# Git

Git 是 Linus Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。

所以git类似与Linux，可以使用Linux命令操作git

# 安装

Linux 平台上安装

### Debian/Ubuntu

Debian/Ubuntu Git 安装命令为：

$ apt-get install libcurl4-gnutls-dev libexpat1-dev gettext \

libz-dev libssl-dev

$ apt-get install git-core

$ git --version

git version 1.8.1.2

### Centos/RedHat

如果你使用的系统是 Centos/RedHat 安装命令为：

$ yum install curl-devel expat-devel gettext-devel \

openssl-devel zlib-devel

$ yum -y install git-core

$ git --version

git version 1.7.1

## Git 配置

Git 提供了一个叫做 git config 的工具，专门用来配置或读取相应的工作环境变量。

这些环境变量，决定了 Git 在各个环节的具体工作方式和行为。这些变量可以存放在以下三个不同的地方：

* /etc/gitconfig 文件：系统中对所有用户都普遍适用的配置。若使用 git config 时用 --system 选项，读写的就是这个文件。
* ~/.gitconfig 文件：用户目录下的配置文件只适用于该用户。若使用 git config 时用 --global 选项，读写的就是这个文件。
* 当前项目的 Git 目录中的配置文件（也就是工作目录中的 .git/config 文件）：这里的配置仅仅针对当前项目有效。每一个级别的配置都会覆盖上层的相同配置，所以 .git/config 里的配置会覆盖 /etc/gitconfig 中的同名变量。

在 Windows 系统上，Git 会找寻用户主目录下的 .gitconfig 文件。主目录即 $HOME 变量指定的目录，一般都是 C:\Documents and Settings\$USER。

### 用户信息

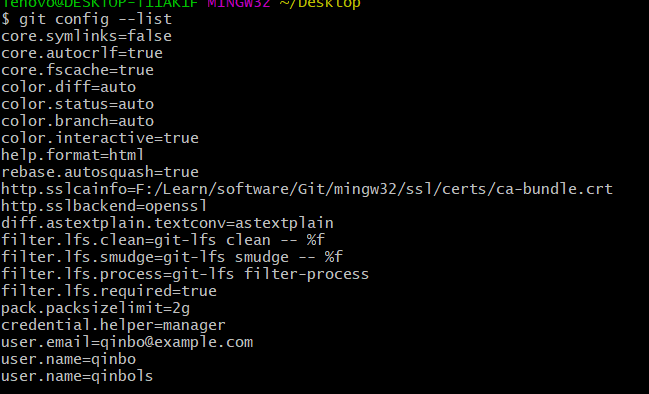
配置个人的用户名称和电子邮件地址：

$ git config --global user.name "runoob"

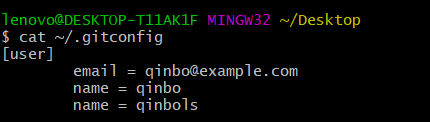
$ git config --global user.email test@runoob.com

### 查看配置信息

要检查已有的配置信息，可以使用 git config --list 命令：

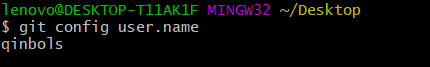


可以在 **~/.gitconfig** 或 **/etc/gitconfig** 看到，如下所示：



也可以直接查阅某个环境变量的设定，只要把特定的名字跟在后面即可，像这样：

$ git config user.name



# Git 工作流程

工作流程如下：

* 克隆 Git 资源作为工作目录。
* 在克隆的资源上添加或修改文件。
* 如果其他人修改了，你可以更新资源。
* 在提交前查看修改。
* 提交修改。
* 在修改完成后，如果发现错误，可以撤回提交并再次修改并提交。



# Git 创建仓库

本章节我们将为大家介绍如何创建一个 Git 仓库。

你可以使用一个已经存在的目录作为Git仓库。

## git init

Git 使用 **git init** 命令来初始化一个 Git 仓库，Git 的很多命令都需要在 Git 的仓库中运行，所以 **git init** 是使用 Git 的第一个命令。

