

Tip: 本文转载自 <http://www.cnblogs.com/tugenhua0707/p/4050072.html>

Git 使用教程

一：Git 是什么？

Git 是目前世界上最先进的分布式版本控制系统。

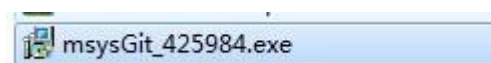
二：SVN 与 Git 的最主要的区别？

SVN 是集中式版本控制系统，版本库是集中放在中央服务器的，而干活的时候，用的都是自己的电脑，所以首先要从中央服务器哪里得到最新的版本，然后干活，干完后，需要把自己做完的活推送到中央服务器。集中式版本控制系统是必须联网才能工作，如果在局域网还可以，带宽够大，速度够快，如果在互联网下，如果网速慢的话，就纳闷了。

Git 是分布式版本控制系统，那么它就没有中央服务器的，每个人的电脑就是一个完整的版本库，这样，工作的时候就不需要联网了，因为版本都是在自己的电脑上。既然每个人的电脑都有一个完整的版本库，那多个人如何协作呢？比如说自己在电脑上改了文件 A，其他人也在电脑上改了文件 A，这时，你们两之间只需把各自的修改推送给对方，就可以互相看到对方的修改了。

三：在 windows 上如何安装 Git？

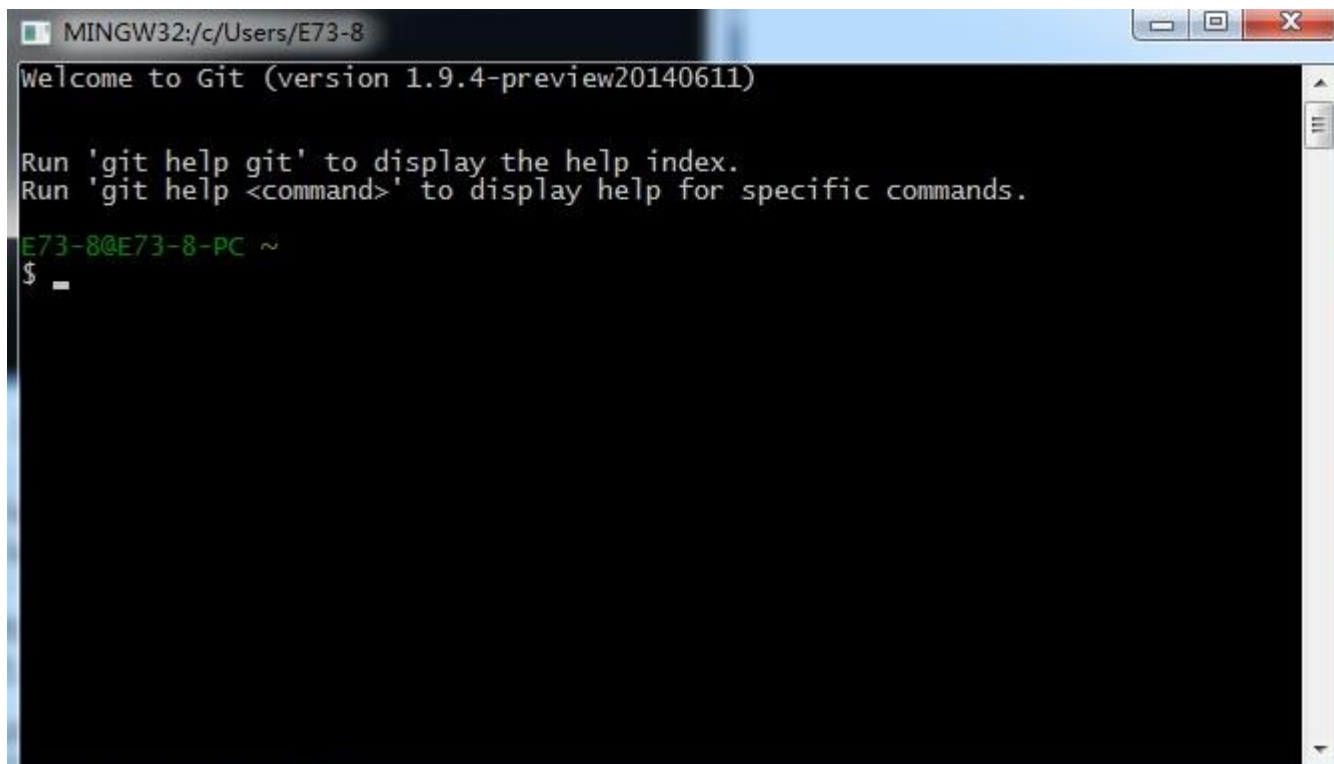
msysgit 是 windows 版的 Git,如下：



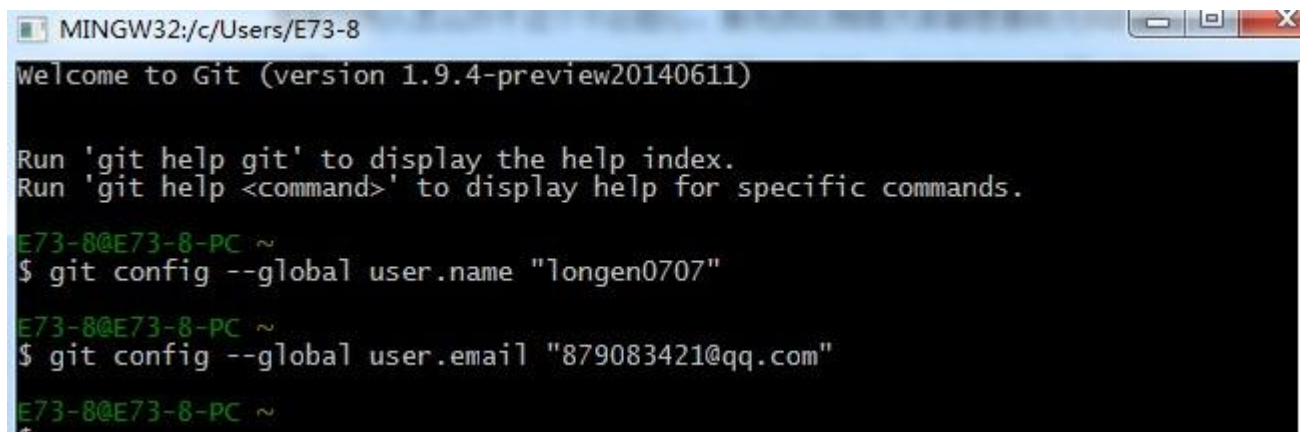
需要从网上下载一个，然后进行默认安装即可。安装完成后，在开始菜单里面找到 "Git --> Git Bash",如下：



会弹出一个类似的命令窗口的东西，就说明 Git 安装成功。如下：

A screenshot of a Git Bash terminal window. The title bar shows the path 'MINGW32:/c/Users/E73-8'. The terminal text reads: 'Welcome to Git (version 1.9.4-preview20140611)', 'Run 'git help git' to display the help index.', and 'Run 'git help <command>' to display help for specific commands.'. The prompt 'E73-8@E73-8-PC ~' is shown in green, followed by a '\$' prompt and a cursor.

安装完成后，还需要最后一步设置，在命令行输入如下：

A screenshot of a Windows command prompt window titled 'MINGW32:/c/Users/E73-8'. The window shows the output of the 'git' command, which includes a welcome message and instructions on how to use 'git help'. It then shows two successful 'git config' commands: one for setting the global user name to 'longen0707' and another for setting the global user email to '879083421@qq.com'. The prompt is currently at the third line, ready for another command.

```
MINGW32:/c/Users/E73-8
Welcome to Git (version 1.9.4-preview20140611)

Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

E73-8@E73-8-PC ~
$ git config --global user.name "longen0707"

E73-8@E73-8-PC ~
$ git config --global user.email "879083421@qq.com"

E73-8@E73-8-PC ~
```

因为 Git 是分布式版本控制系统，所以需要填写用户名和邮箱作为一个标识。

注意：git config --global 参数，有了这个参数，表示你这台机器上所有的 Git 仓库都会使用这个配置，当然你也可以对某个仓库指定的不同的用户名和邮箱。

四：如何操作？

一：创建版本库。

什么是版本库？版本库又名仓库，英文名 repository,你可以简单的理解一个目录，这个目录里面的所有文件都可以被 Git 管理起来，每个文件的修改，删除，Git 都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻还可以将文件“还原”。

所以创建一个版本库也非常简单,如下我是 D 盘 -> www 下 目录下新建一个 testgit 版本库。

```

E73-8@E73-8-PC ~ (master)
$ cd D:

E73-8@E73-8-PC /d
$ cd www

E73-8@E73-8-PC /d/www
$ mkdir testgit

E73-8@E73-8-PC /d/www
$ cd testgit

E73-8@E73-8-PC /d/www/testgit
$ pwd
/d/www/testgit

E73-8@E73-8-PC /d/www/testgit
$

```

pwd 命令是用于显示当前的目录。

1. 通过命令 git init 把这个目录变成 git 可以管理的仓库，如下：

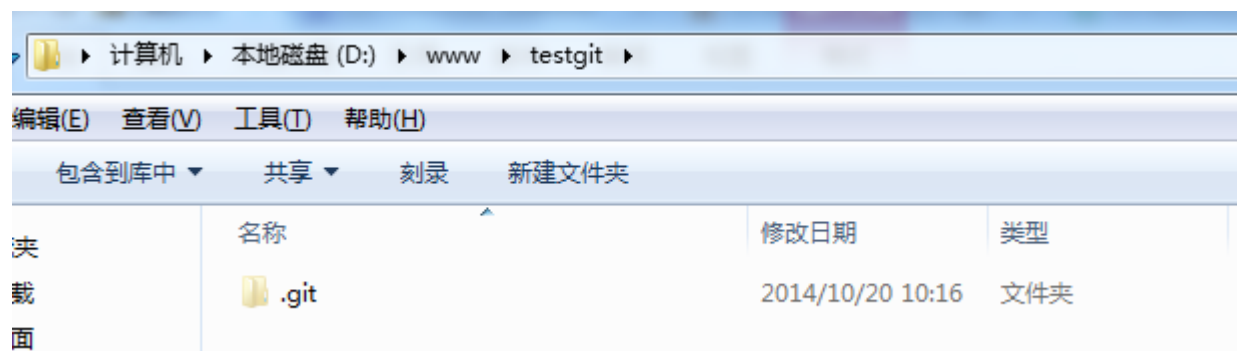
```

E73-8@E73-8-PC /d/www/testgit
$ git init
Initialized empty Git repository in d:/www/testgit/.git/

E73-8@E73-8-PC /d/www/testgit (master)
$

```

这时候你当前 testgit 目录下会多了一个.git 的目录，这个目录是 Git 来跟踪管理版本的，没事千万不要手动乱改这个目录里面的文件，否则，会把 git 仓库给破坏了。如下：



2. 把文件添加到版本库中。

首先要明确下，所有的版本控制系统，只能跟踪文本文件的改动，比如 txt 文件，网页，所有程序的代码等，Git 也不列外，版本控制系统可以告诉你每次的改动，但是图片，视频这些二进制文件，虽能也能由版本控制系统管理，但没法跟踪文件的变化，只能把二进制文件每次改动串起来，也就是知道图片从 1kb 变成 2kb，但是到底改了啥，版本控制也不知道。

