目录

[一、 Git教程 1](#_Toc498778710)

[1. Git简介 1](#_Toc498778711)

[2. 简单介绍 11](#_Toc498778712)

[3. git在windows上安装 13](#_Toc498778713)

[4. 把本地项目提交到Github 18](#_Toc498778714)

[5. 分支和合并 21](#_Toc498778715)

[6. Tags-入门结束篇 28](#_Toc498778716)

[7. 使用国内的Git托管服务 31](#_Toc498778717)

[8. 遇到的问题 38](#_Toc498778718)

1. Git教程

# Git简介

Git是什么？

Git是目前世界上最先进的分布式版本控制系统（没有之一）。

Git有什么特点？简单来说就是：高端大气上档次！

那什么是版本控制系统？

如果你用Microsoft Word写过长篇大论，那你一定有这样的经历：

想删除一个段落，又怕将来想恢复找不回来怎么办？有办法，先把当前文件“另存为……”一个新的Word文件，再接着改，改到一定程度，再“另存为……”一个新文件，这样一直改下去，最后你的Word文档变成了这样：



过了一周，你想找回被删除的文字，但是已经记不清删除前保存在哪个文件里了，只好一个一个文件去找，真麻烦。

看着一堆乱七八糟的文件，想保留最新的一个，然后把其他的删掉，又怕哪天会用上，还不敢删，真郁闷。

更要命的是，有些部分需要你的财务同事帮助填写，于是你把文件Copy到U盘里给她（也可能通过Email发送一份给她），然后，你继续修改Word文件。一天后，同事再把Word文件传给你，此时，你必须想想，发给她之后到你收到她的文件期间，你作了哪些改动，得把你的改动和她的部分合并，真困难。

于是你想，如果有一个软件，不但能自动帮我记录每次文件的改动，还可以让同事协作编辑，这样就不用自己管理一堆类似的文件了，也不需要把文件传来传去。如果想查看某次改动，只需要在软件里瞄一眼就可以，岂不是很方便？

这个软件用起来就应该像这个样子，能记录每次文件的改动：

| **版本** | **文件名** | **用户** | **说明** | **日期** |
| --- | --- | --- | --- | --- |
| 1 | service.doc | 张三 | 删除了软件服务条款5 | 7/12 10:38 |
| 2 | service.doc | 张三 | 增加了License人数限制 | 7/12 18:09 |
| 3 | service.doc | 李四 | 财务部门调整了合同金额 | 7/13 9:51 |
| 4 | service.doc | 张三 | 延长了免费升级周期 | 7/14 15:17 |

这样，你就结束了手动管理多个“版本”的史前时代，进入到版本控制的20世纪。

## Git的诞生

很多人都知道，Linus在1991年创建了开源的Linux，从此，Linux系统不断发展，已经成为最大的服务器系统软件了。

Linus虽然创建了Linux，但Linux的壮大是靠全世界热心的志愿者参与的，这么多人在世界各地为Linux编写代码，那Linux的代码是如何管理的呢？

事实是，在2002年以前，世界各地的志愿者把源代码文件通过diff的方式发给Linus，然后由Linus本人通过手工方式合并代码！

你也许会想，为什么Linus不把Linux代码放到版本控制系统里呢？不是有CVS、SVN这些免费的版本控制系统吗？因为Linus坚定地反对CVS和SVN，这些集中式的版本控制系统不但速度慢，而且必须联网才能使用。有一些商用的版本控制系统，虽然比CVS、SVN好用，但那是付费的，和Linux的开源精神不符。

不过，到了2002年，Linux系统已经发展了十年了，代码库之大让Linus很难继续通过手工方式管理了，社区的弟兄们也对这种方式表达了强烈不满，于是Linus选择了一个商业的版本控制系统BitKeeper，BitKeeper的东家BitMover公司出于人道主义精神，授权Linux社区免费使用这个版本控制系统。

安定团结的大好局面在2005年就被打破了，原因是Linux社区牛人聚集，不免沾染了一些梁山好汉的江湖习气。开发Samba的Andrew试图破解BitKeeper的协议（这么干的其实也不只他一个），被BitMover公司发现了（监控工作做得不错！），于是BitMover公司怒了，要收回Linux社区的免费使用权。

Linus可以向BitMover公司道个歉，保证以后严格管教弟兄们，嗯，这是不可能的。实际情况是这样的：

Linus花了两周时间自己用C写了一个分布式版本控制系统，这就是Git！一个月之内，Linux系统的源码已经由Git管理了！牛是怎么定义的呢？大家可以体会一下。

Git迅速成为最流行的分布式版本控制系统，尤其是2008年，GitHub网站上线了，它为开源项目免费提供Git存储，无数开源项目开始迁移至GitHub，包括jQuery，PHP，Ruby等等。

历史就是这么偶然，如果不是当年BitMover公司威胁Linux社区，可能现在我们就没有免费而超级好用的Git了。

## 1.2集中式vs分布式

Linus一直痛恨的CVS及SVN都是集中式的版本控制系统，而Git是分布式版本控制系统，集中式和分布式版本控制系统有什么区别呢？

先说集中式版本控制系统，版本库是集中存放在中央服务器的，而干活的时候，用的都是自己的电脑，所以要先从中央服务器取得最新的版本，然后开始干活，干完活了，再把自己的活推送给中央服务器。中央服务器就好比是一个图书馆，你要改一本书，必须先从图书馆借出来，然后回到家自己改，改完了，再放回图书馆。



集中式版本控制系统最大的毛病就是必须联网才能工作，如果在局域网内还好，带宽够大，速度够快，可如果在互联网上，遇到网速慢的话，可能提交一个10M的文件就需要5分钟，这还不得把人给憋死啊。

那分布式版本控制系统与集中式版本控制系统有何不同呢？首先，分布式版本控制系统根本没有“中央服务器”，每个人的电脑上都是一个完整的版本库，这样，你工作的时候，就不需要联网了，因为版本库就在你自己的电脑上。既然每个人电脑上都有一个完整的版本库，那多个人如何协作呢？比方说你在自己电脑上改了文件A，你的同事也在他的电脑上改了文件A，这时，你们俩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。

和集中式版本控制系统相比，分布式版本控制系统的安全性要高很多，因为每个人电脑里都有完整的版本库，某一个人的电脑坏掉了不要紧，随便从其他人那里复制一个就可以了。而集中式版本控制系统的中央服务器要是出了问题，所有人都没法干活了。

在实际使用分布式版本控制系统的时候，其实很少在两人之间的电脑上推送版本库的修改，因为可能你们俩不在一个局域网内，两台电脑互相访问不了，也可能今天你的同事病了，他的电脑压根没有开机。因此，分布式版本控制系统通常也有一台充当“中央服务器”的电脑，但这个服务器的作用仅仅是用来方便“交换”大家的修改，没有它大家也一样干活，只是交换修改不方便而已。



当然，Git的优势不单是不必联网这么简单，后面我们还会看到Git极其强大的分支管理，把SVN等远远抛在了后面。

CVS作为最早的开源而且免费的集中式版本控制系统，直到现在还有不少人在用。由于CVS自身设计的问题，会造成提交文件不完整，版本库莫名其妙损坏的情况。同样是开源而且免费的SVN修正了CVS的一些稳定性问题，是目前用得最多的集中式版本库控制系统。

除了免费的外，还有收费的集中式版本控制系统，比如IBM的ClearCase（以前是Rational公司的，被IBM收购了），特点是安装比Windows还大，运行比蜗牛还慢，能用ClearCase的一般是世界500强，他们有个共同的特点是财大气粗，或者人傻钱多。

微软自己也有一个集中式版本控制系统叫VSS，集成在Visual Studio中。由于其反人类的设计，连微软自己都不好意思用了。

分布式版本控制系统除了Git以及促使Git诞生的BitKeeper外，还有类似Git的Mercurial和Bazaar等。这些分布式版本控制系统各有特点，但最快、最简单也最流行的依然是Git！

## 1.3安装Git

最早Git是在Linux上开发的，很长一段时间内，Git也只能在Linux和Unix系统上跑。不过，慢慢地有人把它移植到了Windows上。现在，Git可以在Linux、Unix、Mac和Windows这几大平台上正常运行了。

