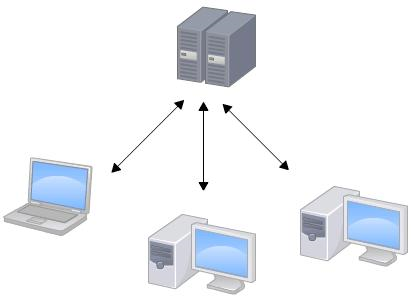
GIT资料

1. 简介

版本控制主要分为两种，集中式版本控制和分布式版本控制。CVS和SVN即使典型的集中式版本控制系统，而Git是目前世界上最先进的分布式版本控制系统。

1. 集中式版本控制：

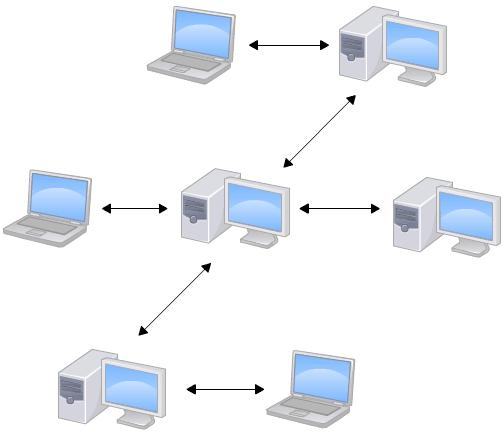
集中式版本控制的仓库是集中存放在中央服务器的，而干活的时候，用的都是自己的电脑，所以要先从中央服务器取得最新的版本，然后开始干活，干完活了，再把自己的活推送给中央服务器。集中式版本控制系统最大的弊端就是必须联网才能工作。在局域网的情况下效率不错，但是通过外放访问的话，由于网络带宽和稳定性的因素将导致效率极低。



1. 分布式版本控制：

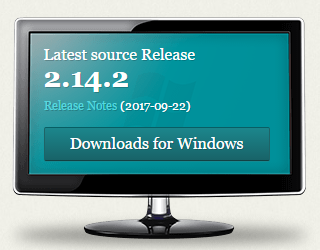
分布式版本控制系统没有“中央服务器”的概念，每个人的电脑上都是一个完整的版本库，这样，你工作的时候，就不需要联网了，因为版本库就在你自己的电脑上。既然每个人电脑上都有一个完整的版本库，那多个人如何协作呢？比方说你在自己电脑上改了文件A，你的同事也在他的电脑上改了文件A，这时，你们俩之间只需把各自的修改推送给对方，就可以互相看到对方的修改了。

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。

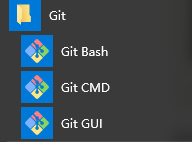


1. Git安装

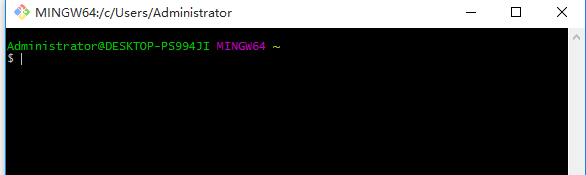
Git提供了不同系统的版本支持，可从Git官网下载相应的版本的，这里以windows版本为例。从官网https://git-scm.com下载最新版本，下载后直接执行安装即可。



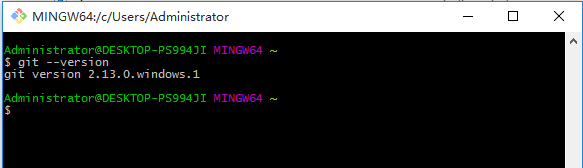
Git安装完毕后，提供多种操作终端，其中Git Bash以Linux命令方式进行操作。



点击Git Bash进入操作终端



输入命令git –version查看当前git版本



补充:Bash常用命令

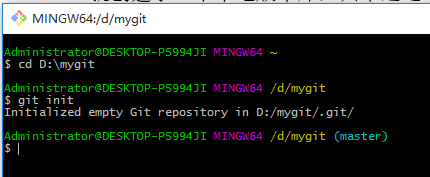
1）、cd : 改变目录。  
2）、cd . . 回退到上一个目录，直接cd进入默认目录  
3）、pwd : 显示当前所在的目录路径。  
4）、ls(ll): 都是列出当前目录中的所有文件，只不过ll(两个ll)列出的内容更为详细。  
5）、touch : 新建一个文件 如 touch index.js 就会在当前目录下新建一个index.js文件。  
6）、rm: 删除一个文件, rm index.js 就会把index.js文件删除。  
7）、mkdir: 新建一个目录,就是新建一个文件夹。  
8）、rm -r : 删除一个文件夹, rm -r src 删除src目录， 好像不能用通配符。  
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

1. Git使用
   1. 创建本地仓库

类似SVN，我们首先需要在客户端创建一个本地仓库，用来存放项目。

1）在任意一个磁盘中新建一个目录，例如：在D盘中创建一个文件夹叫mygit。

2）在Git Bash中进入此文件夹。



3）使用git init命令将mygit目录初始化一个空的本地版本库，这样我们就创建了一个本地版本库，只不过这个库的工作区中没有任何的内容，但是有一个隐藏文件夹“.git”。这个文件夹就是Git的版本库。

