# Git

## 一：Git简介

### 步骤一：什么是Git

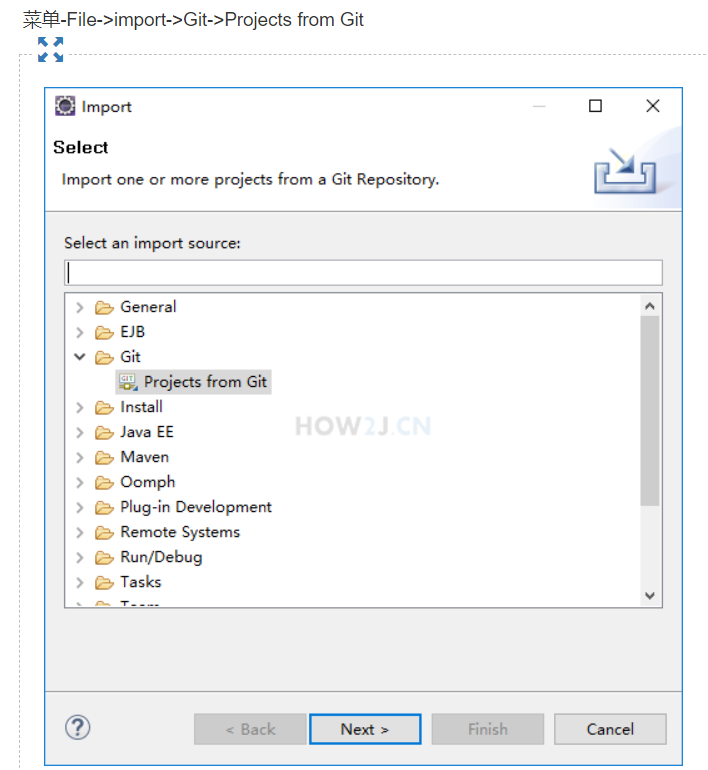
适用于多人开发，把项目放在一个公共空间，需要的时候都可以去获取，有什么改动，都可以进行提交

## 二：Git pull项目

### 步骤一：讲解如何拉一个现成的java项目到eclipse中，并跑起来

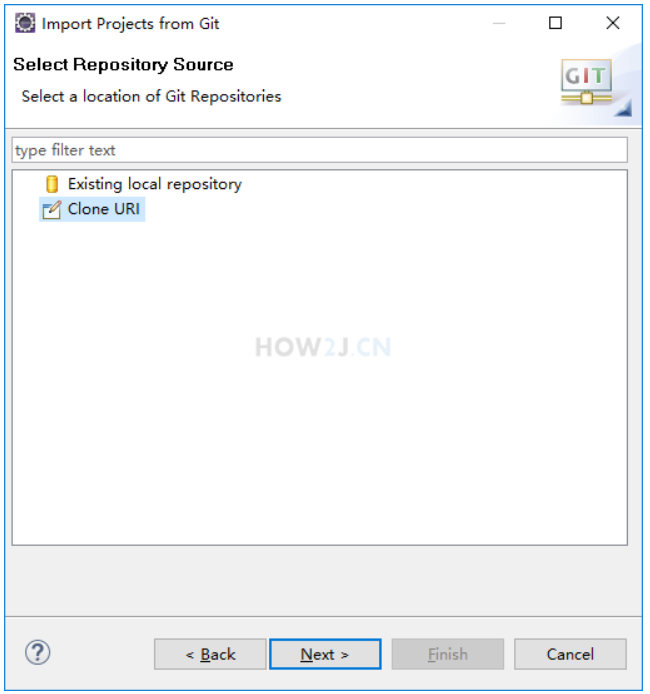
### 步骤二：目前使用的是eclipse ee mar 3.1 版本，内置了对git的支持

