以多次合并到统一分支， 也可以选择在合并之后直接删除被并入的分支。

## Git 分支管理

### **列出分支**

列出分支基本命令：

git branch

没有参数时，git branch 会列出你在本地的分支。

$ git branch\* master

此例的意思就是，我们有一个叫做"master"的分支，并且该分支是当前分支。

当你执行 git init 的时候，缺省情况下 Git 就会为你创建"master"分支。

如果我们要手动创建一个分支。执行 git branch (branchname) 即可。

$ git branch testing

$ git branch\* master

testing

现在我们可以看到，有了一个新分支 testing。

当你以此方式在上次提交更新之后创建了新分支，如果后来又有更新提交， 然后又切换到了"testing"分支，Git 将还原你的工作目录到你创建分支时候的样子

接下来我们将演示如何切换分支，我们用 git checkout (branch) 切换到我们要修改的分支。

$ ls

README

$ echo 'runoob.com' > test.txt

$ git add .

$ git commit -m 'add test.txt'[master 048598f] add test.txt

2 files changed, 1 insertion(+), 3 deletions(-)

delete mode 100644 hello.php

create mode 100644 test.txt

$ ls

README        test.txt

$ git checkout testingSwitched to branch 'testing'

$ ls

README        hello.php

当我们切换到"testing"分支的时候，我们添加的新文件test.txt被移除了, 原来被删除的文件hello.php文件又出现了。切换回"master"分支的时候，它们有重新出现了。

$ git checkout masterSwitched to branch 'master'

$ ls

README        test.txt

我们也可以使用 git checkout -b (branchname) 命令来创建新分支并立即切换到该分支下，从而在该分支中操作。

$ git checkout -b newtestSwitched to a new branch 'newtest'

$ git rm test2.txt

rm 'test2.txt'

$ ls

README        test.txt

$ git commit -am 'removed test2.txt'[newtest 556f0a0] removed test2.txt

1 file changed, 1 deletion(-)

delete mode 100644 test2.txt

$ git checkout masterSwitched to branch 'master'

$ ls

README        test.txt    test2.txt

如你所见，我们创建了一个分支，在该分支的上下文中移除了一些文件，然后切换回我们的主分支，那些文件又回来了。

使用分支将工作切分开来，从而让我们能够在不同上下文中做事，并来回切换。

### **删除分支**

删除分支命令：

git branch -d (branchname)

例如我们要删除"testing"分支：

$ git branch\* master

testing

$ git branch -d testingDeleted branch testing (was 85fc7e7).

$ git branch\* master

### **分支合并**

一旦某分支有了独立内容，你终究会希望将它合并回到你的主分支。 你可以使用以下命令将任何分支合并到当前分支中去：

git merge

$ git branch\* master

newtest

$ ls

README        test.txt    test2.txt

$ git merge newtestUpdating 2e082b7..556f0a0Fast-forward

test2.txt | 1 -

1 file changed, 1 deletion(-)

delete mode 100644 test2.txt

$ ls

README        test.txt

以上实例中我们将 newtest 分支合并到主分支去，test2.txt 文件被删除。

### **合并冲突**

合并并不仅仅是简单的文件添加、移除的操作，Git 也会合并修改。

$ git branch\* master

$ cat test.txt

runoob.com

首先，我们创建一个叫做"change\_site"的分支，切换过去，我们将内容改为 www.runoob.com 。

$ git checkout -b change\_siteSwitched to a new branch 'change\_site'

$ vim test.txt

$ head -1 test.txt

www.runoob.com

$ git commit -am 'changed the site'[change\_site d7e7346] changed the site

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

将修改的内容提交到 "change\_site" 分支中。 现在，假如切换回 "master" 分支我们可以看内容恢复到我们修改前的，我们再次修改test.txt文件。

$ git checkout masterSwitched to branch 'master'

$ head -1 test.txt

www.runoob.com

$ vim test.txt

$ cat test.txt

runoob.com新增加一行

$ git diff

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

index 704cce7..f84c2a4 100644--- a/test.txt+++ b/test.txt@@ -1 +1,2 @@

runoob.com+新增加一行

$ git commit -am '新增加一行'[master 14b4dca] 新增加一行

1 file changed, 1 insertion(+)

现在这些改变已经记录到我的 "master" 分支了。接下来我们将 "change\_site" 分支合并过来。

$ git merge change\_siteAuto-merging test.txt

CONFLICT (content): Merge conflict in test.txtAutomatic merge failed; fix conflicts and then commit the result.