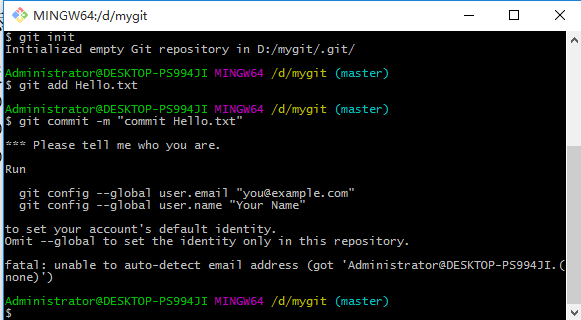
* 1. 添加文件到本地仓库

版本库有了，那么我们就可以向版本库中添加文件了。

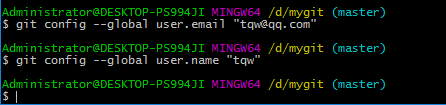
1） 在mygit目录中创建一个文本文件（例如Hello.txt）并填写一些内容

2）在Git Bash中使用命令git add Hello.txt

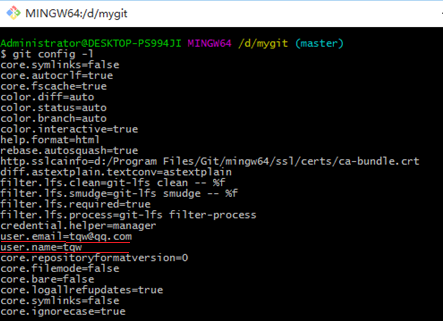
3）再使用命令git commit -m "提交说明" 提交到版本库中



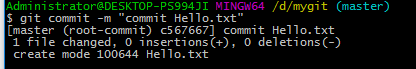
运行命令，我们发现文件成功加入暂存区，但commit失败，这是为什么呢，对了，我们还没有配置用户信息，如何配置，请注意认真读上图信息，git config –global user.emial [you@example.com](mailto:you@example.com)，看到这里，你学会配置用户信息了吗，我们一起来配置一下吧



配置成功了吗，我们通过命令git config –l查看配置信息

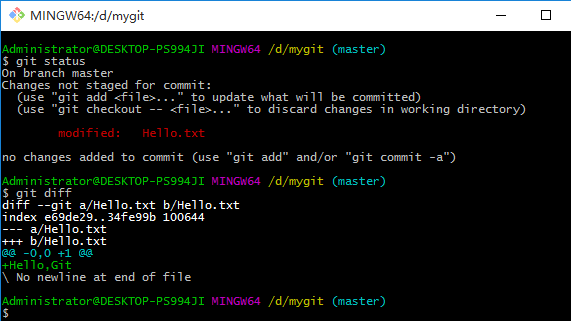


我们再次提交文件，文件成功提交到版本库。

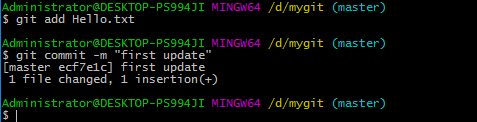


* 1. 修改文件

Hello.txt刚刚已经成功加入版本库，现在我们对工作区中的Hello.txt进行修改操作，添加内容”Hello,Git”,文件一旦发生修改，状态将会发生变化，可以通过命令git status查看



红色的部分告诉我们Hello.txt的文件已经修改，但是还没有提交到版本库。还可以使用git diff命令查看修改前和修改后的内容，修改完毕后，再次使用add,commit命令提交到版本库



1. 暂存区，工作区，版本库

在前面操作中，我们提到了暂存区，工作区，版本库，他们具体指什么，他们自己有什么关系呢？

**工作区（Working Directory）**

在之前创建的mygit目录就是工作区，我们创建的任何文件都是存放在工作区中。



**版本库（Repository）**

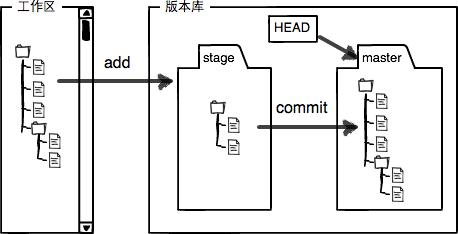
工作区有一个隐藏目录“.git”，这个就是Git的本地版本库。

**暂存区（Repository）**

当我们执行add命令时，工作区文件将放入到暂存区，当执行commit命令后，文件才放入版本库

说明：

Git的版本库里存了很多内容，其中最重要的就一个是暂存区（stage），还有Git为我们自动创建的第一个主分支master，以及指向master的一个指针叫HEAD（HEAD还可以指向其他的分支，作用就是做分支切换）。



