# 1GIT基础

## 1.1Git基础知识

### 1.1GIT简介

Git是一款免费、开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目。Git 是 Linus 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。Git 与常用的版本控制工具 CVS, Subversion 等不同，它采用了分布式版本库的方式，不必服务器端软件支持。Git 的速度很快，这对于诸如 Linux kernel 这样的大项目来说自然很重要。Git最为出色的是它的合并跟踪（merge tracing）能力。

Torvalds 开始着手开发Git 是为了作为一种过渡方案来替代 BitKeeper，后者之前一直是 Linux 内核开发人员在全球使用的主要源代码工具。开放源码社区中的有些人觉得BitKeeper 的许可证并不适合开放源码社区的工作，因此 Torvalds 决定着手研究许可证更为灵活的版本控制系统。尽管最初 Git 的开发是为了辅助 Linux 内核开发的过程，但是我们已经发现在很多其他自由软件项目中也使用了 Git。很多 Freedesktop 的项目迁移到了 Git 上。

作为开源自由原教旨主义项目，Git没有对版本库的浏览和修改做任何的权限限制。目前Git已经可以在windows下使用，主要方法有二：msysgit和Cygwin。Cygwin和Linux使用方法类似，Windows版本的GIT提供了友好的GUI(图形界面)，安装后很快可以上手。

### 1.2GIT和SVN的区别

GIT不仅仅是个版本控制系统，它也是个内容管理系统(CMS),工作管理系统等。

如果你是一个具有使用SVN背景的人，你需要做一定的思想转换，来适应GIT提供的一些概念和特征。

Git 与 SVN 区别点：

1、GIT是分布式的，SVN不是：这是GIT和其它非分布式的版本控制系统，例如SVN，CVS等，最核心的区别。

2、GIT把内容按元数据方式存储，而SVN是按文件：所有的资源控制系统都是把文件的元信息隐藏在一个类似.svn,.cvs等的文件夹里。

3、GIT分支和SVN的分支不同：分支在SVN中一点不特别，就是版本库中的另外的一个目录。

4、GIT没有一个全局的版本号，而SVN有：目前为止这是跟SVN相比GIT缺少的最大的一个特征。

5、GIT的内容完整性要优于SVN：GIT的内容存储使用的是SHA-1哈希算法。这能确保代码内容的完整性，确保在遇到磁盘故障和网络问题时降低对版本库的破坏。

### 1.3集中式和分布式管理的区别

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，而工作的时候，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，然后开始工作，工作完成，再把自己完成的任务推送给中央服务器。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。



图12-1集中式管理

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，效率十分低下。

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，所有人都没法工作了。

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，没有它大家也一样工作，只是交换修改不方便而已。



图12-2分布式管理

## 1.2GIT的安装配置

**12.2.1GIT的安装**

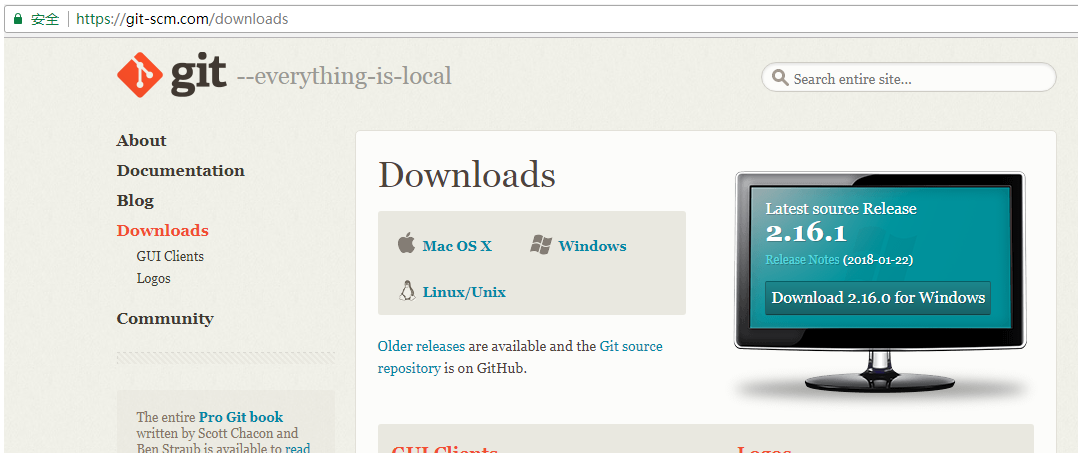
在使用Git前我们需要先安装 Git。Git 目前支持 Linux/Unix、Solaris、Mac和 Windows 平台上运行。