要使用Git，第一步当然是安装Git了。根据你当前使用的平台来阅读下面的文字：

**1.3.1.在Linux上安装Git**

首先，你可以试着输入git，看看系统有没有安装Git：

$ git

The program 'git' is currently not installed. You can install it by typing:

sudo apt-get install git

像上面的命令，有很多Linux会友好地告诉你Git没有安装，还会告诉你如何安装Git。

如果你碰巧用Debian或Ubuntu Linux，通过一条sudo apt-get install git就可以直接完成Git的安装，非常简单。

老一点的Debian或Ubuntu Linux，要把命令改为sudo apt-get install git-core，因为以前有个软件也叫GIT（GNU Interactive Tools），结果Git就只能叫git-core了。由于Git名气实在太大，后来就把GNU Interactive Tools改成gnuit，git-core正式改为git。

如果是其他Linux版本，可以直接通过源码安装。先从Git官网下载源码，然后解压，依次输入：./config，make，sudo make install这几个命令安装就好了。

**1.3.2.在Mac OS X上安装Git**

如果你正在使用Mac做开发，有两种安装Git的方法。

一是安装homebrew，然后通过homebrew安装Git，具体方法请参考homebrew的文档：<http://brew.sh/>。

第二种方法更简单，也是推荐的方法，就是直接从AppStore安装Xcode，Xcode集成了Git，不过默认没有安装，你需要运行Xcode，选择菜单“Xcode”->“Preferences”，在弹出窗口中找到“Downloads”，选择“Command Line Tools”，点“Install”就可以完成安装了。



Xcode是Apple官方IDE，功能非常强大，是开发Mac和iOS App的必选装备，而且是免费的！

**1.3.3.在Windows上安装Git**

实话实说，Windows是最烂的开发平台，如果不是开发Windows游戏或者在IE里调试页面，一般不推荐用Windows。不过，既然已经上了微软的贼船，也是有办法安装Git的。

Windows下要使用很多Linux/Unix的工具时，需要Cygwin这样的模拟环境，Git也一样。Cygwin的安装和配置都比较复杂，就不建议你折腾了。不过，有高人已经把模拟环境和Git都打包好了，名叫msysgit，只需要下载一个单独的exe安装程序，其他什么也不用装，绝对好用。

msysgit是Windows版的Git，从[https://git-for-windows.github.io](https://git-for-windows.github.io/" \t "_blank)下载（网速慢的同学请移步[国内镜像](https://pan.baidu.com/s/1kU5OCOB" \l "list/path=%2Fpub%2Fgit" \t "_blank)），然后按默认选项安装即可。

安装完成后，在开始菜单里找到“Git”->“Git Bash”，蹦出一个类似命令行窗口的东西，就说明Git安装成功！



安装完成后，还需要最后一步设置，在命令行输入：

$ git config --global user.name "Your Name"

$ git config --global user.email "email@example.com"

因为Git是分布式版本控制系统，所以，每个机器都必须自报家门：你的名字和Email地址。你也许会担心，如果有人故意冒充别人怎么办？这个不必担心，首先我们相信大家都是善良无知的群众，其次，真的有冒充的也是有办法可查的。

注意git config命令的--global参数，用了这个参数，表示你这台机器上所有的Git仓库都会使用这个配置，当然也可以对某个仓库指定不同的用户名和Email地址。

网址：

<https://gitlab.com/>

[cacho.zeng@signalfire.cn](mailto:cacho.zeng@signalfire.cn) abc370201

## 1.4创建版本库

什么是版本库呢？版本库又名仓库，英文名repository，你可以简单理解成一个目录，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可以“还原”。

所以，创建一个版本库非常简单，首先，选择一个合适的地方，创建一个空目录：

$ mkdir learngit

$ cd learngit

$ pwd

/Users/michael/learngit

pwd命令用于显示当前目录。在我的Mac上，这个仓库位于/Users/michael/learngit。

如果你使用Windows系统，为了避免遇到各种莫名其妙的问题，请确保目录名（包括父目录）不包含中文。

第二步，通过git init命令把这个目录变成Git可以管理的仓库：

$ git init

Initialized empty Git repository **in** /Users/michael/learngit/.git/

瞬间Git就把仓库建好了，而且告诉你是一个空的仓库（empty Git repository），细心的读者可以发现当前目录下多了一个.git的目录，这个目录是Git来跟踪管理版本库的，没事千万不要手动修改这个目录里面的文件，不然改乱了，就把Git仓库给破坏了。

如果你没有看到.git目录，那是因为这个目录默认是隐藏的，用ls -ah命令就可以看见。

也不一定必须在空目录下创建Git仓库，选择一个已经有东西的目录也是可以的。不过，不建议你使用自己正在开发的公司项目来学习Git，否则造成的一切后果概不负责。

**把文件添加到版本库**

首先这里再明确一下，所有的版本控制系统，其实只能跟踪文本文件的改动，比如TXT文件，网页，所有的程序代码等等，Git也不例外。版本控制系统可以告诉你每次的改动，比如在第5行加了一个单词“Linux”，在第8行删了一个单词“Windows”。而图片、视频这些二进制文件，虽然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，也就是只知道图片从100KB改成了120KB，但到底改了啥，版本控制系统不知道，也没法知道。

不幸的是，Microsoft的Word格式是二进制格式，因此，版本控制系统是没法跟踪Word文件的改动的，前面我们举的例子只是为了演示，如果要真正使用版本控制系统，就要以纯文本方式编写文件。

因为文本是有编码的，比如中文有常用的GBK编码，日文有Shift\_JIS编码，如果没有历史遗留问题，强烈建议使用标准的UTF-8编码，所有语言使用同一种编码，既没有冲突，又被所有平台所支持。

使用Windows的童鞋要特别注意：

千万不要使用Windows自带的**记事本**编辑任何文本文件。原因是Microsoft开发记事本的团队使用了一个非常弱智的行为来保存UTF-8编码的文件，他们自作聪明地在每个文件开头添加了0xefbbbf（十六进制）的字符，你会遇到很多不可思议的问题，比如，网页第一行可能会显示一个“?”，明明正确的程序一编译就报语法错误，等等，都是由记事本的弱智行为带来的。建议你下载[Notepad++](http://notepad-plus-plus.org/)代替记事本，不但功能强大，而且免费！记得把Notepad++的默认编码设置为UTF-8 without BOM即可：



言归正传，现在我们编写一个readme.txt文件，内容如下：

Git **is** a version control system.

Git **is** free software.

一定要放到learngit目录下（子目录也行），因为这是一个Git仓库，放到其他地方Git再厉害也找不到这个文件。

和把大象放到冰箱需要3步相比，把一个文件放到Git仓库只需要两步。

第一步，用命令git add告诉Git，把文件添加到仓库：

$ git add readme.txt

执行上面的命令，没有任何显示，这就对了，Unix的哲学是“没有消息就是好消息”，说明添加成功。

第二步，用命令git commit告诉Git，把文件提交到仓库：

$ git **commit** -m "wrote a readme file"

[master (root-**commit**) cb926e7] wrote a readme file

1 file changed, 2 insertions(+)

**create** mode 100644 readme.txt

简单解释一下git commit命令，-m后面输入的是本次提交的说明，可以输入任意内容，当然最好是有意义的，这样你就能从历史记录里方便地找到改动记录。

嫌麻烦不想输入-m "xxx"行不行？确实有办法可以这么干，但是强烈不建议你这么干，因为输入说明对自己对别人阅读都很重要。实在不想输入说明的童鞋请自行Google，我不告诉你这个参数。

git commit命令执行成功后会告诉你，1个文件被改动（我们新添加的readme.txt文件），插入了两行内容（readme.txt有两行内容）。

为什么Git添加文件需要add，commit一共两步呢？因为commit可以一次提交很多文件，所以你可以多次add不同的文件，比如：

$ git add file1.txt

$ git add file2.txt file3.txt

$ git commit -m "add 3 files."