### 步骤三：使用git方式导入项目

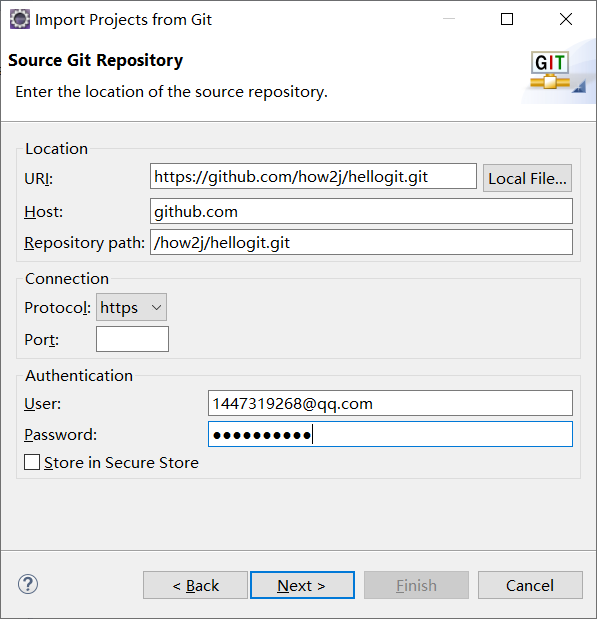


### 步骤四：选择Clone URL

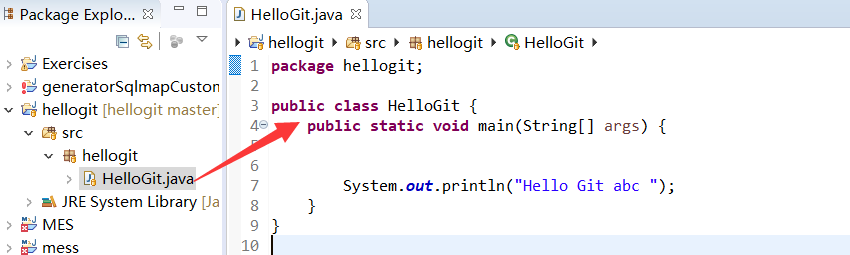
这表示使用某个服务器上的git资源

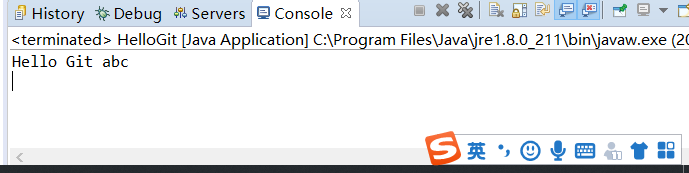


### 步骤五：输入git参数



### 步骤六：运行导入项目





## 三Git创建项目

### 步骤一：把自己的项目共享到Git上

### 步骤二：在Git上新建仓库

首先登陆github.com  
然后点击右上角账号左边的加号，点击New repository创建仓库。  
Git上仓库就相当于项目的意思



### 步骤 三: 输入仓库名称

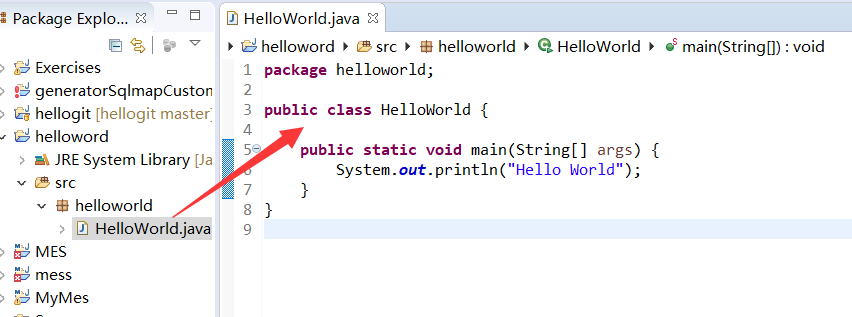
这里输入仓库名称helloworld

### 步骤四：创建成功，得到git地址



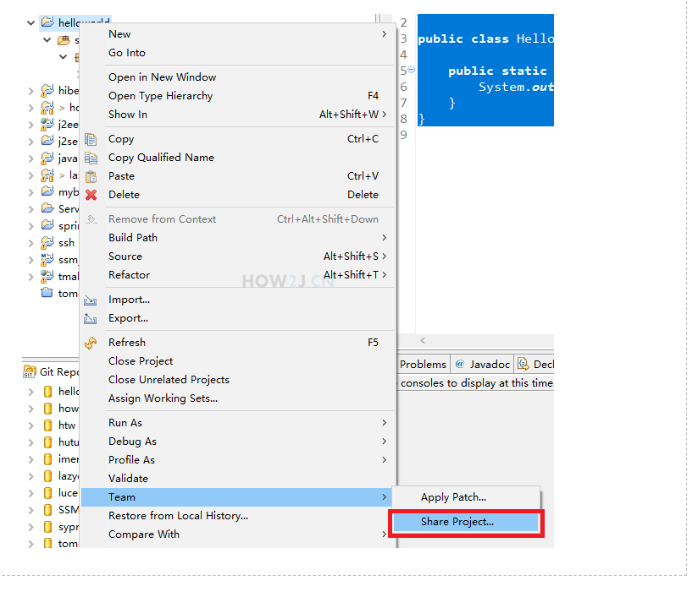
### 步骤五：在Eclipse中创建一个java项目

在Eclipse中创建一个java项目：helloWord，同时建立一个HelloWorld类



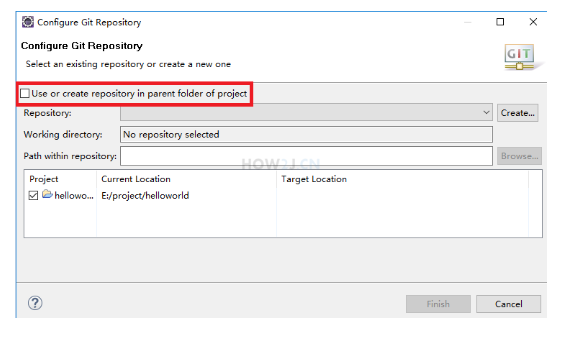
### 步骤六：Share Project

右键项目->Team->Share Project

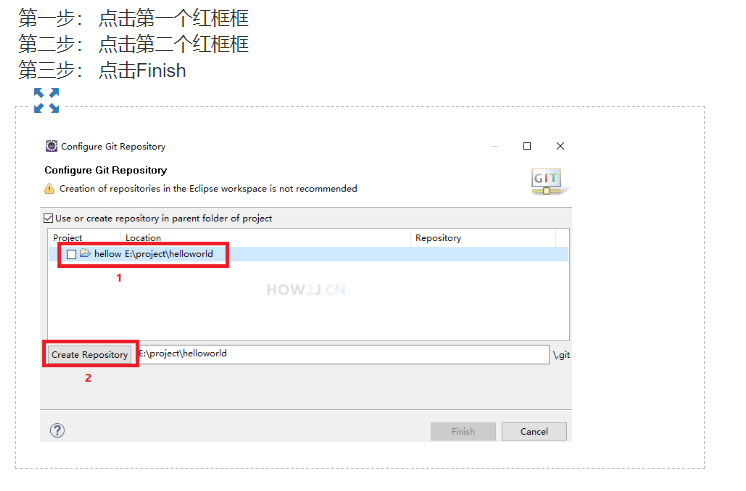


### 步骤七：配置仓库

配置仓库，点击一下Use or create repository in parent folder of project

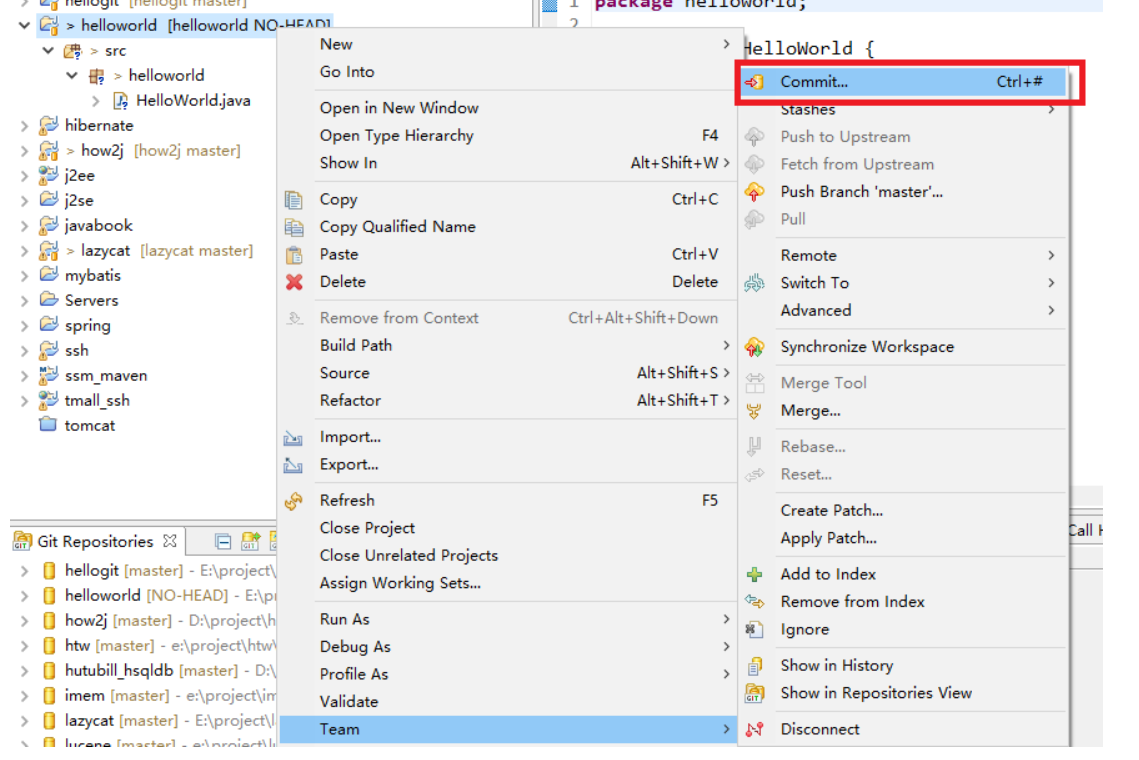


### 步骤八：创建仓库



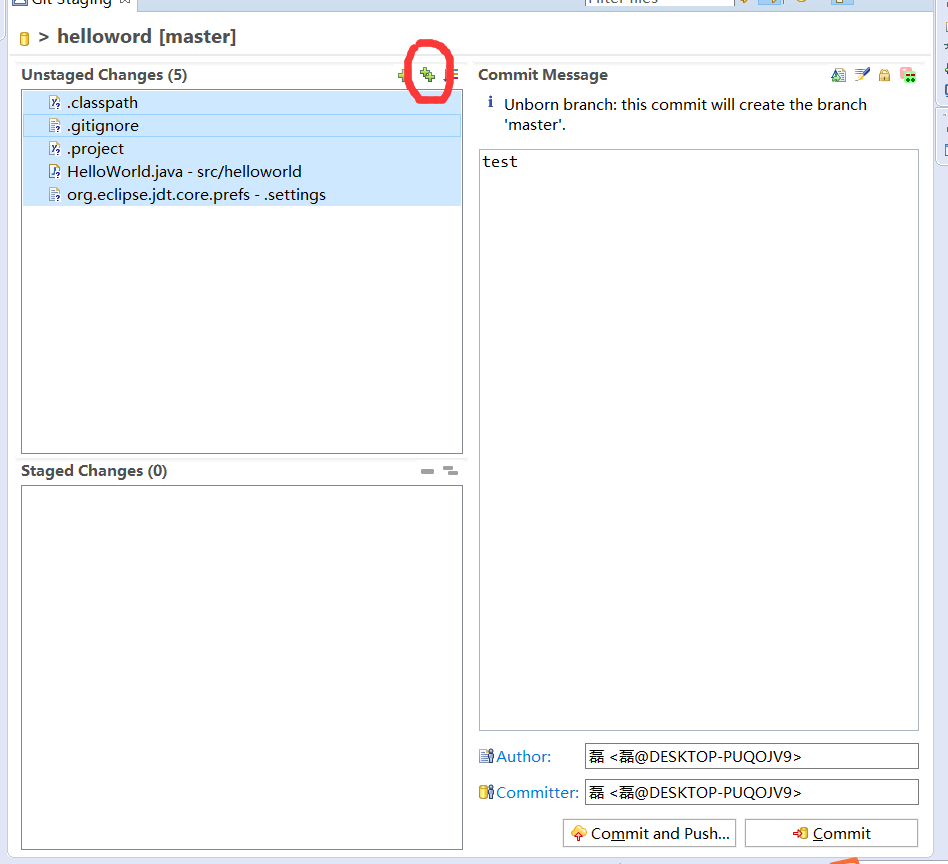
### 步骤九：提交项目

右键项目->Team->commit



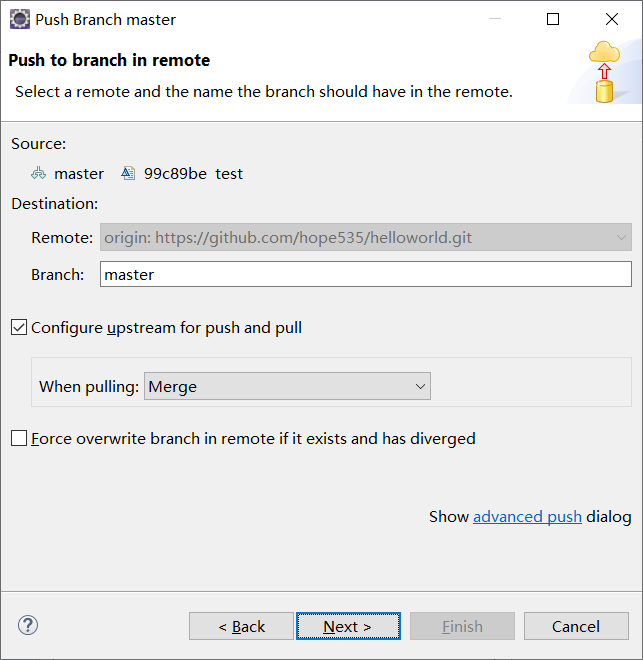
### 步骤十：设置提交信息

设置提交信息：test   
点击红框框，选中所有文件

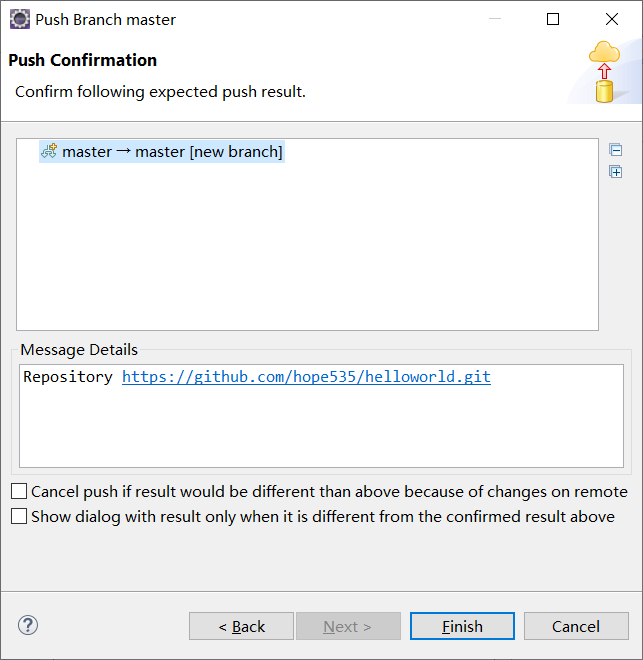


### 步骤十一：设置git参数

### 步骤12：指定分支

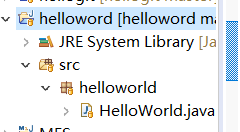


### 步骤13：提交成功



### 步骤14：图标变化

提交成功后，项目，文件夹，包，类下面都会出现小仓库图标，表示同步了，比较[提交项目](https://how2j.cn/k/git/git-create/1343.html" \l "step5649) 步骤里的那些问好小图标，是不一样的。



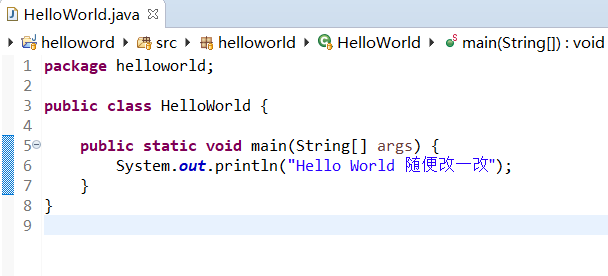
### 步骤15：观察github上的项目



## 四：Git修改自己的代码

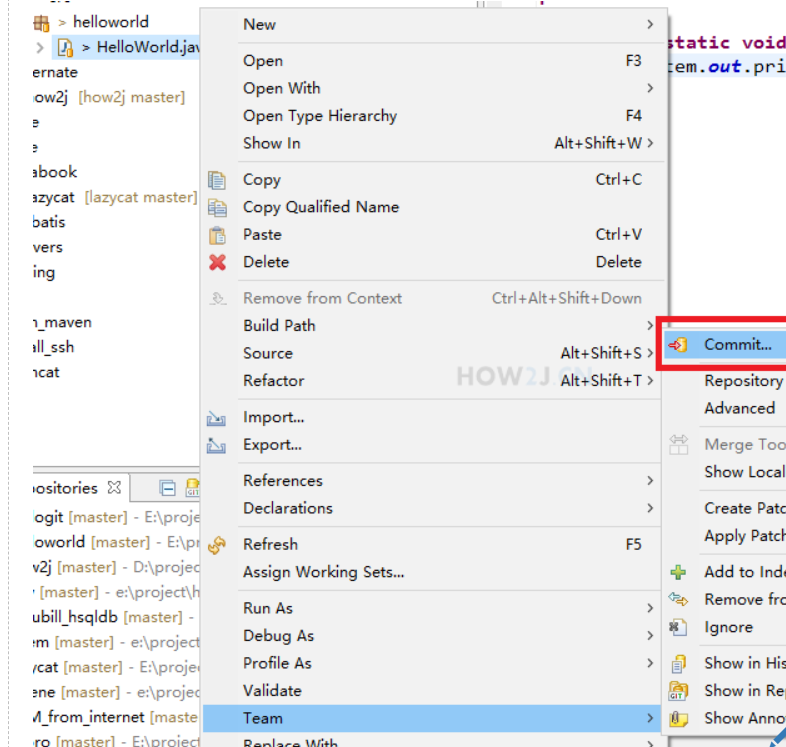
### 步骤1：通过 [创建项目](https://how2j.cn/k/git/git-create/1343.html) 就创建了自己的项目 helloword