Git 各平台安装包下载地址为：http://git-scm.com/downloads

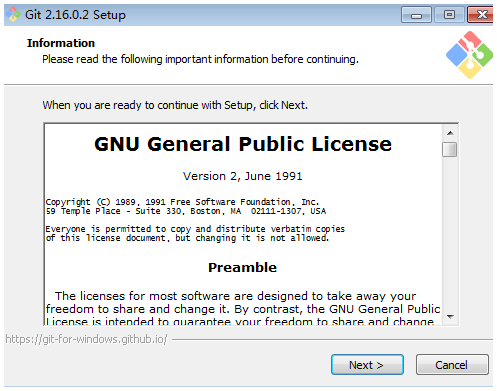
在 Windows 平台上安装 Git 同样轻松，有个叫做 msysGit 的项目提供了安装包，可以到 GitHub 的页面上下载 exe 安装文件并运行：

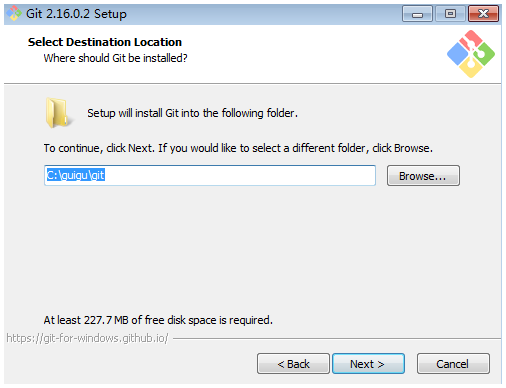
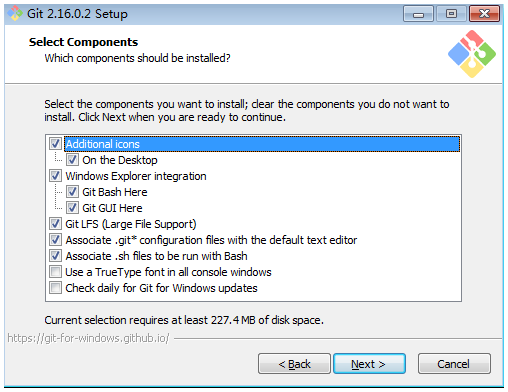
安装包下载地址： <https://git-for-windows.github.io/>