下面先看下 demo 如下演示：

我在版本库 testgit 目录下新建一个记事本文件 readme.txt 内容如下：

11111111

第一步：使用命令 `git add readme.txt` 添加到暂存区里面去。如下：

```
E73-8@E73-8-PC /d/www/testgit (master)
$ git add readme.txt
E73-8@E73-8-PC /d/www/testgit (master)
$
```

如果和上面一样，没有任何提示，说明已经添加成功了。

第二步：用命令 `git commit` 告诉 Git，把文件提交到仓库。

```
E73-8@E73-8-PC /d/www/testgit (master)
$ git commit -m "readme.txt提交"
[master (root-commit) 1f05713] readme.txt提交
1 file changed, 1 insertion(+)
create mode 100644 readme.txt
E73-8@E73-8-PC /d/www/testgit (master)
$
```

是提交的注释

现在我们已经提交了一个 readme.txt 文件了，我们下面可以通过命令 git status 来查看是否还有文件未提交，如下：

```
E73-8@E73-8-PC /d/www/testgit (master)
$ git status
On branch master
nothing to commit, working directory clean
E73-8@E73-8-PC /d/www/testgit (master)
$
```

说明没有任何文件未提交，但是我现在继续来改下 readme.txt 内容，比如我在下面添加一行 2222222222 内容，继续使用 git status 来查看下结果，如下：

```
E73-8@E73-8-PC /d/www/testgit (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")
E73-8@E73-8-PC /d/www/testgit (master)
$
```

上面的命令告诉我们 readme.txt 文件已被修改，但是未被提交的修改。

接下来我想看下 readme.txt 文件到底改了什么内容，如何查看呢？可以使用如下命令：

git diff readme.txt 如下：

```

$ git diff readme.txt
diff --git a/readme.txt b/readme.txt
index d769ca0..26272f6 100644
--- a/readme.txt
+++ b/readme.txt
@@ -1,2 @@
-1111111111111111
\ No newline at end of file
+1111111111111111
+2222222222222222
\ No newline at end of file
E73-8@E73-8-PC /d/www/testgit (master)
$

```

如上可以看到，readme.txt 文件内容从一行 11111111 改成 二行 添加了一行 22222222 内容。

知道了对 readme.txt 文件做了什么修改后，我们可以放心的提交到仓库了，提交修改和提交文件是一样的 2 步(第一步是 git add 第二步是：git commit)。

如下：

```

E73-8@E73-8-PC /d/www/testgit (master)
$ git add readme.txt

E73-8@E73-8-PC /d/www/testgit (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   readme.txt

E73-8@E73-8-PC /d/www/testgit (master)
$ git commit -m "文件增加222222内容"
[master 435ccc9] 文件增加222222内容
1 file changed, 2 insertions(+), 1 deletion(-)

E73-8@E73-8-PC /d/www/testgit (master)
$ git status
On branch master
nothing to commit, working directory clean

E73-8@E73-8-PC /d/www/testgit (master)
$

```

提交文件之前，查看下状态

提交后，继续查看下状态，显示没有可提交的文件

二：版本回退：

如上，我们已经学会了修改文件，现在我继续对 readme.txt 文件进行修改，再增加一行

内容为 3333333333333333.继续执行命令如下：

```
E73-8@E73-8-PC /d/www/testgit (master)
$ git add readme.txt

E73-8@E73-8-PC /d/www/testgit (master)
$ git commit -m "添加readme.txt文件内容为333333"
[master 6fcfc89] 添加readme.txt文件内容为333333
1 file changed, 2 insertions(+), 1 deletion(-)

E73-8@E73-8-PC /d/www/testgit (master)
$
```

现在我已经对 readme.txt 文件做了三次修改了，那么我现在想查看下历史记录，如何查呢？我们现在可以使用命令 `git log` 演示如下所示：

```
E73-8@E73-8-PC /d/www/testgit (master)
$ git log
commit 6fcfc898c63c2c760ea75865312f6242baa2ac92 → 每次提交的版本号
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 10:56:35 2014 +0800

    添加readme.txt文件内容为333333 → 最近一次增加内容为3333

commit 435ccc9d61b8f61de6fcac43d22389eeac8cd7fd
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 10:47:42 2014 +0800

    文件增加222222内容 → 上一次提交增加的内容为22222

commit 1f057136c19a2c6f965be35ae8281b422e664939
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 10:33:01 2014 +0800

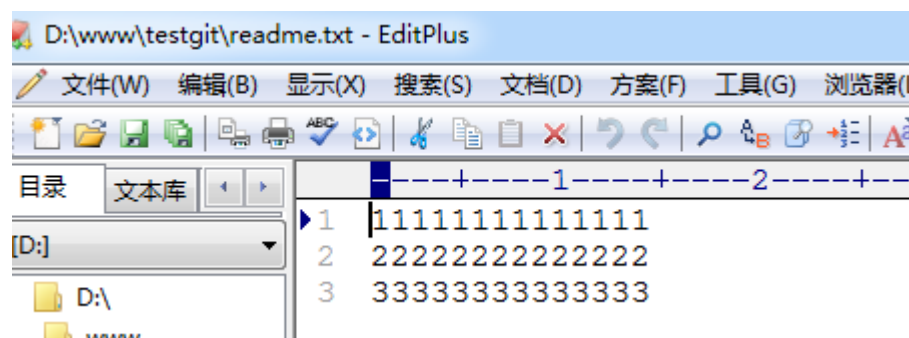
    readme.txt提交

E73-8@E73-8-PC /d/www/testgit (master)
$
```


git log 命令显示从最近到最远的显示日志，我们可以看到最近三次提交，最近的一次是,增加内容为 333333.上一次是添加内容 222222，第一次默认是 111111.如果嫌上面显示的信息太多的话，我们可以使用命令 git log -pretty=oneline 演示如下：

```
E73-8@E73-8-PC /d/www/testgit (master)
$ git log --pretty=oneline
6fcfc898c63c2c760ea75865312f6242baa2ac92 添加readme.txt文件内容为333333
435ccc9d61b8f61de6fcac43d22389eeac8cd7fd 文件增加222222内容
1f057136c19a2c6f965be35ae8281b422e664939 readme.txt提交
E73-8@E73-8-PC /d/www/testgit (master)
```

现在我想使用版本回退操作，我想把当前的版本回退到上一个版本，要使用什么命令呢？可以使用如下 2 种命令，第一种是：git reset --hard HEAD^ 那么如果要回退到上上个版本只需把 HEAD^ 改成 HEAD^^ 以此类推。那如果要回退到前 100 个版本的话，使用上面的方法肯定不方便，我们可以使用下面的简便命令操作：git reset --hard HEAD~100 即可。未回退之前的 readme.txt 内容如下：



D:\www\testgit\readme.txt - EditPlus

文件(W) 编辑(B) 显示(X) 搜索(S) 文档(D) 方案(F) 工具(G) 浏览器(I)

目录 文本库

[D:]

D:\

1 111111111111111

2 22222222222222

3 33333333333333

如果想回退到上一个版本的命令如下操作：

```
1f057136c19a2c6f965be35ae8281b422e664939 README.txt提交
E73-8@E73-8-PC /d/www/testgit (master)
$ git reset --hard HEAD^ 回退到上一个版本
HEAD is now at 435ccc9 文件增加222222内容
E73-8@E73-8-PC /d/www/testgit (master)
$
```

再来查看下 readme.txt 内容如下：通过命令 cat readme.txt 查看

```
E73-8@E73-8-PC /d/www/testgit (master)
$ cat readme.txt
1111111111111111
2222222222222222
E73-8@E73-8-PC /d/www/testgit (master)
$
```

可以看到，内容已经回退到上一个版本了。我们可以继续使用 git log 来查看下

历史记录信息，如下：

```
E73-8@E73-8-PC /d/www/testgit (master)
$ git log
commit 435ccc9d61b8f61de6fcac43d22389eeac8cd7fd
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 10:47:42 2014 +0800

    文件增加222222内容

commit 1f057136c19a2c6f965be35ae8281b422e664939
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 10:33:01 2014 +0800

    readme.txt提交
E73-8@E73-8-PC /d/www/testgit (master)
$
```

我们看到 增加 333333 内容我们没有看到了，但是现在我想回退到最新的版本，

如：有 333333 的内容要如何恢复呢？我们可以通过版本号回退，使用命令方法

如下：

git reset --hard 版本号 ,但是现在的问题假如我已经关掉过一次命令行或者333 内容的版本号我并不知道呢?要如何知道增加 3333 内容的版本号呢?可以通过如下命令即可获取到版本号: git reflog 演示如下:

```
E73-8@E73-8-PC /d/www/testgit (master)
$ git reflog
435ccc9 HEAD@{0}: reset: moving to HEAD^
6fcfc89 HEAD@{1}: commit: 添加readme.txt文件内容为333333
435ccc9 HEAD@{2}: commit: 文件增加222222内容
1f05713 HEAD@{3}: commit (initial): readme.txt提交
E73-8@E73-8-PC /d/www/testgit (master)
$
```

通过上面的显示我们可以知道 ,增加内容 3333 的版本号是 6fcfc89.我们现在可以命令

git reset --hard 6fcfc89 来恢复了。演示如下:

```
E73-8@E73-8-PC /d/www/testgit (master)
$ git reset --hard 6fcfc89
HEAD is now at 6fcfc89 添加readme.txt文件内容为333333
E73-8@E73-8-PC /d/www/testgit (master)
$ cat readme.txt
aaaaaaaaaaaaa
22222222222222
33333333333333
E73-8@E73-8-PC /d/www/testgit (master)
$
```

查看readme.txt内容如下

可以看到 目前已经是最新的版本了。

三：理解工作区与暂存区的区别？

工作区 :就是你在电脑上看到的目录 ,比如目录下 testgit 里的文件(.git 隐藏目录版本库除外)。或者以后需要再新建的目录文件等等都属于工作区范畴。

版本库(Repository)：工作区有一个隐藏目录.git,这个不属于工作区，这是版本库。其中版本库里面存了很多东西，其中最重要的就是 stage(暂存区)，还有 Git 为我们自动创建了第一个分支 master,以及指向 master 的一个指针 HEAD。