### 步骤2：对HelloWorld随便做点修改

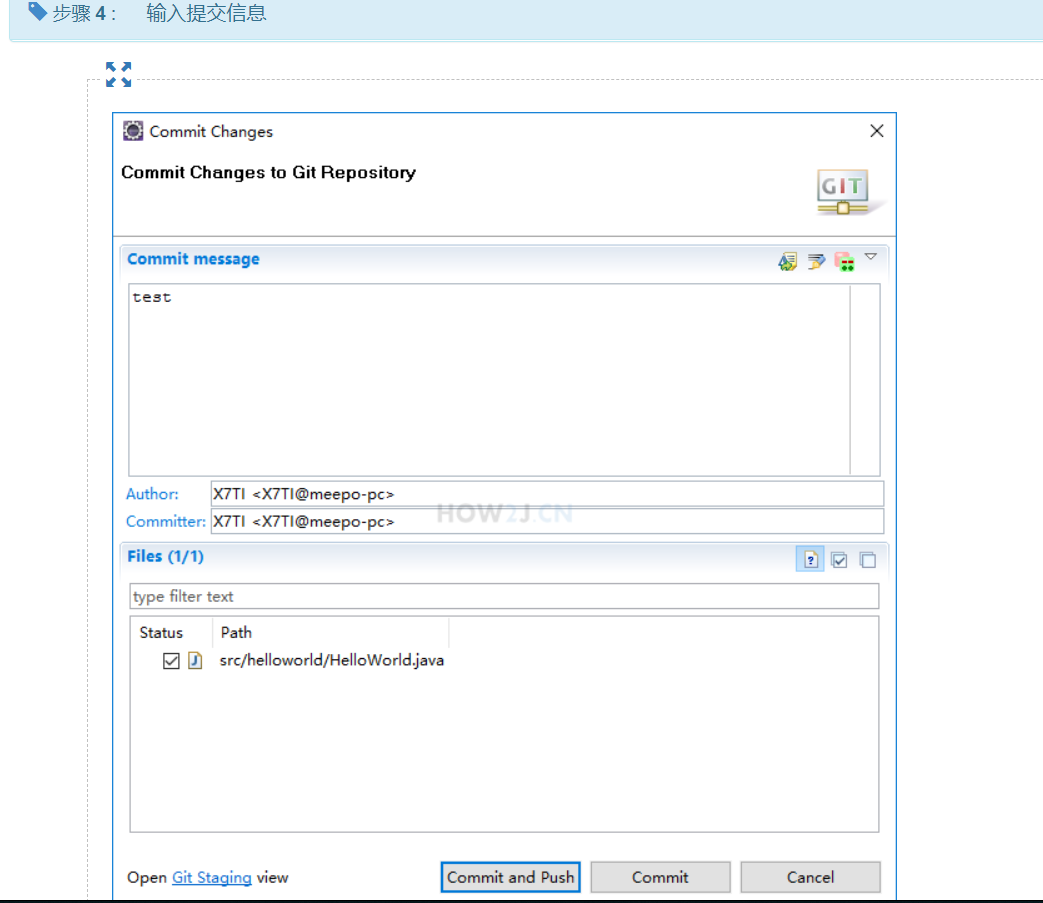


### 步骤3：提交修改

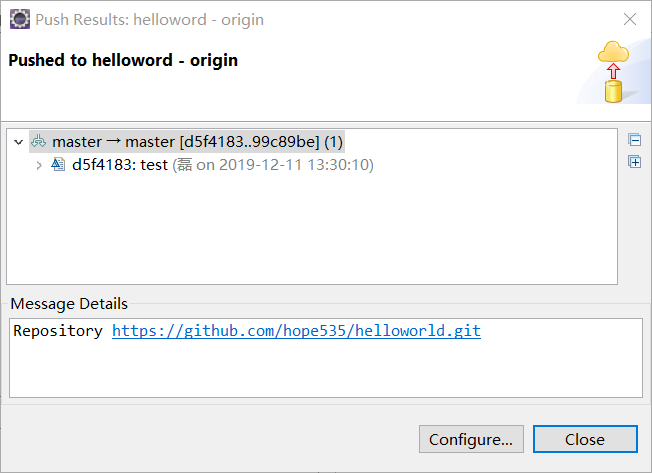
右键HelloWorld.java->Team->Commit



### 步骤4：输入提交信息



### 步骤5：提交成功



## 五：Git修改别人的代码

### 步骤一：修改别人的代码

严格的讲，是不能够直接修改别人的代码的，只能基于别人的项目，克隆一个项目出来到自己的账号上，到了自己的账号上之后，就可以修改了。

修改了之后，再发起一次合并请求，倘若原项目作者介绍你的合并请求，那么你就为原项目做了贡献，也就修改了原项目了。

### 步骤2：Fork

基于别人的项目，做一次克隆，克隆出一个一样的项目到你的账号下，这个行为叫做Fork。

Fork英文是叉子的意思，表示在原项目的基础上，出现了分支。

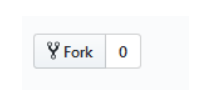


### 步骤3：在github进行Fork操作

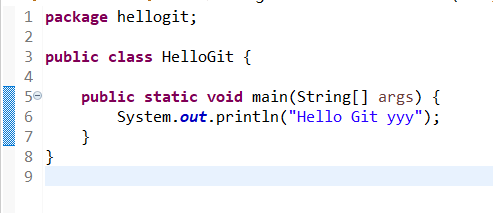
1. 首先用自己的账号密码登陆

2.访问我的hellogit项目：

3. 点击右上角的 Fork 按钮

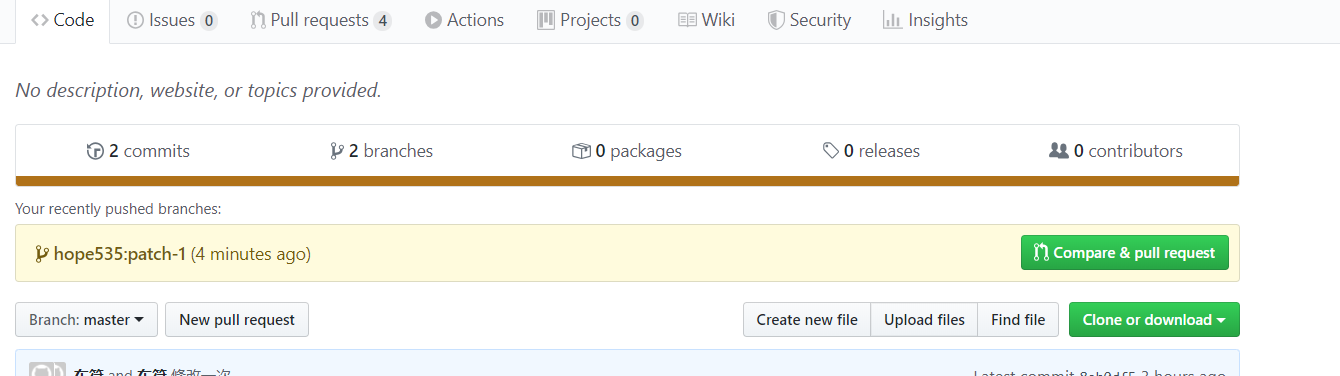


### 步骤4：随便做点修改



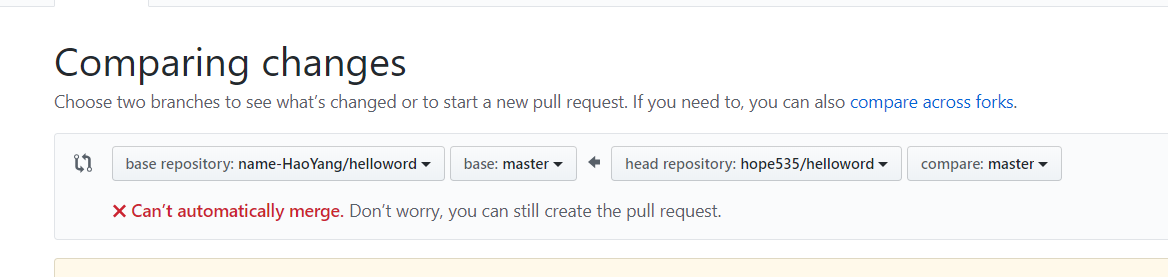
### 步骤5：发起New Pull Request

回到github的hellogit项目下，然后点击New Pull Request，即发起一次合并请求。  
  
既然是合并请求，为什么叫Pull Request呢？ 这是站在原项目主的角度的一个动词，即发起一次请求，希望原项目主，可以把你的改动，**拉进**到他的项目里去。