**小结**

现在总结一下今天学的两点内容：

初始化一个Git仓库，使用git init命令。

添加文件到Git仓库，分两步：

第一步，使用命令git add <file>，注意，可反复多次使用，添加多个文件；

第二步，使用命令git commit，完成。

# 简单介绍

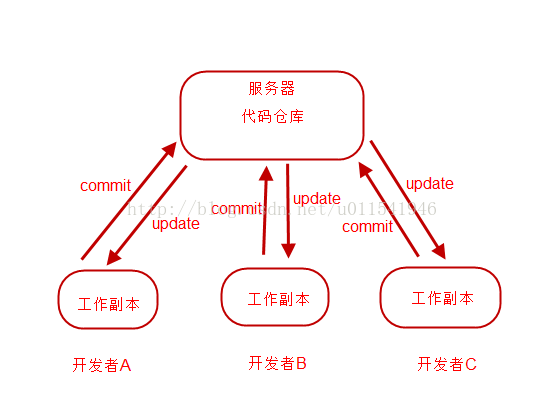
其实很早之前，就想学习和介绍版本控制工具SVN或者GIT。最近开始写Jenkins的文章，马上就要利用Git来做持续集成构建，所以，还是在这里插入一个新的工具介绍的入门介绍。这里先打算介绍如何使用Git这个版本控制工具，第一篇，先简单介绍下版本控制相关的概念。

## 什么是Git

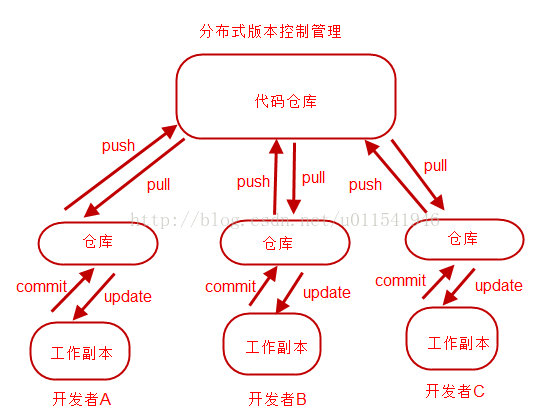
       其实Git就是一个开源的版本控制系统，什么是版本控制系统呢？Version Control System, 简称VCS。版本控制系统简单来说就是用来跟踪文件或者文件夹的变化版本历史记录的一个系统。目前版本控制系统分两类：集中式版本控制（Centralised VCS ）和分布式版本控制（Distributed vcs），集中式版本控制以SVN为代表，而分布式版本控制以Git为代表。

## 2.2 CVCS VS DVCS

集中式版本控制用图表示工作原理，大概是这样



分布式版本控制系统用下图表示。



两者的优缺点就不对比了，毕竟用都没用过，就不要去进行对比，只有自己使用过了才能去评价。

## 2.3 Git是不是就等于Github

       我们经常听到Git和Github，那么是不是有人就让认为两个是相等的关系。其实不是的，Git要远比Github功能多，Github只是一个能够提供提交代码到在线网站的一个网站而已。当前，Github提供虚拟接口，让你的项目可以提交到Github上托管。

# git在windows上安装

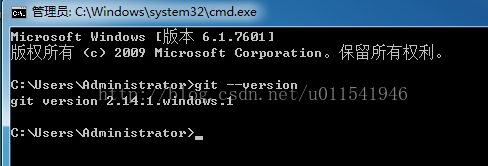
本篇介绍Git在windows上的安装，和简单应用。Git支持Wcindows和Linux和Mac系统的安装，实际工作项目中，一般是安装到linux系统上作为一个git服务器来使用。我们这里，作为学习使用，安装到自己的windows电脑就可以。

Git官网和下载

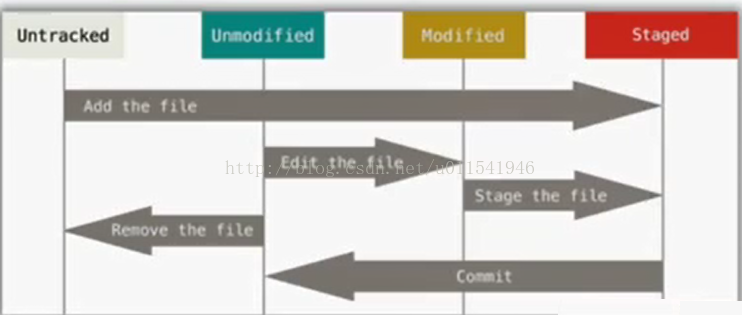
浏览器打开<https://git-scm.com/>，点击下载。



       点击Windows版本开始下载。又是很无语，官网下载速度巨慢，一两个小时都下载不完，有需要的[点击这里](https://pan.baidu.com/s/1hs9tZ3u" \t "_blank)。下载完后，直接双击进行安装，全面默认安装，一直点击Next直到安装结束。那么安装结束，如何测试是否安装成功呢? 在cmd 输入git --version,如下图，可以看到安装的版本号。



## Git文件三种状态转换图

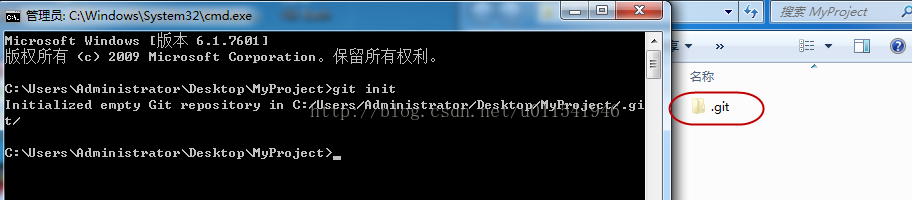


       上面这个图片是github上的，我们借助这张图来理解文件的三种状态。首先，图片中的Unmodified 和Modified两个状态合并成一个状态，叫Unstaged。所以，三种状态分别是Untracked、Unstaged、Staged. 第一种状态，Untracked，也就是说git压根就不知道这个文件，对应的中文就是未被追踪的状态；第二个状态，Unstaged，文件以及添加到git仓库中，但是还没有提交，处于等待提交的状态；第三种状态，Staged，到了Staged状态，意味着文件可以执行提交操作。一旦执行了提交操作，文件就由Staged状态变成Unstaged状态。整个过程就是这样，三个状态形成环状。

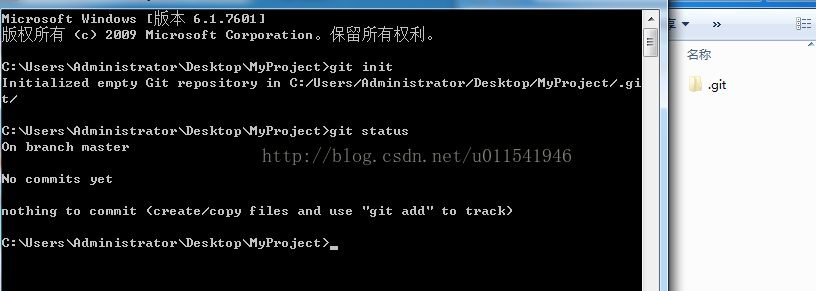
## 3.2实际项目来演示文件三种状态

**3.2.1 Untracked状态**

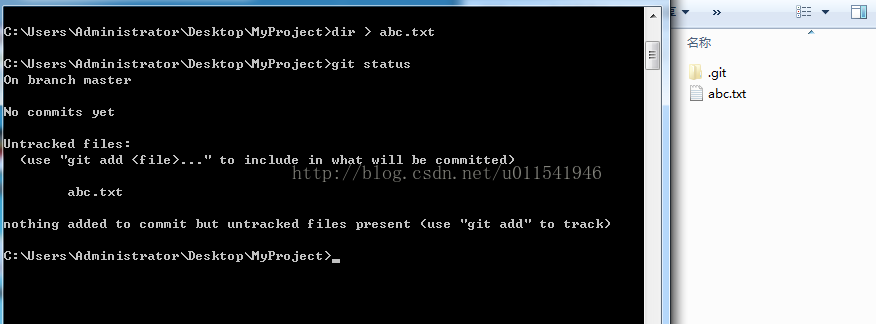
       桌面创建一个空文件夹，当做我们的项目文件夹，然后cd到该文件夹的路径，输入 git init命令回车,可以看到在项目文件夹下创建了一个.git的文件夹。这个步骤就是git仓库的初始化操作，就是告诉git我们的文件夹GitProjects已经添加到了git仓库管理。



