

# Git使用教程

## 必须记住的六条命令。

- `cd`:用来切换工作目录,最常用的一个命令。简单来讲, `cd A文件夹` 就是进入到 `A文件夹` 里面的意思。
- `git status .`: 查看当前路径下的的状态。git下**最最常用**的一个命令。
- `git add .`: 把工作区的所有变化, (就是你的所有改动), 都添加到 版本库/暂存区。
- `git commit -m "提交时说明信息"`: 更进一步提交, 并说明提交log。
- `git push`: 把版本库的所有更新内容, 都推送到远程服务器。(就是推代码/推上去)
- `git pull`: 把代码从远程服务器拉取到本地。(俗称拉代码)

当我们修改了本地代码, 向远程服务器推送时, 我们的操作步骤如下:

1. `git add .`
2. `git commit -m "提交时说明信息"`
3. `git push`

当我们想更新本地代码, 就是把服务器上最新的代码拉取下来, 只需要执行一个命令。

`git pull`

## 这三条命令建议记住。

- `git log`:查看提交历史, 与各次的提交说明。
- `git diff`:比较工作区与暂存区的差异, 就是比较看看你到底都做了什么修改。
- `git clone url地址`: 将远程服务器上项目克隆到新创建的目录中 (第一次拉项目时使用, 后面的更新都用 `git pull`了)。

## 其他问题

- 操作时 双击 `tab` 键的自动提示/补全功能。
- `q` 或者 `:q` 等命令代表退出(quit)。
- `ctrl+f`, `ctrl+b` 快捷键在terminal可以翻页, 就是 上一页, 下一页

# 文章正文

git是一个分布式版本控制系统。简单来讲, 如果有几个人同时开发维护一个项目的代码, 那么我们就找个中央服务器, 放置一份公共的代码, 每个人在各自的电脑上去修改各自的代码, 然后修改完, 提交到中央服务器。这样大家拉代码时, 就能更新到其他人修改的内容了。。

Notice:代码只是为了便于说明。版本控制系统, 管理的是文件, 所以任何文件都可以。图片啦, 视频文件啦, 二进制文件啦, 没有什么不可以的。只是我们为了行文方便, 直接说代码文件。

本教程会讲述 命令行 操作。不过也有很多人 使用图形化界面软件 比如 `source tree` 或者 俗称的 `小乌龟` 软件 来操作。但是基底的原理是一样的。`source tree` 软件操作也只是命令行的 封装。并且图形化操作更加直观一些。熟悉了命令行, 图形化软件操作自然也会。不会命令行, 图形化软件操作也可以会, 但是会理解的比较肤浅。更重要的是会了其中一个, 学习另一个就非常容易了。

本篇文章之所以采用 终端(terminal)命令行的方式, 除了我本人平时一直使用命令行操作之外, 还有重要的一点, 命令行具有更广泛的适用性。换句话说, 你熟悉了 git的命令行, 那么利用命令行进行其他操作, 比如java, python, 运行测试脚本等, 对你来说很easy, 理解了最底层的原理, 学习图形化软件也会很容易, 毕竟图形化软件那么多, 你永远学不完的, 但是理解了底层的, 就能以不变应万变。

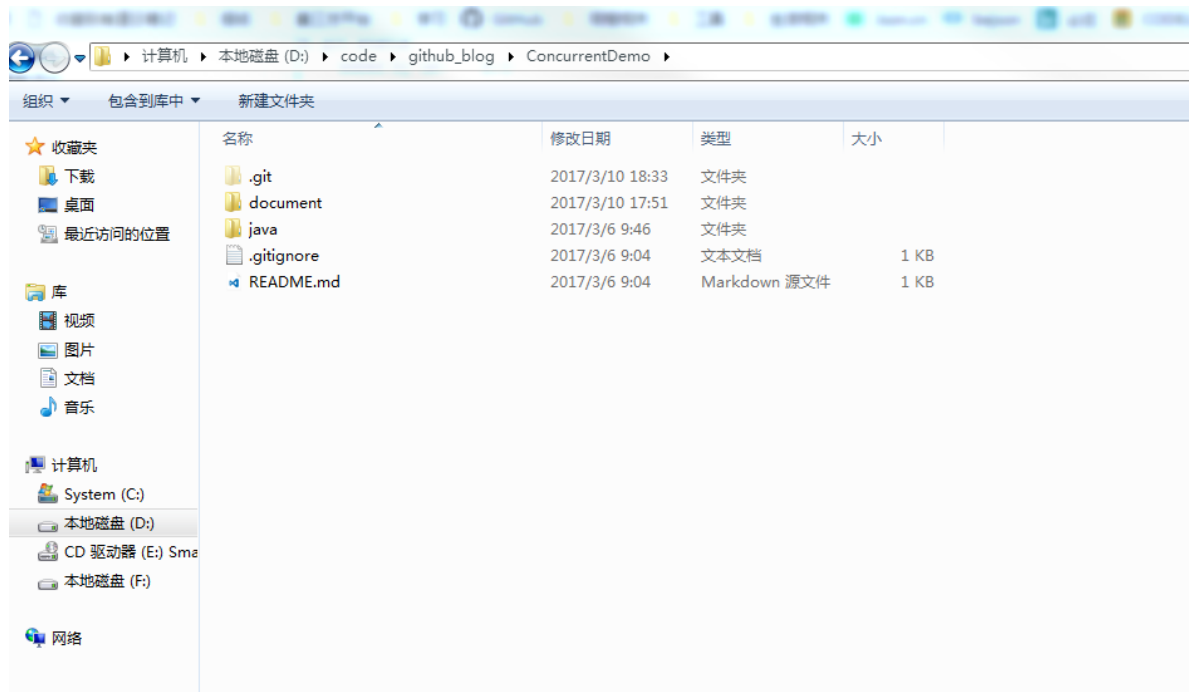
# 理解几个概念

工作区 (Working Directory) , 版本库 (Repository) /暂存区 , (中央/远程) 服务器.

- **服务器**的概念已经清楚了。叫做 中央服务器/远程服务器都行。
- **工作区**:就是你电脑的工作目录
- **版本库**:工作区有一个隐藏的 `.git` 文件夹, 这个是叫做 版本库(有些文章也叫 **暂存区**, 不管叫什么, 知道这个意思就好)。`.git` 是隐藏文件夹。该文件内的内容很重要, 因为git的控制配置等信息, 都在这个隐藏文件夹里。电脑如果设置不显示隐藏文件夹, 那么就会看不到。

我电脑上的一个项目, 可以看到什么是工作区, 暂存区.

图片名称: 工作区与暂存区.png



## 为什么存在一个 版本库?

我修改过的代码, 直接从 **工作区**提交到**服务器**不就行了嘛, 为什么还要这么麻烦。 `svn` 等集中式版本管理系统就是这么做的, 简单明了, 但是如果你没网络时怎么办? 所以有了 **版本库**, 那么你可以把代码先从工作区提交到版本库, 等待有网络了, 可以再提交到服务器。

## `.gitignore` 文件是干啥的?

工作区的目录下面, 总会存在很多乱七八糟的文件, 比如你本地的配置, 编译生成的中间文件等, 这些文件你不想(或不能)提交到 服务器。那怎么办呢。就把这些文件的规则写到 `.gitignore` 文件中, 这样git就会 ignore(忽略)这些文件, git就会像没看到这些文件一样。

比如我的 `.gitignore` 文件有些内容如下:

```
*.apk
*.class
bin/
gen/

# Windows thumbnail db
Thumbs.db

# OSX files
.DS_Store
```

这几句话的意思是 所有apk后缀的文件, class后缀的文件都忽略, bin/ 和 gen/ 目录下的文件也忽略。说的通俗一点, 就是你git别管这些文件了, 这些和git没屁关系。我不管怎么倒腾这些文件, 都和git没关系。另外, .gitignore 文件中, # 号开头的行 代表注释, 就像 编程文件中的 // 开头的行一样。

## 几个简单的命令

怎么创建一个被git控制的项目, 后面再讲, 这里先讲述几个基本命令。

在此之前, 先熟悉一个 cd 命令。

cd 命令用来切换工作目录, linux, mac环境下最常用的一个命令。简单来讲, cd A文件夹 就是进入到A文件夹里面的意思。比如我要进入d盘我的代码文件夹, 输入命令 cd /d/code/github\_blog/ 然后回车。

假设你在一个项目中修改了某些文件。

你想看看当前目录下是什么状态, 命令 git status 。

. : 一个点代表当前目录, .. : 两个点 代表上级目录。(这和git无关, 这是 计算机基本的常识)。那么请思考, cd .. ,这个命令是啥意思?