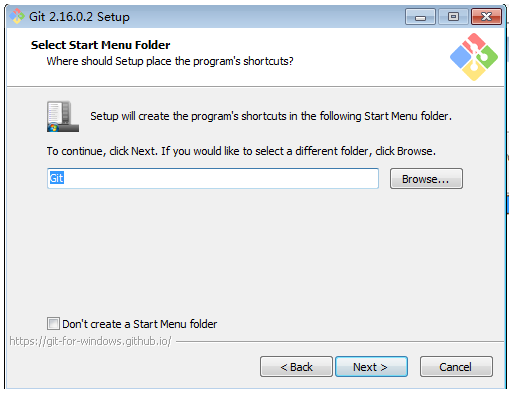
本文将以Git-2.16.0.2-64-bit为例

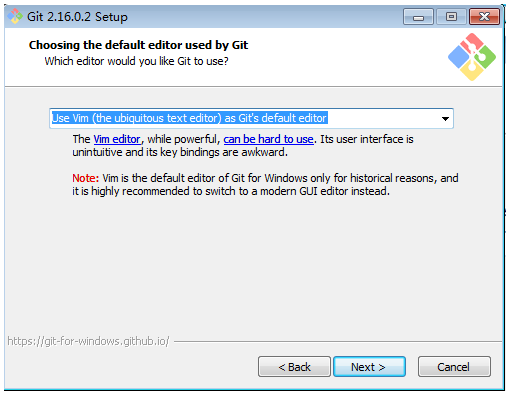


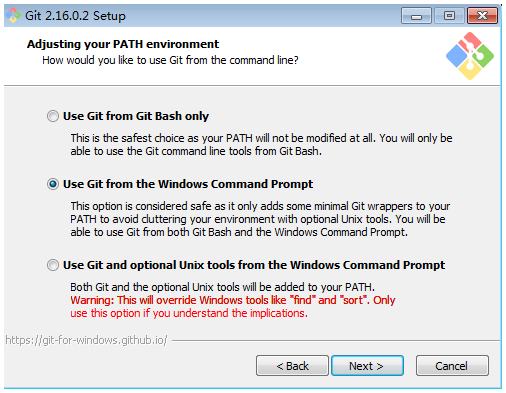
进行安装

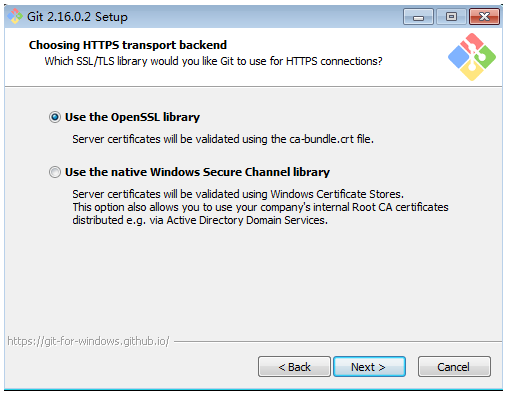


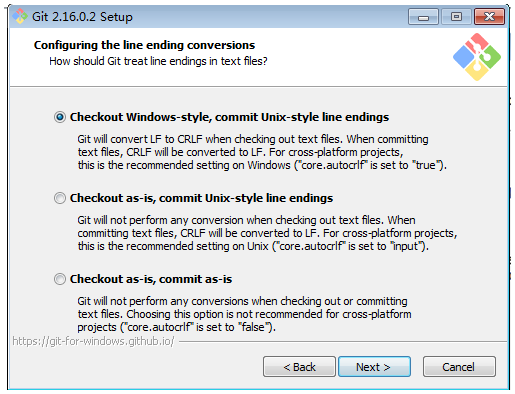
  


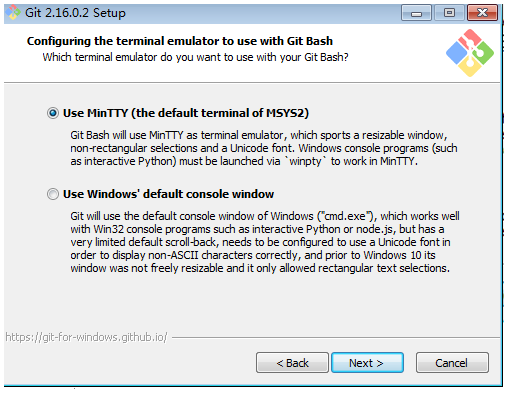


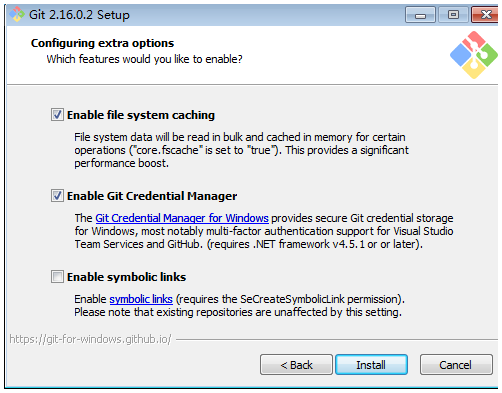


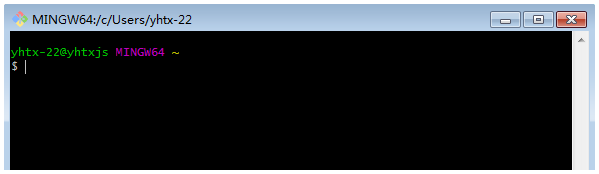












**12.2.2Git的配置**

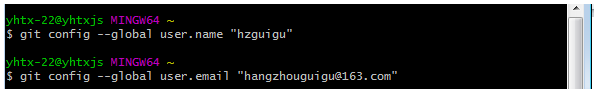
**12.2.2.1配置用户信息**

用户名和邮箱地址的作用:

* 用户名和邮箱地址是本地Git客户端的一个变量，不随git库而改变。
* 每次commit都会用用户名和邮箱纪录。
* github的contributions统计就是按邮箱来统计的。