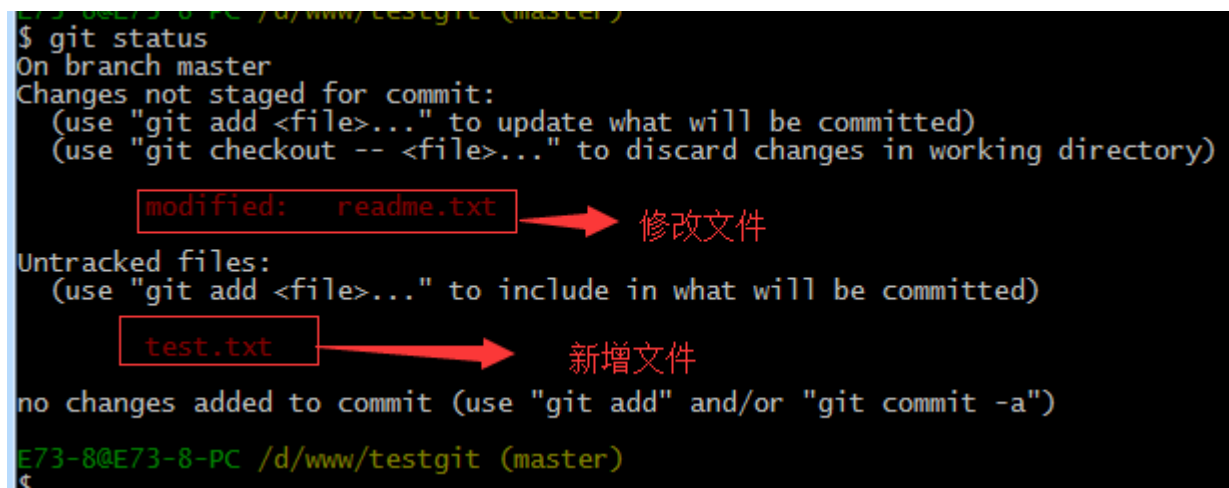
我们前面说过使用 Git 提交文件到版本库有两步：

第一步：是使用 `git add` 把文件添加进去，实际上就是把文件添加到暂存区。

第二步：使用 `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上。

我们继续使用 demo 来演示下：

我们在 `readme.txt` 再添加一行内容为 44444444 ,接着在目录下新建一个文件为 `test.txt` 内容为 test，我们先用命令 `git status` 来查看下状态，如下：



```
E73-8@E73-8-PC /d/www/testgit (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

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

    test.txt

no changes added to commit (use "git add" and/or "git commit -a")
E73-8@E73-8-PC /d/www/testgit (master)
$
```

现在我们先使用 `git add` 命令把 2 个文件都添加到暂存区中，再使用 `git status` 来查看下状态，如下：

```

$ git add readme.txt
E73-8@E73-8-PC /d/www/testgit (master)
$ git add test.txt
E73-8@E73-8-PC /d/www/testgit (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   readme.txt
        new file:   test.txt
E73-8@E73-8-PC /d/www/testgit (master)
$

```

接着我们可以使用 git commit 一次性提交到分支上，如下：

```

$ git commit -m "一次性提交所有文件，包括新建文件test.txt"
[master 4612fa5] 一次性提交所有文件，包括新建文件test.txt
2 files changed, 3 insertions(+), 1 deletion(-)
create mode 100644 test.txt
E73-8@E73-8-PC /d/www/testgit (master)
$ git status
On branch master
nothing to commit, working directory clean
E73-8@E73-8-PC /d/www/testgit (master)
$

```

继续查看下状态

四：Git 撤销修改和删除文件操作。

一：撤销修改：

比如我现在在 readme.txt 文件里面增加一行 内容为 555555555555 ,我们先通过命令查看如下：

```

E73-8@E73-8-PC /d/www/testgit (master)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
5555555555555555
E73-8@E73-8-PC /d/www/testgit (master)
$

```

在我未提交之前，我发现添加 555555555555 内容有误，所以我得马上恢复以前的版本，现在我可以有如下几种方法可以做修改：

第一：如果我知道要删掉那些内容的话，直接手动更改去掉那些需要的文件，然后 add 添加到暂存区，最后 commit 掉。

第二：我可以按以前的方法直接恢复到上一个版本。使用 `git reset --hard HEAD^`

但是现在我不想使用上面的 2 种方法，我想直接使用撤销命令该如何操作呢？

首先在做撤销之前，我们可以先用 `git status` 查看下当前的状态。如下所示：

```
$ 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)

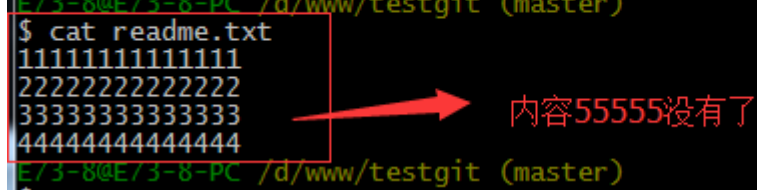
        modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")
E73-8@E73-8-PC /d/www/testgit (master)
$
```

可以发现，Git 会告诉你，`git checkout -- file` 可以丢弃工作区的修改，如下命令：

`git checkout -- readme.txt`，如下所示：

```
E73-8@E73-8-PC /d/www/testgit (master)
$ git checkout -- readme.txt
E73-8@E73-8-PC /d/www/testgit (master)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
E73-8@E73-8-PC /d/www/testgit (master)
$
```



内容55555没有了

命令 `git checkout --readme.txt` 意思就是，把 `readme.txt` 文件在工作区做的修改全部撤销，这里有 2 种情况，如下：

1. `readme.txt` 自动修改后，还没有放到暂存区，使用 撤销修改就回到和本库一模一样的状态。
2. 另外一种 `readme.txt` 已经放入暂存区了，接着又作了修改，撤销修改就回到添加暂存区后的状态。

对于第二种情况，我想我们继续做 demo 来看下，假如现在我对 `readme.txt` 添加一行 内容为 66666666666666，我 `git add` 增加到暂存区后，接着添加内容 7777777，我想通过撤销命令让其回到暂存区后的状态。如下所示：

```
E73-8@E73-8-PC /d/www/testgit (master)
$ cat readme.txt
111111111111111
222222222222222
333333333333333
444444444444444
566666666666666

```

添加一条内容为6666666

```
E73-8@E73-8-PC /d/www/testgit (master)
$ git add readme.txt

```

先放到暂存区中

```
E73-8@E73-8-PC /d/www/testgit (master)
$ cat readme.txt
111111111111111
222222222222222
333333333333333
444444444444444
666666666666666
777777777777777

```

接着添加内容7777777，但是没有添加到暂存区

```
E73-8@E73-8-PC /d/www/testgit (master)
$ git checkout -- readme.txt

```

直接使用撤销命令，把未添加到暂存区内容撤销掉

```
E73-8@E73-8-PC /d/www/testgit (master)
$ cat readme.txt
111111111111111
222222222222222
333333333333333
444444444444444
666666666666666

```

继续查看下内容，发现内容777777已经撤销掉了

```
E73-8@E73-8-PC /d/www/testgit (master)
$

```

注意：命令 `git checkout -- readme.txt` 中的 `--` 很重要，如果没有 `--` 的话，那么命令变成创建分支了。

二：删除文件。

假如我现在版本库 testgit 目录添加一个文件 b.txt,然后提交。如下：


```
E73-8@E73-8-PC /d/www/testGit (master)
$ git add b.txt
E73-8@E73-8-PC /d/www/testGit (master)
$ git commit -m "添加b.txt文件"
[master /fcb8ee] 添加b.txt文件
1 file changed, 1 insertion(+)
create mode 100644 b.txt
E73-8@E73-8-PC /d/www/testGit (master)
$ rm b.txt
E73-8@E73-8-PC /d/www/testGit (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:    a.txt
        deleted:    b.txt
        deleted:    test.txt

no changes added to commit (use "git add" and/or "git commit -a")
E73-8@E73-8-PC /d/www/testGit (master)
$
```

添加b.txt文件

接着提交b.txt文件

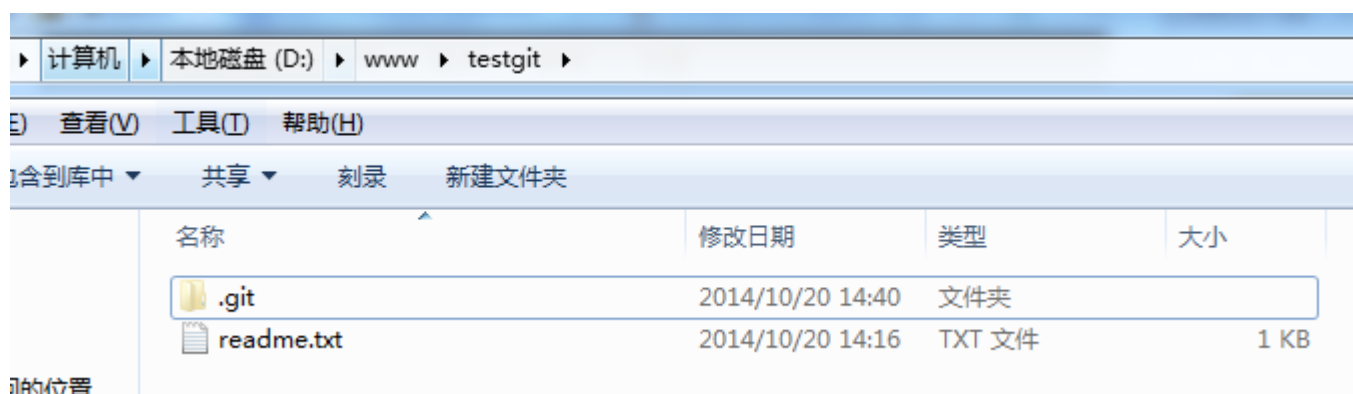
我们可以直接在目录下删掉文件或者使用命令rm b.txt

继续查看 b.txt文件已经删除了，此时有2个选择，一：直接commit掉。二：从版本库中恢复被删掉的文件

如上：一般情况下，可以直接在文件目录中把文件删了，或者使用如上 rm 命令：

rm b.txt，如果我想彻底从版本库中删掉了此文件的话，可以再执行 commit

命令 提交掉，现在目录是这样的，

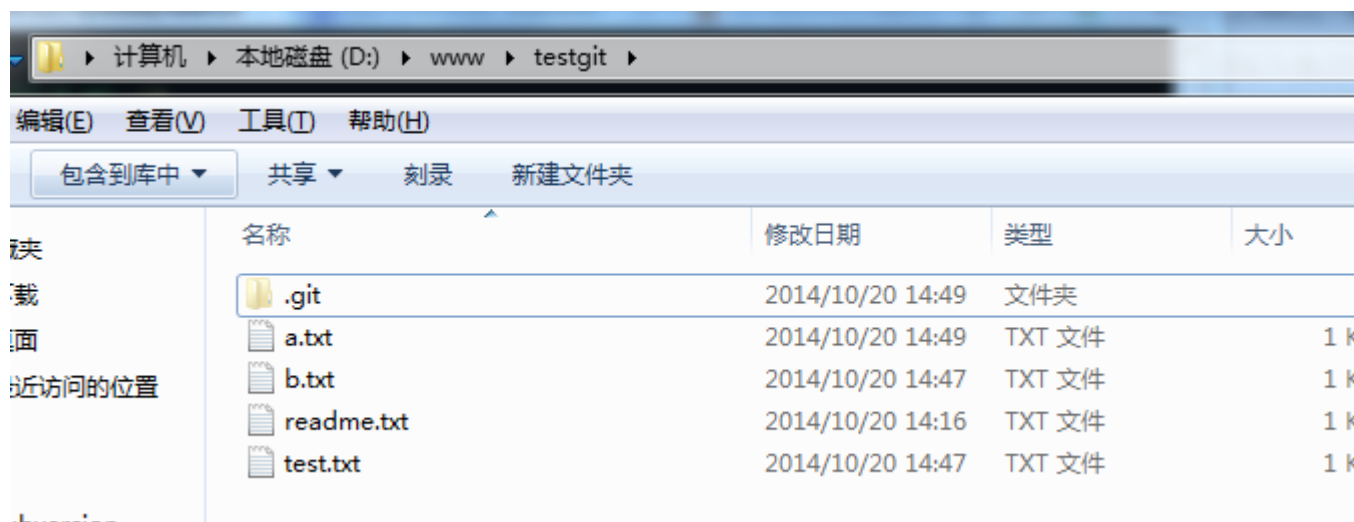


只要没有 commit 之前，如果我想在版本库中恢复此文件如何操作呢？

可以使用如下命令 `git checkout -- b.txt` , 如下所示 :

```
E73-8@E73-8-PC /d/www/testGit (master)
$ git checkout -- b.txt
E73-8@E73-8-PC /d/www/testGit (master)
$ git checkout test.txt
E73-8@E73-8-PC /d/www/testGit (master)
$ git checkout -- a.txt
E73-8@E73-8-PC /d/www/testGit (master)
$ git commit -m "提交b.txt和test.txt文件和a.txt文件"
On branch master
nothing to commit, working directory clean
E73-8@E73-8-PC /d/www/testGit (master)
$
```