$ cat test.txt <<<<<<< HEAD

runoob.com新增加一行=======

www.runoob.com>>>>>>> change\_site

我们将前一个分支合并到 "master" 分支，一个合并冲突就出现了，接下来我们需要手动去修改它。

$ vim test.txt

$ cat test.txt

www.runoob.com新增加一行

$ git diff

diff --cc test.txt

index f84c2a4,bccb7c2..0000000--- a/test.txt+++ b/test.txt@@@ -1,2 -1,1 +1,2 @@@- runoob.com+ www.runoob.com

+新增加一行

在 Git 中，我们可以用 git add 要告诉 Git 文件冲突已经解决

$ git status -s

UU test.txt

$ git add test.txt

$ git status -s

M test.txt

$ git commit[master 88afe0e] Merge branch 'change\_site'

现在我们成功解决了合并中的冲突，并提交了结果。

# Git 查看提交历史

在使用 Git 提交了若干更新之后，又或者克隆了某个项目，想回顾下提交历史，我们可以使用 git log 命令查看。

针对我们前一章节的操作，使用 git log 命令列出历史提交记录如下：

$ git log

commit 88afe0e02adcdfea6844bb627de97da21eb10af1Merge: 14b4dca d7e7346Author: w3cschool <w3c@w3cschool.cc>Date: Sun Mar 1 15:03:42 2015 +0800

Merge branch 'change\_site'

Conflicts:

test.txt

commit 14b4dcadbdc847207651d5a9fae0d315057f346eAuthor: w3cschool <w3c@w3cschool.cc>Date: Sun Mar 1 14:53:15 2015 +0800

新增加一行

commit d7e734640da06055e107eaf29cf350b3f1de1c2cAuthor: w3cschool <w3c@w3cschool.cc>Date: Sun Mar 1 14:48:57 2015 +0800

changed the site

commit 556f0a0637978097b82287ac665a717623b21f3fAuthor: w3cschool <w3c@w3cschool.cc>Date: Sun Mar 1 14:40:34 2015 +0800

removed test2.txt

我们可以用 --oneline 选项来查看历史记录的简洁的版本。

$ git log --oneline88afe0e Merge branch 'change\_site'14b4dca 新增加一行

d7e7346 changed the site556f0a0 removed test2.txt2e082b7 add test2.txt048598f add test.txt85fc7e7 test comment from w3cschool.cc

这告诉我们的是，此项目的开发历史。

我们还可以用 --graph 选项，查看历史中什么时候出现了分支、合并。以下为相同的命令，开启了拓扑图选项：

$ git log --oneline --graph\* 88afe0e Merge branch 'change\_site'|\ | \* d7e7346 changed the site\* | 14b4dca 新增加一行|/ \* 556f0a0 removed test2.txt\* 2e082b7 add test2.txt\* 048598f add test.txt\* 85fc7e7 test comment from w3cschool.cc

现在我们可以更清楚明了地看到何时工作分叉、又何时归并。

你也可以用 '--reverse'参数来逆向显示所有日志。

$ git log --reverse --oneline85fc7e7 test comment from w3cschool.cc048598f add test.txt2e082b7 add test2.txt556f0a0 removed test2.txt

d7e7346 changed the site14b4dca 新增加一行88afe0e Merge branch 'change\_site'

如果只想查找指定用户的提交日志可以使用命令：git log --author , 例如，比方说我们要找 Git 源码中 Linus 提交的部分：

$ git log --author=Linus --oneline -581b50f3 Move 'builtin-\*' into a 'builtin/' subdirectory3bb7256 make "index-pack" a built-in377d027 make "git pack-redundant" a built-in

b532581 make "git unpack-file" a built-in112dd51 make "mktag" a built-in

如果你要指定日期，可以执行几个选项：--since 和 --before，但是你也可以用 --until 和 --after。

例如，如果我要看 Git 项目中三周前且在四月十八日之后的所有提交，我可以执行这个（我还用了 --no-merges 选项以隐藏合并提交）：

$ git log --oneline --before={3.weeks.ago} --after={2010-04-18} --no-merges5469e2d Git 1.7.1-rc2

d43427d Documentation/remote-helpers: Fix typos and improve language272a36b Fixup: Second argument may be any arbitrary string

b6c8d2d Documentation/remote-helpers: Add invocation section5ce4f4e Documentation/urls: Rewrite to accomodate transport::address00b84e9 Documentation/remote-helpers: Rewrite description03aa87e Documentation: Describe other situations where -z affects git diff77bc694 rebase-interactive: silence warning when no commits rewritten636db2c t3301: add tests to use --format="%N"