但是你敲命令的时候, 记不清楚了, 或者 打错了了。看看termial会有什么反应?

```
$ git satds
git: 'satds' is not a git command. See 'git --help'.

Did you mean this?
    status
```

有很多的概念和操作, 都和什么git无关, 都是计算机领域中的基本常识, 或者 所有(至少绝大部分) 软件都遵循的操作常识, git自然也同样遵循这样概念和操作, 这些内容我会用斜体的 计算机常识 来标注。

\$ git satds 一行, \$ 代表的是命令行的开始, 后面的内容代表的就是你的输入内容。而 它的下一行就是系统反馈/回应 你的内容。(或者说系统输出)。(计算机常识)

你看到什么? git会问你, 它不认识这个命令啊, 你是不是敲错了, 你可以用 git --help 寻求帮助哦。另外你是不是想打 status 这个词呢? 所以git的命令根本 不用背, 有个简单的印象就好。

甚至, 你在敲打命令的时候, 根本不用敲完, 输入头几个字符, 然后直接 敲击 tab 键, 看看会发生什么?

```
$ git sta //敲击 tab
stage      stash      status
```

系统直接提示你了，sta开头的命令有三个，就看你想用哪个了，这是为了你记忆。而你敲击 `git stat` 之后，再敲击 `tab`，再看看会发生什么。因为此时 `stat` 开头的命令只剩下一个了 `status`，所以你也只能打这个命令了。所以git自动帮你补全了。。

点击 `tab` 键的自动提示/补全功能很有用，绝大部分命令行操作都有这个快捷键，毕竟那么长的命令，文件路径等，记忆很难，打字也很累。不过有些terminal情况，是点击一次 `tab` 键，而有些则是双击 `tab` 键，反正你可以总是双击 `tab` 键，这总不会错。(计算机常识)

费了好大劲，我们终于输入了正确的字符。看看 命令行 会输出什么内容。

图片名称: `git_status.png`

```
Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$ git status .
On branch master
Your branch is up-to-date with 'origin/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)

        modified:   java/src/thread_runnable/CachedThreadPool.java
        deleted:    java/src/thread_runnable/DirectThread.java

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

        test0908.txt

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

我在测试前，做了以下几件事情：

- 新建一个 `test0908.txt` 文件。
- `CachedThreadPool.java` 文件中修改了一些内容。
- 删除了 `DirectThread` 文件。

然后你看看 git 输出的内容。我们逐行进行分析。

```
On branch master
Your branch is up-to-date with 'origin/master'.
```

第一行不用管，这是 分支(branch)的概念，基础教程不涉及分支。

第二行：你的分支与远程分支已经同步了。（就是远程服务器并不比你的代码新）

```
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)

        modified:   java/src/thread_runnable/CachedThreadPool.java
        deleted:    java/src/thread_runnable/DirectThread.java
```

翻译一下：

改变还没有到提交阶段呢。  
(使用 ``git add <file>...`` 命令 来更新 你的提交。)  
(使用 ``git checkout -- <file>...`` 命令来放弃你工作区的修改 )  
然后下面 列出了你修改的具体文件。  
``modified``代表修改，``deleted``代表 删除。(还有add表示增加，像svn中就直接简写M, D, A了，如果看到了这些简写了，要明白什么意思)

提示内容已经非常明明白白的告诉你了，你的修改内容，以及你下一步可以怎么做了。  
你可以使用 `git add` 命令来提交;也可以使用 `git checkout` 来放弃修改。(就是 把工作区重新变干净, 把你修改的东西都恢复了, 就像 `ctrl+z` 一样)。

然后还有几行:

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

    test0908.txt

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

翻译如下:

```
未跟踪的文件
(使用 `git add <file>...` 命令会包括下面这些提交)
    你添加的文件名
    现在还没走到commit地步呢。(可以使用 `git add` 或者 `git commit -a`)
```

首先 `test0908.txt` 为什么是 `Untracked files`,因为我刚才就说了, 我的这个文件是 新添加的, `git` 之前没见过这个文件(`git`刚刚第一次见到这个文件, 所以感觉很面生, 不认识啊)。所以它说这个文件未跟踪, 而上面那两个文件 `CachedThreadPool.java` 和 `DirectThread.java` 这两个文件, 因为之前早就添加了,所以`git`系统会认识这两个文件。

那么现在工作区就是这个样子了。

我想看看我具体到某个文件进行了什么修改。该怎么操作呢。

`git diff` 操作。

```
Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$ git diff
diff --git a/java/src/thread_runnable/CachedThreadPool.java b/java/src/thread_runnable/CachedThreadPool.java
index b1ba60f..5c31277 100644
--- a/java/src/thread_runnable/CachedThreadPool.java
+++ b/java/src/thread_runnable/CachedThreadPool.java
@@ -5,6 +5,10 @@ import java.util.concurrent.Executors;

public class CachedThreadPool{
+
+    /**
+     * 这么简单的东西，到底有什么学不会的。
+     */
+
+    public static void main(String[] args) {
+        ExecutorService exec = Executors.newCachedThreadPool();
@@ -15,7 +19,6 @@ public class CachedThreadPool{
+    }
+
+    exec.shutdown();
+    System.out.println("main end " + Thread.currentThread().getName());
+
+}

diff --git a/java/src/thread_runnable/DirectThread.java b/java/src/thread_runnable/DirectThread.java
deleted file mode 100644
index 135fece..0000000
--- a/java/src/thread_runnable/DirectThread.java
+++ /dev/null
@@ -1,24 +0,0 @@
-package thread_runnable;

-
-
-public class DirectThread extends Thread{
-    protected int countDown = 5;
-
-    public String status() {
-        return " == == (" + (countDown>0 ? countDown : "Over") + ") , " + Thread.currentThread().getName();
-    }
-    @Override
-    public void run() {
-        // TODO Auto-generated method stub
-        while(countDown-- > 0){
-            System.out.println(status());
-        }
-    }
-
-    public static void main(String[] args) {
-        System.out.println("DirectThread --- main start, " + Thread.currentThread().getName());
-        new DirectThread().start();
-        System.out.println("DirectThread --- main end, " + Thread.currentThread().getName());
-    }
-}

Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$
```

这个内容显示的有些乱七八糟。

首先你看到 它列出了第一个文件

```
diff --git a/java/src/thread_runnable/CachedThreadPool.java
b/java/src/thread_runnable/CachedThreadPool.java
```

前面一个a，后面一个b，其实就是代表你修改前后的文件。(不用关心这些)。

然后下面，是具体内容，

- 所有 + 开头的，代表的都是你添加的内容，
- 所有 - 开头的，代表的都是你删除的内容。
- 那些既没有"+"也没有 "-"开头的行，就是和你修改区域的相关上下文，有了这些上下文，可以更好地帮你回想起来，你到底都修改了什么了。

并且它们显示的颜色也不同。

然后它列出了你修改的第二个文件。

```
diff --git a/java/src/thread_runnable/DirectThread.java
b/java/src/thread_runnable/DirectThread.java
deleted file mode 100644
```

它也提示你了，你把这个文件删除了。

而具体的提示，则全部是红色的 - 号区域。为什么是全部是 - 号区域，因为你把这个文件都删除了，那自然是相当于你把所有的内容都删除了。

为什么 你添加的 test0908.txt 文件没有被这个命令提示，因为 这个文件还没有被跟踪，再说，也没必要显示啊。因为这个文件的所有内容 都是你新添加的。

我把terminal的界面调整到最大了。所以可以全部输出，如果文件改动很多/terminal界面太小，一屏幕输出不完。那么 ctrl+f, ctrl+b 快捷键分别显示 上一页，下一页，q 或者 :q 等命令代表退出 (quit)。(计算机常识)

上面虽然解释了 git diff 命令的意思。但是这个显示的确让人眼花缭乱。而 source tree 等图形化工具，关于这个对比显示，的确直观了很多。

图形化界面是类似下面这样显示的，看着 直观了许多。

图片名称: git\_diff\_图形化界面.png

```

Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$ git diff
diff --git a/java/src/thread_runnable/CachedThreadPool.java b/java/src/thread_runnable/CachedThreadPool.java
index b1ba60f..5c31277 100644
--- a/java/src/thread_runnable/CachedThreadPool.java
+++ b/java/src/thread_runnable/CachedThreadPool.java
@@ -5,6 +5,10 @@ import java.util.concurrent.Executors;

public class CachedThreadPool{

+    /**
+     * 这么简单的东西，到底有什么学不会的。
+     */
+
    public static void main(String[] args) {
        ExecutorService exec = Executors.newCachedThreadPool();
@@ -15,7 +19,6 @@
    }

    exec.shutdown();
    System.out.println("main end " + Thread.currentThread().getName());
}

diff --git a/java/src/thread_runnable/DirectThread.java b/java/src/thread_runnable/DirectThread.java
deleted file mode 100644
index 135fece..0000000
--- a/java/src/thread_runnable/DirectThread.java
+++ /dev/null
@@ -1,24 +0,0 @@
-package thread_runnable;