再来看看我们 testgit 目录 , 添加了 3 个文件了。如下所示 :

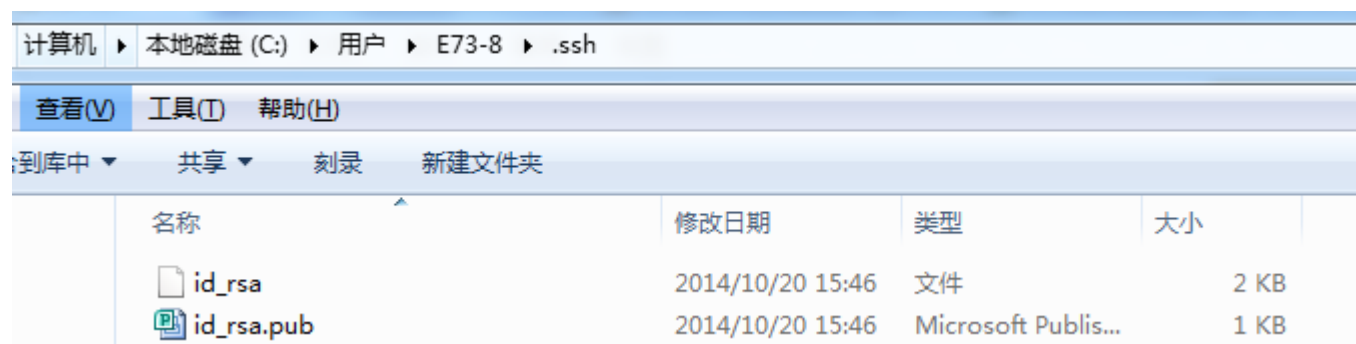


五：远程仓库。

在了解之前 , 先注册 github 账号 , 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 , 所以需要一点设置 :


第一步：创建 SSH Key。在用户主目录下，看看有没有.ssh 目录，如果有，再看看这个目录下有没有 id_rsa 和 id_rsa.pub 这两个文件，如果有的话，直接跳过此如下命令，如果没有的话，打开命令行，输入如下命令：

ssh-keygen -t rsa -C "youremail@example.com"，由于我本地此前运行过一次，所以本地有，如下所示：



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。


第二步：登录 github,打开“ settings” 中的 SSH Keys 页面，然后点击 “Add SSH Key” ,填上任意 title，在 Key 文本框里黏贴 id_rsa.pub 文件的内容。



ExploreGistBlogHelp


You don't have any verified emails. We recommend [verifying](#) at least one email.

Email verification helps our support team verify ownership if you lose account access and allows you to

 **tugenhua0707**

[Profile](#)

[Account settings](#)

[Emails](#) 

[Notification center](#)

[Billing](#)

SSH keys

[Security](#)

[Applications](#)


[Repositories](#)


[Organizations](#)


Need help? Check out our guide to [generating SSH keys](#) or troubleshoot [common issues](#)

SSH Keys

This is a list of SSH keys associated with your account. Remove any keys that you no longer need.



GitHub for Windows - PC-200912151212
71:2a:59:e0:dc:86:e2:9e:00:96:ad:b4:bb:7c:56:23
Added on 9 May 2013 —  No recent activity



my ssh key
7e:0a:07:d0:31:c5:5e:b3:c9:42:a4:40:5d:a0:70:ec
Added on 17 Oct 2014 — Last used on 17 Oct 2014

Add an SSH Key

Title

测试远程demo

Key

ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAq83ZSBe6BT9VM5329K9Ksa3zMsl
2s6MyECC9GPtTlyCN7fyNXGWY5CIHetsrddA3D2Vja8g6iYeT7N+6tJadcH1
5XF707B594jfBOAxec.ODKzPv+VtB9V63BgudWheBvyBVBhqiQncQhgwwU
cY4OeSVLsjRWoz3/WE CFm6pGrY2ry2IFRIyelDjzFHe1y1izl8mAiNSCzO5J
UTOzW5AaOQ== tugenhua0707@qq.com

Add key

第二步点击








最后点击此按钮

点击 Add Key , 你就应该可以看到已经添加的 key。

Need help? Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH Problems](#)

SSH KeysAdd


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

	GitHub for Windows - PC-200912151212 71:2a:59:e0:dc:86:e2:9e:00:96:ad:b4:bb:7c:56:23 Added on 9 May 2013 —  No recent activity	
	my ssh key 7e:0a:07:d0:31:c5:5e:b3:c9:42:a4:40:5d:a0:70:ec Added on 17 Oct 2014 — Last used on 17 Oct 2014	
	测试远程demo e6:f3:3f:67:eb:b6:84:eb:30:bc:59:bd:a5:2c:70:12 Added on 20 Oct 2014 — Never used	

一：如何添加远程库？


现在的情景是：我们已经在本地创建了一个 Git 仓库后，又想在 github 创建一个 Git 仓库，并且希望这两个仓库进行远程同步，这样 github 的仓库可以作为备份，又可以其他人通过该仓库来协作。

首先，登录 github 上，然后在右上角找到“create a new repo” 创建一个新的仓库。如下：




[Explore](#) [Gist](#) [Blog](#) [Help](#)

You don't have any verified emails. We recommend [verifying](#) at least one email.
Email verification helps our support team verify ownership if you lose account access and allows you to



Owner


 tugenhua0707 ▾

/


Repository name

Great repository names are short and memorable. Need inspiration? How about [shiny-d](#)

Description (optional)

☒  **Public**

Anyone can see this repository. You choose who can commit.


☐  **Private**

You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**

This will allow you to `git clone` the repository immediately. Skip this step if you have already

Add .gitignore: **None** ▾

Add a license: **None** ▾ 

Create repository

第三步创建

在 Repository name 填入 `testgit`，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：

Quick setup — if you've done this kind of thing before

[Set up in Desktop](#)

or

[HTTP](#)[SSH](#)<https://github.com/tughua0707/testgit.git>

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
touch README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/tughua0707/testgit.git
git push -u origin master
```

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

```
git remote add origin https://github.com/tughua0707/testgit.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)

目前，在 GitHub 上的这个 testgit 仓库还是空的，GitHub 告诉我们，可以从这个仓库克隆出新的仓库，也可以把一个已有的本地仓库与之关联，然后，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 testgit 仓库下运行命令：

```
git remote add origin https://github.com/tughua0707/testgit.git
```

所有的如下：

```
$ git remote add origin https://github.com/tughua0707/testgit.git
fatal: remote origin already exists.

E73-8@E73-8-PC /d/www/testGit (master)
$ git push -u origin master
Username for 'https://github.com': tughua0707@qq.com
Password for 'https://tughua0707@qq.com@github.com':
Counting objects: 23, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (14/14), done.
Writing objects: 100% (23/23), 1.83 KiB | 0 bytes/s, done.
Total 23 (delta 4), reused 0 (delta 0)
To https://github.com/tughua0707/testgit.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.

E73-8@E73-8-PC /d/www/testGit (master)
```

这个是我的，要用自己的

把本地仓库分支master内容推送到元仓库去

把本地库的内容推送到远程，使用 `git push` 命令，实际上是把当前分支 `master` 推送到远程。

由于远程库是空的，我们第一次推送 `master` 分支时，加上了 `-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了，上面的要输入 github 的用户名和密码如下所示：

tugenhua0707 / testgit

Description

Short description of this repository

Website


Website for this repository (optional)

 8 commits

 1 branch

 0 releases



 branch: **master** ▾

testgit / +

删掉了c.txt文件



longen0707 authored an hour ago

latest

 [a.txt](#)

添加b.txt文件

 [b.txt](#)


添加b.txt文件

 [readme.txt](#)

添加文件a.txt

 [test.txt](#)

一次性提交所有文件，包括新建文件test.txt

 **readme.txt**

```
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
```

从现在起，只要本地作了提交，就可以通过如下命令：

```
git push origin master
```

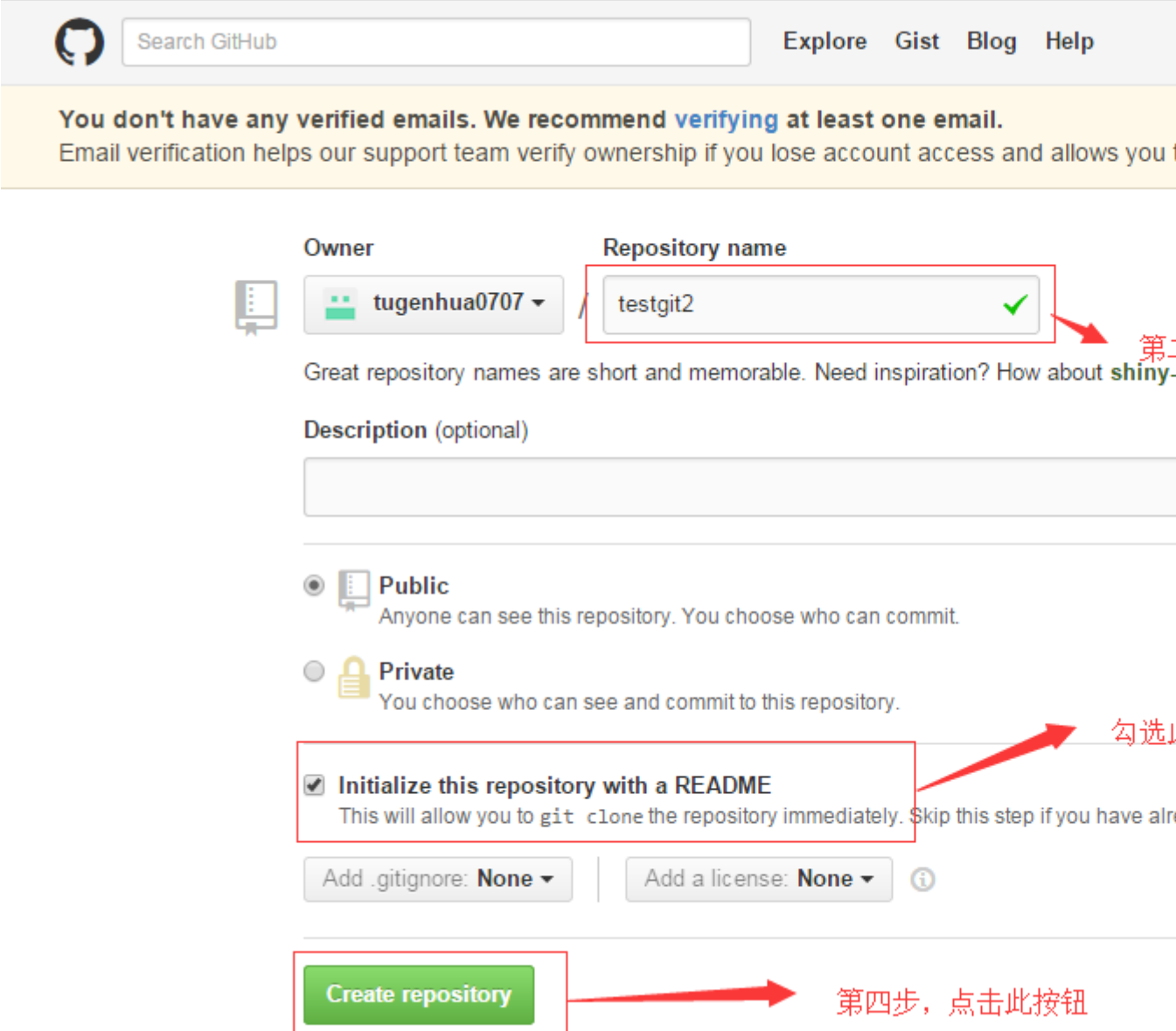
把本地 master 分支的最新修改推送到 github 上了，现在你就拥有了真正的分布式版本库了。