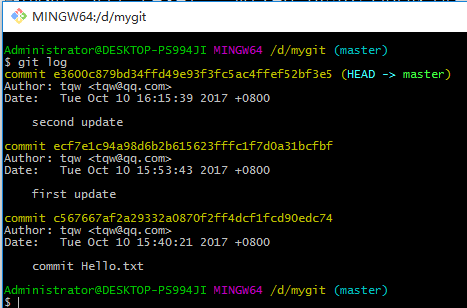
我们把文件往Git版本库里添加的时候，是分两步执行的：

第一步是用git add把文件添加进去，实际上就是把新建或修改的文件添加到暂存区。

第二步是用git commit执行提交，实际上就是把暂存区的所有内容提交到当前master分支。

1. 版本回退

1) 可以使用git log命令查看最近的提交日志列表，当初会有相应的版本号、用户信息、提交时间和提交说明等内容。



说明：

commit：提交的版本号

Author：提交人信息

Date：提交时间

再后面的信息就是提交说明

2）使用git reset 命令执行版本回退

reset命令有三个参数，这三个参数将导致回退的内容将在在不同的作用域中（工作区、暂存区、版本库）。

--soft，这个参数表示在版本库中回退到旧的版本。而最新的版本内容会移到暂存区保存。

--mixed， 这个参数表示在版本库中进行版本回退，同时重置暂存区。（默认选项）

--hard，这个参数表示表示在版本库中进行版本回退，同时重置暂存区和工作区。

例如：git reset --soft HEAD^ 表示在版本库的master分支中回退到上一个版本

HEAD说明：

1） HEAD表示回退到当前最新版本

例如：git reset --mixed HEAD (回退到当前最新的版本，并重置暂存区)

2） HEAD^表示回退到上一个版本

例如：git reset --mixed HEAD^ (回退到上一个版本，并重置暂存区)

3） HEAD~2表示回退到上两个版本（其中2可以替换成其他数字，表示回退到第几个版本）

例如：git reset --mixed HEAD~2 (回退到上两个版本，并重置暂存区)

版本号说明：

可以将HEAD替换成版本号，表示回退到指定的版本。版本号不一定需要全部输入，只需要输入前面几个字符即可。

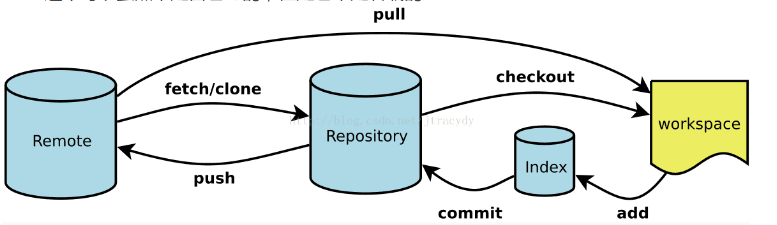
例如：git reset --mixed 版本号 （回退到指定版本，并将版本内容重置到暂存区）

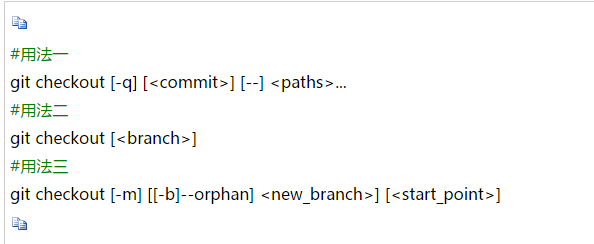
使用git reset filename撤销暂存区的操作。

撤销暂存区后，工作区的内容依然被保留，可以在工作区从新修改好内容后再次add到暂存区。

1. 版本检出checkout

版本回退后，只是版本库中指针发生改变，本地工作中内容并没有发生变化，如何将版本库中内容检出到本地工作区，此时需要用到checkout命令。





$ git checkout -- . 或写作 git checkout .

用暂存区的所有文件直接覆盖本地文件，不给用户任何确认的机会！

$ git checkout -- filename

#用暂存区中filename文件来覆盖工作区中的filename文件。

$ git checkout commit\_id -- file\_name



#如果不加commit\_id，那么git checkout -- file\_name 表示恢复文件到本地版本库中最新的状态



1. 删除文件和文件夹

1） 使用git rm filename命令删除工作区中的文件

例如：git rm Hello.txt

注意：删除之后，记得要将此操作commit到版本库中，因为此时删除的动作还只是在暂存区中，并未同步到版本库。



2) 使用git rm 目录名 -r -f 删除文件夹及其下所有的文件

例如：git rm demo -r –f

注意：执行完同样需要commit到版本库中

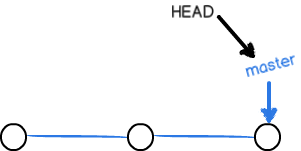
1. 本地分支

1）分支的作用

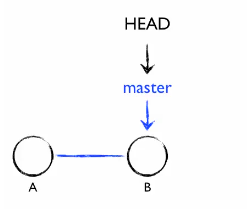
分支在实际中有什么用呢？假设你准备开发一个新功能，但是需要两周才能完成，第一周你写了50%的代码，如果立刻提交，由于代码还没写完，不完整的代码库会导致别人不能干活了。如果等代码全部写完再一次提交，又存在丢失每天进度的巨大风险。有了分支，就不用怕了。你创建了一个属于你自己的分支，别人看不到，还继续在原来的分支上正常工作，而你在自己的分支上干活，想提交就提交，直到开发完毕后，再一次性合并到原来的主分支上，这样，既安全，又不影响别人工作。