更多 git log 命令可查看：[http://git-scm.com/docs/git-log](http://git-scm.com/docs/git-log" \t "http://www.runoob.com/git/_blank)

# Git 标签

如果你达到一个重要的阶段，并希望永远记住那个特别的提交快照，你可以使用 git tag 给它打上标签。

比如说，我们想为我们的 w3cschoolcc 项目发布一个"1.0"版本。 我们可以用 git tag -a v1.0 命令给最新一次提交打上（HEAD）"v1.0"的标签。

-a 选项意为"创建一个带注解的标签"。 不用 -a 选项也可以执行的，但它不会记录这标签是啥时候打的，谁打的，也不会让你添加个标签的注解。 我推荐一直创建带注解的标签。

$ git tag -a v1.0

当你执行 git tag -a 命令时，Git 会打开你的编辑器，让你写一句标签注解，就像你给提交写注解一样。

现在，注意当我们执行 git log --decorate 时，我们可以看到我们的标签了：

$ git log --oneline --decorate --graph

\* 88afe0e (HEAD, tag: v1.0, master) Merge branch 'change\_site'

|\

| \* d7e7346 (change\_site) changed the site

\* | 14b4dca 新增加一行

|/

\* 556f0a0 removed test2.txt

\* 2e082b7 add test2.txt

\* 048598f add test.txt

\* 85fc7e7 test comment from w3cschool.cc

如果我们忘了给某个提交打标签，又将它发布了，我们可以给它追加标签。

例如，假设我们发布了提交 85fc7e7(上面实例最后一行)，但是那时候忘了给它打标签。 我们现在也可以：

$ git tag -a v0.9 85fc7e7

$ git log --oneline --decorate --graph

\* 88afe0e (HEAD, tag: v1.0, master) Merge branch 'change\_site'

|\

| \* d7e7346 (change\_site) changed the site

\* | 14b4dca 新增加一行

|/

\* 556f0a0 removed test2.txt

\* 2e082b7 add test2.txt

\* 048598f add test.txt

\* 85fc7e7 (tag: v0.9) test comment from w3cschool.cc

如果我们要查看所有标签可以使用以下命令：

$ git tag

v0.9

v1.0

指定标签信息命令：

git tag -a <tagname> -m "w3cschool.cc标签"

PGP签名标签命令：

git tag -s <tagname> -m "w3cschool.cc标签"

# Git 远程仓库(Github)

Git 并不像 SVN 那样有个中心服务器。

目前我们使用到的 Git 命令都是在本地执行，如果你想通过 Git 分享你的代码或者与其他开发人员合作。 你就需要将数据放到一台其他开发人员能够连接的服务器上。

本例使用了 Github 作为远程仓库，你可以先阅读我们的 [Github 简明教程。](http://www.runoob.com/w3cnote/git-guide.html" \t "http://www.runoob.com/git/_blank)

## 添加远程库

要添加一个新的远程仓库，可以指定一个简单的名字，以便将来引用,命令格式如下：

git remote add [shortname] [url]

本例以Github为例作为远程仓库，如果你没有Github可以在官网[https://github.com/](https://github.com/" \t "http://www.runoob.com/git/_blank)注册。

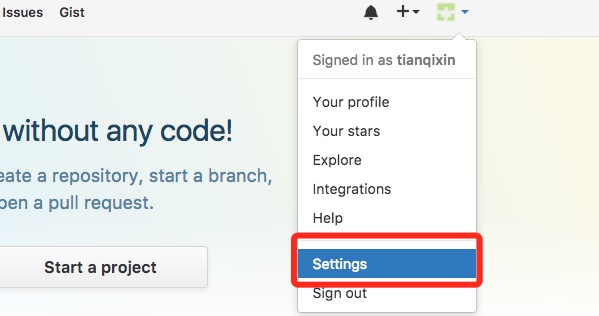
由于你的本地Git仓库和GitHub仓库之间的传输是通过SSH加密的，所以我们需要配置验证信息：