二：如何从远程库克隆？

上面我们了解了先有本地库，后有远程库时候，如何关联远程库。

现在我们想，假如远程库有新的内容了，我想克隆到本地来 如何克隆呢？

首先，登录 github，创建一个新的仓库，名字叫 testgit2.如下：



The screenshot shows the GitHub 'Create repository' form. The 'Owner' is 'tugenhua0707'. The 'Repository name' is 'testgit2', which is highlighted with a red box and a red arrow pointing to the text '第二步' (Step 2). Below the name, there is a note: 'Great repository names are short and memorable. Need inspiration? How about shiny-'. The 'Description (optional)' field is empty. The 'Public' radio button is selected, with the text 'Anyone can see this repository. You choose who can commit.' below it. The 'Private' radio button is unselected, with the text 'You choose who can see and commit to this repository.' below it. The checkbox 'Initialize this repository with a README' is checked, highlighted with a red box, and has a red arrow pointing to the text '勾选' (checked). Below this, there are two dropdown menus: 'Add .gitignore: None' and 'Add a license: None'. At the bottom, the 'Create repository' button is highlighted with a red box and a red arrow pointing to the text '第四步，点击此按钮' (Step 4, click this button).

Owner: tugenhua0707

Repository name: testgit2

Description (optional):

Public: Anyone can see this repository. You choose who can commit.

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

☒ Initialize this repository with a README
This will allow you to `git clone` the repository immediately. Skip this step if you have already cloned the repository.

Add .gitignore: None | Add a license: None

Create repository

如下，我们看到：

Description

Short description of this repository

Website

Website for this repository (optional)

1 commit

1 branch

0 releases

branch: master testgit2 / +

Initial commit

tugenhua0707 authored 17 seconds ago

latest commit

README.md

Initial commit

README.md

testgit2

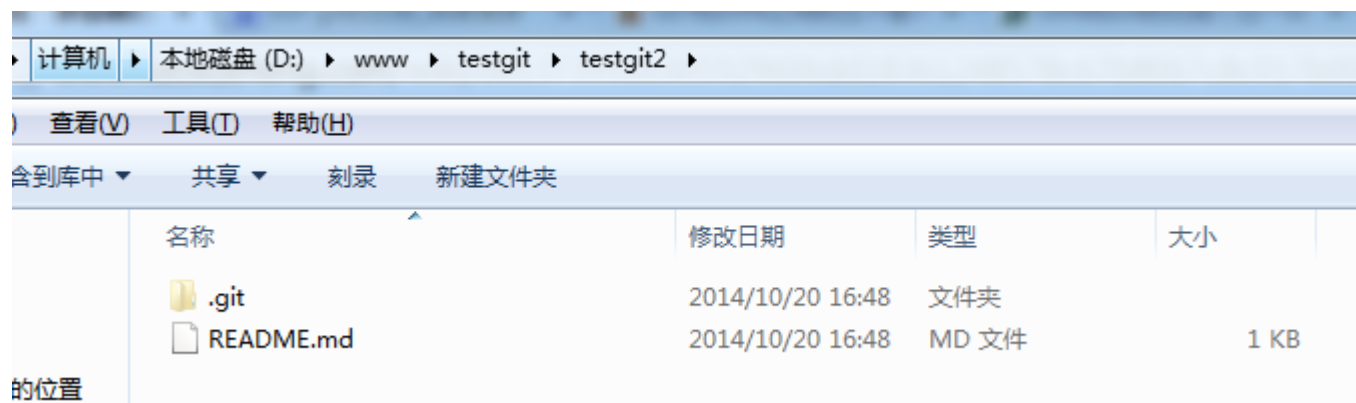
现在，远程库已经准备好了，下一步是使用命令 `git clone` 克隆一个本地库了。

如下所示：

```
$ git clone https://github.com/tugenhua0707/testgit2
Cloning into 'testgit2'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.
E73-8@E73-8-PC /d/www/testGit (master)
```

这是我github地址，你们
的要换成自己的

接着在我本地目录下 生成 testgit2 目录了，如下所示：



六：创建与合并分支。

在 版本回填退里，你已经知道，每次提交，Git 都把它们串成一条时间线，这条时间线就是一个分支。截止到目前，只有一条时间线，在 Git 里，这个分支叫主 分支，即 master 分支。HEAD 严格来说不是指向提交，而是指向 master，master 才是指向提交的，所以，HEAD 指向的就是当前分支。

首先，我们来创建 dev 分支，然后切换到 dev 分支上。如下操作：

```
E73-8@E73-8-PC /d/www/testGit (master)
$ git checkout -b dev
Switched to a new branch 'dev'

E73-8@E73-8-PC /d/www/testGit (dev)
$ git branch
* dev
  master

E73-8@E73-8-PC /d/www/testGit (dev)
$
```

git checkout 命令加上 -b 参数表示创建并切换，相当于如下 2 条命令

git branch dev

git checkout dev

git branch 查看分支，会列出所有的分支，当前分支前面会添加一个星号。然后我们在 dev 分支上继续做 demo，比如我们现在在 readme.txt 再增加一行 777777777777

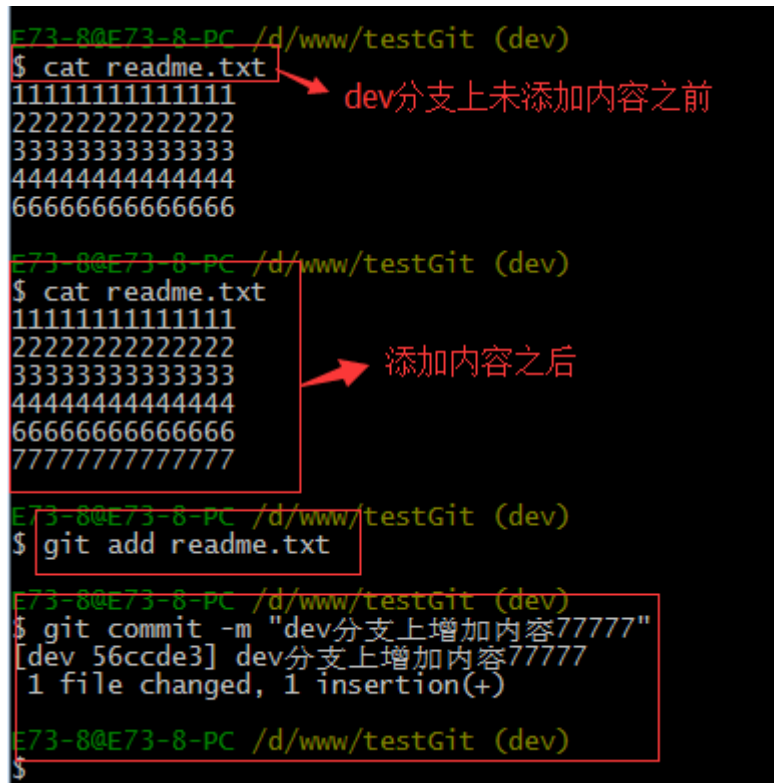
首先我们先来查看下 readme.txt 内容，接着添加内容 77777777，如下：

```
E73-8@E73-8-PC /d/www/testGit (dev)
$ cat readme.txt
11111111111111
22222222222222
33333333333333
44444444444444
66666666666666

E73-8@E73-8-PC /d/www/testGit (dev)
$ cat readme.txt
11111111111111
22222222222222
33333333333333
44444444444444
66666666666666
77777777777777

E73-8@E73-8-PC /d/www/testGit (dev)
$ git add readme.txt

E73-8@E73-8-PC /d/www/testGit (dev)
$ git commit -m "dev分支上增加内容77777"
[dev 56ccde3] dev分支上增加内容77777
1 file changed, 1 insertion(+)
```



现在 dev 分支工作已完成，现在我们切换到主分支 master 上，继续查看 readme.txt 内容如下：

```
E73-8@E73-8-PC /d/www/testGit (dev)
$ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.

E73-8@E73-8-PC /d/www/testGit (master)
$ cat readme.txt
11111111111111
22222222222222
33333333333333
44444444444444
66666666666666

E73-8@E73-8-PC /d/www/testGit (master)
$
```



现在我们可以把 dev 分支上的内容合并到分支 master 上了，可以在 master 分支上，使用如下命令 `git merge dev` 如下所示：

```
E73-8@E73-8-PC /d/www/testGit (master)
$ git merge dev
Updating 2a4fd81..56ccde3
Fast-forward
 readme.txt | 1 +
 1 file changed, 1 insertion(+)

E73-8@E73-8-PC /d/www/testGit (master)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
7777777777777777

E73-8@E73-8-PC /d/www/testGit (master)
$
```

在master分支上合并dev分支内容

继续查看内容，多了一条777777

`git merge` 命令用于合并指定分支到当前分支上，合并后，再查看 `readme.txt` 内容，可以看到，和 dev 分支最新提交的是完全一样的。

注意到上面的 *Fast-forward* 信息，Git 告诉我们，这次合并是“快进模式”，也就是直接把 master 指向 dev 的当前提交，所以合并速度非常快。

合并完成后，我们可以接着删除 dev 分支了，操作如下：

```
E73-8@E73-8-PC /d/www/testGit (master)
$ git branch -d dev
Deleted branch dev (was 56ccde3).

E73-8@E73-8-PC /d/www/testGit (master)
$ git branch
* master

E73-8@E73-8-PC /d/www/testGit (master)
$
```

删除dev分支

查看分支的命令

总结创建与合并分支命令如下：

查看分支：`git branch`

创建分支：git branch name

切换分支：git checkout name

创建+切换分支：git checkout -b name

合并某分支到当前分支：git merge name

删除分支：git branch -d name

一：如何解决冲突？

下面我们还是一步一步来，先新建一个新分支，比如名字叫 fenzhi1，在
readme.txt 添加一行内容 8888888，然后提交，如下所示：

```
$ git checkout -b fenzhil
Switched to a new branch 'fenzhil'

E73-8@E73-8-PC /d/www/testGit (fenzhil)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
7777777777777777

E73-8@E73-8-PC /d/www/testGit (fenzhil)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
7777777777777777
8888888888888888

E73-8@E73-8-PC /d/www/testGit (fenzhil)
$ git add readme.txt

E73-8@E73-8-PC /d/www/testGit (fenzhil)
$ git commit -m "添加内容888888"
[fenzhil b03ae4b] 添加内容888888
1 file changed, 1 insertion(+)

E73-8@E73-8-PC /d/www/testGit (fenzhil)
$
```