-
-
-public class DirectThread extends Thread{
-    protected int countDown = 5;

-    public String status() {
-        return "###{ " + (countDown>0 ? countDown : "Over") + ")", " + Thread.currentThread().getName();
-    }

-    @Override
-    public void run() {
-        // TODO Auto-generated method stub
-        while(countDown-- > 0){
-            System.out.println(status());
-        }
-    }

-    public static void main(String[] args) {
-        System.out.println("DirectThread --- main start, " + Thread.currentThread().getName());
-        new DirectThread().start();
-        System.out.println("DirectThread --- main end, " + Thread.currentThread().getName());
-    }
-}
Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$

```

(其实eclipse等IDE，都附带了类似的工具帮助你比较你都修改了什么，显示结果用图形化界面形式来显示，比较直观，不过这里就不做具体说明了。但是 `git diff` 的确比较少用，因为这种terminal输出看的眼睛都花了)

我们 已经知道了工作区的状态，也知道修改了哪些内容。  
那么下面该做什么呢。

`git status` . 时，已经很清楚的提示下一步命令是什么了。

我们先把文件从 工作区提交到 版本库。

本次提交时，你可以只添加某一个文件，其他文件你没修完还不想提交呢: `git add`

`java/src/thread_runnable/CachedThreadPool.java`

也可以 一次提交两个文件(文件中间空格分割): `git add`

`java/src/thread_runnable/CachedThreadPool.java test0908.txt`

你当然也可以一起提交所有的修改。 `git add .` (一般常用的就是这个命令,修改了就全部添加，省的麻烦)

思考一下最后一个命令为什么是这样，不要忘记一个点 `.` 代表什么。

更别忘记了刚才强调的 `tab` 快捷键， 否则那么长的文件路径打字累不累啊。(当然，你复制粘贴也可以的)

好。我们把 所有文件提交了，

`$ git add .`

这一步没有没有任何输出。

此时 `git status` . 看看 什么状态。



图片名称: git\_status\_after\_add.png

```
Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$ git add .

Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$ git status .
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   java/src/thread_runnable/CachedThreadPool.java
        deleted:    java/src/thread_runnable/DirectThread.java
        new file:   test0908.txt

Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$
```

提示区域已经告诉我们可以怎么做了。

你可以使用 `git reset HEAD <file>...` 来恢复上一步操作。

这里提示了怎么进行恢复。而上面那句话是啥？你可以进行 committed 啊,(当然, git直接提示使用 `git commit` 进行更进一步的提交才完美。)

然后 我执行 下面命令, 进行更进一步的提交操作。

```
git commit -m "我这只是一次提交测试, 进行教学的提交测试"
```

命令的 `-m "提交说明"` 是添加说明的。

此时你的代码改动都已经放到了 版本库了。

然后再 `git status .`,看看提示了什么:

图片名称: git\_status\_after\_commit.png

```
Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$ git status .
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   java/src/thread_runnable/CachedThreadPool.java
        deleted:    java/src/thread_runnable/DirectThread.java
        new file:   test0908.txt

Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$ git commit -m "我这只是一次提交测试, 进行教学的提交测试"
[master 0e522fe] 我这只是一次提交测试, 进行教学的提交测试
3 files changed, 5 insertions(+), 25 deletions(-)
delete mode 100644 java/src/thread_runnable/DirectThread.java
create mode 100644 test0908.txt

Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$ git status .
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean

Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$
```

下面几句话值得注意:

```
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
```



`origin`是原点，远程的意思。所以这句话是这个意思。

你可以 使用 `git push` 来向远程 发布你的 变化。(其实就是推送到远程服务器>)

那么我们就 `git push` 吧。。把 版本库的东西，推送到远程服务器。因为你写了代码，本来就是为了提交到远程服务器嘛。。

我们来看看 `git push` 之后，提示了什么。

图片名称: `git_push.png`

```
Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$ git push
warning: push.default is unset; its implicit value has changed in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the traditional behavior, use:

    git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

    git config --global push.default simple

When push.default is set to 'matching', git will push local branches
to the remote branches that already exist with the same name.

Since Git 2.0, Git defaults to the more conservative 'simple'
behavior, which only pushes the current branch to the corresponding
remote branch that 'git pull' uses to update the current branch.

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

Enter passphrase for key '/c/Users/Administrator/.ssh/id_rsa':
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (7/7), 671 bytes | 0 bytes/s, done.
Total 7 (delta 4), reused 0 (delta 0)
To git@git.oschina.net:yaowen369/ConcurrentDemo.git
 5c53c8f..0e522fe master -> master
Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$
```

这次操作输出的内容也是让人眼花缭乱，上面的那一部分同样是相关提示，告诉我们可以使用更详细的命令，这个区域不用管。我也没研究过什么内容，关注一下我红框画出来的内容。

其实在输入 `git push` 命令时，输出会停留在下面这一行，等待你输入密码。

Enter passphrase for key '/c/Users/Administrator/.ssh/id\_rsa':

这个是我们最初设置git环境时，设置的ssh密码，比如我的电脑上设置的是 123456。输入了密码，然后才能输出下面那些内容，代表此时我们的操作已经完成了。

有些人的电脑上，使用 `git push` 时，并没有输入密码这一步骤，那是因为他们最初配置git环境时，把密码这步省略了(或者记住了密码)，所以他们不用输入密码。

下面这几句内容。

```
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
writing objects: 100% (7/7), 671 bytes | 0 bytes/s, done.
Total 7 (delta 4), reused 0 (delta 0)
```

是一些push时的 提示内容，不必关心(反正我也看不懂)。。再下面两句。

```
To git@git.oschina.net:yaowen369/ConcurrentDemo.git
5c53c8f..0e522fe master -> master
```

To 代表的意思是目的地，你本地提交到了哪里？因为我的这个项目放在了 `oschina` 托管网站上了，(oschina地址:<http://git.oschina.net/>) 所以就是说，我的代码被提交到了 那个地址了。本地master分支到 远程master分支。

`oschina` 和github一样，都是代码托管网站。（当然，你托管其他文件自然也可以）。类似的网站有很多，还有 `gitlab` 之类的，不过github最有名罢了。所谓托管，意思就是相当于他们提供了一个远程服务器，供你放置你的文件。这个问题下面的内容会讲到。

到了这个步骤，我们修改的内容，都被提交到了远程服务器上，虽然有时候敲击命令时，`termial` 提示了我们很多乱七八糟看不懂的内容，但是这不重要，看不懂也很正常，完全看懂也没必要，因为我们的目的已经达到了，就是把本地修改 提交到了 远程服务器上了。

此时我们再使用 `git status`，看看当前状态是什么。

图片名称: `git_status_after_pull.png`

```
Enter passphrase for key '/c/Users/Administrator/.ssh/id_rsa':
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (7/7), 671 bytes | 0 bytes/s, done.
Total 7 (delta 4), reused 0 (delta 0)
To git@git.oschina.net:yaowen369/ConcurrentDemo.git
5c53c8f..0e522fe master -> master

Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean

Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$
```

上面的意思是说

你当前的分支已经和`origin/master`一样新了(就是内容一致了)。  
没有什么东西要提交，你的工作区很干净。

此时，我打开了我的mac电脑，我的mac电脑上也有这个项目。我有时候在公司电脑上写代码，有时候在自己的mac电脑上写代码，所以有了git，都可以方便的在不同电脑上切换。

我在自己的mac电脑上，进入到对应的代码目录，使用了 `git pull` 命令，看看什么反应。

有些时候，也会要求你输入 `ssh` 的命令，才能拉代码，直接输入之前设置的123456密码就行了。

图片名称: `git_pull.png`

```
morandeMacBook-Pro:ConcurrentDemo yw$ git pull
remote: Counting objects: 7, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 7 (delta 4), reused 0 (delta 0)
Unpacking objects: 100% (7/7), done.
From https://git.oschina.net/yaowen369/ConcurrentDemo
5c53c8f..0e522fe master -> origin/master
Updating 5c53c8f..0e522fe
Fast-forward
 java/src/thread_runnable/CachedThreadPool.java | 5 ++++
 java/src/thread_runnable/DirectThread.java      | 24 -----
 test0908.txt                                     | 1 +
3 files changed, 5 insertions(+), 25 deletions(-)
delete mode 100644 java/src/thread_runnable/DirectThread.java
create mode 100644 test0908.txt
morandeMacBook-Pro:ConcurrentDemo yw$
```

最上面的那几句话不用看(我也看不懂,也完全没看懂的必要)。  
只看最重要的部分