检查当前项目的文件状态



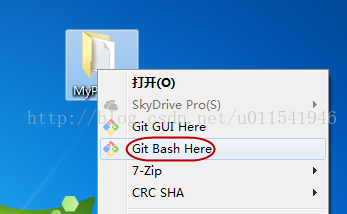
当前显示没有提交记录。在项目文件下添加一个文件abc.txt后，再次检查文件状态。



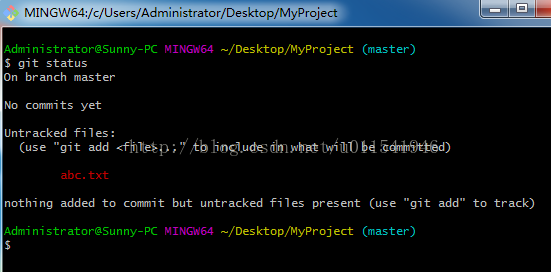
       这个时候文件的状态就是Untracked状态，此时此刻，git才知道这个仓库下出现了一个没有被管理的文件，会提醒你用 git add file 这个命令去把Untracked状态的文件添加到仓库管理范围，这样这个文件就可以被提交。

**3.2.2 Unstaged状态**

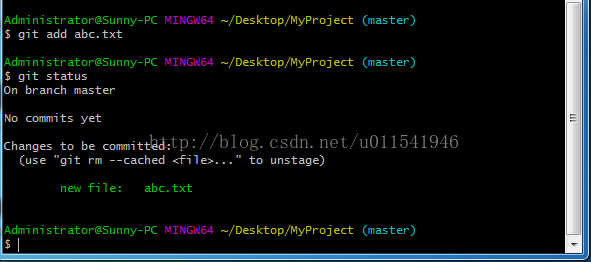
       一般我们操作或者输入git 命令是在git bash中进行，如何打开git bash呢，找到桌面你的项目，例如GitProject,右键这个项目，选择git bash here。



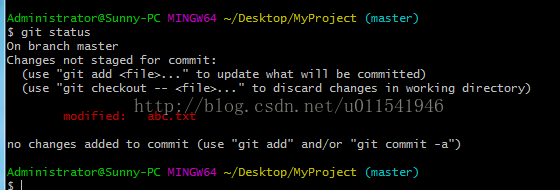
在git bash上查看仓库文件状态。



使用 git add filename 把文件添加到仓库



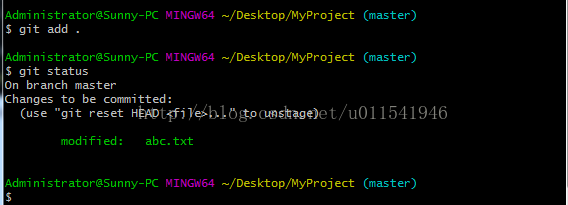
       此时的abc.txt文件的状态就是Unmodifed，上面可以看到new file，所以是没有被修改的状态。接下来，我们需要把这个文件的内容进行修改。这里，我们把abc.txt 里面的内容删除，然后随意输入一些字符，保存，然后在git bash查看文件状态。



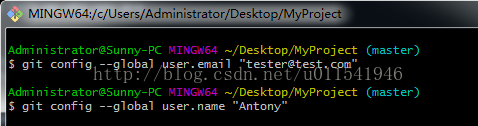
这个时候，文件abc.txt的状态就变成了Modified。Unmodified和Modified两种状态组成了Unstaged，

**3.2.3 Staged状态**

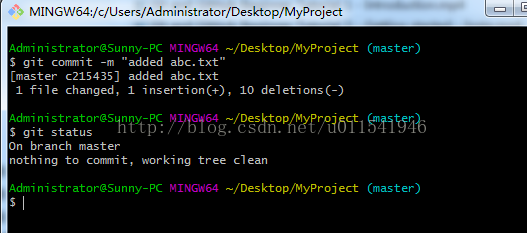
        上面的图，我们看到abc.txt 显示红色的modified的状态，我们需要通过git add file 来把这个文件状态从modified改成Staged，任何文件到了Staged状态，就意味了可以执行提交命令。所以，git会提醒你， Changes to be committed，变成了可提交状态。git add .中这个点表示任何文件。



       在做commit命令之前，我们需要创建一个身份，就是告诉git这次提交是谁做的。因为可能存在重复的name，所以这里还需要设置email，一般来说email肯定不会有相同的。



       这里设置了两个全局变量，一个是user.email 和user.name，以后在这个机器上的提交都算到这个用户名下。你可以通过 git config user.name查看当前用户是谁。 下面，我们开始用commit命令提交到仓库。



       上面的参数-m表示message，就是告诉别人，你这次提交的内容是什么，这个-m参数不是强制性的。提交完后，Staged状态又变成了Unstaged，如果有新的文件，就会变成Untracked状态，又开始三种状态之间切换。

总结：

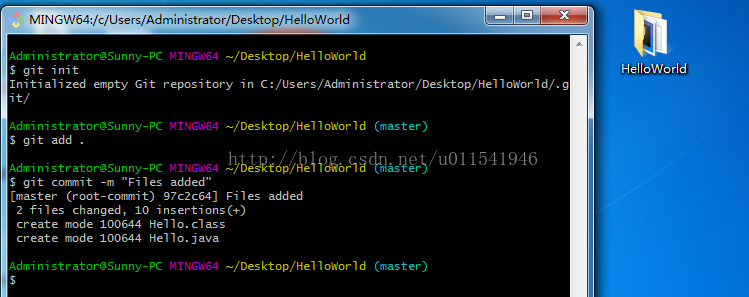
       Git的三种文件状态就是上面介绍的过程，第一次玩这个命令，绝对会有脑袋疼的感觉。三种状态，加上不同命令，会搞晕你。暂时没法理解也没关系，以后用多了git就自然而然知道了。知道了哪些状态下可以输入什么git命令。有了一次项目代码成功提交的经验之后，就会很好学会了git的基本使用。

# 把本地项目提交到Github

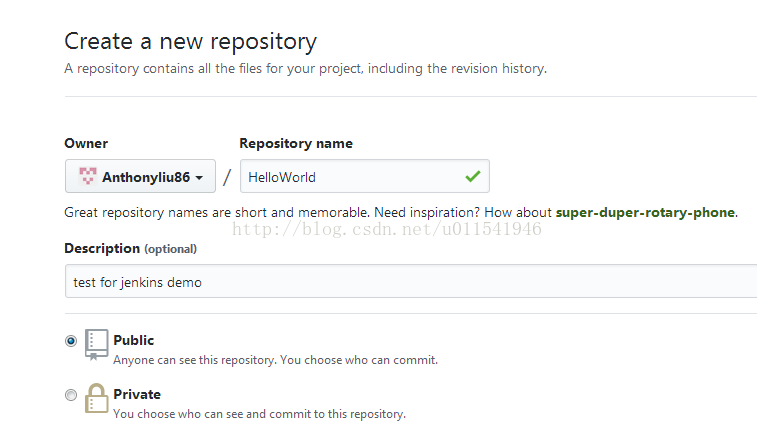
前面一篇文章，我们介绍了如何在windows上安装git软件，和git得文件三种状态的介绍。知道了一些简单的git命令。这篇，我们介绍如何把本地的一个项目提交到自己的github网站上面。所以，在做本练习之前，你需要去github网站申请一个账号。