新建并切换分支

内容为添加之前的内容

添加8888内容后

同样，我们现在切换到 master 分支上来，也在最后一行添加内容，内容为 99999999，如下所示：


```
E73-8@E73-8-PC /d/www/testGit (fenzhi1)
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)

E73-8@E73-8-PC /d/www/testGit (master)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
7777777777777777

E73-8@E73-8-PC /d/www/testGit (master)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
7777777777777777
9999999999999999

E73-8@E73-8-PC /d/www/testGit (master)
$ git add readme.txt

E73-8@E73-8-PC /d/www/testGit (master)
$ git commit -m "在master分支上新增内容99999"
[master 418595b] 在master分支上新增内容99999
1 file changed, 1 insertion(+)
```

切换到master分支上

未添加内容之前

添加内容9999之后

/d/www/testGit (master)

\$ 3-8@E73-8-PC

现在我们需要在 master 分支上来合并 fenzhi1，如下操作：

```

$ git merge fenzhil
Auto-merging readme.txt
CONFLICT (content): Merge conflict in readme.txt
Automatic merge failed; fix conflicts and then commit the result.

E73-8@E73-8-PC /d/www/testGit (master|MERGING)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
(use "git push" to publish your local commits)

You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   readme.txt

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

        testgit2/

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

E73-8@E73-8-PC /d/www/testGit (master|MERGING)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
5555555555555555
6666666666666666
7777777777777777
<<<<<< HEAD
9999999999999999
=====
8888888888888888
>>>>>> fenzhil

```

在master分支上合并fenzhil

产生冲突

查看状态

查看readme.txt内容

冲突代码

Git 用<<<<<< , ===== , >>>>>>标记出不同分支的内容，其中<<<HEAD 是指主分支修改的内容，>>>>>fenzhil 是指 fenzhil 上修改的内容，我们可以修改下如下后保存：

```
E73-8@E73-8-PC /d/www/testGit (master|MERGING)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
7777777777777777
9999999999999999

E73-8@E73-8-PC /d/www/testGit (master|MERGING)
$ git add readme.txt

E73-8@E73-8-PC /d/www/testGit (master|MERGING)
$ git commit -m "conflict fixed"
[master 672c256] conflict fixed

E73-8@E73-8-PC /d/www/testGit (master)
$
```

查看内容，修改成和主干上代码一样的

如果我想查看分支合并的情况的话，需要使用命令 `git log`. 命令行演示如下：

```
273 0ae273b PC /d/www/testgit (master)
$ git log
commit 672c25679deef1281775f0a800058ac1358234b8
Merge: 418595b b03ae4b
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 18:01:16 2014 +0800
```

conflict fixed

```
commit 418595b4c39b47820cf00241a291505713a649e8
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 17:52:47 2014 +0800
```

在master分支上新增内容99999

```
commit b03ae4b0c7fa088df21c097178ac3b3ad01dbee8
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 17:48:42 2014 +0800
```

添加内容888888

```
commit 56ccde3b9b86b16c1dbb6670eaaf1f1f7ad40c06
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 17:25:43 2014 +0800
```

dev分支上增加内容77777

```
commit 2a4fd81d920ba228941438f3262fe1ae60f1f5be
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 14:57:32 2014 +0800
```

删掉了c.txt文件

```
commit fed1d562614e581bcb4b8bb925408f01e039e113
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 14:56:07 2014 +0800
```

删掉了c.txt文件

```
commit 7fcb8ee84f9d4d5ebae7b001424794108956424a
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 14:40:15 2014 +0800
```

添加b.txt文件

```
commit d8bb7b49d053019e4807d427756d8e1331cb2fef
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 14:27:40 2014 +0800
```

添加文件a.txt

```
commit 4612fa5c71b1ece119c75199702add45ba5c3157
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 11:53:26 2014 +0800
```

一次性提交所有文件，包括新建文件test.txt

```
commit 6fcfc898c63c2c760ea75865312f6242baa2ac92
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 10:56:35 2014 +0800
```

添加readme.txt文件内容为333333

二：分支管理策略。

通常合并分支时，git 一般使用“Fast forward”模式，在这种模式下，删除分支后，会丢掉分支信息，现在我们来使用带参数 `-no-ff` 来禁用“Fast forward”模式。首先我们来做 demo 演示下：

1. 创建一个 dev 分支。
2. 修改 readme.txt 内容。
3. 添加到暂存区。
4. 切换回主分支(master)。
5. 合并 dev 分支，使用命令 `git merge -no-ff -m “注释” dev`
6. 查看历史记录

截图如下：

```
E73-8@E73-8-PC /d/www/testgit (master)
$ git checkout -b dev
Switched to a new branch 'dev'
E73-8@E73-8-PC /d/www/testgit (dev)
$ git add readme.txt
E73-8@E73-8-PC /d/www/testgit (dev)
$ git commit -m "add merge"
[dev f595301] add merge
1 file changed, 1 insertion(+)
E73-8@E73-8-PC /d/www/testgit (dev)
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 4 commits.
(use "git push" to publish your local commits)
E73-8@E73-8-PC /d/www/testgit (master)
$ git merge --no-ff -m "merge with no-ff" dev
Merge made by the 'recursive' strategy.
 readme.txt | 1 +
1 file changed, 1 insertion(+)
E73-8@E73-8-PC /d/www/testgit (master)
$ git branch -d dev
Deleted branch dev (was f595301).
E73-8@E73-8-PC /d/www/testgit (master)
$ git branch
* master
E73-8@E73-8-PC /d/www/testgit (master)
$ git log --graph --pretty=oneline --abbrev-commit
* 91dfe16 merge with no-ff
* f595301 add merge
* 672c256 conflict fixed
* b03ae4b 添加内容888888
* 418595b 在master分支上新增内容99999
* 56ccde3 dev分支上增加内容77777
* 2a4fd81 删掉了c.txt文件
* fed1d56 删掉了c.txt文件
* 7fcb8ee 添加b.txt文件
* d8bb7b4 添加文件a.txt
* 4612fa5 一次性提交所有文件，包括新建文件test.txt
* 6fcfc89 添加readme.txt文件内容为333333
* 435ccc9 文件增加222222内容
* 1f05713 readme.txt提交
E73-8@E73-8-PC /d/www/testgit (master)
$
```

创建一个dev分支

合并dev分支 -no-ff 表示禁用 fast forward

删除dev分支

版本号

被删除的分支信息还在

分支策略：首先 master 主分支应该是非常稳定的，也就是用来发布新版本，一般情况下不允许在上面干活，干活一般情况下在新建的 dev 分支上干活，干完后 比如上要发布 或者说 dev 分支代码稳定后可以合并到主分支 master 上来。

七： bug 分支：

在开发中，会经常碰到 bug 问题，那么有了 bug 就需要修复，在 Git 中，分支是很强大的，每个 bug 都可以通过一个临时分支来修复，修复完成后，合并分支，然后将临时的分支删除掉。

比如我在开发中接到一个 404 bug 时候 我们可以创建一个 404 分支来修复它，但是，当前的 dev 分支上的工作还没有提交。比如如下：

```
$ git status
On branch dev
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")
E73-8@E73-8-PC /d/www/testGit (dev)
$
```

并不是我不想提交，而是工作进行到一半时候，我们还无法提交，比如我这个分支 bug 要 2 天完成，但是我 issue-404 bug 需要 5 个小时内完成。怎么办呢？还好，Git 还提供了一个 stash 功能，可以把当前工作现场“隐藏起来”，等以后恢复现场后继续工作。如下：

```
no changes added to commit (use "git add" and/or "git commit -a")
E73-8@E73-8-PC /d/www/testGit (dev)
$ git stash
Saved working directory and index state WIP on dev: 91dfe16 merge with no-ff
HEAD is now at 91dfe16 merge with no-ff

E73-8@E73-8-PC /d/www/testGit (dev)
$ git status
On branch dev
nothing to commit, working directory clean

E73-8@E73-8-PC /d/www/testGit (dev)
$
```

将当前的工作现场隐藏起来

查看状态，是干净的

所以现在我可以通过创建 issue-404 分支来修复 bug 了。

首先我们要确定在那个分支上修复 bug ,比如我现在是在主分支 master 上来修复的，现在我要在 master 分支上创建一个临时分支，演示如下：


```
E73-8@E73-8-PC /d/www/testGit (master)
$ git checkout -b issue-404
Switched to a new branch 'issue-404'

E73-8@E73-8-PC /d/www/testGit (issue-404)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
5555555555555555
6666666666666666
7777777777777777
8888888888888888
9999999999999999
0101010101010101
bbbbbbbbbbbbbbbb

E73-8@E73-8-PC /d/www/testGit (issue-404)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
5555555555555555
6666666666666666
7777777777777777
8888888888888888
9999999999999999
0101010101010101
aaaaaaaaaaaaaaaa

E73-8@E73-8-PC /d/www/testGit (issue-404)
$ git add readme.txt

E73-8@E73-8-PC /d/www/testGit (issue-404)
$ git commit -m "fix bug 404"
[issue-404 5198735] fix bug 404
1 file changed, 1 insertion(+), 1 deletion(-)

E73-8@E73-8-PC /d/www/testGit (issue-404)
$
```

在master分支上创建临时分支issue-404

未修改前查看readme.txt内容

修改后把readme.txt内容最后一行bbbbbb改成aaaaaa

修复完成后，切换到 master 分支上，并完成合并，最后删除 issue-404 分支。

演示如下：

```

E73-8@E73-8-PC /d/www/testGit (issue-404)
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 8 commits.
(use "git push" to publish your local commits)

E73-8@E73-8-PC /d/www/testGit (master)
$ git merge --no-ff -m "merge bug fix 404" issue-404
Merge made by the 'recursive' strategy.
 readme.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

E73-8@E73-8-PC /d/www/testGit (master)
$ cat readme.txt
11111111111111111111
22222222222222222222
33333333333333333333
44444444444444444444
66666666666666666666
77777777777777777777
99999999999999999999
01010101010101010101
aaaaaaaaaaaaaaaa

```

切换到master分支上

合并分支issue-404内容

合并分支后查看内容如下，和issue-404内容一致

```

E73-8@E73-8-PC /d/www/testGit (master)
$ git branch -d issue-404
Deleted branch issue-404 (was 5198735).

E73-8@E73-8-PC /d/www/testGit (master)
$

```

在master分支上删除临时分支issue-404

现在，我们回到 dev 分支上干活了。

```

E73-8@E73-8-PC /d/www/testGit (master)
$ git checkout dev
Switched to branch 'dev'

E73-8@E73-8-PC /d/www/testGit (dev)
$ git status
On branch dev
nothing to commit, working directory clean