```
From https://git.oschina.net/yaowen369/ConcurrentDemo
```

```
5c53c8f..0e522fe master -> origin/master
Updating 5c53c8f..0e522fe
Fast-forward
 java/src/thread_runnable/CachedThreadPool.java | 5 ++++-
 java/src/thread_runnable/DirectThread.java      | 24 -----
 test0908.txt                                     | 1 +
```

这个文件从 `oschina` 远程服务器拉下来了, 并且这次 拉取 更新了那些文件呢。。就是下面三个。

```
java/src/thread_runnable/CachedThreadPool.java | 5 ++++-
java/src/thread_runnable/DirectThread.java      | 24 -----
test0908.txt                                     | 1 +
```

这不就是我在自己办公电脑上修改的那三个文件吗? 并且看到后面的 `+`, `-` 符号, 它还告诉你了, 有些我们添加内容了, 有些我们删除内容了。所以到了这个时候,  
我们通过 `status`, `add`, `commit`, `push`, `pull` 这五个简单的命令, 我们就能简单的使用 `git`了。。这已经能满足我们日常80%的需求了。(对于一个人开发的项目, 而不是 团队多人开发模式来讲, 这五个命令已经能满足95%的日常需求了)

至于有些命令的提示输出内容等, 看不懂就看不懂了, 都看懂又有啥用。我们只要会用就可以了。比如 `git push` 命令之后, `terminal`提示输出了那么多的内容, 你只要简单的能看懂, 我们推送成功(还是失败)了这就ok了。其他的不用管。

## 冲突问题

可是人生哪有处处都如意的时候呢, 代码也是如此。如果A和B都同时修改了同一个文件会发生什么呢, 此时就会发生冲突。

比如张三修改了 `A.java`文件上传到了中央服务器, 然后李四在本地也修改了 `A.java`文件, 李四想提交文件时, 会提示因为冲突而无法提交。系统要求你先把代码拉下来, 合并了`A.java`的冲突(这个合并过程, 其实有时候`git`会自动合并, 但是复杂的合并`git`做不了, 所以就要求李四自己去合并冲突), 然后李四才能提交上去。。。

当然, 你工作电脑上提交, 然后 自己私人笔记本上又修改了同一个文件, 这和上面是同样的意思, 只是 我们用张三李四来方便表达。

所以为了避免潜在冲突一个好习惯就是 你在修改你的代码之前, 先 `git pull`一下, 把服务器的最新代码拉下来,这是一个好习惯。

因此我们开发时, 有时候早晨来了第一件事情, 就是先把代码 `git pull`一下, 进行更新。这是个好的开发习惯。避免你写了很多代码, 你同事也写了很多代码, 然后冲突了, 你们俩合并的时候, 比较浪费时间。

我现在就实际做一个冲突的demo, 演示冲突是怎么发生的, 又怎么解决。其实都很好解决。。

我在自己的mac电脑上修改了 `FixedThreadPool.java` 文件, 然后 在mac电脑上 操作了 `add`, `commit`, `push` 操作, 提交到了远程服务器上。

而同时, 我在自己的工作电脑上, 也对 `FixedThreadPool.java` 文件 进行了修改。然后我在自己的工作电脑上, 执行了 `add`, `commit` 操作, 现在我要提交了, 我执行了 `push` 操作, 看看会发生什么。

图片名称: git\_push\_error\_because\_confilt.png

```
MINGW32/d/code/github_blog/ConcurrentDemo
(use "git checkout -- <file>..." to discard changes in working directory)
        modified:   java/src/thread_runnable/FixedThreadPool.java
no changes added to commit (use "git add" and/or "git commit -a")
Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$ git add .
g
Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$ git commit -m "我在工作电脑上修改了改文件，待会去pull，看看会发生什么"
[master 5b90fa3] 我在工作电脑上修改了改文件，待会去pull，看看会发生什么
1 file changed, 3 insertions(+)

Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$ git push
warning: push.default is unset; its implicit value has changed in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the traditional behavior, use:

    git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

    git config --global push.default simple

When push.default is set to 'matching', git will push local branches
to the remote branches that already exist with the same name.

Since Git 2.0, Git defaults to the more conservative 'simple'
behavior, which only pushes the current branch to the corresponding
remote branch that 'git pull' uses to update the current branch.

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

Enter passphrase for key '/c/Users/Administrator/.ssh/id_rsa':
To git@git.oschina.net:yaowen369/ConcurrentDemo.git
! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'git@git.oschina.net:yaowen369/ConcurrentDemo.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.
Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$ !
```

此时再看看对方 termial提示了什么。输出了一大串内容，前面的内容不用关心。我们只看最后一段主要内容。

```
To git@git.oschina.net:yaowen369/ConcurrentDemo.git
! [rejected]        master -> master (fetch first)
error: failed to push some refs to
'git@git.oschina.net:yaowen369/ConcurrentDemo.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.
```

翻译其中的主要内容：

```
! [拒绝]          master -> master (fetch first)
错误：向'git@git.oschina.net:yaowen369/ConcurrentDemo.git'执行 `push`时，发生了某些错误。
提示：更新之所以被拒绝是因为 远程分支包含了你本地没有的内容，这通常是因为 另一个库推送了同样的文件(ref是索引的意思，可以翻译成文件)。你可以在推送之前先合并这些远程的变化(比如，试试 git pull)。
```

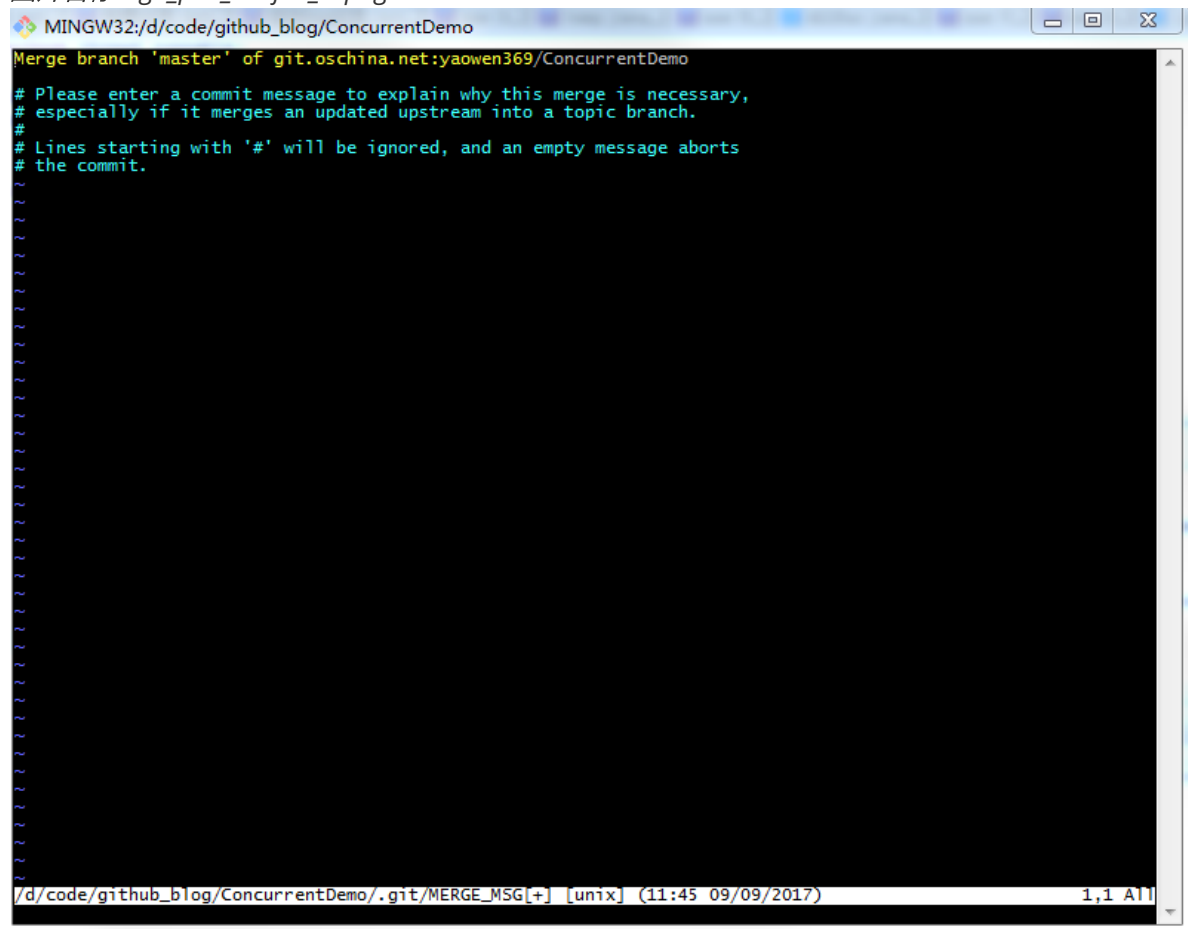
你可以看看 `git push --help`中的 `Note about fast-forwards`了解更多的细节。

其实在命令行中，help是很有用的，可以提示很多有用的帮助信息，不过有些termial要求直接 命令后面输入 help就好了，有些要求输入 `-help`，有些要求输入 `--help`，或者有些直接输入 `-h`/`--h`也行，但是我们始终要有这个意识，因为太多东西不用记忆，有个大概的印象就好。(计算机常识)

看到这些提示内容，即使你第一次碰到这个问题，你下一步准备怎么做？人家已经给你提示了啊。直接 `git pull` 啊。

**Notice:**在你输入 `git pull` 时，有时候terminal会要求你输入密码，有时候不会，但是 很快的，terminal 就会完全的跳转到一个新的页面，这应该是你第一个碰到这种情况。

图片名称: `git_pull_confilit_vi.png`



这其实是个 `vi` 编辑器，(`vim` 是 `vi` 的升级版，因为 `vim` 颜色高亮做的比较好，看起来更舒服)。

我们在之前执行 `git commit -m "相关提交的log内容"` 这个命令时，直接输入一行提交说明内容，所以没那么复杂，我们就写一句话说明一下而已(搞那么复杂干啥子)。但是你如果执行 `git commit`，不带 `-m` 然后你直接敲击回车，也会进入这个 `vi` 页面。(因为你没使用一行的提交说明模式，系统会以为你想长篇大论的去写提交信息呢，所以专门给你准备个编译器，你好好写吧，想写多少，就写多少)

说简单点，`vi` 和你的 `word`, `notepad`, `sublime text` 没啥区别，包括和你电脑上新建个 `文本文档`，都是一回事，都只是一个 **文本编译器**，但是这个 `vi` 的历史可比后面的那几个历史早太多了，上世纪八十年代，电脑图形化界面还没发明呢，当时电脑操作都是黑乎乎的命令窗口操作(其实现在 `window`, `linux` 也可以直接黑乎乎的terminal操作，只是那么多命令，大家都记不住，有了鼠标和图形化界面，黑乎乎的命令行操作都被忘记了，只剩下程序员使用terminal了)。 `word` 等更无从谈起，但是大家很多时候也要编译文本啊，又没有 `word` 等，所以 `vi` 就是一个当时环境下的 terminal 环境操作的 **文本编译器**，完全 键盘操作，有无数复杂的 快捷键，你使用 `vi` 操作，完全不用接触鼠标，所以操作也比较快(当然是在你比较熟悉快捷键的情况下，否则你就尴尬了)。

我们这里呢，不讨论 `vi`。 `vi` 的操作是另一个话题。(其实也不是难，而是那么多复杂的快捷键组记忆着比较困难而已)。

我们可以看看 它上面的内容说了什么。