## 本地把一个项目添加到git仓库

下面的HelloWorld是我桌面的一个项目文件夹。里面有一个Hello.java和一个Hello.class文件。



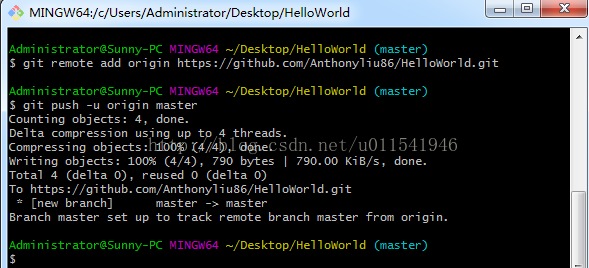
## 4.2 Github上创建一个project，待会要用



点击确定创建按钮。



       上面github的项目地址URL可以拷贝下来，下面步骤备用。根据上面图的一些git命令，教我们如何把一个本地小项目给提交到github上的仓库。



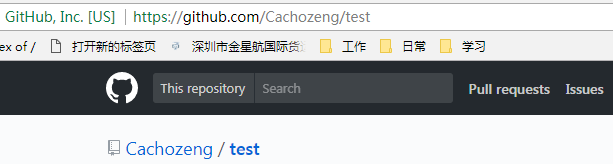
       在执行git push -u origin master命名时候，会弹出一个登陆github的登录框，输入用户名和密码，开始上传到github仓库。到这里，本地仓库上传到github上的仓库就完成了。其实我们只是用了命令的方式，还有一种方式是采用SSH。假如你在本地写了一个selenium的自动化测试框架或者项目，你可以通过本篇方式上传到github仓库。

## 从远程库克隆到本地

上次我们讲了从本地库上传到远程库，如何关联远程库。

现在，假设我们从零开发，那么最好的方式是先创建远程库，然后，从远程库克隆。

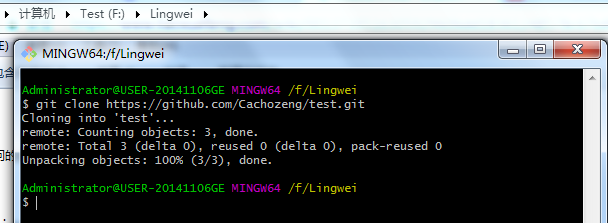
复制你要克隆的项目路径: <https://github.com/Cachozeng/test>



现在，远程库已经准备好了，下一步是用命令git clone克隆一个本地库：

在你要克隆的本地库路径右键“git base here”，输入命令

git clone <https://github.com/Cachozeng/test.git> 然后进入的路径查看是否增加了test文件夹



# 分支和合并

在版本回退里，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。

一开始的时候，master分支是一条线，Git用master指向最新的提交，再用HEAD指向master，就能确定当前分支，以及当前分支的提交点：



每次提交，master分支都会向前移动一步，这样，随着你不断提交，master分支的线也越来越长：

当我们创建新的分支，例如dev时，Git新建了一个指针叫dev，指向master相同的提交，再把HEAD指向dev，就表示当前分支在dev上：



你看，Git创建一个分支很快，因为除了增加一个dev指针，改改HEAD的指向，工作区的文件都没有任何变化！

不过，从现在开始，对工作区的修改和提交就是针对dev分支了，比如新提交一次后，dev指针往前移动一步，而master指针不变：



假如我们在dev上的工作完成了，就可以把dev合并到master上。Git怎么合并呢？最简单的方法，就是直接把master指向dev的当前提交，就完成了合并：



所以Git合并分支也很快！就改改指针，工作区内容也不变！

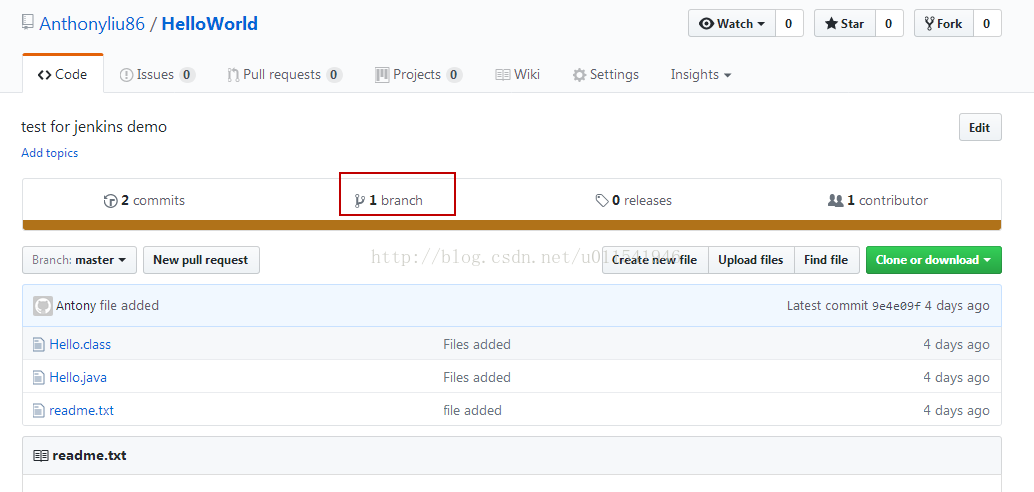
合并完分支后，甚至可以删除dev分支。删除dev分支就是把dev指针给删掉，删掉后，我们就剩下了一条master分支：



真是太神奇了，你看得出来有些提交是通过分支完成的吗？

本篇介绍Git中分支和合并概念和相关操作。先来理解下什么是分支，分支就相当于一颗树的不同分叉出去的树枝。有分支就有主干，所以git中的主干叫master或者main。其实，一个公司的软件开发，如果开发经理不是脑残，那么一定就会进行代码分支管理。例如一个项目A，V3.0版本作为这个项目的master分支，V3.1.0作为这个项目的分支1，V3.2.0作为这个项目的分支2。整体上三个版本大的软件功能差不多，稍微有点细微的差别，不同版本开发不同新的功能。如果分支1或者分支2的功能开发完善，功能稳定，那么就很有可能把新功能的代码提交到master branch保存。实际上，很多公司确实就是这么做的。

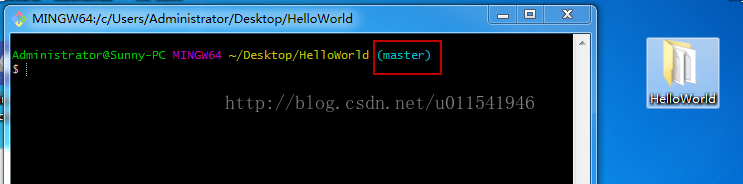
       我先举例一下什么是master branch。本篇的前一篇，我把在我github上上传了一个叫HelloWorld的项目。打开，这个可以显示这个项目有一个branch。



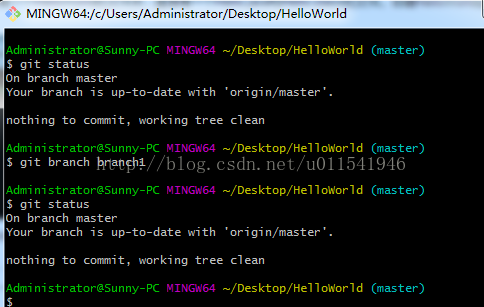
如果只有1个branch的话，那么这个就是master branch。好了，现在解释清楚了什么叫分支，接下来介绍如何新建branch。

## 如何新建分支

      接着前面一篇内容，桌面有一个HelloWorld的文件夹，里面有一个Hello.java和Hello.class的文件。右键HelloWorld这个文件夹，选择git bash，这个时候默认就进入了该项目的master分支。