使用以下命令生成SSH Key：

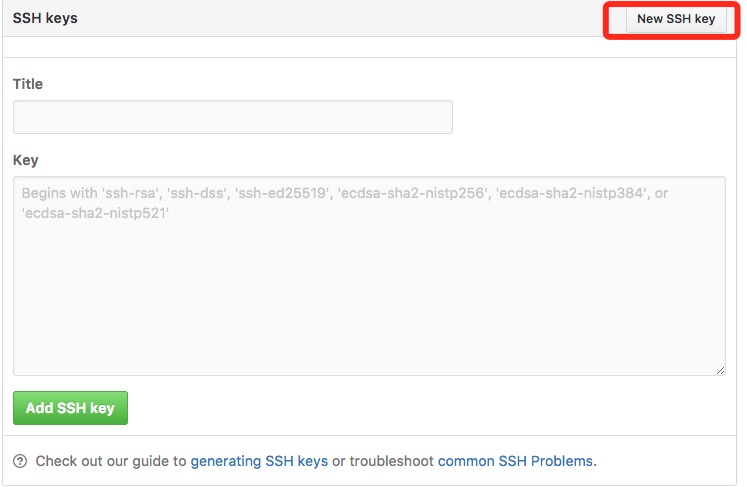
$ ssh-keygen -t rsa -C "youremail@example.com"

后面的 **your\_email@youremail.com** 改为你在 github 上注册的邮箱，之后会要求确认路径和输入密码，我们这使用默认的一路回车就行。成功的话会在~/下生成.ssh文件夹，进去，打开 id\_rsa.pub，复制里面的 key。

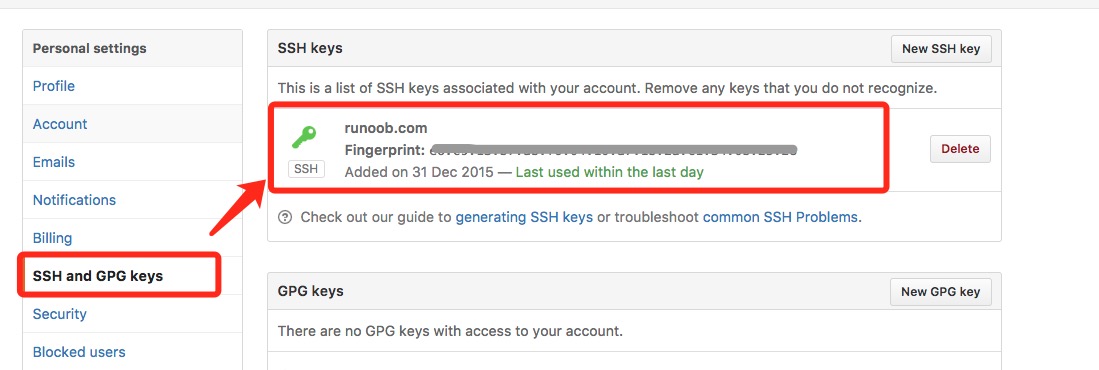
回到 github 上，进入 Account => Settings（账户配置）。



左边选择 **SSH and GPG keys**，然后点击 **New SSH key** 按钮,title 设置标题，可以随便填，粘贴在你电脑上生成的 key。