```
Please enter a commit message to explain why this merge is necessary,  
  
Lines starting with '#' will be ignored
```



翻译内容：

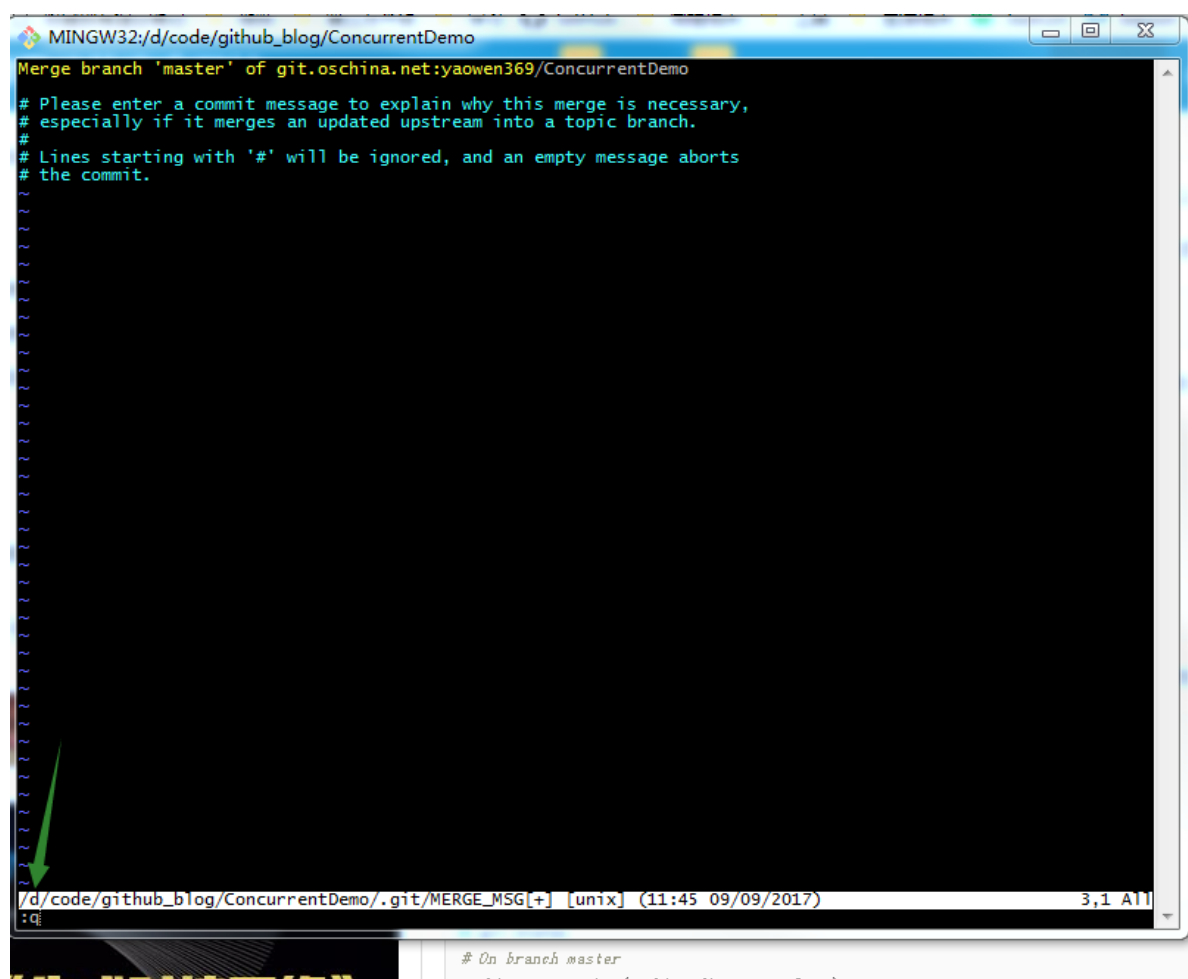
请输入一些提交内容来解释为什么这次合并是必须的。  
以`#`号开头的行都会被忽略（注释的作用）

其实上面那句 Merge branch 'master' of git.oschina.net:yaowen369/ConcurrentDemo,就是它默认帮你生成的 提交信息。

反正你不用管（因为你要搞定这个，这是另一个学习内容，但是你学习这些完全没必要，虽然也不难。这也是source tree等软件的好处，使用了 source tree等图形化软件，你怎么着也不会碰到vi界面），还记得 怎么退出不？

输入 `:q`, 字符 `q` 代表是退出(quit)的意思，不过这个 `vi` 的退出要 一个冒号+`q`，所以你输入 `:q`，直接退出就好了。（git有些界面退出也是 `:q`，只是你输入命令操作的时候，git自动帮你前缀一个冒号了，所以给你省事了而已）

你输入 `:q`，注意左下角。



vi当中 `:` 开头的都是命令模式，命令模式显示都在左下角。你输入 `:q` 回车后，左下角出现了这么一行红色的文字。

E37: No write since last change (add ! to override)

你输入 `:q` 竟然没用，相关区域提示你，因为这个文件你啥都没改动，所以你要加个 `!` 号去覆盖(其实就是强制退出模式)。然后你直接输入 `:q!` 就好了。（直接输入 `:` 就好了，那行红色的提示就消失了，然后你接着输入 `q!` 就好了，你试图用键盘上的 `delete` 等按键去删除红色文字没用的）

还要讨论vi的相关内容。但是不讨论用户十有八九又会碰到这个问题，当年我第一次碰到vi问题，连怎么退出都搞不定，急的满头大汗。所以不得不讨论。

终于我们退出了 `vi`，看看具体提示了什么。。