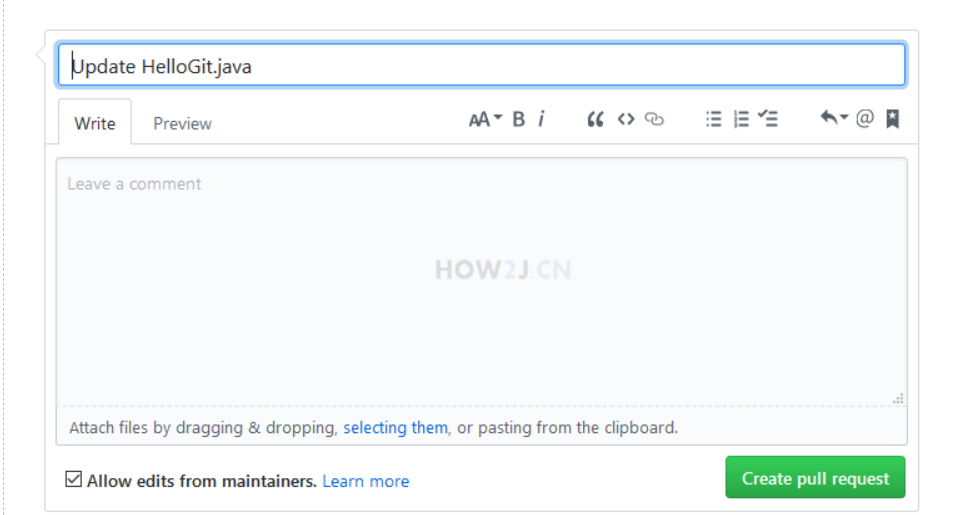


### 步骤6：比较改变的地方

比较文件和原项目中的文件有什么区别，然后点击Create pull request



### 步骤7：填写修改原因



### 步骤8：修改提交完成

这样pull request就发出去了，然后就没你什么事了。

剩下的就是原项目主自己判断，是否要接受你的这次pull request了。

# 一个小时学会Git

## 本控制概要 工作区 暂存区 本地仓库 远程仓库

### 1.1什么是版本控制

版本控制是一种在开发的过程中用于管理我们对文件、目录或工程等内容的修改历史，方便查看更改历史记录，备份以便恢复以前的版本的软件工程技术。

没有进行版本控制或者版本本身缺乏正确的流程管理，在软件开发过程中将会引入很多问题，如软件代码的一致性、软件内容的冗余、软件过程的事物性、软件开发过程中的并发性、软件源代码的安全性、以及软件的整合等问题

### 1.2常用术语

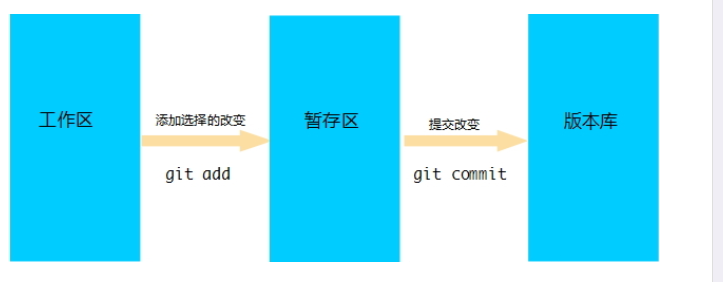
**1)、仓库（Repository）**

受版本控制的所有文件修订历史的共享数据库

**2)、工作空间（Workspace)**   
本地硬盘或Unix 用户帐户上编辑的文件副本

**3)、工作树/区（Working tree）**  
工作区中包含了仓库的工作文件。您可以修改的内容和提交更改作为新的提交到仓库。

**4)、暂存区（Staging area）**  
暂存区是工作区用来提交更改（commit）前可以暂存工作区的变化。



**5)、索引（Index）**  
索引是暂存区的另一种术语。

**6)、签入（Checkin）**  
将新版本复制回仓库

**7)、签出（Checkout）**  
从仓库中将文件的最新修订版本复制到工作空间

**8)、提交（Commit）**  
对各自文件的工作副本做了更改，并将这些更改提交到仓库

**9)、冲突（Conflict）**  
多人对同一文件的工作副本进行更改，并将这些更改提交到仓库

**10)、合并（Merge）**  
将某分支上的更改联接到此主干或同为主干的另一个分支

**11)、分支（Branch）**  
从主线上分离开的副本，默认分支叫master

**12)、锁（Lock）**  
获得修改文件的专有权限。

**13)、头（HEAD）**  
头是一个象征性的参考，最常用以指向当前选择的分支。

**14)、修订（Revision）**  
表示代码的一个版本状态。Git通过用SHA1 hash算法表示的ID来标识不同的版本。

**15)、标记（Tags）**  
标记指的是某个分支某个特定时间点的状态。通过标记，可以很方便的切换到标记时的状态

## 二、Git安装与配置

### 2.1、什么是Git

Git是目前世界上最先进的分布式版本控制系统

优点：

* 适合分布式开发，强调个体。
* 公共服务器压力和数据量都不会太大。
* 速度快、灵活。
* 任意两个开发者之间可以很容易的解决冲突。
* 离线工作。

缺点：

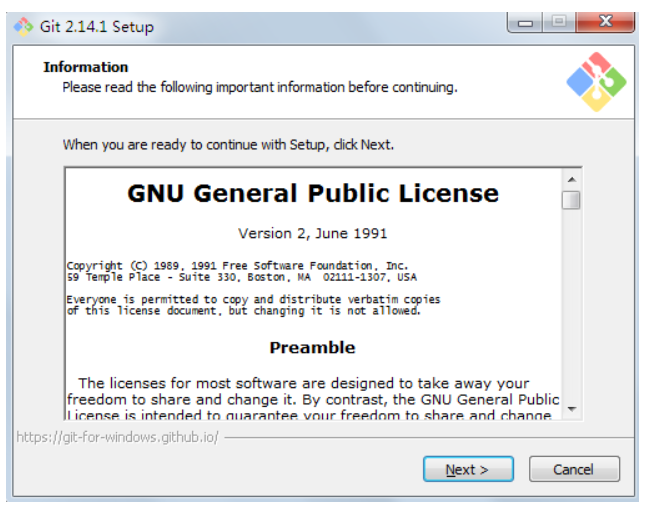
* 模式上比SVN更加复杂。
* 不符合常规思维。
* 代码保密性差，一旦开发者把整个库克隆下来就可以完全公开所有代码和版本信息

### 2.2搭建Git工作环境

#### 2.2.1下载git

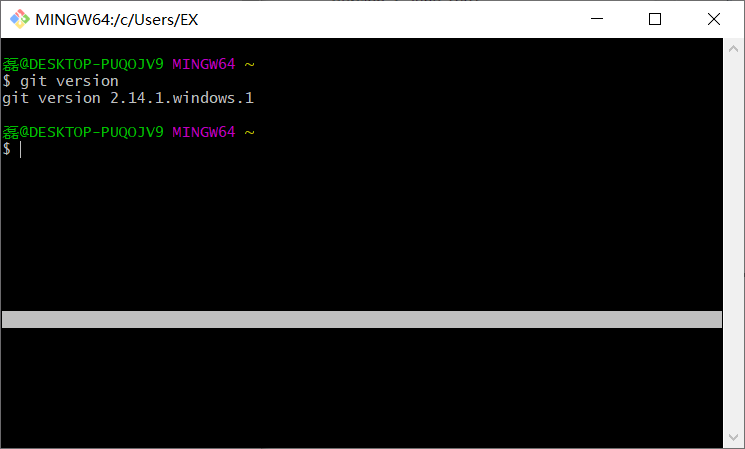
打开 [git官网](https://git-scm.com/" \t "_blank)，下载git对应操作系统的版本。