添加成功后界面如下所示

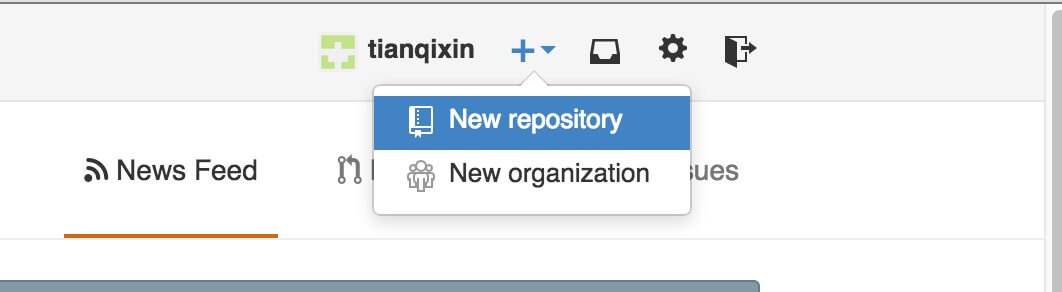


为了验证是否成功，输入以下命令：

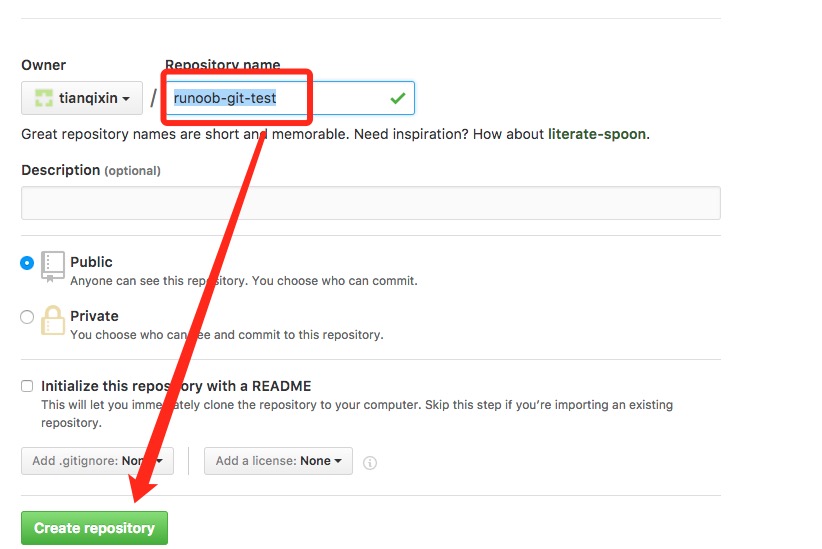
$ ssh -T git@github.comHi tianqixin! You've successfully authenticated, but GitHub does not provide shell access.

以下命令说明我们已成功连上 Github。

之后登录后点击" New repository " 如下图所示：



之后在在Repository name 填入 runoob-git-test(远程仓库名) ，其他保持默认设置，点击"Create repository"按钮，就成功地创建了一个新的Git仓库：



创建成功后，显示如下信息：



以上信息告诉我们可以从这个仓库克隆出新的仓库，也可以把本地仓库的内容推送到GitHub仓库。

现在，我们根据 GitHub 的提示，在本地的仓库下运行命令：

$ mkdir runoob-git-test # 创建测试目录

$ cd runoob-git-test/ # 进入测试目录

$ echo "# 菜鸟教程 Git 测试" >> README.md # 创建 README.md 文件并写入内容

$ ls # 查看目录下的文件

README

$ git init # 初始化

$ git add README.md # 添加文件

$ git commit -m "添加 README.md 文件" # 提交并备注信息[master (root-commit) 0205aab] 添加 README.md 文件

1 file changed, 1 insertion(+)

create mode 100644 README.md

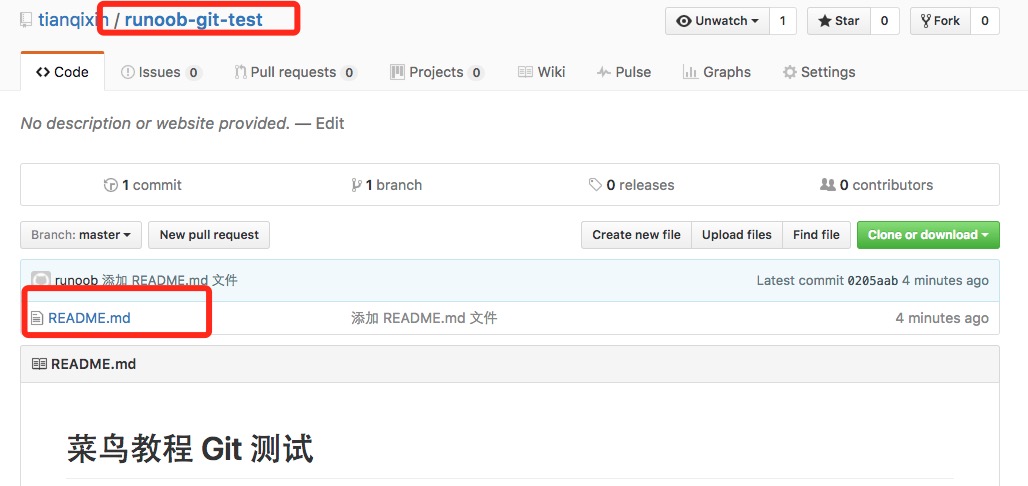
# 提交到 Github

$ git remote add origin git@github.com:tianqixin/runoob-git-test.git

$ git push -u origin master

以下命令请根据你在Github成功创建新仓库的地方复制，而不是根据我提供的命令，因为我们的Github用户名不一样，仓库名也不一样。

接下来我们返回 Github 创建的仓库，就可以看到文件已上传到 Github上：



## 查看当前的远程库

要查看当前配置有哪些远程仓库，可以用命令：

git remote

### **实例**

$ git remote

origin

$ git remote -v

origin    git@github.com:tianqixin/runoob-git-test.git (fetch)

origin    git@github.com:tianqixin/runoob-git-test.git (push)