2) 分支的概念

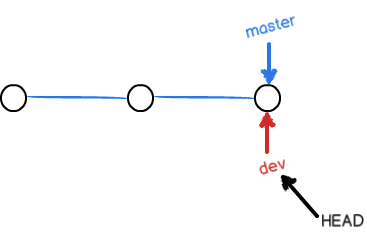
每次提交，Git都把它们串成一条时间线，这条时间线就是一个分支。截止到目前，只有一条时间线，在Git里，这个分支叫主分支，即master分支。HEAD严格来说不是指向提交，而是指向master，master才是指向提交的，所以，HEAD指向的就是当前分支。



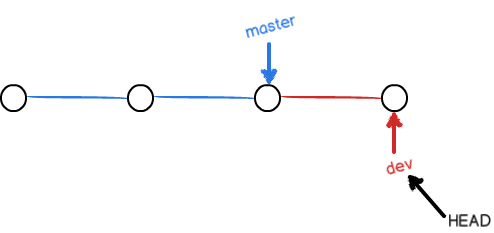
每次提交，master分支都会向前移动一步，这样，随着你不断提交，master分支的线也越来越长：



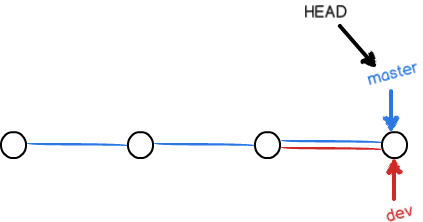
当我们创建新的分支，例如dev时，Git新建了一个指针叫dev，指向master相同的提交，再把HEAD指向dev，就表示当前分支在dev上：



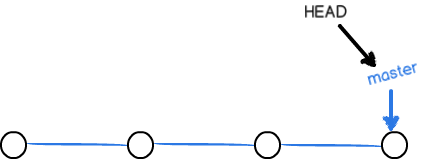
从现在开始，对工作区的修改和提交就是针对dev分支了，比如新提交一次后，dev指针往前移动一步，而master指针不变：



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：



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：



3) 分支的使用

1. 创建分支: git branch 分支名

例如：git branch dev



2.切换分支: git checkout 分支名

例如: git checkout dev



3. 查看分支: git branch

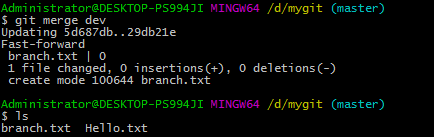


说明：\*号表标识前正在操作的分支

4. 合并分支：git merge 分支名

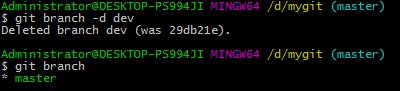
例如：git merge dev我们先切换到master分之下，然后合并dev分支的内容