#### 2.2.2安装Git

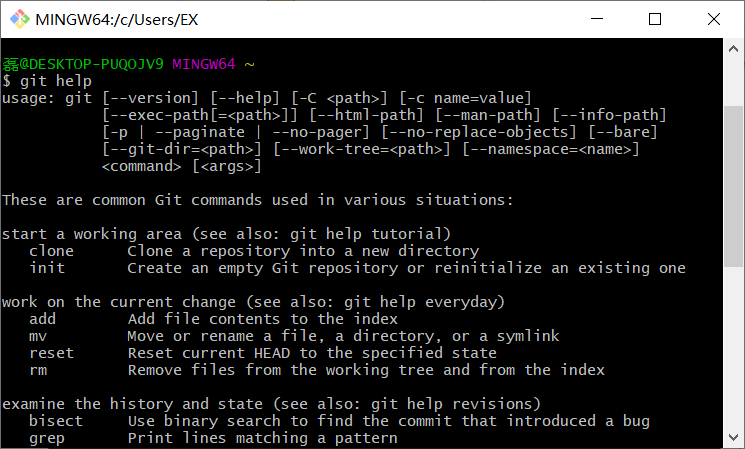


一直Next默认就好了，如果需要设置就要仔细读一下安装界面上的选项。

#### 2.2.3启动Git



Git CMD：Windows风格的命令行

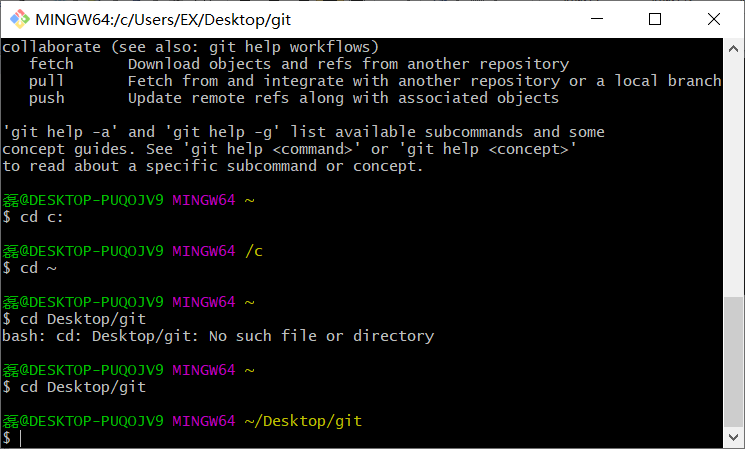


#### 2.2.4、Linux与Mac OS安装Git

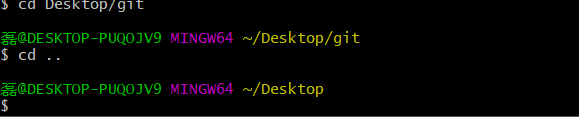
#### 2.2.5Bash基本操作命令

1）、cd : 改变目录。

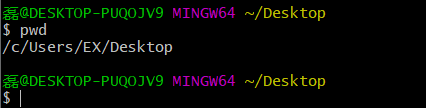
　　cd ~ 回Home（windows是当前用户所在目录）



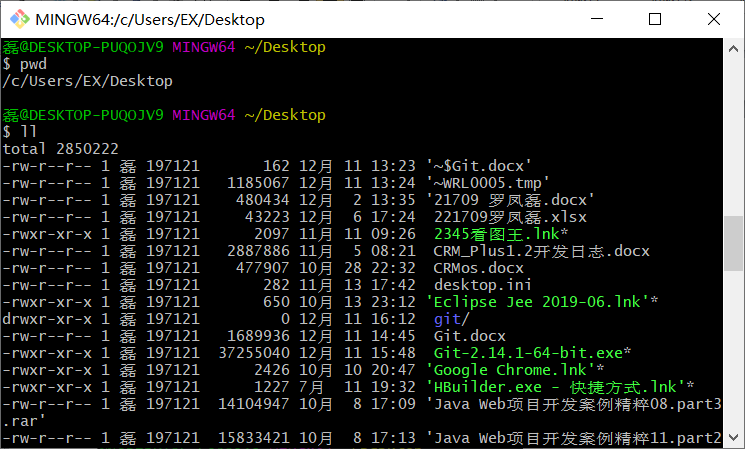
2）、cd . . 回退到上一个目录，直接cd进入默认目录



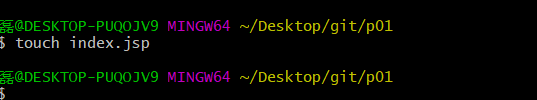
3）、pwd : 显示当前所在的目录路径。

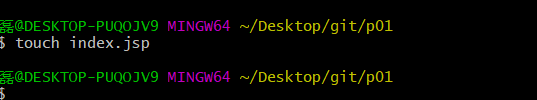


4）、ls(ll): 都是列出当前目录中的所有文件，只不过ll(两个ll)列出的内容更为详细。

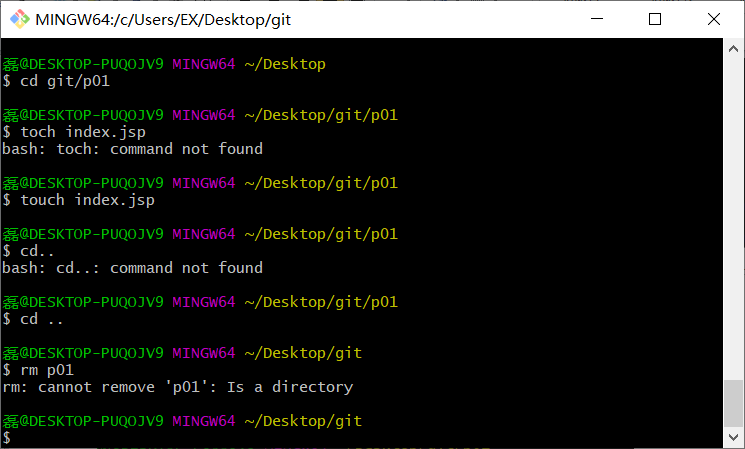


5）、touch : 新建一个文件 如 touch index.js 就会在当前目录下新建一个index.js文件。

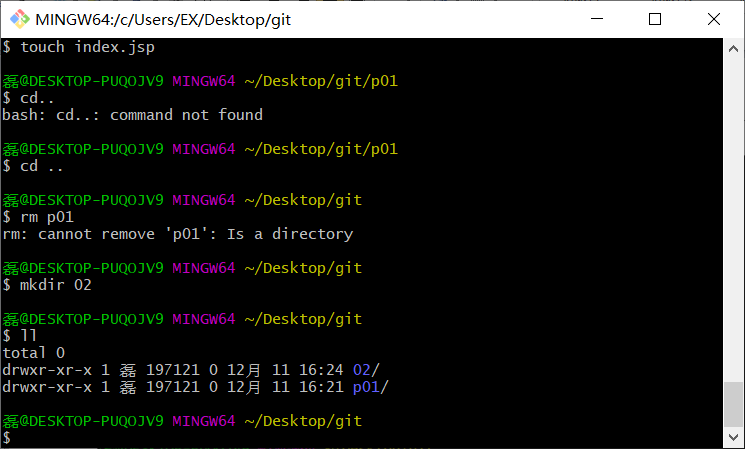




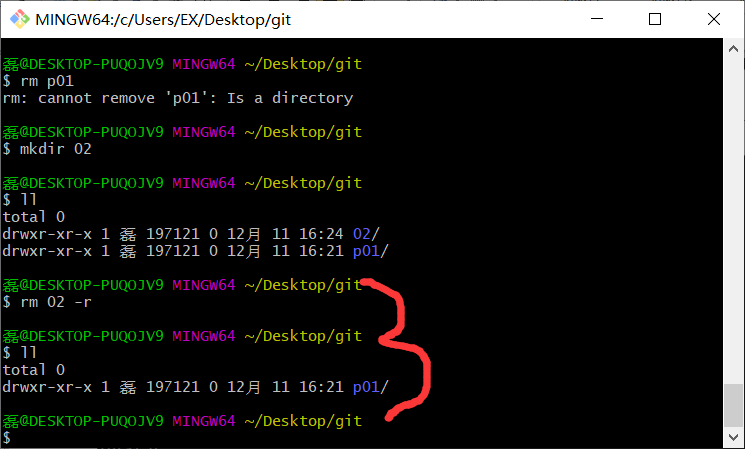
6）、rm: 删除一个文件, rm index.js 就会把index.js文件删除。



7）、mkdir: 新建一个目录,就是新建一个文件夹。



8）、rm -r : 删除一个文件夹, rm -r src 删除src目录， 好像不能用通配符。

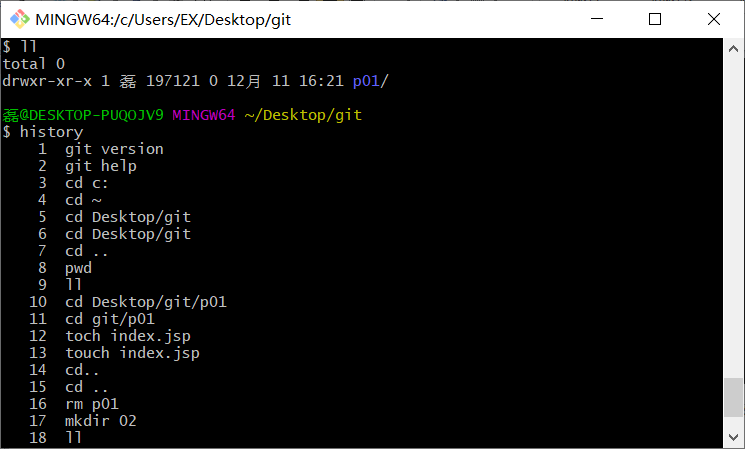


9）、mv 移动文件, mv index.html src index.html 是我们要移动的文件, src 是目标文件夹,当然, 这样写,必须保证文件和目标文件夹在同一目录下。

10）、reset 重新初始化终端/清屏。

11）、clear 清屏。

12）、history 查看命令历史。

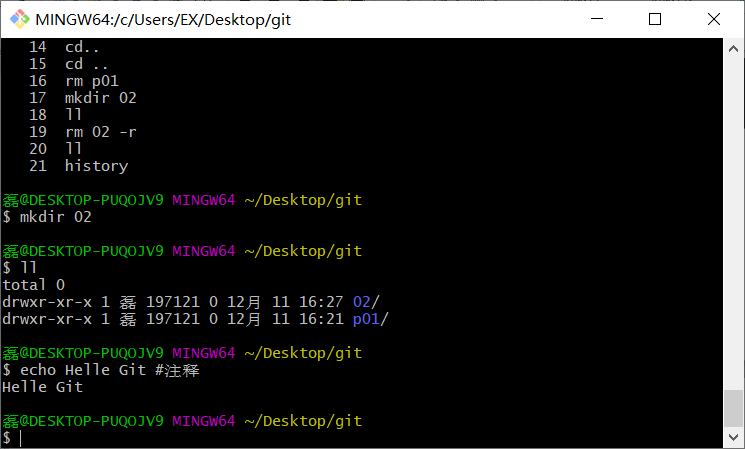


13）、help 帮助。

14）、exit 退出。

15）、#表示注释

16）、输出与注释

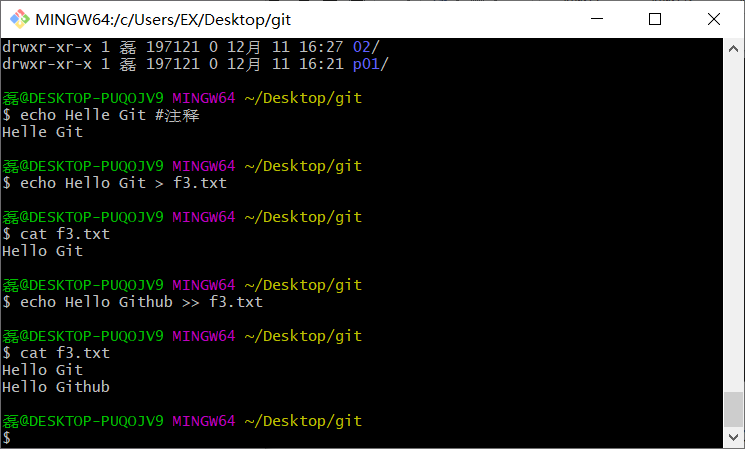


17）、创建文件

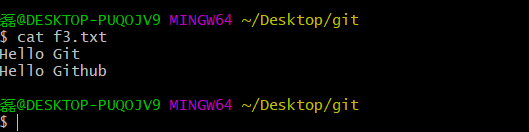
小于号：命令默认从键盘获得的输入，改成从文件，或者其它打开文件以及设备输入

>> 是追加内容

> 是覆盖原有内容



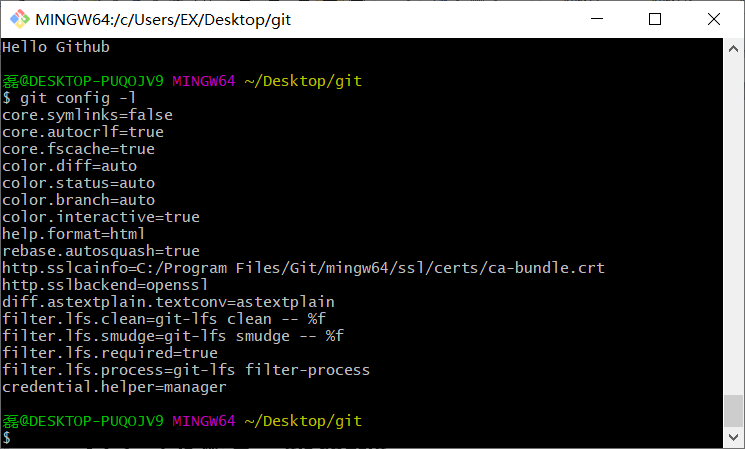
 18、显示文件内容 cat



### 2.3、Git配置 – git config

#### 2.3.1、查看配置 – git config –l

使用git config -l 可以查看现在的git环境详细配置



查看不同级别的配置文件：

#查看系统config

git config --system --list

#查看当前用户（global）配置

git config --global --list

#查看当前仓库配置信息

git config --local --list

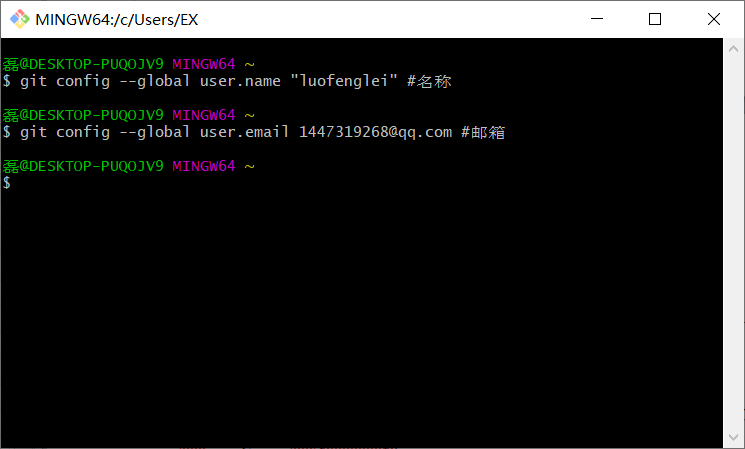
#### 2.3.2、Git配置文件分类

在Windows系统中，Git在$HOME目录中查找.gitconfig文件（一般位于C:\Documents and Settings$USER下）

**Git相关的配置文件有三个：**

1）、 /etc/gitconfig：包含了适用于系统所有用户和所有项目的值。(Win：C:\Program Files\Git\mingw64\etc\gitconfig)