执行时加上 -v 参数，你还可以看到每个别名的实际链接地址。

## 提取远程仓库

Git 有两个命令用来提取远程仓库的更新。

1、从远程仓库下载新分支与数据：

git fetch

该命令执行完后需要执行git merge 远程分支到你所在的分支。

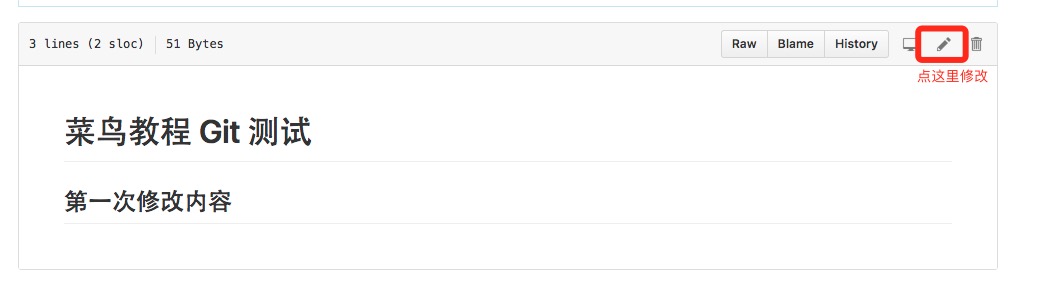
2、从远端仓库提取数据并尝试合并到当前分支：

git merge

该命令就是在执行 git fetch 之后紧接着执行 git merge 远程分支到你所在的任意分支。

假设你配置好了一个远程仓库，并且你想要提取更新的数据，你可以首先执行 **git fetch [alias]** 告诉 Git 去获取它有你没有的数据，然后你可以执行 **git merge [alias]/[branch]** 以将服务器上的任何更新（假设有人这时候推送到服务器了）合并到你的当前分支。

接下来我们在 Github 上点击" README.md" 并在线修改它:



然后我们在本地更新修改。

$ git fetch origin

remote: Counting objects: 3, done.

remote: Compressing objects: 100% (2/2), done.

remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0Unpacking objects: 100% (3/3), done.From github.com:tianqixin/runoob-git-test

0205aab..febd8ed master -> origin/master

以上信息"0205aab..febd8ed master -> origin/master" 说明 master 分支已被更新，我们可以使用以下命令将更新同步到本地：

$ git merge origin/masterUpdating 0205aab..febd8edFast-forward

README.md | 1 +

1 file changed, 1 insertion(+)

查看 README.md 文件内容：

$ cat README.md # 菜鸟教程 Git 测试## 第一次修改内容

## 推送到远程仓库

推送你的新分支与数据到某个远端仓库命令:

git push [alias] [branch]

以上命令将你的 [branch] 分支推送成为 [alias] 远程仓库上的 [branch] 分支，实例如下。

$ touch runoob-test.txt # 添加文件

$ git add runoob-test.txt

$ git commit -m "添加到远程"

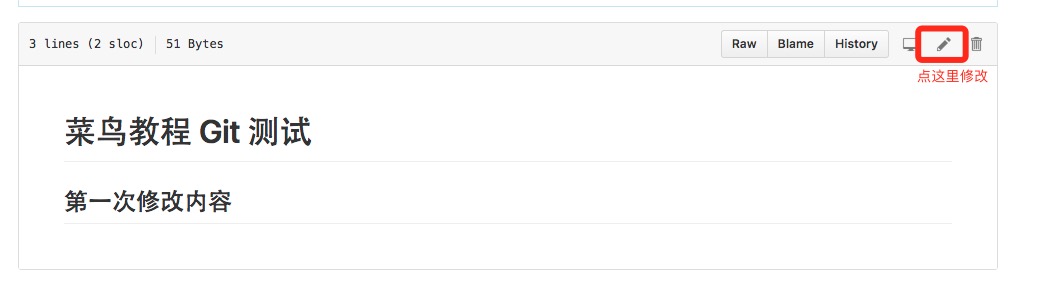
master 69e702d] 添加到远程

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 runoob-test.txt

$ git push origin master # 推送到 Github

重新回到我们的 Github 仓库，可以看到文件以及提交上来了：



## 删除远程仓库

删除远程仓库你可以使用命令：

git remote rm [别名]