这样，我们就把dev分支的内容合并到了master分支上



5. 删除分支：git branch -d 分支名

例如：git branch -d dev



6.内容比较 git diff

git diff 用于比较内容的差别。

比较规则：

先比较暂存区内容，如果暂存区为空时则比较当前版本库中HEAD所指向的分支内容。

1）比较当前的工作区和当前分支的内容差别

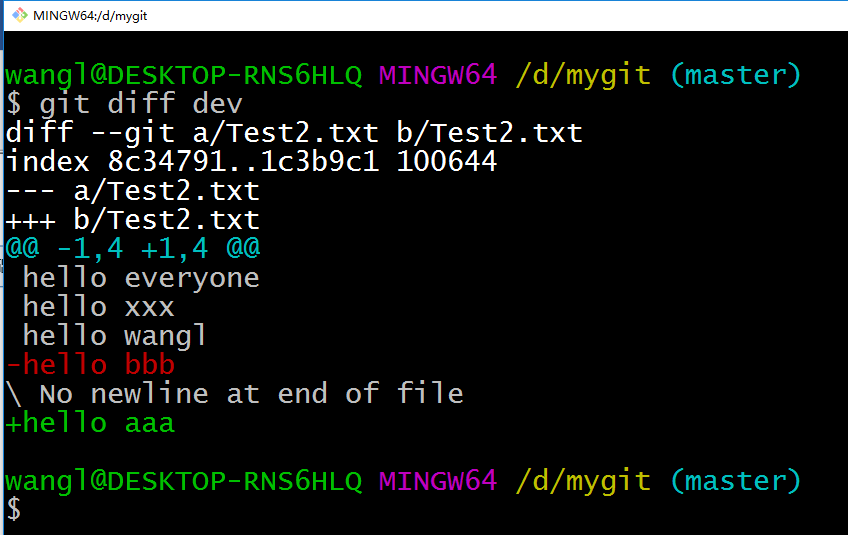
例如：git diff



说明：git diff 默认是比较所有文件的内容。也可以指定文件名，用于比较当前文件的内容。例如：git diff Hello.txt

2) 比较当前的工作区和指定分支的内容差别

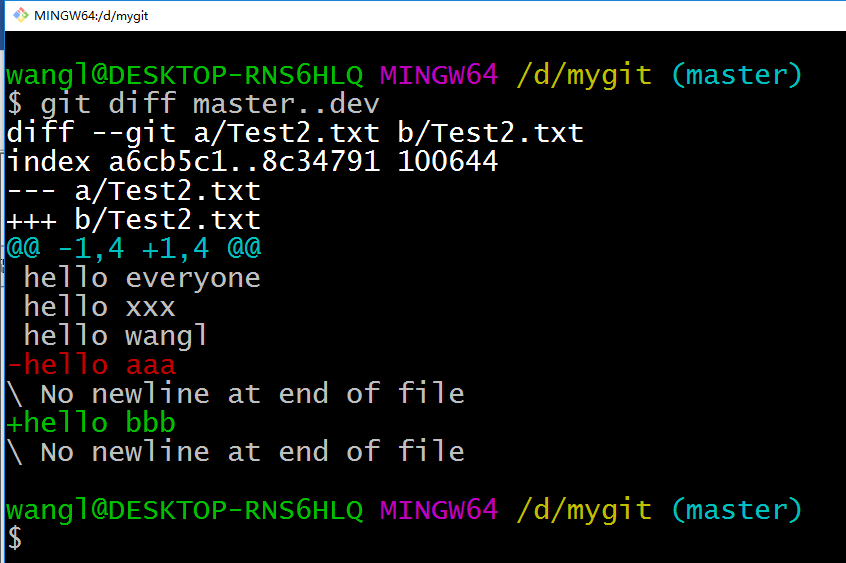
例如: git diff 分支名



说明：同样可以指定具体某个文件进行内容比。例如：git diff dev Hello.txt

3) 比较两个分支之间的内容差别

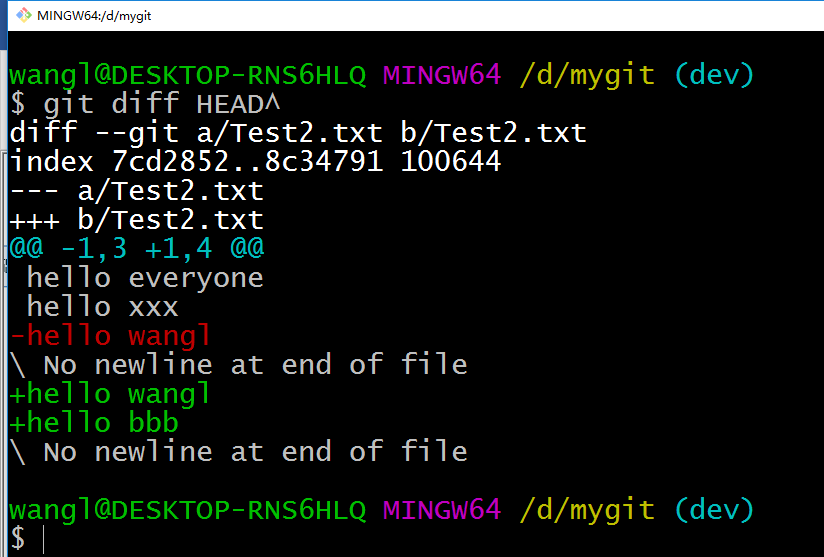
例如：git diff 分支名..分支名



说明：同样可以指定具体某个文件进行内容比。例如：git diff master..dev hello.txt

4) 比较当前工作区与当前分支上一个版本的内容

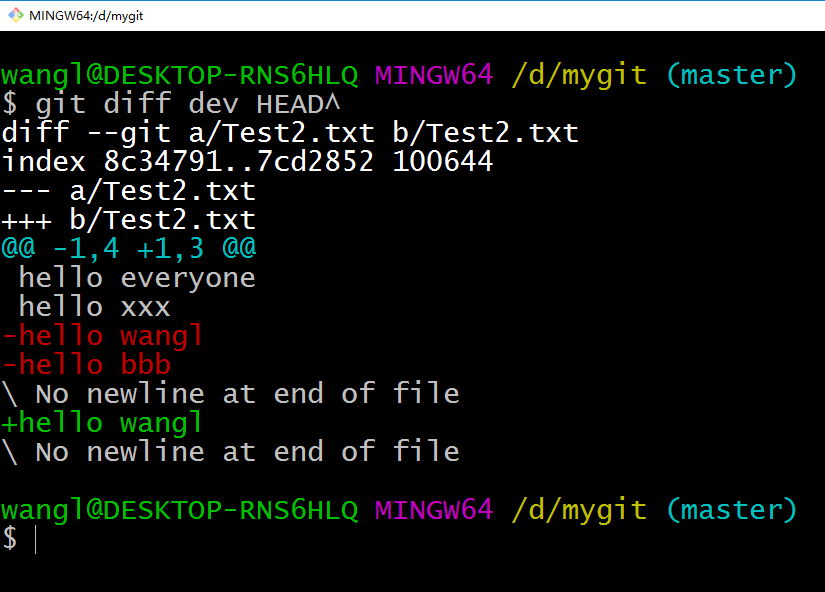
例如：git diff HEAD^



说明：还可以比较上n个版的内容。例如：git diff HEAD~2 (比较上两个版本的内容)。

并且可以指定文件名，比较个某个文件内容。例如：git diff HEAD~2 Hello.txt

5) 比较当前工作区与指定分支上一个版本的内容



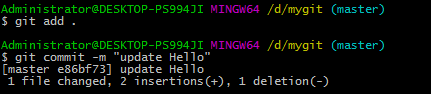
说明：也可以比较上n个版的内容和比较个指定文件的内容。同上

1. 版本冲突

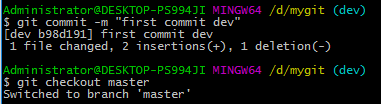
在合并分支时，并不一定都成功。这是因为当两个分支都做了修改并且都提交到了版本库中，这样在合并的时候可能就会产生版本冲突。