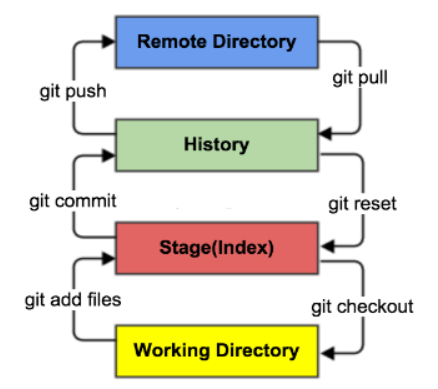
#### 2.3.2、设置用户名与邮箱（用户标识，必要）



## 三、Git理论基础

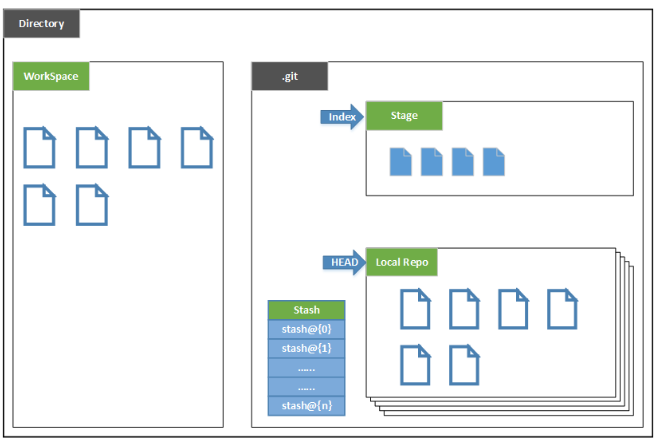
### 3.1、工作区域

Git本地有三个工作区域：工作目录（Working Directory）、暂存区（Stage/Index）、资源库（Repository或Git Directory）。如果在加上远程的Git仓库（Remote Directory）就可以分为四个工作区域。文件在这四个区域之间的转换关系如下：



* Workspace：工作区，就是你平时存放项目代码的地方
* Index/Stage：暂存区。用于临时存放你的改动，事实上他只是一个文件，保存即将提交文件列表信息
* Repository：仓库区（或本地仓库），就是安全存放数据的位置，这里面有你提交到版本的数据。其中HEAD指向最新放入仓库的版本
* Remote：远程仓库，托管代码的服务器，可以简单的认为是你项目组中的一台电脑用于远程数据交换

本地的三个区域确切的说应该是git仓库中HEAD指向的版本



* Directory：使用Git管理的一个目录，也就是一个仓库，包含我们的工作空间和Git的管理空间。
* WorkSpace：需要通过Git进行版本控制的目录和文件，这些目录和文件组成了工作空间。
* .git：存放Git管理信息的目录，初始化仓库的时候自动创建。
* Index/Stage：暂存区，或者叫待提交更新区，在提交进入repo之前，我们可以把所有的更新放在暂存区。
* Local Repo：本地仓库，一个存放在本地的版本库；HEAD会只是当前的开发分支（branch）。
* Stash：隐藏，是一个工作状态保存栈，用于保存/恢复WorkSpace中的临时状态。