### **实例**

$ git remote -v

origin    git@github.com:tianqixin/runoob-git-test.git (fetch)

origin    git@github.com:tianqixin/runoob-git-test.git (push)

# 添加仓库 origin2

$ git remote add origin2 git@github.com:tianqixin/runoob-git-test.git

$ git remote -v

origin    git@github.com:tianqixin/runoob-git-test.git (fetch)

origin    git@github.com:tianqixin/runoob-git-test.git (push)

origin2    git@github.com:tianqixin/runoob-git-test.git (fetch)

origin2    git@github.com:tianqixin/runoob-git-test.git (push)

# 删除仓库 origin2

$ git remote rm origin2

$ git remote -v

origin    git@github.com:tianqixin/runoob-git-test.git (fetch)

origin    git@github.com:tianqixin/runoob-git-test.git (push)

# 如何解决failed to push some refs to git

听语音

* |
* 浏览：22287
* |
* 更新：2015-07-28 09:55
* |
* 标签：[it](https://jingyan.baidu.com/tag?tagName=it" \t "https://jingyan.baidu.com/article/_blank)

[](http://jingyan.baidu.com/album/f3e34a12a25bc8f5ea65354a.html?picindex=1)1

[](http://jingyan.baidu.com/album/f3e34a12a25bc8f5ea65354a.html?picindex=2)2

[](http://jingyan.baidu.com/album/f3e34a12a25bc8f5ea65354a.html?picindex=3)3

[](http://jingyan.baidu.com/album/f3e34a12a25bc8f5ea65354a.html?picindex=4)4

[](http://jingyan.baidu.com/album/f3e34a12a25bc8f5ea65354a.html?picindex=5)5

[](http://jingyan.baidu.com/album/f3e34a12a25bc8f5ea65354a.html?picindex=6)6

[](http://jingyan.baidu.com/album/f3e34a12a25bc8f5ea65354a.html?picindex=7)7

[分步阅读](http://jingyan.baidu.com/album/f3e34a12a25bc8f5ea65354a.html)

Administrator@PC-20150110FGWU /K/cocos2d/yc (master)

$ git push -u origin master

To git@github.com:yangchao0718/cocos2d.git

 ! [rejected]        master -> master (non-fast-forward)

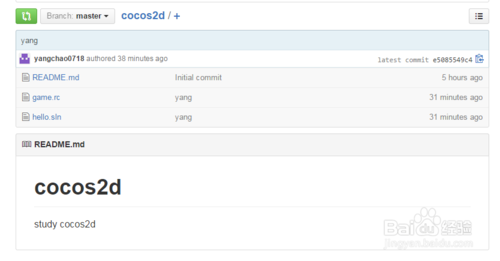
error: failed to push some refs to 'git@github.com:yangchao0718/cocos2d.git

hint: Updates were rejected because the tip of your current branch is behin

hint: its remote counterpart. Integrate the remote changes (e.g.

hint: 'git pull ...') before pushing again.

hint: See the 'Note about fast-forwards' in 'git push --help' for details.

[](http://jingyan.baidu.com/album/f3e34a12a25bc8f5ea65354a.html?picindex=1)

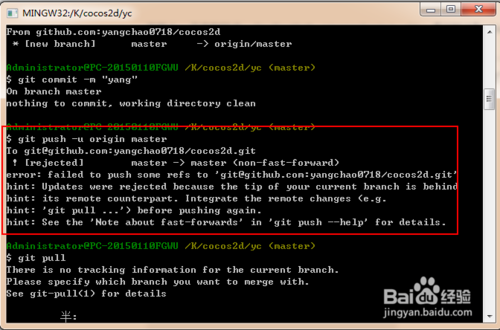
## 工具/原料

git

## 方法/步骤

1

在使用git 对源代码进行push到gitHub时可能会出错，信息如下

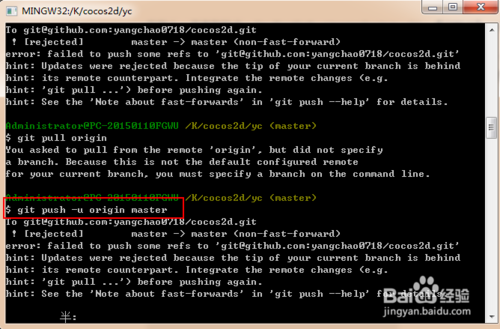
[](http://jingyan.baidu.com/album/f3e34a12a25bc8f5ea65354a.html?picindex=2)