在执行完成 **git init** 命令后，Git 仓库会生成一个 .git 目录，该目录包含了资源的所有元数据，其他的项目目录保持不变（不像 SVN 会在每个子目录生成 .svn 目录，Git 只在仓库的根目录生成 .git 目录）。

### 使用方法

使用当前目录作为Git仓库，我们只需使它初始化。

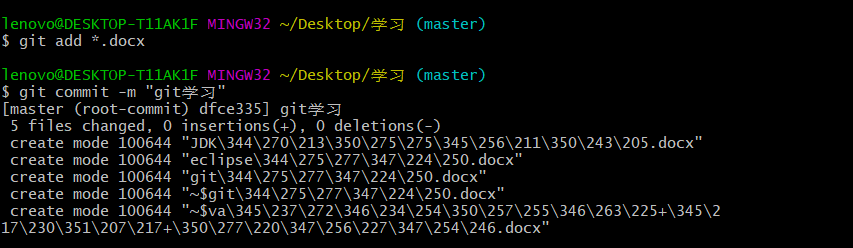
git init

该命令执行完后会在当前目录生成一个 .git 目录。

使用我们指定目录作为Git仓库。

git init newrepo

初始化后，会在 newrepo 目录下会出现一个名为 .git 的目录，所有 Git 需要的数据和资源都存放在这个目录中。



## git基本操作

### git clone

我们使用 **git clone** 从现有 Git 仓库中拷贝项目（类似 **svn checkout**）。

克隆仓库的命令格式为：

git clone <repo>

如果我们需要克隆到指定的目录，可以使用以下命令格式：

git clone <repo> <directory>

**参数说明：**

* **repo:**Git 仓库。
* **directory:**本地目录。

比如，要克隆 Ruby 语言的 Git 代码仓库 Grit，可以用下面的命令：

$ git clone git://github.com/schacon/grit.git

执行该命令后，会在当前目录下创建一个名为grit的目录，其中包含一个 .git 的目录，用于保存下载下来的所有版本记录。

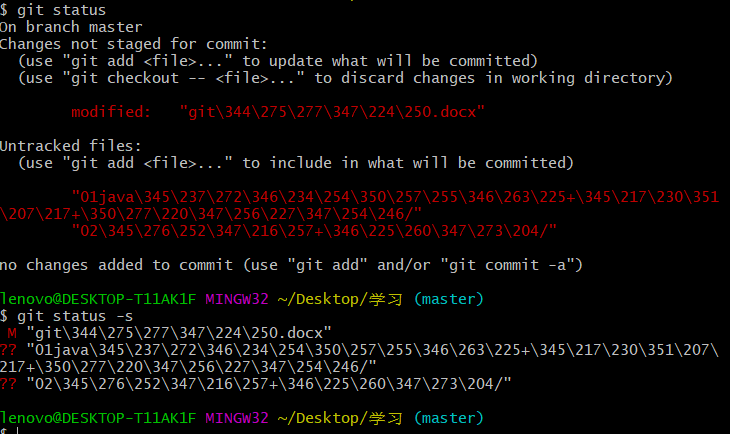
如果要自己定义要新建的项目目录名称，可以在上面的命令末尾指定新的名字：

$ git clone git://github.com/schacon/grit.git mygrit

### git status

git status 以查看在你上次提交之后是否有修改。

我演示该命令的时候加了 -s 参数，以获得简短的结果输出。如果没加该参数会详细输出内容：



### git diff

执行 git diff 来查看执行 git status 的结果的详细信息。

git diff 命令显示已写入缓存与已修改但尚未写入缓存的改动的区别。git diff 有两个主要的应用场景。

* 尚未缓存的改动：**git diff**
* 查看已缓存的改动： **git diff --cached**
* 查看已缓存的与未缓存的所有改动：**git diff HEAD**
* 显示摘要而非整个 diff：**git diff --stat**