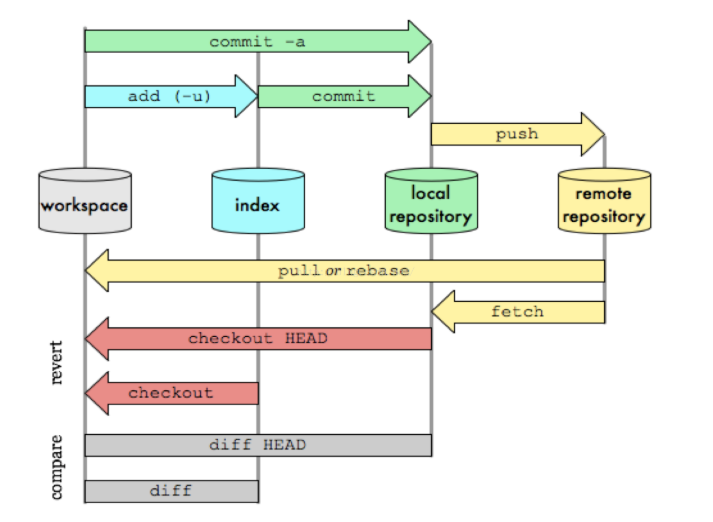
### 3.2工作流程

git的工作流程一般是这样的：

1. 在工作目录中添加、修改文件；
2. 将需要进行版本管理的文件放在暂存区域；
3. 将暂存区域的文件提交到git仓库。

因此，git管理的文件有三种状态：已修改（modified），已暂存（staged），

已提交（committed）

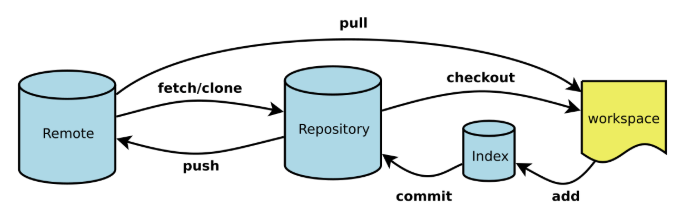


## 四、Git操作

### 4.1、创建工作目录与常用指令

工作目录（WorkSpace)一般就是你希望Git帮助你管理的文件夹，可以是你项目的目录，也可以是一个空目录，建议不要有中文。

日常使用只要记住下图6个命令：



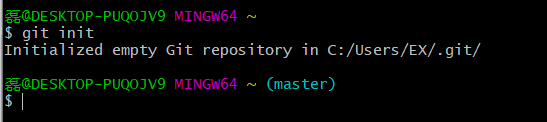
4.2、获取GIT仓库

创建本地仓库的方法有两种：一种是创建全新的仓库，另一种 是克隆远程仓库

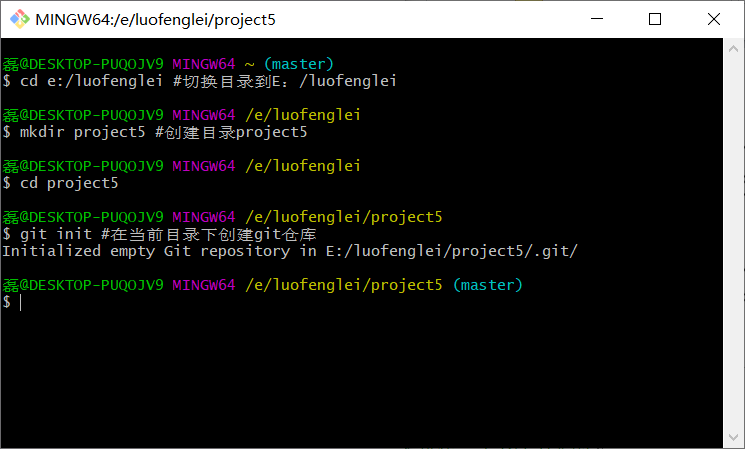
4.2.1

需要用GIT管理的项目的根目录执行

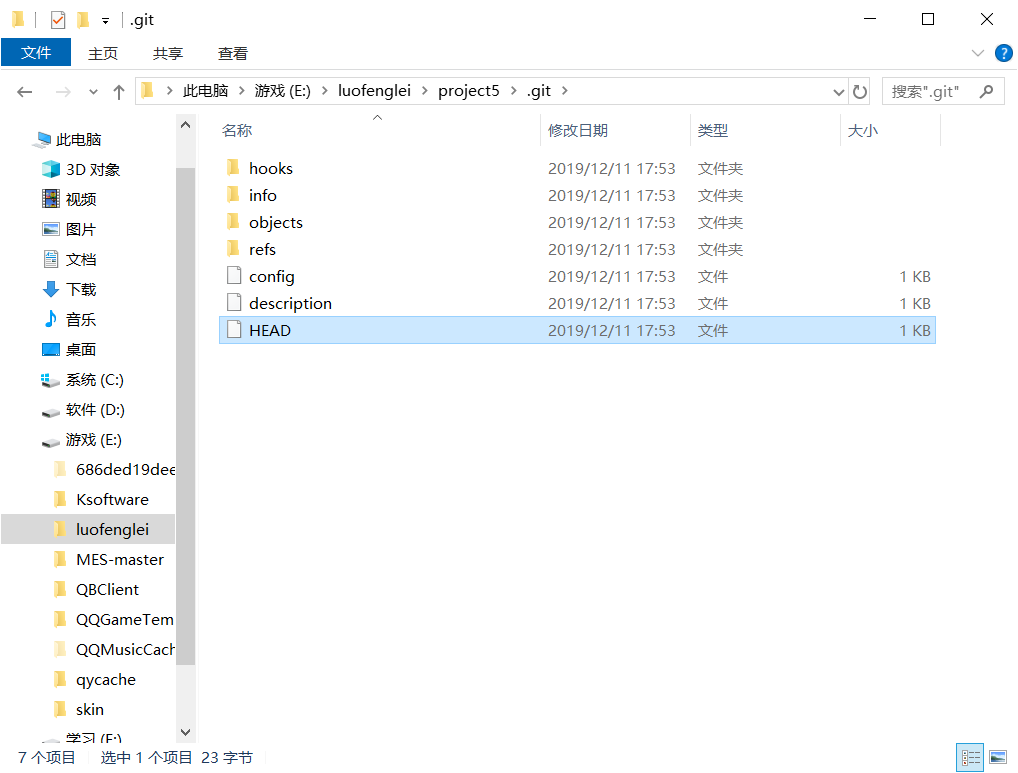
# 在当前目录新建一个Git代码库



执行：



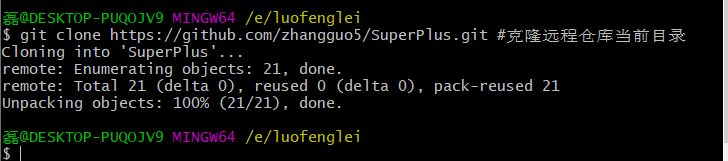
结果：



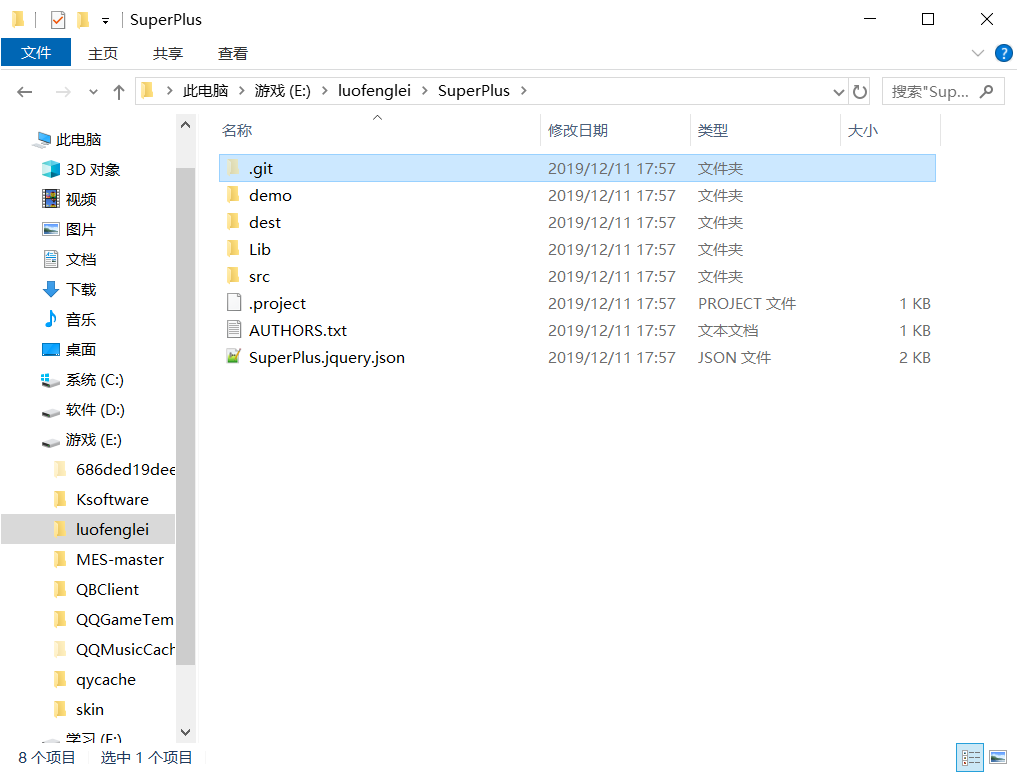
执行后可以看到，仅仅在项目目录多出了一个.git目录，关于版本等的所有信息都在这个目录里面。

### 4.2.2、克隆远程仓库

 另一种方式是克隆远程目录，由于是将远程服务器上的仓库完全镜像一份至本地，而不是取某一个特定版本，所以用clone而不是checkout，语法格式如下：



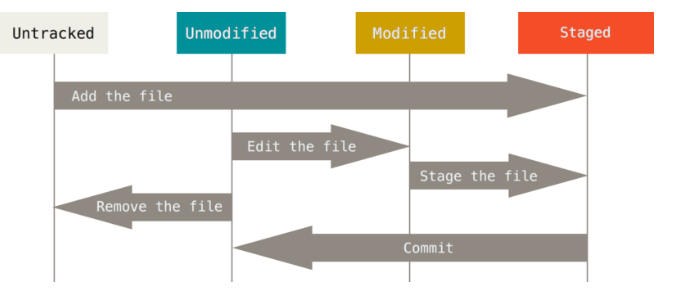
结果：



### 4.3GIT文件操作

版本控制就是对文件的版本控制，要对文件进行修改、提交等操作，首先要知道文件当前在什么状态，不然可能会提交了现在还不想提交的文件，或者要提交的文件没提交上。GIT不关心文件两个版本之间的具体差别，而是关心文件的整体是否有改变，若文件被改变，在添加提交时就生成文件新版本的快照，而判断文件整体是否改变的方法就是用SHA-1算法计算文件的校验和。

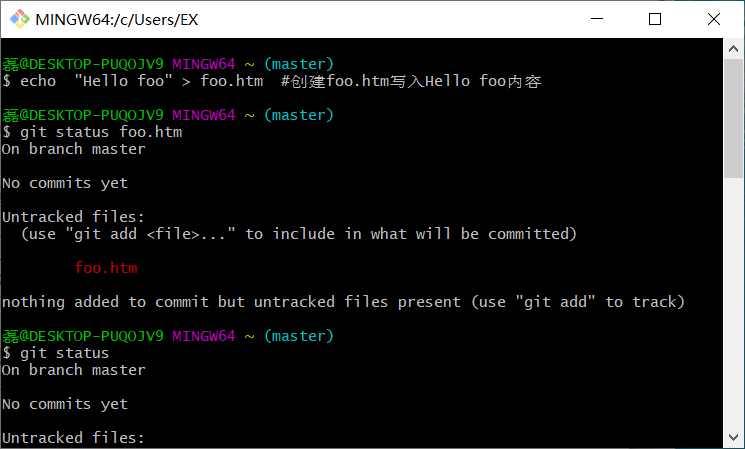
#### 4.3.1、文件4种状态



* **Untracked**: 未跟踪, 此文件在文件夹中, 但并没有加入到git库, 不参与版本控制. 通过git add 状态变为Staged.
* **Unmodify**: 文件已经入库, 未修改, 即版本库中的文件快照内容与文件夹中完全一致. 这种类型的文件有两种去处, 如果它被修改, 而变为Modified. 如果使用git rm移出版本库, 则成为Untracked文件
* **Modified**: 文件已修改, 仅仅是修改, 并没有进行其他的操作. 这个文件也有两个去处, 通过git add可进入暂存staged状态, 使用git checkout则丢弃修改过, 返回到unmodify状态, 这个git checkout即从库中取出文件, 覆盖当前修改
* **Staged**: 暂存状态. 执行git commit则将修改同步到库中, 这时库中的文件和本地文件又变为一致, 文件为Unmodify状态. 执行git reset HEAD filename取消暂存, 文件状态为Modified

#### 4.3.2、查看文件状态

命令：

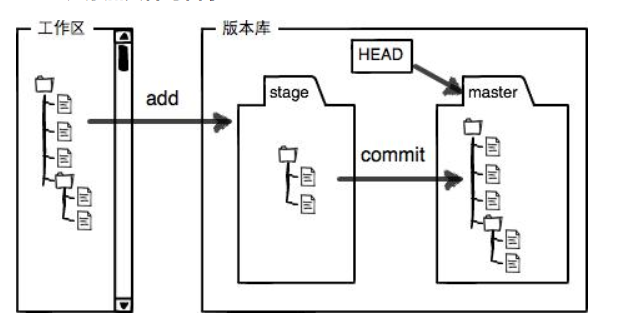


结果：

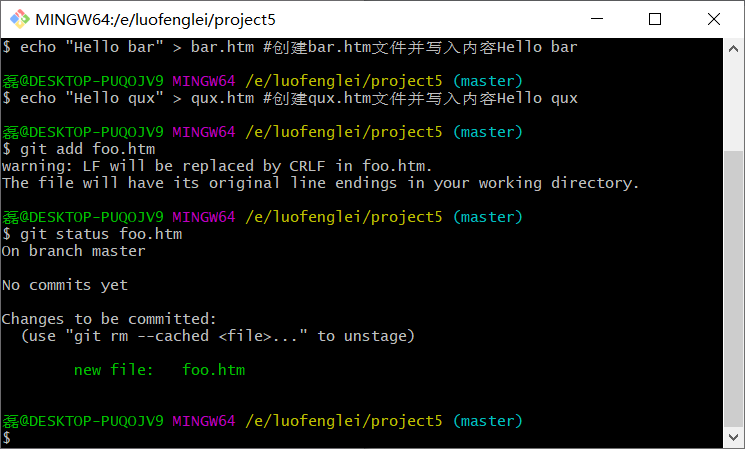
foo.htm文件的状态为untracked（未跟踪），提示通过git add可以暂存

GIT在这一点做得很好，在输出每个文件状态的同时还说明了怎么操作，像上图就有怎么暂存、怎么跟踪文件、怎么取消暂存的说明。

### 4.3.3、添加文件与目录



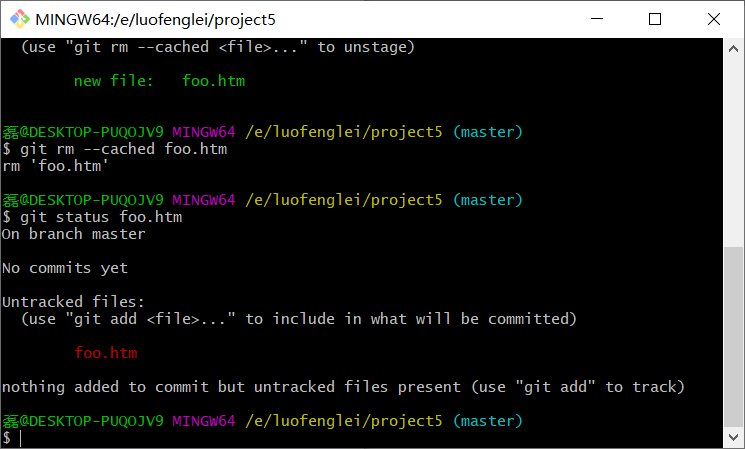
执行：



### 4.3.4、移除文件与目录（撤销add）

当执行如下命令时，会直接从暂存区删除文件，工作区则不做出改变

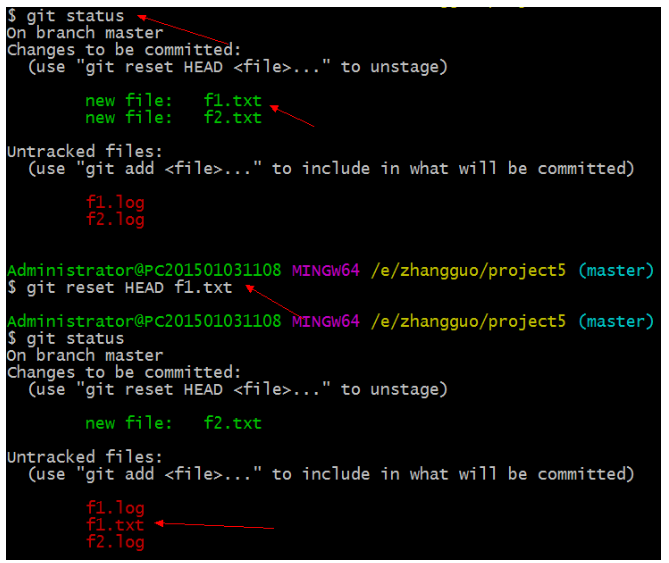
执行命令



通过重写目录树移除add文件：

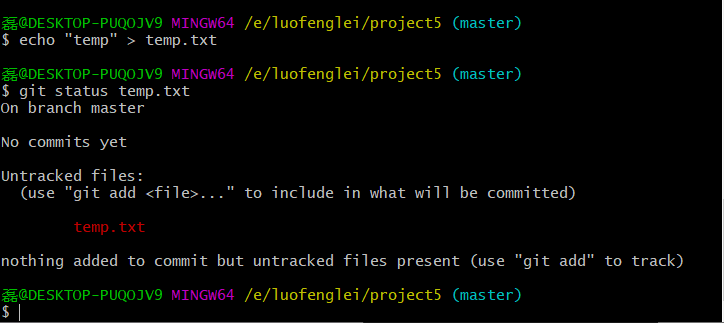
当执行 “git reset HEAD” 命令时，暂存区的目录树会被重写，被 master 分支指向的目录树所替换，但是工作区不受影响。