```

从master分支切换到dev分支上

目前干净的

工作区是干净的，那么我们工作现场去哪里呢？我们可以使用命令 `git stash list` 来查看下。如下：

```

E73-8@E73-8-PC /d/www/testGit (dev)
$ git stash list
stash@{0}: WIP on dev: 91dfe16 merge with no-ff
stash@{1}: WIP on master: 91dfe16 merge with no-ff
stash@{2}: WIP on master: 91dfe16 merge with no-ff

```

工作现场还在，Git 把 stash 内容存在某个地方了，但是需要恢复一下，可以使用如下 2 个方法：

1. git stash apply 恢复，恢复后，stash 内容并不删除，你需要使用命令 git stash drop 来删除。
2. 另一种方式是使用 git stash pop,恢复的同时把 stash 内容也删除了。

演示如下

```
E73-8@E73-8-PC /d/www/testGit (dev)
$ git stash list
stash@{0}: WIP on dev: 91dfe16 merge with no-ff
stash@{1}: WIP on master: 91dfe16 merge with no-ff
stash@{2}: WIP on master: 91dfe16 merge with no-ff

E73-8@E73-8-PC /d/www/testGit (dev)
$ git stash drop
Dropped refs/stash@{0} (d228a8c52dcf5d89aec877f0c9be774a73eb8a34)

E73-8@E73-8-PC /d/www/testGit (dev)
$ git stash list
stash@{0}: WIP on master: 91dfe16 merge with no-ff
stash@{1}: WIP on master: 91dfe16 merge with no-ff

E73-8@E73-8-PC /d/www/testGit (dev)
$ git stash list
stash@{0}: WIP on master: 91dfe16 merge with no-ff
stash@{1}: WIP on master: 91dfe16 merge with no-ff

E73-8@E73-8-PC /d/www/testGit (dev)
$ git stash drop
Dropped refs/stash@{0} (683d3fe8c3416d95e8dd25d3055a5b0f376d8f0c)

E73-8@E73-8-PC /d/www/testGit (dev)
$ git stash drop
Dropped refs/stash@{0} (753a3b3dad781bab43560494b836bab1860cd5e)

E73-8@E73-8-PC /d/www/testGit (dev)
$ git stash list

E73-8@E73-8-PC /d/www/testGit (dev)
```

删除前

删除一条

剩下2条

继续删2条

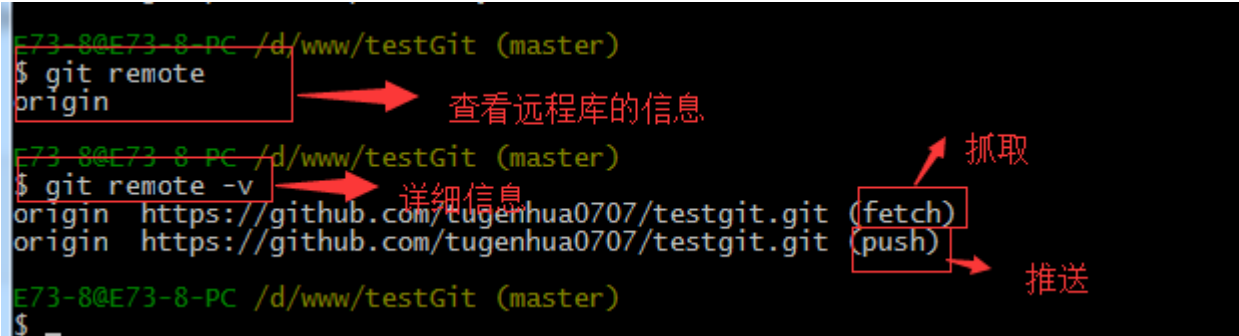
没有了

八：多人协作。

当你从远程库克隆时候 实际上 Git 自动把本地的 master 分支和远程的 master 分支对应起来了，并且远程库的默认名称是 origin。

1. 要查看远程库的信息 使用 `git remote`
2. 要查看远程库的详细信息 使用 `git remote -v`

如下演示：



```
E73-8@E73-8-PC /d/www/testGit (master)
$ git remote
origin
E73-8@E73-8-PC /d/www/testGit (master)
$ git remote -v
origin https://github.com/tugenhua0707/testgit.git (fetch)
origin https://github.com/tugenhua0707/testgit.git (push)
E73-8@E73-8-PC /d/www/testGit (master)
$
```

The screenshot shows a terminal window with three commands and their outputs. Red boxes and arrows are used to highlight specific parts and add annotations:

- A red box around `$ git remote` has an arrow pointing to the text "查看远程库的信息" (View remote repository information).
- A red box around `$ git remote -v` has an arrow pointing to the text "详细信息" (Detailed information).
- A red box around the `(fetch)` part of the output for the `origin` remote has an arrow pointing to the text "抓取" (Fetch).
- A red box around the `(push)` part of the output for the `origin` remote has an arrow pointing to the text "推送" (Push).

一：推送分支：

推送分支就是把该分支上所有本地提交到远程库中，推送时，要指定本地分支，这样，Git 就会把该分支推送到远程库对应的远程分支上：

使用命令 `git push origin master`

比如我现在的 github 上的 readme.txt 代码如下：



本地的 readme.txt 代码如下：

```
E73-8@E73-8-PC /d/www/testGit (master)
$ cat readme.txt
11111111111111
22222222222222
33333333333333
44444444444444
66666666666666
77777777777777
99999999999999
01010101010101
aaaaaaaaaaaaaa
E73-8@E73-8-PC /d/www/testGit (master)
```

本地readme.txt内容这样的

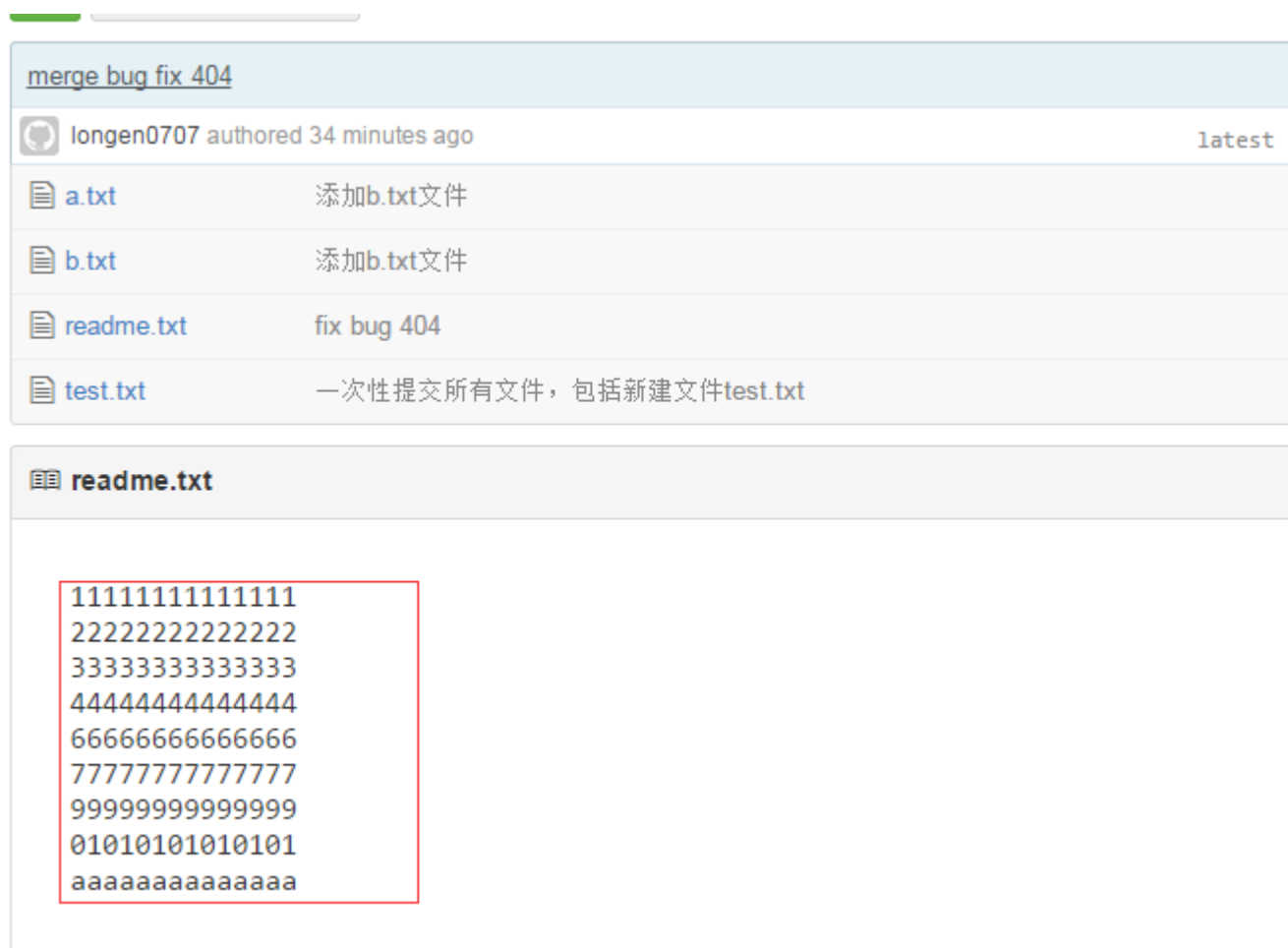
现在我想把本地更新的 readme.txt 代码推送到远程库中，使用命令如下：

```

E73-8@E73-8-PC /d/www/testGit (master)
$ git push origin master
Username for 'https://github.com': tughnhua0707@qq.com
Password for 'https://tughnhua0707@qq.com@github.com':
Counting objects: 28, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (26/26), done.
Writing objects: 100% (26/26), 2.44 KiB | 0 bytes/s, done.
Total 26 (delta 10), reused 0 (delta 0)
To https://github.com/tughnhua0707/testgit.git
2a4fd81..9b2f706 master -> master
E73-8@E73-8-PC /d/www/testGit (master)
$

```

我们可以看到如上，推送成功，我们可以继续来截图 github 上的 readme.txt 内容 如下：



可以看到 推送成功了，如果我们现在要推送到其他分支，比如 dev 分支上，我们还是那个命令 `git push origin dev`

那么一般情况下，那些分支要推送呢？

1. master 分支是主分支，因此要时刻与远程同步。
2. 一些修复 bug 分支不需要推送到远程去，可以先合并到主分支上，然后把主分支 master 推送到远程去。

二：抓取分支：

多人协作时，大家都会往 master 分支上推送各自的修改。现在我们可以模拟另外一个同事，可以在另一台电脑上（注意要把 SSH key 添加到 github 上）或者同一台电脑上另外一个目录克隆，新建一个目录名字叫 testgit2

但是我首先要把 dev 分支也要推送到远程去，如下

```
E73-8@E73-8-PC /d/www/testGit (master)
$ git push origin dev
Username for 'https://github.com': tughenhu0707@qq.com
Password for 'https://tughenhu0707@qq.com@github.com':
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/tughenhu0707/testgit.git
 * [new branch]      dev -> dev

E73-8@E73-8-PC /d/www/testGit (master)
$
```