```
Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$ git pull
Enter passphrase for key '/c/Users/Administrator/.ssh/id_rsa':
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 6 (delta 4), reused 0 (delta 0)
Unpacking objects: 100% (6/6), done.
From git.oschina.net:yaowen369/ConcurrentDemo
 0e522fe..c63c40b  master    -> origin/master
Auto-merging java/src/thread_runnable/FixedThreadPool.java
[No write since last change]
/usr/bin/bash: q: command not found
shell returned 127
Press ENTER or type command to continue
[No write since last change]
/usr/bin/bash: q: command not found
shell returned 127
Press ENTER or type command to continue
Merge made by the 'recursive' strategy.
 java/src/thread_runnable/FixedThreadPool.java | 4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)
Administrator@zr MINGW32 /d/code/github_blog/ConcurrentDemo (master)
$
```

红色方框内的内容不要管，那是因为我第一次在`vi`中输入命令时，输错了 `:!q`，所以`vi`提示 `q`命令找不到，我又重新进入输入了 `:q!`，就ok了。。

我们看重点内容：

```
Auto-merging java/src/thread_runnable/FixedThreadPool.java

Merge made by the 'recursive' strategy.
 java/src/thread_runnable/FixedThreadPool.java | 4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)
```

注意：`Auto-merging` 自动合并了。也就是说，因为我刚才测试的冲突比较简单，所以 `git` 自动比较合并了。（比如张三修改了文件的第一行，李四修改了文件的最后一行，这种简单的，`git` 就能自动合并，但是张三李四都修改了文件的第一行，那就只能手动合并了）。

那么此时你使用 `git status` 来查看状态，

```
$ git status .
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
```

你的工作区很干净，没啥可commit的，但是 注意上面那句话

`Your branch is ahead of 'origin/master' by 2 commits.`

你当前的分支领先 远程分支 两次提交。

为啥是两次，因为本来你就commit了一次，然后 `git pull` 时，又自动合并commit一次，所以就两次了。

不用关心几次，看到重点就对了，你当前的分支领先远程分支。

另外，扯了这么大大一段，别忘记了最初咱们的目的是什么，咱们最初的目标就是 推送代码到远程服务器。

所以接下来直接 `git push`，将 代码直接推送到远程服务器就好了。

这次的冲突因为我制造的比较简单，所以自动合并了，但是有些冲突比较复杂，`git` 无法自动合并，那么此时就需要你手动合并了。



下面的demo直接借用了网上别人的代码冲突内容，我针对输出进行解释。

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

系统提示你，`test.txt` 文件冲突了，自动合并失败了，你需要解决冲突，然后并提交。。

好，我打开 `test.txt` 文件，会看到下面的情况。

```
Git has a mutable index called stage.
Git tracks changes of files.
<<<<<<< HEAD
Creating a new branch is quick & simple.
=====
Creating a new branch is quick AND simple.
>>>>>>> feature1
```

其中冲突的部分都是用 `<<<<<<<`，`=====`，`>>>>>>>` 来进行标记了，他们都代表了不同分支(或者远程/本地)不同的内容，你自己看着代码，把 `<<<<<<<`，`=====`，`>>>>>>>` 进行删除，该删除的代码部分也进行删除。

如果是你自己写的代码，你肯定知道该删除那些内容，如果不是你写的代码，比如李四写的，那么你要叫着李四讨论，把你们两个的代码合并掉。李四写的代码你又不了解业务逻辑，你不和他讨论，就瞎合并，这在多人团队中，是大忌。

因为你可能把李四写的代码给覆盖掉。而程序员基本上写完某个文件就不再关心了，所以李四也不知道你把他的内容覆盖掉了，这肯定会引起问题，如果测试人员能发现业务逻辑不对，那还好，最多被测试人员提个代码臭骂一顿，但如果测试不能发现，那等着线上事故吧。

比如上面的那段冲突，我们合并成如下形式，并进行文件保存。

```
Creating a new branch is quick and simple.
```

合并之后，相当于你又重新 修改了文件。

所以在此重新进行提交步骤。。

```
$ git add readme.txt
$ git commit -m "老子把冲突合并了"
[master 59bc1cb] 老子把冲突合并了
```

最后再push就好了。。

到了这里我们就理解了平时的提交代码，拉取代码的步骤，以及怎么解决冲突。

我们再来学习一个命令。`git log`：

`git log`：查看之前每次提交的说明信息：

```
MINGW32:/d/code/github_blog/ConcurrentDemo
commit fb095209cc6adf53a98035cc7661d109a2024de9
Merge: 5b90fa3 c63c40b
Author: yaowen <yw43194@1y.com>
Date: Sat Sep 9 11:45:40 2017 +0800

    Merge branch 'master' of git.oschina.net:yaowen369/ConcurrentDemo

commit 5b90fa38a6dcc7f79227d0f8b19cd79295d28e12
Author: yaowen <yw43194@1y.com>
Date: Sat Sep 9 11:31:17 2017 +0800

    我在工作电脑上修改了改文件，待会去pull，看看会发生什么

commit c63c40bf3d1a36b80905ab79625a4d33231614d
Author: yaowen <yaowen369@163.com>
Date: Sat Sep 9 11:23:39 2017 +0800

    mac上操作，教学测试，随便修改一个文件，制造冲突

commit 0e522fe146096657489a141a441b68d2d2a8da57
Author: yaowen <yw43194@1y.com>
Date: Fri Sep 8 16:50:18 2017 +0800

    我这只是一次提交测试，进行教学的提交测试

commit 5c53c8fe17bb319ef73978d524750b3f493ff9fb
Author: yaowen <yw43194@1y.com>
Date: Fri Mar 10 17:52:35 2017 +0800

    增加文档的修改

commit fbc9c728eb32a52e4558d052ff32cf83167ecb95
Author: yaowen <yaowen369@163.com>
Date: Thu Mar 9 23:18:05 2017 +0800

    补充文档

commit f02a96237815ea72e19edbd66b09d155e26c4469
Author: yaowen <yaowen369@163.com>
Date: Thu Mar 9 22:19:23 2017 +0800

    多线程添加

commit 9a11fadae9fee1d280fa5431b7e0d2858abfcbb4
Author: yaowen <yw43194@1y.com>
Date: Tue Mar 7 18:20:37 2017 +0800

    java多线程大概完成了

:
```

直接看输出应该一目了然了。每次提交的版本号。作者，时间，提交的信息说明 都直接列出来了。咱们之前在 `git commit -m "说明信息"`，这里就有用了，否则那么多次提交，谁也没本事都记住啊。

尤其是你注意最上面那个 说明信息：

```
commit fb095209cc6adf53a98035cc7661d109a2024de9
Merge: 5b90fa3 c63c40b
Author: yaowen <yw43194@1y.com>
Date: Sat Sep 9 11:45:40 2017 +0800
```