场景案例:

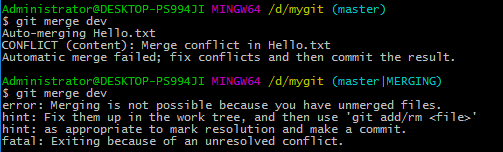
1) 在master主分支中对Hello文件进行修改并提交



2）切换到dev分支也对Hello文件进行内容修改并提交



3）切换回master分支合并dev分支的内容，这是就产生版本冲突，合并失败

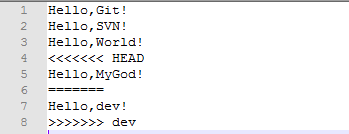


原因分析：

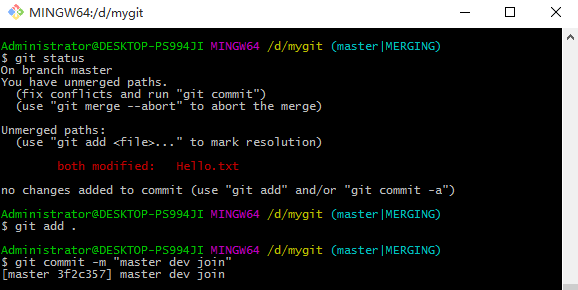
因为master分支和dev分支各自都分别有新的提交。这种情况下，Git无法执行“快速合并”，只能试图把各自的修改合并起来，但这种合并就导致了冲突的原因。

解决办法：

我们查看文件内容，Git用<<<<<<<，=======，>>>>>>>标记出不同分支的内容。

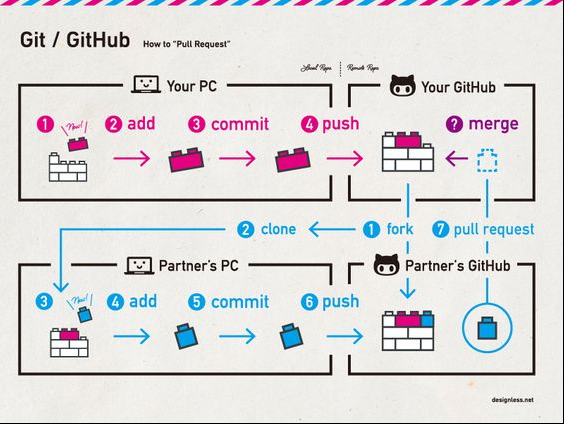


因此我们手动将内容修改合并后再进行提交以解决此问题。



1. Github

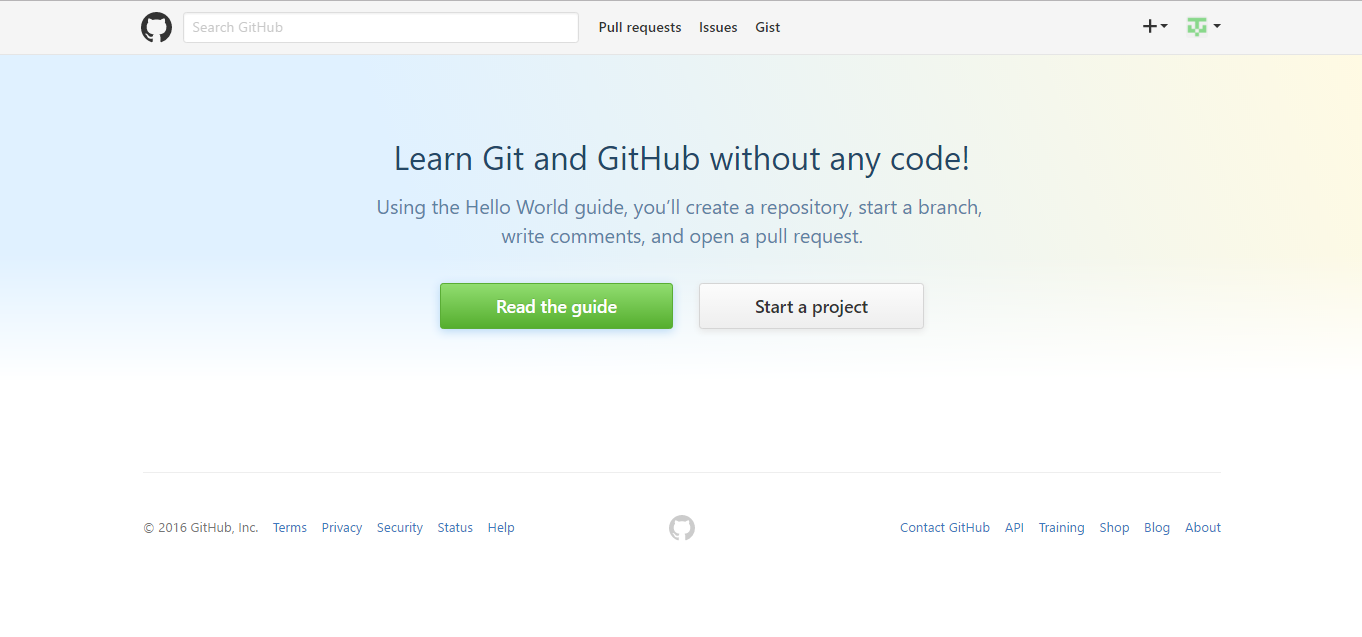
GitHub 是一个面向开源及私有软件项目的托管平台，因为只支持 Git 作为唯一的版本库格式进行托管，故名 GitHub。它于 2008 年 4 月 10 日正式上线，除了 Git 代码仓库托管及基本的 Web 管理界面以外，还提供了订阅、讨论组、文本渲染、在线文件编辑器、协作图谱（报表）、代码片段分享（Gist）等功能。作为开源代码库以及版本控制系统，Github拥有超过900万开发者用户。随着越来越多的应用程序转移到了云上，Github已经成为了管理软件开发以及发现已有代码的首选方法。https://github.com/