示例：把f1.txt文件从暂存区撤回工作区

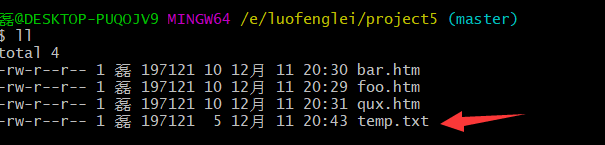


移除所有未跟踪文件

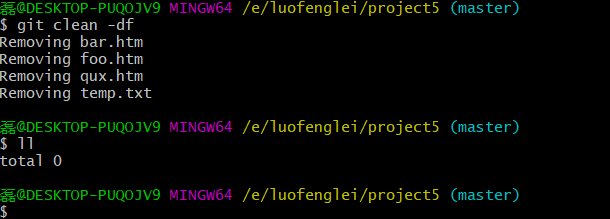
示例：



移除前：



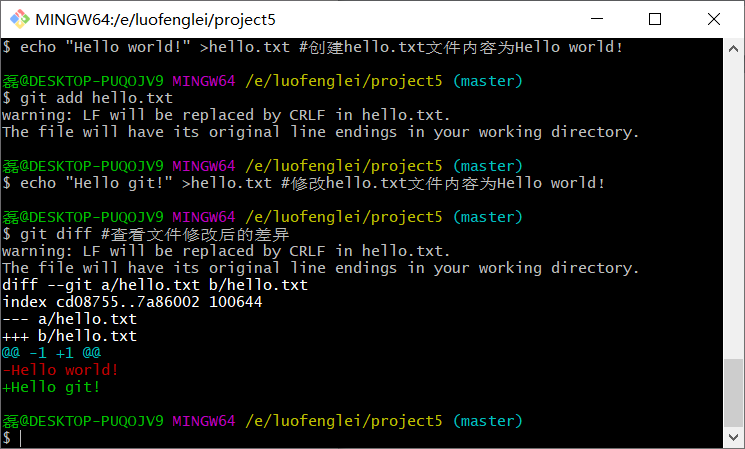
执行移除：



### 4.3.5、查看文件修改后的差异

用"git status"只能查看对哪些文件做了改动，如果要看改动了什么，可以用：

命令：

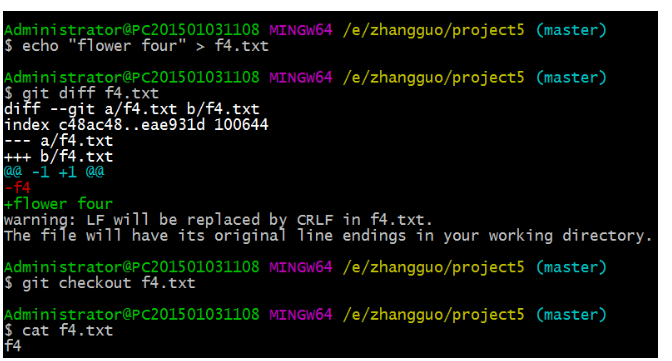


 ---a表示修改之前的文件，+++b表示修改后的文件

### 4.3.6签出

检出命令git checkout是git最常用的命令之一，同时也是一个很危险的命令，因为这条命令会重写工作区

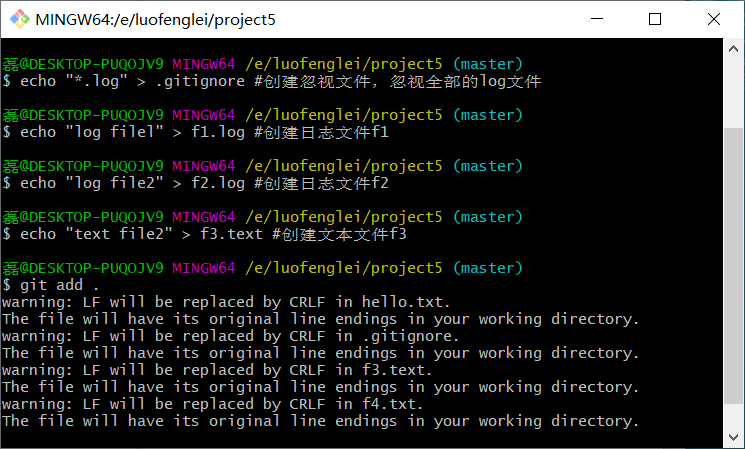
示例：



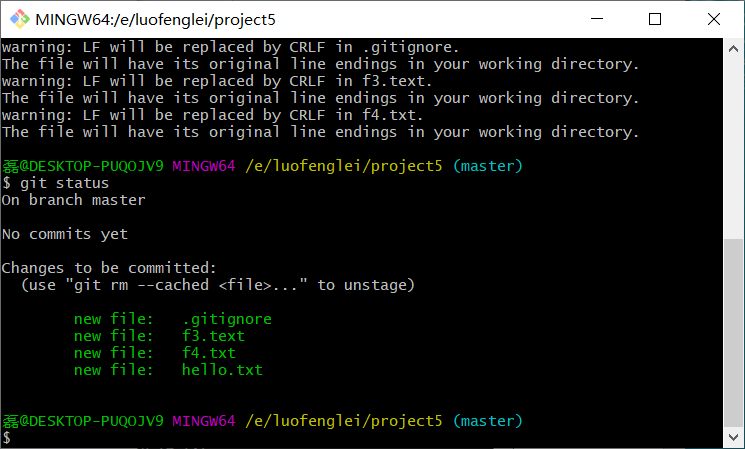
### 4.3.6、忽略文件

示例：

创建一个.gitignore文件忽视所有的日志文件



查看状态：

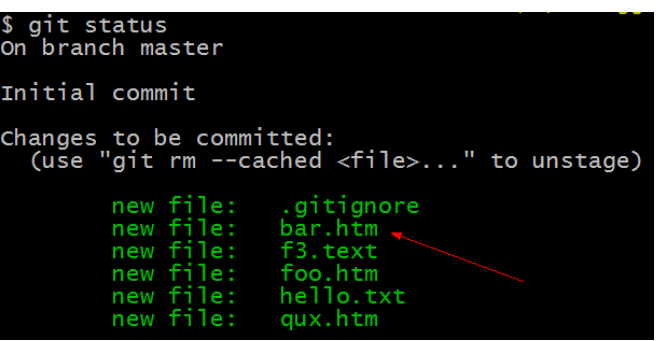


### 4.3.7、提交

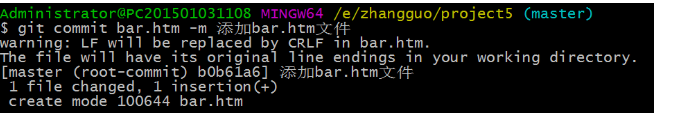
通过add只是将文件或目录添加到了index暂存区，使用commit可以实现将暂存区的文件提交到本地仓库。

示例：

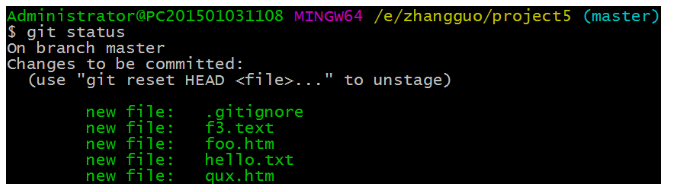
提交前的状态



提交：



提交后的状态：



**修订提交**

如果我们提交过后发现有个文件改错了，或者只是想修改提交说明，这时可以对相应文件做出修改，将修改过的文件通过"git add"添加到暂存区，然后执行以下命令：

#修订提交

git commit --amend

**撤销提交（commit）**

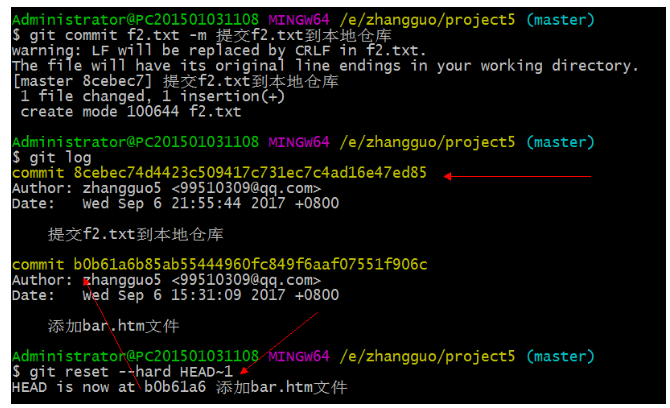
原理就是放弃工作区和index的改动，同时HEAD指针指向前一个commit对象

#撤销上一次的提交

git reset --hard HEAD~1

 要通过git log查看提交日志，也可直接指定提交编号或序号

示例：



### 4.3.8、日志与历史

查看提交日志可以使用git log指令，语法格式如下：

#查看提交日志

git log [<options>] [<revision range>] [[\--] <path>…?]

示例：

# [Git介绍,安装,Git+Git flow使用](https://www.cnblogs.com/wubaiqing/archive/2011/12/18/2271724.html)

## 一：什么是Git

Git是Linux 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， 特点是快速,开源,分布式管理系统.它可以对代码的修改进行回滚,将错误的代码剔除，或者简单地跟踪哪些人修改了代码的哪些行的内容.

## 二：对比：集中化的版本控制

集中化的版本控制系统通常采用两种方式：

有些提供了文件锁来防止多个用户的并行访问,这些系统对文件进行加锁,这样在某个时间只有一个开发人员对中心仓库具有写入权限.  
另外一些工具,例如 SVN,允许多个开发人员同时对相同的文件进行编辑,并提供了一些机制稍后合并这些修改。

CVS,SVN 等工具都是代码部署到服务器上,每次需要远程更新到本地,解决冲突在此提交.  
　　这样弊端很明显,如果这台服务器挂掉几个小时,可能在这几个小时内,你的代码都无法去提交.

## 三：**简单配置Gitosis**