```
Merge branch 'master' of git.oschina.net:yaowen369/ConcurrentDemo
```

最上面的是 版本号(就是那一长串奇怪的字符串),git的版本号是一长串字符串，而svn的版本号就是很简单的1, 2, 3, 4阿拉伯数字,简单来讲，因为svn你每次提交拉取时，都是直接与中央服务器交互，而git则是先与版本库(暂存区)交互，多人都使用git来提交代码，他们本地的电脑时间都不一定准确，不能使用1, 2, 3, 4作为版本号，因为不能简单的依据时间戳来比较。而svn则可以直接使用服务器时间。

这次提交信息，还很贴心的给你提示了，你这次提交其实是一个 合并冲突操作 `merge`，并且提交的说明信息 `Merge branch 'master' of git.oschina.net:yaowen369/ConcurrentDemo`，就是刚才合并时系统帮我们生成的。（因为当时我们在vi界面并没有修改默认的提交说明信息啊）。

注意右下角的冒号，因为我这个项目提交很多多次了，所以log说明比较多，一页显示不完，你还记得怎么上下翻页，怎么退出log提示不？(翻页一般用不到，因为我们一般看提交log也都是看最近的几次说明，不过怎么退出这肯定是要知道额)

那么下面我们要讨论怎么结合github使用。

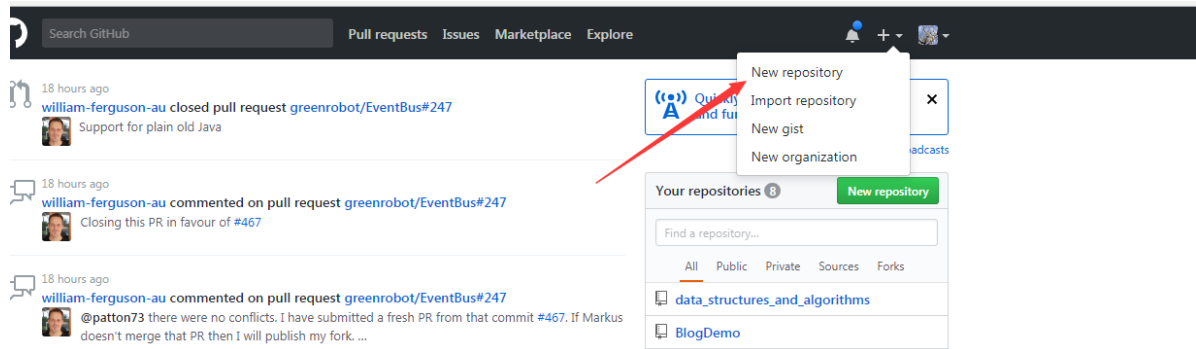
## github

我们为什么要使用github，因为我们需要一个远程服务器啊。

在本文最初的时候，就说了需要一个远程服务器，我们上面那么多的操作，都是客服端的操作。都是假设我们已经搭建好了远程服务器，而在公司里，也已经搭建好了代码服务器，所以我们平时的代码等都是发布到那里的，但是单独的个人的小项目，你代码托管到哪里呢？当然，你可以用自己的电脑搭建个git服务器，但是这是一个非常复杂的过程。所以我们可以托管到github上之类的，这样不就给我们省了很多事情吗？反正又不要钱，对吧。

github账号的创建，ssh key的上传自己去google搜索吧，我们就直接来创建一个项目..

图片名字：github\_new\_repository.png

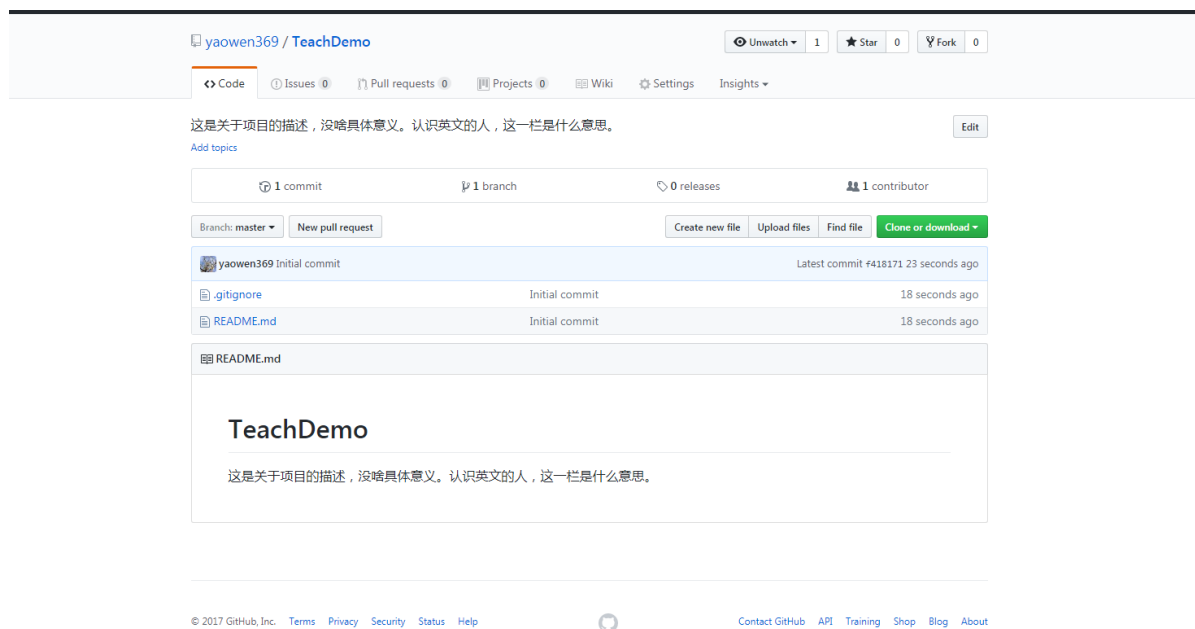


注意那个箭头，点击加号， **New repository**，这就是创建一个新项目的意思。

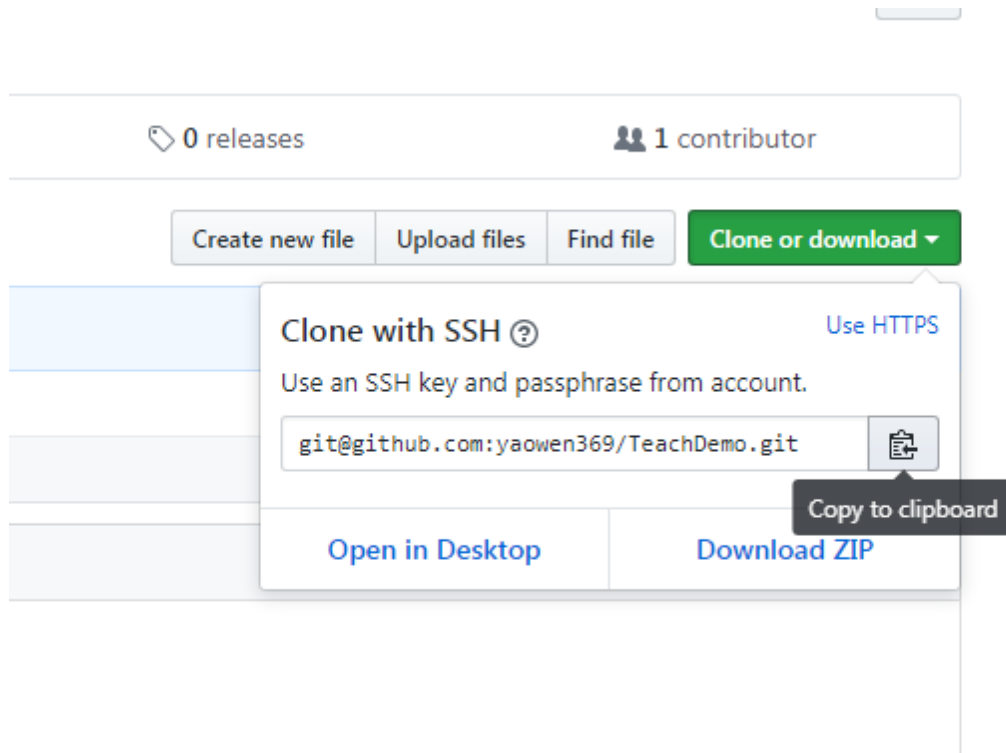
关于这个界面：

- **Repository name**: 项目的名字，我这次的项目就叫做 **TeachDemo** (这个名字后面会说明，和你eclipse的项目名字保持一致最好)。
- **Description (optional)**，描述(可选)，这个不用多介绍了。
- 我们只能创建 **Public**，因为 **private** 是要money的。(不过oschina上你可以免费创建私有项目，你可以创建个项目，放你的一些电子书之类的不那么隐私的资料)。
- **Initialize this repository with a README**，是否初始化一个readme文件，是markdown格式的，建议 勾选吧。
- **Add .gitignore**：这时候再说看不懂这栏啥意思，那就说明之前的文章都白写了。既然你写java代码，那就选择个java的。
- **Add a license**：这个不用管。

此时我们填写和勾选都已经完毕了，点击蓝色确认按钮吧 **Create repository**，此时进入了这个页面。



此时该项目创建ok了。。。你会看到项目当中给你初始化了两个文件，`.gitignore`，`README.md` 文件，这都是刚才我们勾选要求创建的。注意左边的蓝色的 **Clone or download**按钮，点击会出现下面界面，然后鼠标放在右边那个小图标上，会直接给予提示 `Copy to clipboard`。(这几个英文要是还看不懂，那让别学编程了)。

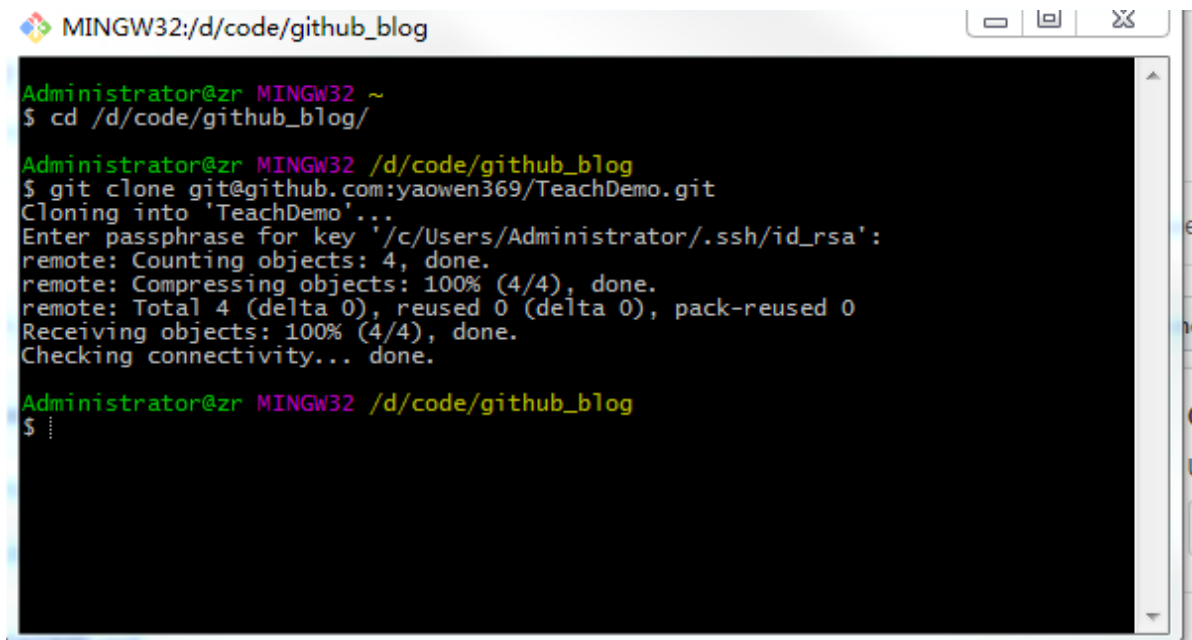


也就是说，我们点击那个小图标，会把前面的那个 ssh地址。

`git@github.com:yaowen369/TeachDemo.git` 复制到剪切板，(这不就是 `ctrl+c` 嘛)。

我们的项目已经创建完毕了，换句话说，我们已经在远程服务器上创建了这个项目，那么下面我们的本地就已经可以用了。

在你的电脑上，看你平时喜欢把代码项目放在哪里，就像我平时代码都是放在D盘的某个文件夹，所以我是这样操作的。