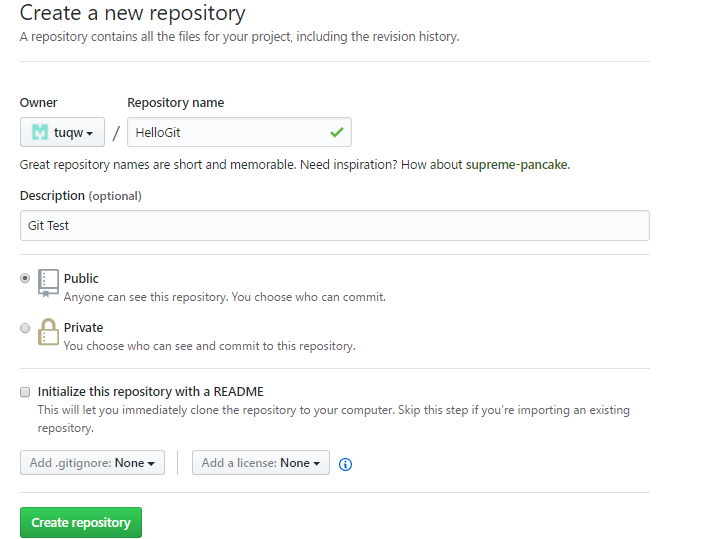
访问www.github.com官网，在首页填写相应信息，点击Sign up for GitHub