### git commit

使用 git add 命令将想要快照的内容写入缓存区， 而执行 git commit 将缓存区内容添加到仓库中。

Git 为你的每一个提交都记录你的名字与电子邮箱地址，所以第一步需要配置用户名和邮箱地址。

$ git config --global user.name 'runoob'

$ git config --global user.email test@runoob.com

接下来我们写入缓存，并提交对 hello.php 的所有改动。在首个例子中，我们使用 -m 选项以在命令行中提供提交注释。

$ git add hello.php

$ git status -s

A README

A hello.php

$ $ git commit -m '第一次版本提交'

[master (root-commit) d32cf1f] 第一次版本提交

2 files changed, 4 insertions(+)

create mode 100644 README

create mode 100644 hello.php

现在我们已经记录了快照。如果我们再执行 git status:

$ git status

# On branch master

nothing to commit (working directory clean)

以上输出说明我们在最近一次提交之后，没有做任何改动，是一个"working directory clean：干净的工作目录"。

如果你没有设置 -m 选项，Git 会尝试为你打开一个编辑器以填写提交信息。 如果 Git 在你对它的配置中找不到相关信息，默认会打开 vim。屏幕会像这样：

# Please enter the commit message for your changes. Lines starting

# with '#' will be ignored, and an empty message aborts the commit.

# On branch master

# Changes to be committed:

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

#

# modified: hello.php

#

~

~

".git/COMMIT\_EDITMSG" 9L, 257C

如果你觉得 git add 提交缓存的流程太过繁琐，Git 也允许你用 -a 选项跳过这一步。命令格式如下：

git commit -a

我们先修改 hello.php 文件为以下内容：

<?php

echo '菜鸟教程：www.runoob.com';

echo '菜鸟教程：www.runoob.com';

?>

再执行以下命令：

git commit -am '修改 hello.php 文件'

[master 71ee2cb] 修改 hello.php 文件

1 file changed, 1 insertion(+)

### git reset HEAD

git reset HEAD 命令用于取消已缓存的内容。

### 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>

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 \*

### git mv

git mv 命令用于移动或重命名一个文件、目录、软连接。

# 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)

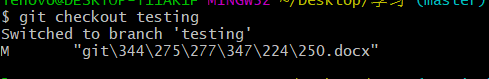
切换分支命令:

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



### 列出分支

列出分支基本命令：

git branch

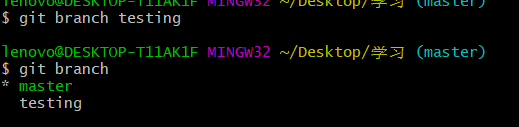
没有参数时，git branch 会列出你在本地的分支。



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。

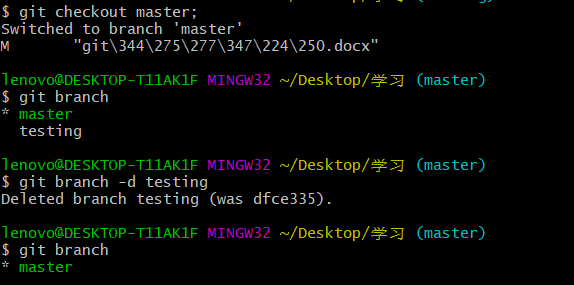


### 删除分支

删除分支命令：

git branch -d (branchname)

例如我们要删除"testing"分支：



### 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 newtest

Updating 2e082b7..556f0a0

Fast-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\_site

Switched 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 master

Switched to branch 'master'

$ head -1 test.txt

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\_site

Auto-merging test.txt

CONFLICT (content): Merge conflict in test.txt

Automatic 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

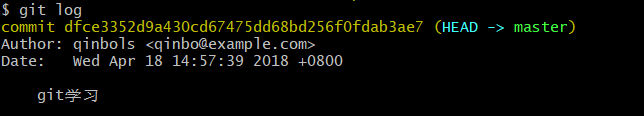
[master 88afe0e] Merge branch 'change\_site'