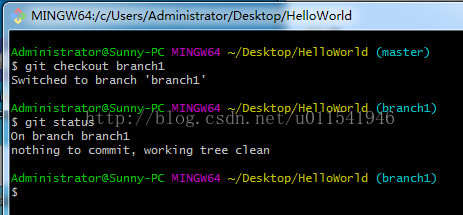
输入命令 git branch branchName 来新建一个分支。



通过上面一些命令，我们发现这里创建了branch1的分支，但是当前还是在master分支，那么如何才能切换到新建的分支呢。

## 5.2如何切换到新分支

在git中切换到新的分支，用的命令是 git checkout branchName, checkout的意思是迁出的意思。



从上面图看出，我从master分支切换到了branch1分支。接下来，我们在branch1分支下添加一些新内容。



我们新增了文件，先把文件提交到仓库中。

创建并切换到分支：git check -b branchName

因为创建、合并和删除分支非常快，所以Git鼓励你使用分支完成某个任务，合并后再删掉分支，这和直接在master分支上工作效果是一样的，但过程更安全。

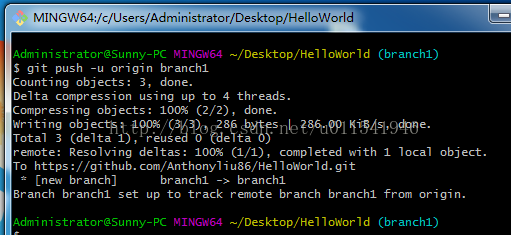
这里再来练习一下之前的git 常用commond。

1）git status 查看文件状态

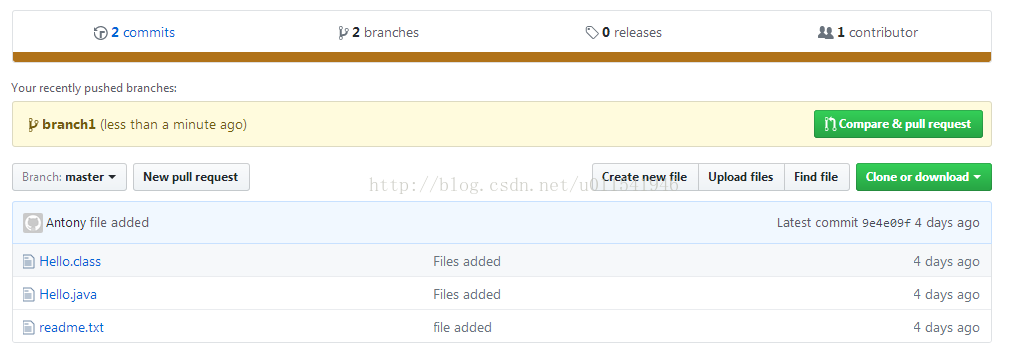
2）git add a.txt 把a.txt添加到可提交状态

3）git commit -m "added a.txt" 把文件提交

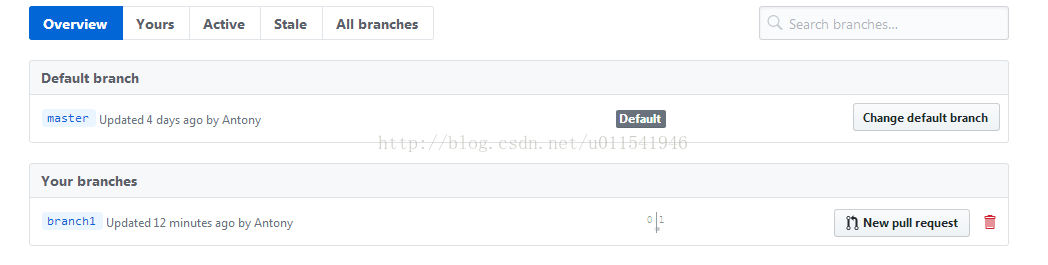
从右侧文件显示来看，看不出哪个文件属于哪个branch。这里，我们先把本地仓库代码推送到远程git服务器，这里就是 github。



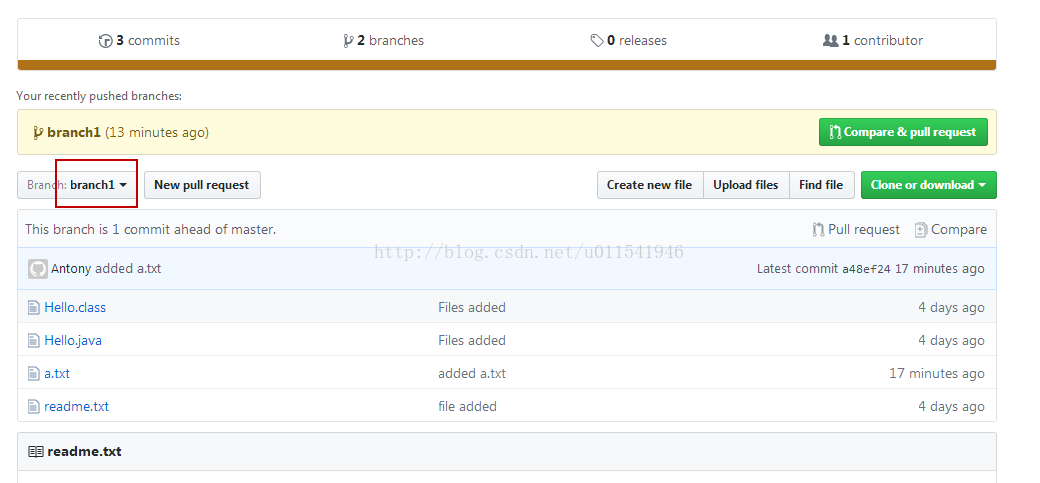
有些人，这里可能要问，为什么要从本地仓库推送到远程服务器上。如果看了git入门系列第一篇，就明白。git工作原理图中说明了，远程有一个git服务器，每个开发本地有一个本地代码仓库，只有每个开发人员本地的仓库代码推送到服务器上，其他的开发人员才能用他的代码。好了，我们再次登录github，就发现有两个branch。



从上面看到master 这个branch里没有我们新建的a.txt文件，但是上面显示了2 branchs，我们点击这个2 branchs。



点击淡蓝色字体的 branch1，进入观察文件内容



这个和master的这张图对比，就有区别了，显示增加了一个a.txt文件。

## 5.3如何把分支合并到master

这里需先checkout到master 然后输入 git merge branch1，就可以看到master下加了一个a.txt文件。

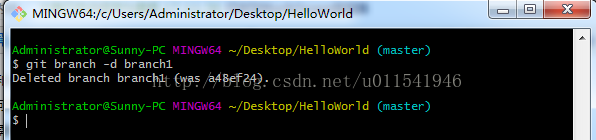


如果你要在github上看到效果，这里你需要执行 git push -u origin master命令，这样github下的master branch下才可以看到a,txt文件。

## 5.4如何把一个分支删除

俗话说，有添加就有删除，那么如何把一个branch删除呢？

先看看本地仓库如何删除一个分支



然后看看如何在github上删除一个分支



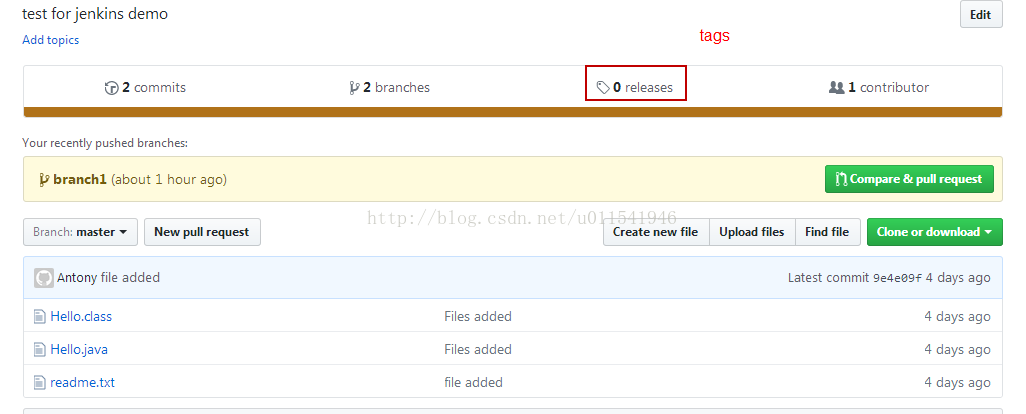
好了，关于git的分支和合并以及删除分支就介绍到这里，需要几下每个操作对应的git 命令。