把dev分支推送到远程

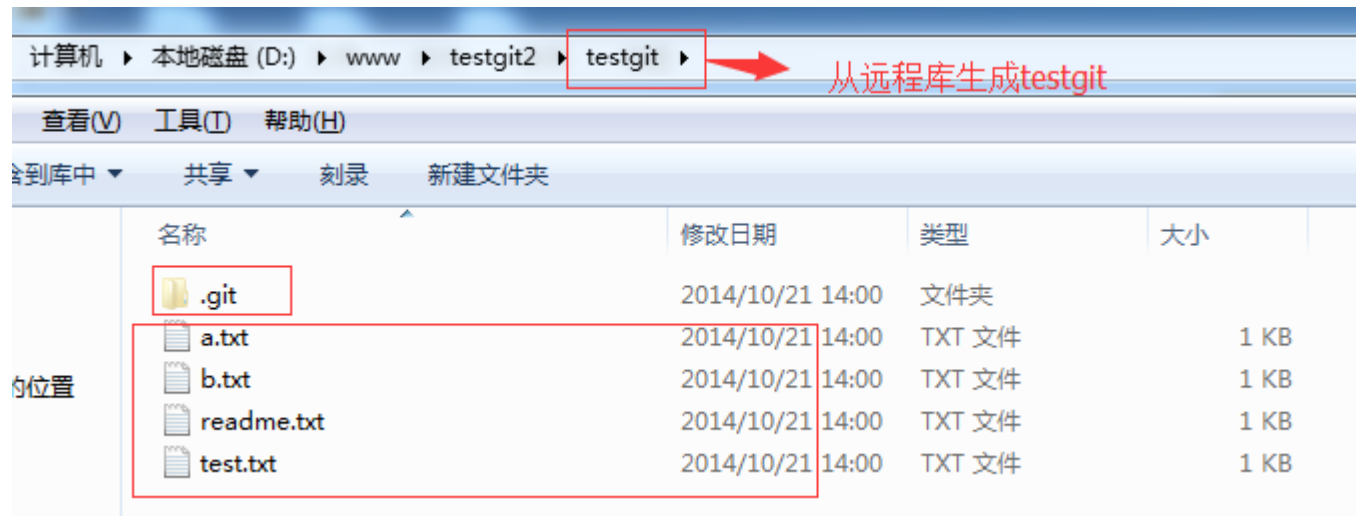
接着进入 testgit2 目录，进行克隆远程的库到本地来，如下：

```
E73-8@E73-8-PC /d/www/testgit2 (master)
$ git clone https://github.com/tughenhu0707/testgit
Cloning into 'testgit'...
remote: Counting objects: 49, done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 49 (delta 15), reused 48 (delta 14)
Unpacking objects: 100% (49/49), done.
Checking connectivity... done.

E73-8@E73-8-PC /d/www/testgit2 (master)
$
```

克隆远程库

现在目录下生成有如下所示：



现在我们的小伙伴要在 dev 分支上做开发，就必须把远程的 origin 的 dev 分支

到本地来，于是可以使用命令创建本地 dev 分支：`git checkout -b dev`

`origin/dev`

现在小伙伴们就可以在 dev 分支上做开发了，开发完成后把 dev 分支推送到远程库时。

如下：


```
E73-8@E73-8-PC /d/www/testgit2/testgit (master)
$ git checkout -b dev origin/dev
Branch dev set up to track remote branch dev from origin.
Switched to a new branch 'dev'

E73-8@E73-8-PC /d/www/testgit2/testgit (dev)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
7777777777777777
9999999999999999
0101010101010101

E73-8@E73-8-PC /d/www/testgit2/testgit (dev)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
7777777777777777
9999999999999999
0101010101010101
aaaaaaaaaaaaaaaa

E73-8@E73-8-PC /d/www/testgit2/testgit (dev)
$ git add readme.txt

E73-8@E73-8-PC /d/www/testgit2/testgit (dev)
$ git commit -m "readme.txt上增加aaaaaaaa内容"
[dev fd74bb1] readme.txt上增加aaaaaaaa内容
1 file changed, 1 insertion(+), 1 deletion(-)

E73-8@E73-8-PC /d/www/testgit2/testgit (dev)
$ git push origin dev
Username for 'https://github.com': tugenhua0707@qq.com
Password for 'https://tugenhua0707@qq.com@github.com':
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 356 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To https://github.com/tugenhua0707/testgit
91dfe16..fd74bb1 dev -> dev

E73-8@E73-8-PC /d/www/testgit2/testgit (dev)
$
```

创建远程origin的dev分支到本地来

修改前readme.txt文件内容

添加aaaaaa内容后的文件

把现在的dev分支推送到远程去

小伙伴们已经向 origin/dev 分支上推送了提交，而我在我的目录文件下也对同样的文件同个地方作了修改，也试图推送到远程库时，如下：

```
E73-8@E73-8-PC /d/www/testgit (master)
$ git checkout dev
Switched to branch 'dev'

E73-8@E73-8-PC /d/www/testgit (dev)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
7777777777777777
9999999999999999
0101010101010101

E73-8@E73-8-PC /d/www/testgit (dev)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
7777777777777777
9999999999999999
0101010101010101
aaaaaaaaaaaaaa

E73-8@E73-8-PC /d/www/testgit (dev)
$ git add readme.txt

E73-8@E73-8-PC /d/www/testgit (dev)
$ git commit -m "我自己分支dev上同样提交readme.txt文件"
[dev 63da7a4] 我自己分支dev上同样提交readme.txt文件
1 file changed, 1 insertion(+)

E73-8@E73-8-PC /d/www/testgit (dev)
$ git push origin dev
Username for 'https://github.com': tugenhua0707@qq.com
Password for 'https://tugenhua0707@qq.com@github.com':
To https://github.com/tugenhua0707/testgit.git
! [rejected]        dev -> dev (fetch first)
error: failed to push some refs to 'https://github.com/tugenhua0707/testgit.git'

hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

E73-8@E73-8-PC /d/www/testgit (dev)
$
```

切换目录到我的dev分支上

修改之前内容

给readme.txt文件添加内容aaaaaaa后

推送到远程库时发生错误，不同的人推同样的文件，修改同个文件同一个地方报错

由上面可知：推送失败，因为我的小伙伴最新提交的和我试图推送的有冲突，解决的办法也很简单，上面已经提示我们，先用 git pull 把最新的提交从 origin/dev 抓下来，然后在本地合并，解决冲突，再推送。

```

E73-8@E73-8-PC /d/www/testgit (dev)
$ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1)
Unpacking objects: 100% (3/3), done.
From https://github.com/tugenhua0707/testgit
   91dfe16..fd74bb1  dev       -> origin/dev
There is no tracking information for the current branch.
Please specify which branch you want to merge with.
See git-pull(1) for details

    git pull <remote> <branch>

If you wish to set tracking information for this branch you can do so with

    git branch --set-upstream-to=origin/<branch> dev
E73-8@E73-8-PC /d/www/testgit (dev)
$

```

git pull 也失败了，原因是没有指定本地 dev 分支与远程 origin/dev 分支的链接，根据提示，设置 dev 和 origin/dev 的链接：如下：

```

E73-8@E73-8-PC /d/www/testgit (dev)
$ git branch --set-upstream dev origin/dev
The --set-upstream flag is deprecated and will be removed. Consider using --track or --set-upstream-to
Branch dev set up to track remote branch dev from origin.

E73-8@E73-8-PC /d/www/testgit (dev)
$ git pull
Auto-merging readme.txt
CONFLICT (content): Merge conflict in readme.txt
Automatic merge failed; fix conflicts and then commit the result.

E73-8@E73-8-PC /d/www/testgit (dev|MERGING)
$

```

pull成功了，但是有冲突，需要解决，再pull

这回 *git pull* 成功，但是合并有冲突，需要手动解决，解决的方法和分支管理中的 解决冲突完全一样。解决后，提交，再 push：

我们可以先来看看 readme.txt 内容了。

```

E73-8@E73-8-PC /d/www/testgit (dev|MERGING)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
7777777777777777
9999999999999999
01010101010101
<<<<<<< HEAD
aaaaaaaaaaaaaa

=====
aaaaaaaaaaaaaa
>>>>>>> fd74bb10291a19708e7c503def84fca4a2481594
E73-8@E73-8-PC /d/www/testgit (dev|MERGING)
$

```

现在手动已经解决完了，我接在需要再提交，再 push 到远程库里面去。如下所示：

```

E73-8@E73-8-PC /d/www/testgit (dev|MERGING)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
7777777777777777
9999999999999999
01010101010101
aaaaaaaaaaaaaa

```

手动解决后的文件是这样的

```

E73-8@E73-8-PC /d/www/testgit (dev|MERGING)
$ git add readme.txt

E73-8@E73-8-PC /d/www/testgit (dev|MERGING)
$ git commit -m "merge & fix readme.txt"
[dev bbaf5ad] merge & fix readme.txt

```

继续推送到远程库中 恭喜你success了

```

E73-8@E73-8-PC /d/www/testgit (dev)
$ git push origin dev
Username for 'https://github.com': tugenhua0707@qq.com
Password for 'https://tugenhua0707@qq.com@github.com':
Counting objects: 10, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 567 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
To https://github.com/tugenhua0707/testgit.git
   fd74bb1..bbaf5ad  dev -> dev

E73-8@E73-8-PC /d/www/testgit (dev)
$

```

因此：多人协作工作模式一般是这样的：

1. 首先，可以试图用 `git push origin branch-name` 推送自己的修改.
2. 如果推送失败，则因为远程分支比你的本地更新早，需要先用 `git pull` 试图合并。
3. 如果合并有冲突，则需要解决冲突，并在本地提交。再用 `git push origin branch-name` 推送。

Git 基本常用命令

`mkdir` : `XX` (创建一个空目录 `XX` 指目录名)

`pwd` : 显示当前目录的路径。

`git init` 把当前的目录变成可以管理的 git 仓库，生成隐藏 `.git` 文件。

`git add XX` 把 `xx` 文件添加到暂存区去。

`git commit -m "XX"` 提交文件 `-m` 后面的是注释。

`git status` 查看仓库状态

`git diff XX` 查看 `XX` 文件修改了那些内容

`git log` 查看历史记录

`git reset --hard HEAD^` 或者 `git reset --hard HEAD~` 回退到上一个版本

(如果想回退到100个版本,使用 git reset --hard HEAD~100)

cat XX 查看 XX 文件内容

git reflog 查看历史记录的版本号 id

git checkout -- XX 把 XX 文件在工作区的修改全部撤销。

git rm XX 删除 XX 文件

git remote add origin <https://github.com/tughnua0707/testgit> 关联
一个远程库

git push -u(第一次要用-u 以后不需要) origin master 把当前 master 分支
推送到远程库

git clone <https://github.com/tughnua0707/testgit> 从远程库中克隆

git checkout -b dev 创建 dev 分支 并切换到 dev 分支上

git branch 查看当前所有的分支

git checkout master 切换回 master 分支

git merge dev 在当前的分支上合并 dev 分支

git branch -d dev 删除 dev 分支

git branch name 创建分支

`git stash` 把当前的工作隐藏起来 等以后恢复现场后继续工作

`git stash list` 查看所有被隐藏的文件列表

`git stash apply` 恢复被隐藏的文件，但是内容不删除

`git stash drop` 删除文件

`git stash pop` 恢复文件的同时 也删除文件

`git remote` 查看远程库的信息

`git remote -v` 查看远程库的详细信息

`git push origin master` Git 会把 master 分支推送到远程库对应的远程分支
上