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：



我们可以用 --oneline 选项来查看历史记录的简洁的版本。



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：

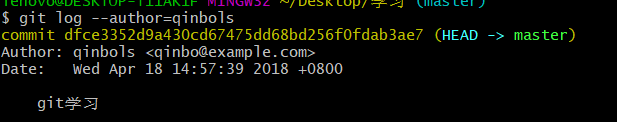
$ git log --oneline --graph

可以用 '--reverse'参数来逆向显示所有日志。

$ git log --reverse --oneline

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 -5



# 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 runoob 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

指定标签信息命令：

git tag -a <tagname> -m "runoob.com标签"

PGP签名标签命令：

git tag -s <tagname> -m "runoob.com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)

## 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 "_blank)注册。

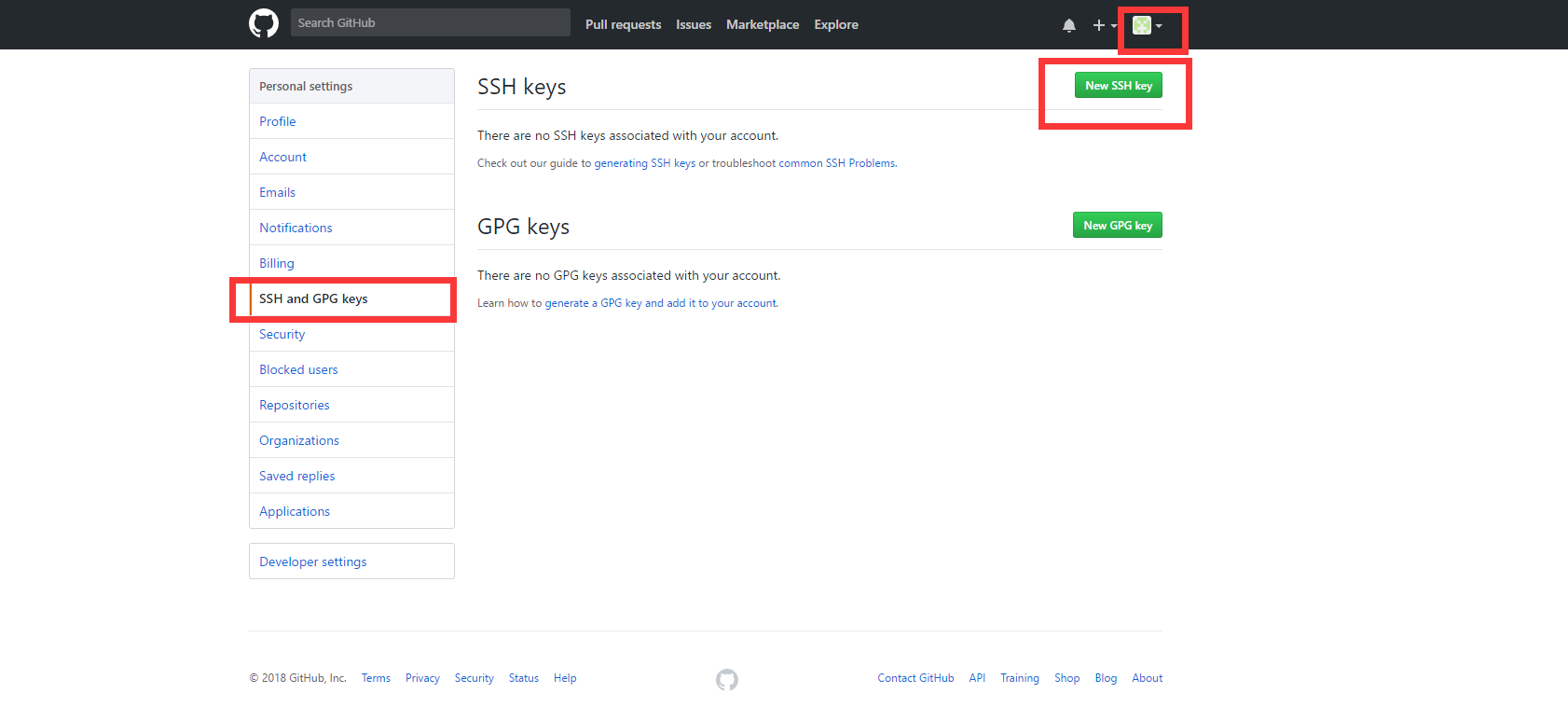
由于你的本地Git仓库和GitHub仓库之间的传输是通过SSH加密的，所以我们需要配置验证信息：