# Tags-入门结束篇

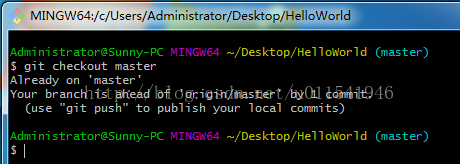
本篇介绍Tags, 如果学习过HTML的，应该知道，字面意思就是一个标签。在git中，前面我们介绍了branch。为什么有了分支，还要有标签呢。实际上，tags是branch中的一些小阶段的标记。例如，我在branch1这个分支，通过创建不同Tags，有V1.0和V1.1等等。这些V1.0和V1.1，只是我们那些在分支中功能比较稳定的代码做一个标签，例如release，就是这个点的代码可以用来发布。还有一个，就是我可以通过切换不同tags，来观察代码不同，相当于就查看历史版本记录。

## 选择在什么branch进行创建tag

这里，我们还是拿前面文章的HelloWorld举例，我们这里在master中创建tag，先来看看github上的状态。



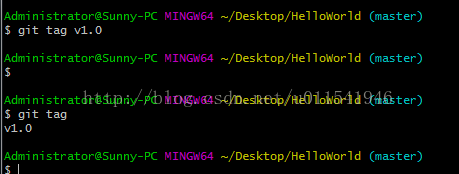
我特意圈出一个红圈，这个就表示tags，当前有0个tags。我们在本地bash shell输入checkout 到master命令。



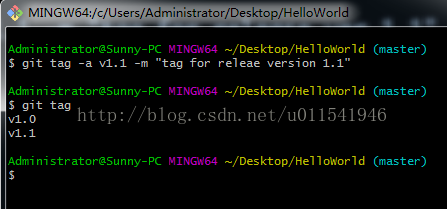
上面图片，显示当前就在master分支。

## 6.2创建一些tags

输入git tag tagname来创建tag

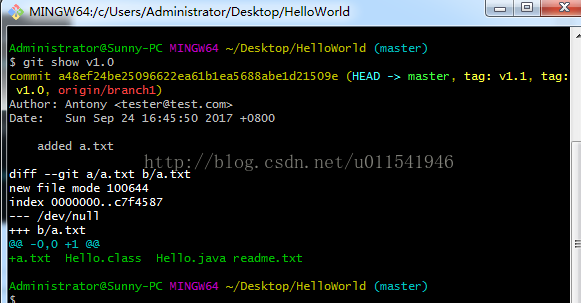


还可以创建带注释的tag

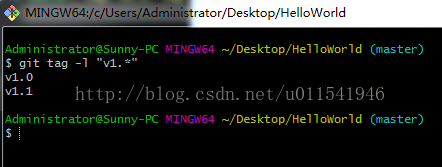


## 6.3显示tag

上面我们知道git tag可以显示当前所有tag的名称，如果需要具体某一个tag的详情信息，输入命令 git show v1.0

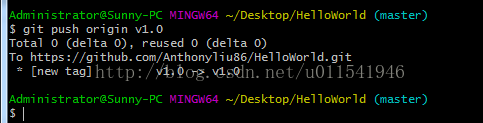


还可以输入匹配的多个tag，显示V1.开头的tags$$

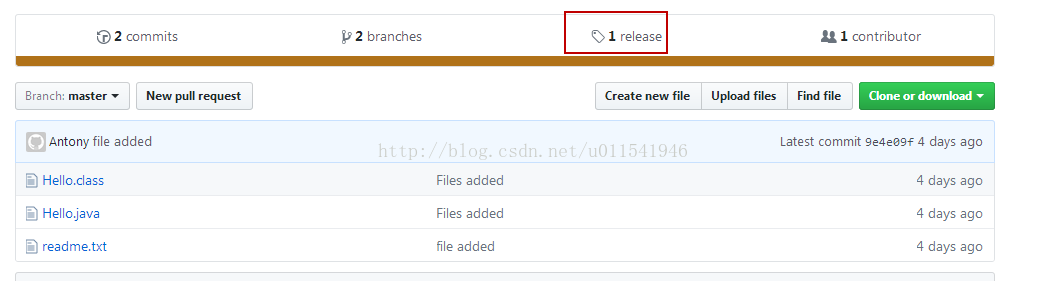


## 6.4推送tags到远程服务器

这里介绍如何推送tag到github网站，命令是 git push origin  tagname; 如果有多个tag需要push，执行这个命令git pust --tags

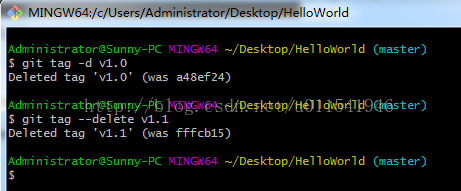


刷新github，可以看到 0 release 变成了 1release

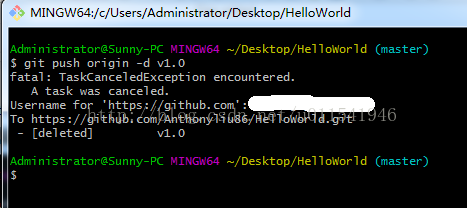


## 6.5删除tags

先看看本地仓库删除，输入命令 git tag -d v1.0



如何远程删除github上的tag



      这个可以去github上验证是否被删除了这个release节点。本篇关于git中tags的操作就介绍到这里。作为本系列的结束篇，我想git的入门或者基本操作都介绍完了。其他的知识点或者问题解决过程，应该是在实际工作中才会出现，出现了问题，大家要去网上多查查资料，肯定有解决办法。

# 使用国内的Git托管服务

使用GitHub时，国内的用户经常遇到的问题是访问速度太慢，有时候还会出现无法连接的情况（原因你懂的）。

如果我们希望体验Git飞一般的速度，可以使用国内的Git托管服务——[码云](https://gitee.com/)（[gitee.com](https://gitee.com/)）。

和GitHub相比，码云也提供免费的Git仓库，并且，免费版本还包含私有库。此外，还集成了代码质量检测、项目演示等功能。对于团队协作开发，码云还提供了项目管理、代码托管、文档管理的服务，5人以下小团队免费。

使用码云和使用GitHub类似，我们在码云上注册账号并登录后，需要先上传自己的SSH公钥。选择右上角用户头像 -> 菜单“修改资料”，然后选择“SSH公钥”，填写一个便于识别的标题，然后把用户主目录下的.ssh/id\_rsa.pub文件的内容粘贴进去：



点击“确定”即可完成并看到刚才添加的Key：

**Windows如何获取公钥**

1. 首先你要安装git工具(参考第三点)
2. 右键鼠标，选中 “Git Bash here”，当然你也可以在windows的 “开始”--->“所以程序”，或者安装目录打开它
3. 输入指令，进入.ssh文件夹cd ~/.ssh/

如果提示 “ No such file or directory”，你可以手动的创建一个 .ssh文件夹即可

命令为：mkdir ~/.ssh

1. 配置全局的name和email（参考前面设置提交者信心）

这里是的你github或者bitbucket的name和email

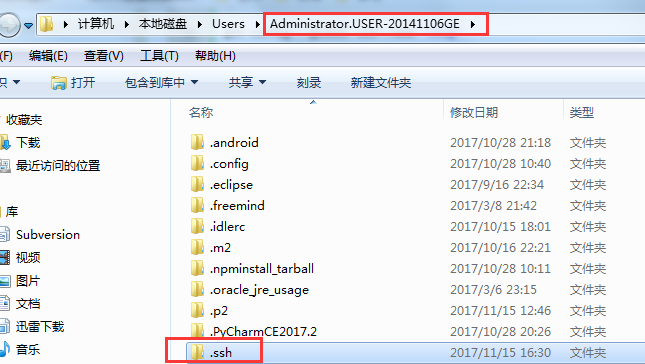
5、生成SSH key 命令：ssh-keygen -t rsa -C "394369786@qq.com"