2

此时很多人会尝试下面的命令把当前分支代码上传到master分支上。

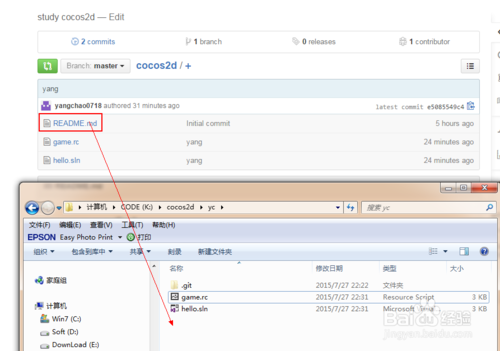
$ git push -u origin master

但依然没能解决问题

[](http://jingyan.baidu.com/album/f3e34a12a25bc8f5ea65354a.html?picindex=3)

3

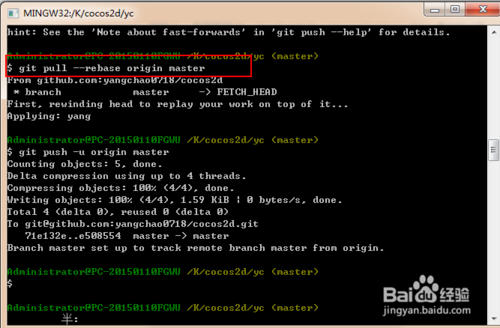
出现错误的主要原因是github中的README.md文件不在本地代码目录中

[](http://jingyan.baidu.com/album/f3e34a12a25bc8f5ea65354a.html?picindex=4)

4

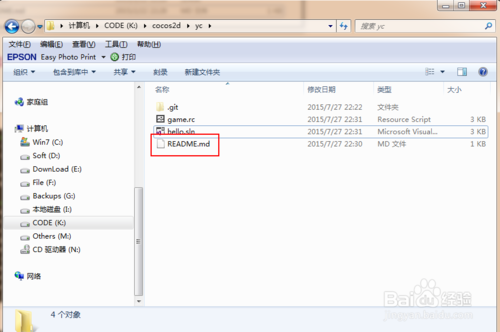
可以通过如下命令进行代码合并【注：pull=fetch+merge]

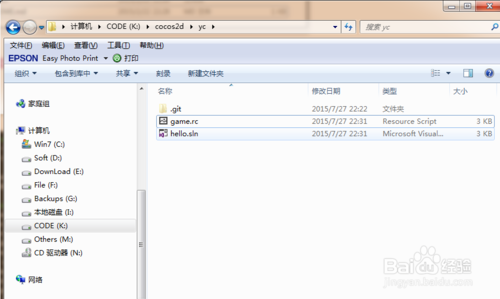
git pull --rebase origin master

[](http://jingyan.baidu.com/album/f3e34a12a25bc8f5ea65354a.html?picindex=5)

5

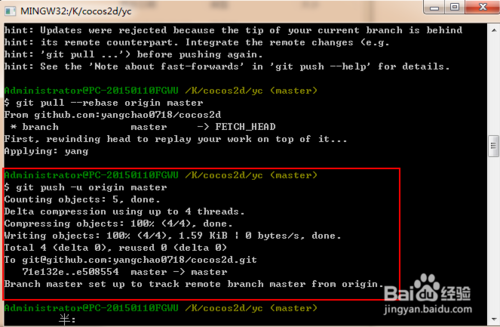
执行上面代码后可以看到本地代码库中多了README.md文件

[](http://jingyan.baidu.com/album/f3e34a12a25bc8f5ea65354a.html?picindex=6)

[](http://jingyan.baidu.com/album/f3e34a12a25bc8f5ea65354a.html?picindex=7)

6

此时再执行语句 git push -u origin master即可完成代码上传到github

[](http://jingyan.baidu.com/album/f3e34a12a25bc8f5ea65354a.html?picindex=8)

Git pull -p 同步远程分支到本地