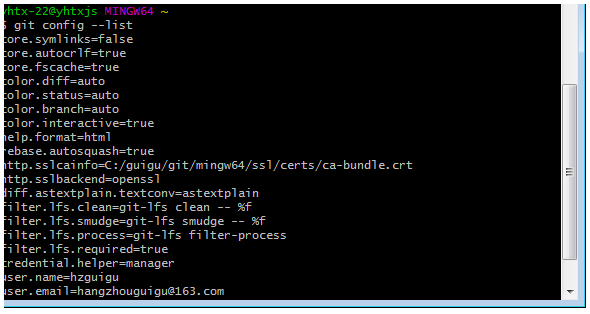
$ git config --global user.name "hzguigu"

$ git config --global user.email "hangzhouguigu@163.com"



**12.2.2.2查看配置信息**

要检查已有的配置信息，可以使用 git config --list 命令：



## 1.3GIT工作流程和术语

### 1.3.1Git的工作流程

一般工作流程如下：

* 克隆 Git 资源作为工作目录。
* 在克隆的资源上添加或修改文件。
* 如果其他人修改了，你可以更新资源。
* 在提交前查看修改。
* 提交修改。
* 在修改完成后，如果发现错误，可以撤回提交并再次修改并提交。



图12-17 GIT工作流程图

### 1.3.2GIT工作区、暂存区和版本库

Git 工作区、暂存区和版本库概念

* 工作区：就是你在电脑里能看到的目录。
* 暂存区：英文叫stage, 或index。一般存放在 ".git目录下" 下的index文件（.git/index）中，所以我们把暂存区有时也叫作索引（index）。
* 版本库：工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库。

下面这个图展示了工作区、版本库中的暂存区和版本库之间的关系：

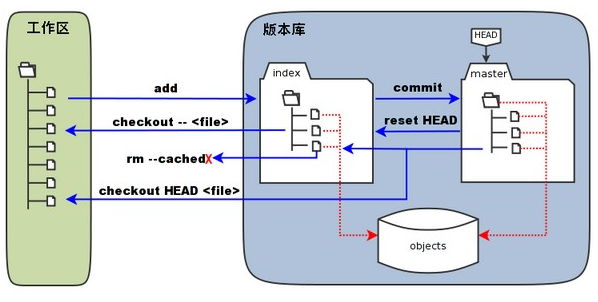


图12-18术语关系图

图中左侧为工作区，右侧为版本库。在版本库中标记为 "index" 的区域是暂存区（stage, index），标记为 "master" 的是 master 分支所代表的目录树。

图中我们可以看出此时 "HEAD" 实际是指向 master 分支的一个"游标"。所以图示的命令中出现 HEAD 的地方可以用 master 来替换。

图中的 objects 标识的区域为 Git 的对象库，实际位于 ".git/objects" 目录下，里面包含了创建的各种对象及内容。

当对工作区修改（或新增）的文件执行 "git add" 命令时，暂存区的目录树被更新，同时工作区修改（或新增）的文件内容被写入到对象库中的一个新的对象中，而该对象的ID被记录在暂存区的文件索引中。

当执行提交操作（git commit）时，暂存区的目录树写到版本库（对象库）中，master 分支会做相应的更新。即 master 指向的目录树就是提交时暂存区的目录树。

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。

当执行 "git rm --cached <file>" 命令时，会直接从暂存区删除文件，工作区则不做出改变。

当执行 "git checkout ." 或者 "git checkout -- <file>" 命令时，会用暂存区全部或指定的文件替换工作区的文件。这个操作很危险，会清除工作区中未添加到暂存区的改动。

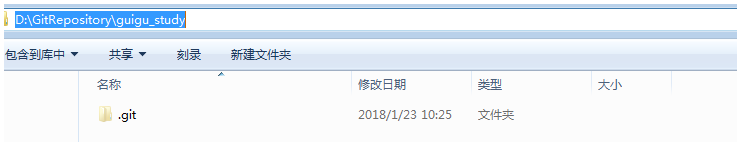
当执行 "git checkout HEAD ." 或者 "git checkout HEAD <file>" 命令时，会用 HEAD 指向的 master 分支中的全部或者部分文件替换暂存区和以及工作区中的文件。这个命令也是极具危险性的，因为不但会清除工作区中未提交的改动，也会清除暂存区中未提交的改动。