完成注册后，会自动登陆，接下来在下面的页面点击Start a Project开始创建一个项目

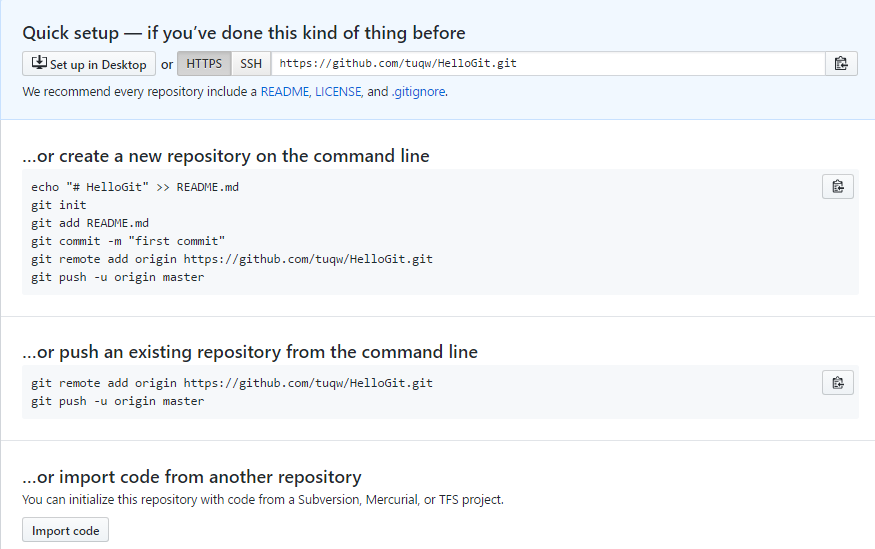


接下来会跳转到一下这个页面



填写仓库名（Repository name），其他可以默认值即可，点击Create Repository,这样就创建好了一个空的远程仓库，

会自动跳转到以下页面，该页面显示了当前仓库地址，创建仓库，连接远程仓库，推送文件的常用命令



在Github中创建一个空的远程仓库，但是什么项目和内容也没有。

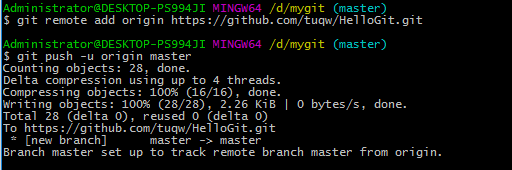
* 将本地版本库的项目和内容同步到远程仓库。

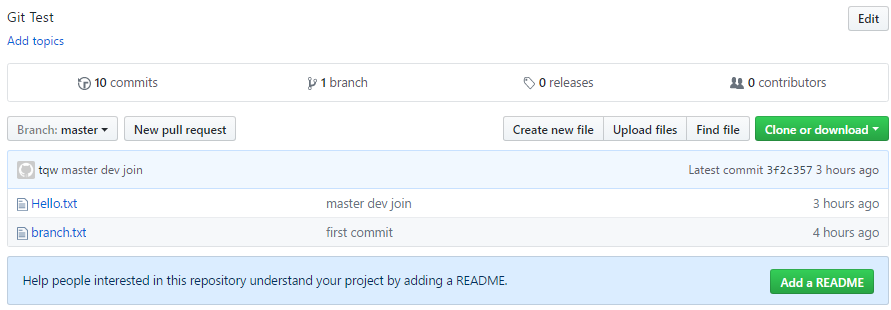
使用以下命令：

语法: git remote add <自定义远程版本库别名> <github版本库地址>

同步完成之后，再使用push命令将内容推送到远程版本库的指定分支上

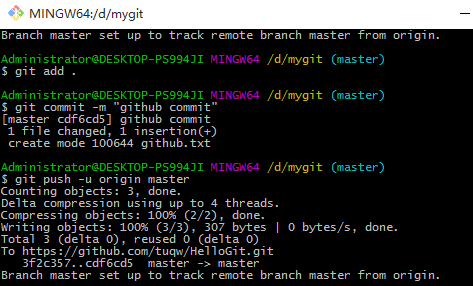
git push <自定义远程版本库别名> master

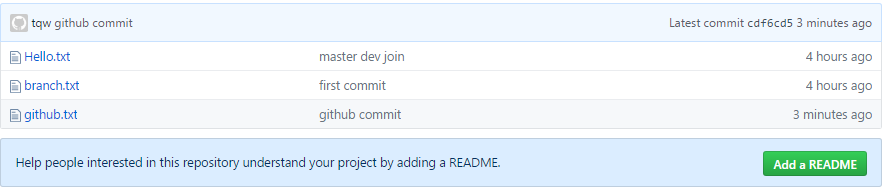




此时，我们发现本地版本库中内容以及提交到远程仓库中。

如果我们本地仓库再次有更新，如何实现又如何提交到远程版本库中，我们仍然是先add，然后commit，最后push到远程库中



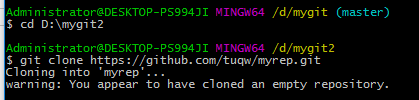


* Clone远程仓库到本地

Github中创建好远程版本库并且有项目和内容，就可以clone到本地开发，使用以下命令：

语法：git clone -o <远程版本库别名> <github版本库地址>

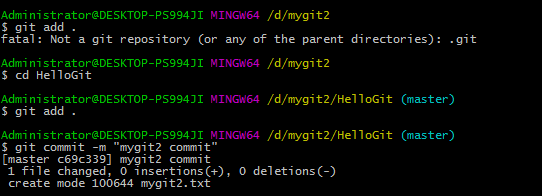
如果远程版本库已经有别名，可以直接克隆

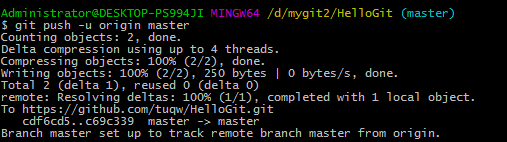


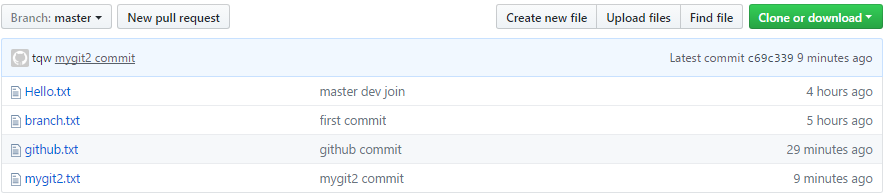
查看远程仓库信息



将远程仓库内容克隆到本地后，可以进行修改，然后再次条件到远程仓库







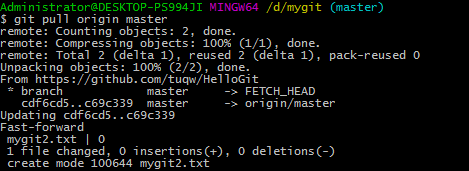
此时，远程仓库已经改变，名称为mygit的本地仓库如何更新到最新修改呢

拉取远程版本库的分支更新到本地分支并执行合并（更新）

git pull <远程版本库别名> <远程分支>:<本地分支>

例如:

git pull origin master:master



1. Coding码市

Coding 是一个面向开发者的云端开发平台[1]  ，目前提供代码托管，运行空间，质量控制，项目管理等功能。此外，还提供社会化协作功能，包含了社交元素，方便开发者进行技术讨论和协作。

官网：https://coding.net/

1. 在Eclipse上使用Git