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）。



### Github 安装

* [下载 git OSX 版](http://code.google.com/p/git-osx-installer/downloads/list?can=3)
* [下载 git Windows 版](http://msysgit.github.io/)
* [下载 git Linux 版](http://book.git-scm.com/2_installing_git.html)

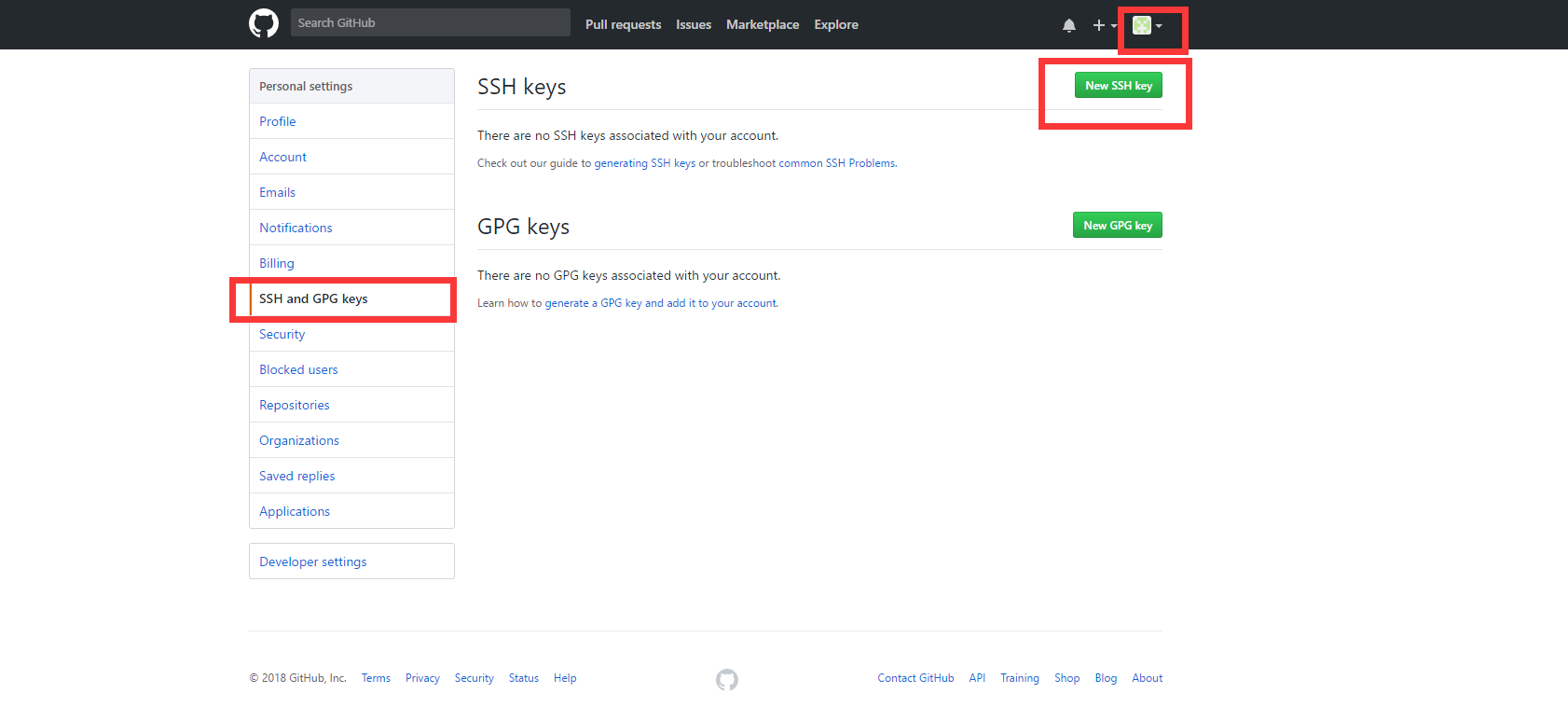
### 配置Git

首先在本地创建ssh key；

$ ssh-keygen -t rsa -C "your\_email@youremail.com"

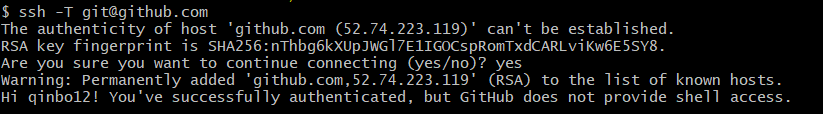