连续按三次回车，这里设置的密码就为空了，并且创建了key。

最后得到了两个文件：id\_rsa和id\_rsa.pub

1. 复制SSH key

打开Admin目录进入.ssh文件夹，用记事本打开id\_rsa.pub，复制里面的内容添加到你github或者bitbucket ssh设置里即可。



1. 测试是否添加成功

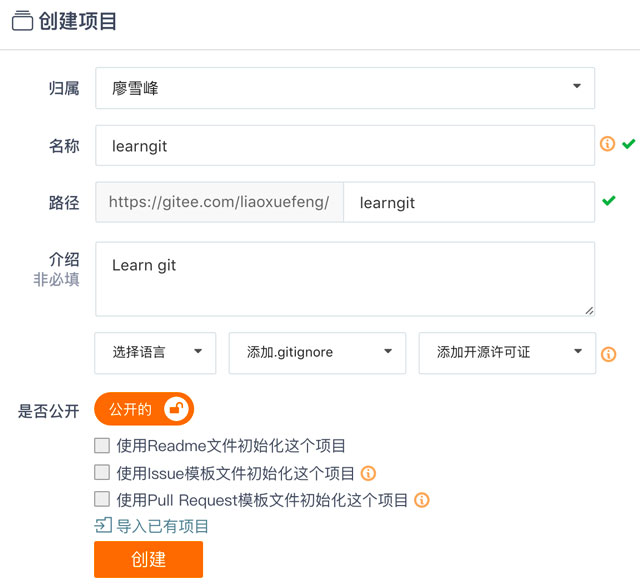
ssh -T git@bitbucket.org

ssh git@github.com



如果我们已经有了一个本地的git仓库（例如，一个名为learngit的本地库），如何把它关联到码云的远程库上呢？

首先，我们在码云上创建一个新的项目，选择右上角用户头像 -> 菜单“控制面板”，然后点击“创建项目”：



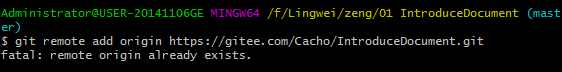
项目名称最好与本地库保持一致：

然后，我们在本地库上使用命令git remote add把它和码云的远程库关联：

git remote add origin https://gitee.com/Cacho/IntroduceDocument.git

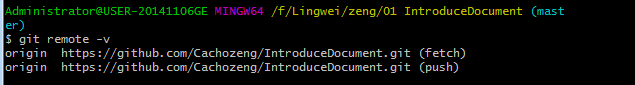
之后，就可以正常地用git push和git pull推送了！

如果在使用命令git remote add时报错：



这说明本地库已经关联了一个名叫origin的远程库，此时，可以先用git remote -v查看远程库信息：git remote -v

可以看到，本地库已经关联了origin的远程库，并且，该远程库指向GitHub。



我们可以删除已有的GitHub远程库：

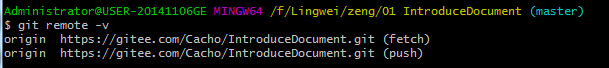
git remote rm origin

再关联码云的远程库（注意路径中需要填写正确的用户名）：

git remote add origin https://gitee.com/Cacho/IntroduceDocument.git

此时，我们再查看远程库信息：

git remote –v



现在可以看到，origin已经被关联到码云的远程库了。通过git push命令就可以把本地库推送到Gitee上。

有的小伙伴又要问了，一个本地库能不能既关联GitHub，又关联码云呢？

答案是肯定的，因为git本身是分布式版本控制系统，可以同步到另外一个远程库，当然也可以同步到另外两个远程库。

使用多个远程库时，我们要注意，git给远程库起的默认名称是origin，如果有多个远程库，我们需要用不同的名称来标识不同的远程库。

仍然以learngit本地库为例，我们先删除已关联的名为origin的远程库：

git remote rm origin

然后，先关联GitHub的远程库：

git remote add github https://github.com/Cachozeng/IntroduceDocument.git

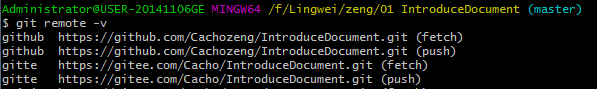
注意，远程库的名称叫github，不叫origin了。

接着，再关联码云的远程库：

git remote add gitte https://gitee.com/Cacho/IntroduceDocument.git

同样注意，远程库的名称叫gitee，不叫origin。

现在，我们用git remote -v查看远程库信息，可以看到两个远程库：



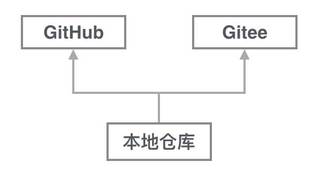
如果要推送到GitHub，使用命令：

git push github master

如果要推送到码云，使用命令：

git push gitee master

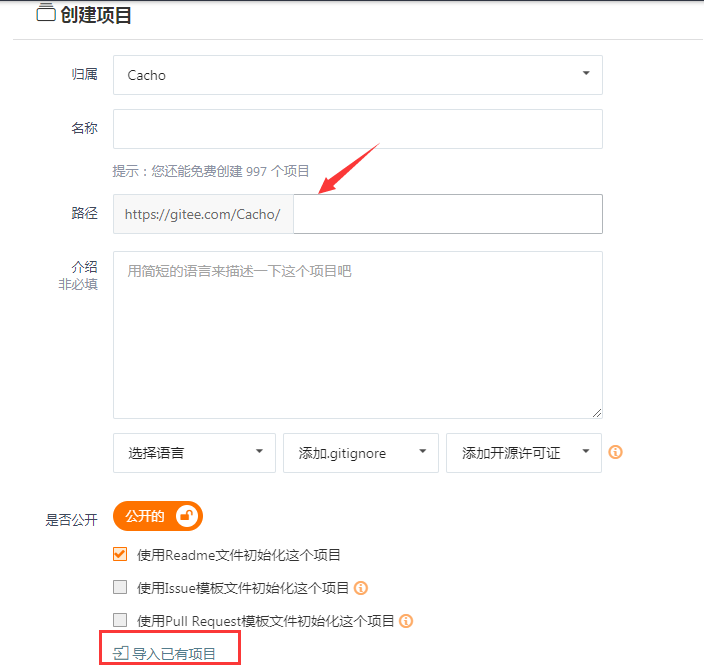
这样一来，我们的本地库就可以同时与多个远程库互相同步：



码云也同样提供了Pull request功能，可以让其他小伙伴参与到开源项目中来。你可以通过Fork我的仓库：[https://gitee.com/liaoxuefeng/learngit](https://gitee.com/liaoxuefeng/learngit" \t "_blank)，创建一个your-gitee-id.txt的文本文件， 写点自己学习Git的心得，然后推送一个pull request给我，这个仓库会在码云和GitHub做双向同步。

如何Fork某个项目：

1. 进入到创建项目页面，填入克隆路径，点击导入已有项目



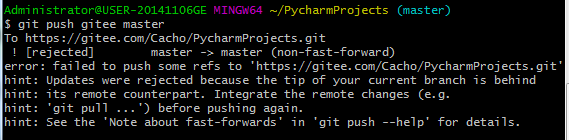
1. 输入要导入项目的路径



1. 克隆到本地路径

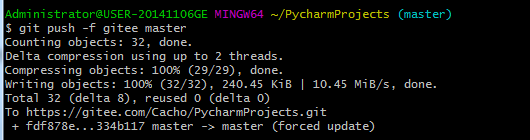
# 遇到的问题

1. 远程创建了含有Readme的仓库, 本地初始化并添加了远程仓库后, push失败, 出现提示：



**解决办法：强制提交**

1. 强推，即利用强覆盖方式用你本地的代码替代git仓库内的内容: git push –f



2. 先把git的东西fetch到你本地然后merge后再push

- $ git fetch

- $ git merge

这2句命令等价于

$ git pull

可是, 有时候还会出现问题:

上面出现的 [branch "master"]是需要明确(.git/config)如下的内容

[branch "master"]

remote = origin

merge = refs/heads/master

这等于告诉git2件事:

1，当你处于master branch, 默认的remote就是origin。

2，当你在master branch上使用git pull时，没有指定remote和branch，那么git就会采用默认的remote（也就是origin）来merge在master branch上所有的改变

如果不想或者不会编辑config文件的话，可以在bush上输入如下命令行：

$ git config branch.master.remote origin

$ git config branch.master.merge refs/heads/master

之后再重新git pull下。最后git push你的代码吧。

1. 开发过程push数据到远程出错：RT ! [rejected] master -> master (fetch first)

**解决办法：先pull 命令：**git pull origin master