```
Administrator@zr MINGW32 ~
$ cd /d/code/github_blog/

Administrator@zr MINGW32 /d/code/github_blog
$ git clone git@github.com:yaowen369/TeachDemo.git
Cloning into 'TeachDemo'...
Enter passphrase for key '/c/Users/Administrator/.ssh/id_rsa':
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
Checking connectivity... done.

Administrator@zr MINGW32 /d/code/github_blog
$
```

```
cd /d/code/github_blog/

git clone git@github.com:yaowen369/TeachDemo.git
```

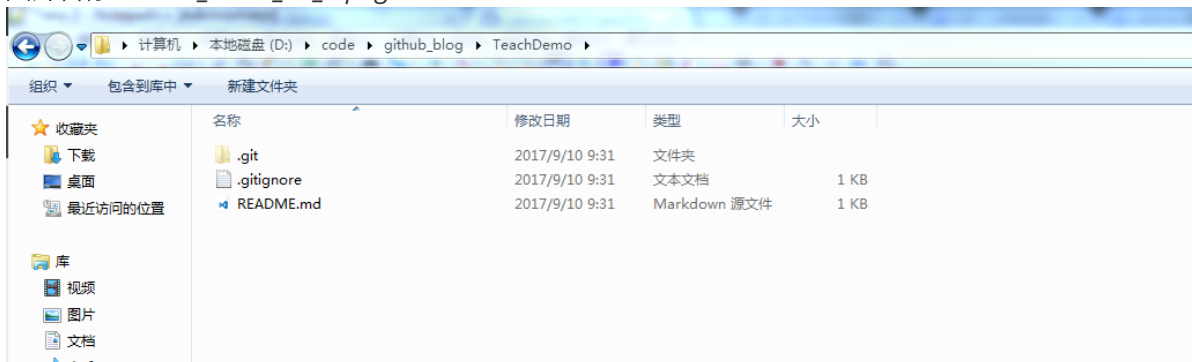
我先通过cd命令进入了 d盘的 code/github\_blog/ 文件夹下，然后我再clone这个代码。

先来解释这两条命令。

- `git clone` : 将远程服务器上的项目克隆到新创建的目录中，解释简单点，将你在远程服务器上的项目，第一次拉到本地,供你在本地使用。其实严格来讲，该过程是将项目代码，从服务器拉到本地版本库，然后再从版本库解析到工作区。不过你关心那么多干嘛？记得这个命令是你在第一次想拉取某个本地没有的项目时使用就行了。(为什么是第一次？因为你第一次使用 `git clone` 在本地弄好之后，今后再更新服务器上的代码就使用 `git pull` 了。 )。
- 那么现在你理解github那个 标签为什么叫做 `clone or download` 了吧。(如果你想查看github项目，直接下载也行。clone也行)

好的。现在我们已经将 服务器上的代码，拉到本地了。但是这个目录不是你eclipse下的项目目录。你又不想重现创建项目，我想将eclipse下的某个项目直接和github的远程服务器发生关联，因为你平时写代码都是用eclipse啊。其实很简单，将我D盘这个文件夹的所有文件，都复制到 你eclipse的某个项目的目录下，就好了。。

图片名称: *teach\_demo\_on\_d.png*



说简单点，将该目录下的 所有文件 都复制到 你eclipse的项目的代码路径下就ok了。。。

注意以下几点：

某些人的电脑上可能没有打开 `显示隐藏文件` 选项，所以要打开这个，因为你复制过程中，真正起作用的就是 `.git` 文件夹，最重要的隐藏文件都漏掉了，那你复制还有啥用？

因为我们在 github 上给项目取的名字就叫做 `TeachDemo`，为了保险起见，我们建议这个名字和 eclipse 工程中你的项目名称保持一致。

另外 关于代码路径，建议不要带中文字符，因为有些时候，带中文路径的代码，编译等可能有很奇怪的问题，所以你看我的电脑中代码的文件路径全都是英文的。

我们已经拷贝完成了，然后你再 利用 terminal 进入到 你的 eclipse 下的项目路径，然后 `git status`，你看看会发生什么。下面的操作就会了吧。

## 结束

到了这里，我们的基础教程都已经讲完了，只要求你记住六个名字，六个英文字符而已。会了以上的内容，对于你一个人开发项目来说，基本上 已经能应付95%的需求了。更进一步的内容，你可以去[廖雪峰的博客](#)上学习。他写的比较通俗易懂，并且涵盖了平时团队多人开发所使用的所有基本操作了。

有以下几点想说：

- 整篇文章，就是在讲述 `status`, `add`, `commit`, `push`, `pull`，外加一个 `cd`，记住这六个命令，六个单词就够了，
- 更重要的是，这几个命令，借助 `git status`，tab 键自动补全/提示功能，下一步该做什么很简单啊，有些东西掌握了方法剩下的学习成本很低的。
- 包括什么学习 java，学习 python，你会了其中任何一门，剩下的掌握其他的，都很容易。你学习 java，那么多的 api 方法你怎么记得住？所以要知道查询 API 文档的重要性，只要有大概印象，然后知道怎么搜索，能上 google，百度这就够了，否则那么多东西谁能记得住呢？
- 当然，不管是 git，还是 java，或者任何一个领域学科。你要想深入的去理解学习，那真的很难，要想精通，那对于我们这种人来说基本是不可能的，git 的命令多如牛毛，你要想成为 git 的专家级人才，那学习过程，难的令人发指，但是问题是，对于我们平时使用来说，就那几个命令就够了啊，你要想成为该领域的专家那就是 另外一回事了。
- 另外，英语水平不要太差，不过我想大学出来的，再加上各种翻译软件，google，百度，这也不是个什么难的问题 *必须记住的六条命令*。
  - `cd`: 用来切换工作目录，最常用的一个命令。简单来讲，`cd A文件夹` 就是进入到 `A文件夹` 里面的意思。
  - `git status .`: 查看当前路径下的状态。git 下 **最最常用** 的一个命令。
  - `git add .`: 把工作区的所有变化，(就是你的所有改动)，都添加到 版本库/暂存区。
  - `git commit -m "提交时说明信息"`: 更进一步提交，并说明提交 log。
  - `git push`: 把版本库的所有更新内容，都推送到远程服务器。(就是推代码/推上去)
  - `git pull`: 把代码从远程服务器拉取到本地。(俗称拉代码)

当我们修改了本地代码，向远程服务器推送时，我们的操作步骤如下：

1. `git add .`
2. `git commit -m "提交时说明信息"`
3. `git push`

当我们想更新本地代码，就是把服务器上最新的代码拉取下来，只需要执行一个命令。

`git pull`

### 这三条命令建议记住。

- `git log`: 查看提交历史，与各次的提交说明。
- `git diff`: 比较工作区与暂存区的差异，就是比较看看你到底都做了什么修改。
- `git clone url地址`: 将远程服务器上项目克隆到新创建的目录中（第一次拉项目时使用，后面的更新都用 `git pull` 了）。

### 其他问题

- 操作时 双击 `tab` 键的自动提示/补全功能。
- `q` 或者 `:q` 等命令代表退出(quit)。
- `ctrl+f`, `ctrl+b` 快捷键在terminal可以翻页，就是 上一页，下一页