后面的your\_email@youremail.com改为你在github上注册的邮箱，之后会要求确认路径和输入密码，我们这使用默认的一路回车就行。成功的话会在~/下生成.ssh文件夹，进去，打开id\_rsa.pub，复制里面的key。

回到github上，进入 Account Settings（账户配置），左边选择SSH Keys，Add SSH Key,title随便填，粘贴在你电脑上生成的key。



为了验证是否成功，在git bash下输入：

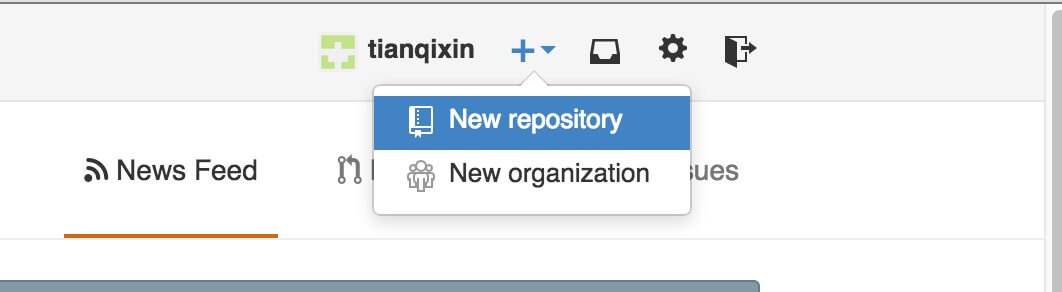
$ ssh -T git@github.com



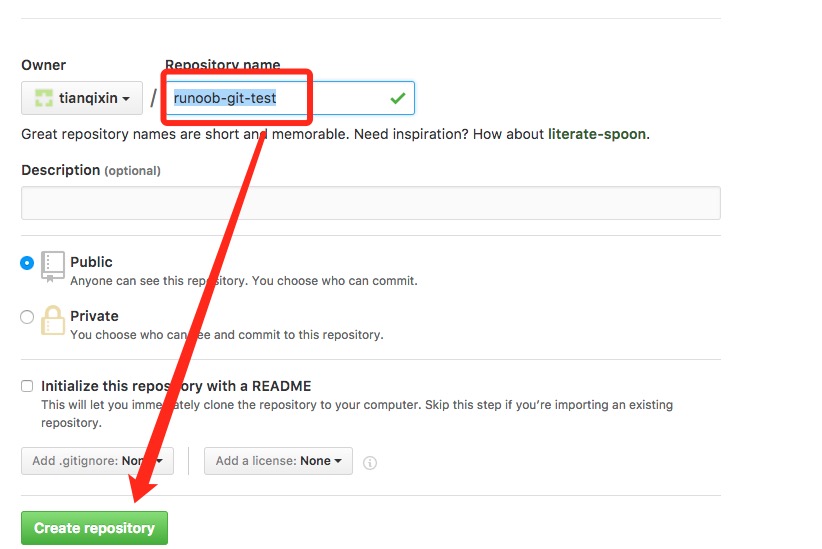
如果是第一次的会提示是否continue，输入yes就会看到：You've successfully authenticated, but GitHub does not provide shell access 。这就表示已成功连上github。

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。

接下来我们要做的就是把本地仓库传到github上去，在此之前还需要设置username和email，因为github每次commit都会记录他们。

$ git config --global user.name "your name"

$ git config --global user.email "your\_email@youremail.com"

进入要上传的仓库，右键git bash，添加远程地址：

$ git remote add origin git@github.com:yourName/yourRepo.git

后面的yourName和yourRepo表示你再github的用户名和刚才新建的仓库，加完之后进入.git，打开config，这里会多出一个remote "origin"内容，这就是刚才添加的远程地址，也可以直接修改config来配置远程地址。

<h3创建新仓库< h3="" style="color: rgb(51, 51, 51); font-family: "Helvetica Neue", Helvetica, "PingFang SC", "Hiragino Sans GB", "Microsoft YaHei", "Noto Sans CJK SC", "WenQuanYi Micro Hei", Arial, sans-serif; font-size: 13px; font-style: normal; font-variant-ligatures: normal; font-variant-caps: normal; font-weight: normal; letter-spacing: normal; orphans: 2; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 2; word-spacing: 0px; -webkit-text-stroke-width: 0px; background-color: rgb(255, 255, 255);">

创建新文件夹，打开，然后执行 git init 以创建新的 git 仓库。

</h3创建新仓库<>

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

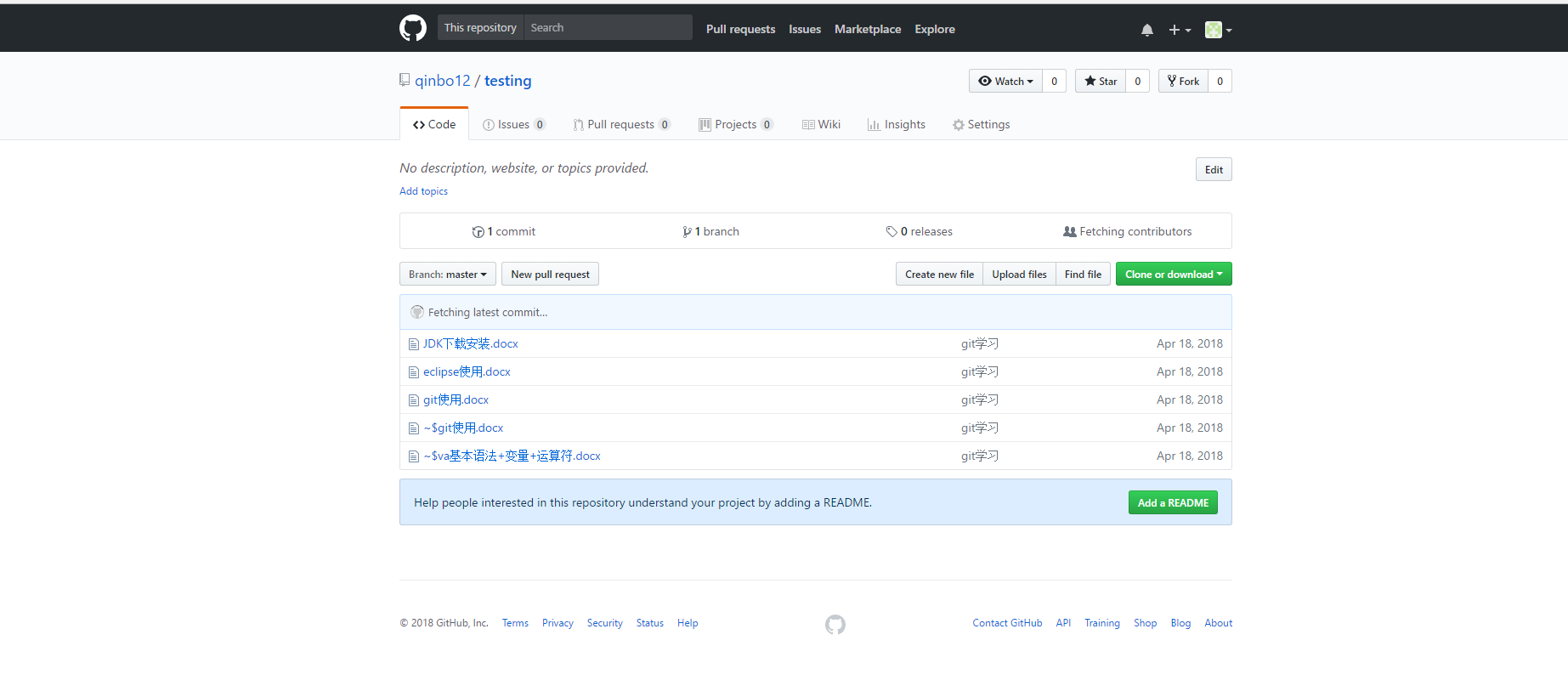
# 提交到 Github

$ 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：



### 检出仓库

执行如下命令以创建一个本地仓库的克隆版本：

git clone /path/to/repository

如果是远端服务器上的仓库，你的命令会是这个样子：

git clone username@host:/path/to/repository

## 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。

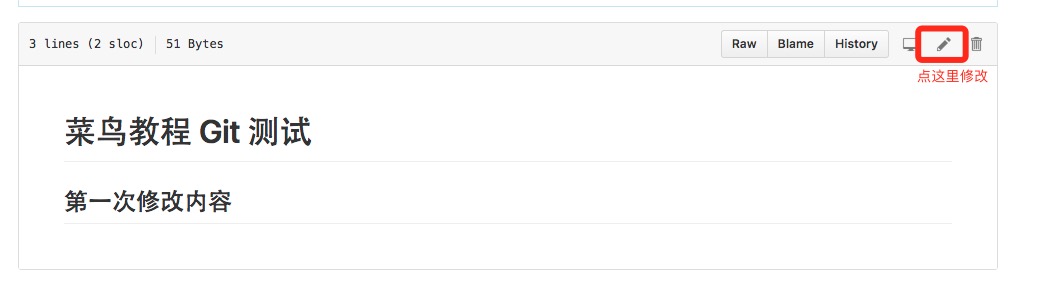
2、从远端仓库提取数据并尝试合并到当前分支：

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 0

Unpacking 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/master

Updating 0205aab..febd8ed

Fast-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"

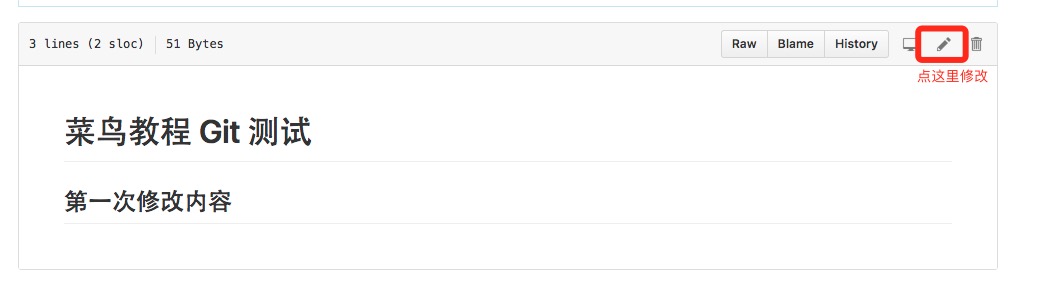
master 69e702d] 添加到远程

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)