# 2GIT仓库管理

## 2.1GIT创建本地仓库

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”。

Git 使用git init命令来初始化一个 Git 仓库，Git 的很多命令都需要在 Git 的仓库中运行，所以git init是使用 Git 的第一个命令。执行完成git init命令后，Git 仓库会生成一个 .git 目录，该目录包含了资源的所有元数据，其他的项目目录保持不变（不像 SVN 会在每个子目录生成 .svn 目录，Git 只在仓库的根目录生成 .git 目录）。





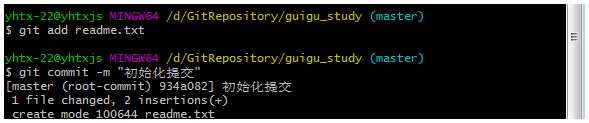
**把文件添加到版本库**

以下内容后面章节还会有详细介绍

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

1. 在版本库GitRepository目录下新建一个记事本文件 readme.txt

2、用命令 git commit告诉Git，把文件提交到仓库.

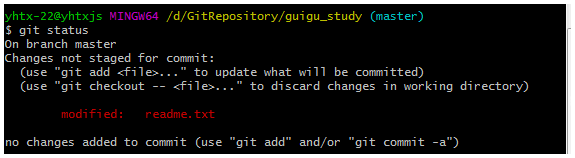


-m 代表添加注释

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

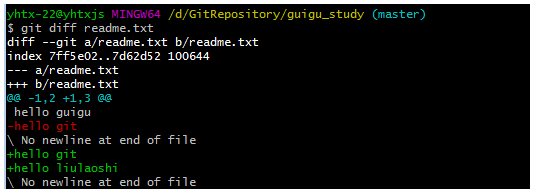


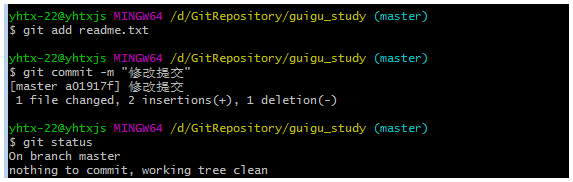
4、继续来修改readme.txt内容，比如我在下面添加一行Hello guigu内容，继续使用git status来查看下结果，如下：



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

git diff readme.txt 如下：





## 2.2GIT创建远程仓库使用Github

### 2.2.1添加远程仓库

Git是分布式版本控制系统，同一个Git仓库，可以分布到不同的机器上。怎么分布呢？最早，肯定只有一台机器有一个原始版本库，此后，别的机器可以“克隆”这个原始版本库，而且每台机器的版本库其实都是一样的，并没有主次之分。

其实一台电脑上也是可以克隆多个版本库的，只要不在同一个目录下。不过，现实生活中是不会有人这么傻的在一台电脑上搞几个远程库玩，因为一台电脑上搞几个远程库完全没有意义，而且硬盘挂了会导致所有库都挂掉。

实际情况往往是这样，找一台电脑充当服务器的角色，每天24小时开机，其他每个人都从这个“服务器”仓库克隆一份到自己的电脑上，并且各自把各自的提交推送到服务器仓库里，也从服务器仓库中拉取别人的提交。

完全可以自己搭建一台运行Git的服务器，不过现阶段，为了学Git先搭个服务器绝对是小题大作。好在这个世界上有个叫GitHub的神奇的网站，从名字就可以看出，这个网站就是提供Git仓库托管服务的，所以，只要注册一个GitHub账号，就可以免费获得Git远程仓库。

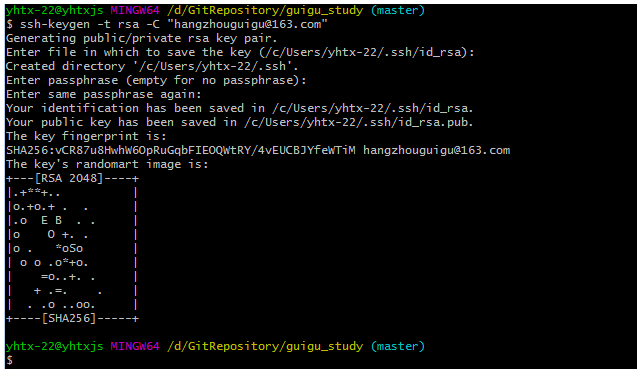
本例以Github为例作为远程仓库，如果你没有Github可以在官网**https://github.com/注册**。**由于你的本地Git仓库和GitHub仓库之间的传输是通过SSH加密的，所以我们需要配置验证信息**：

使用以下命令生成SSH Key

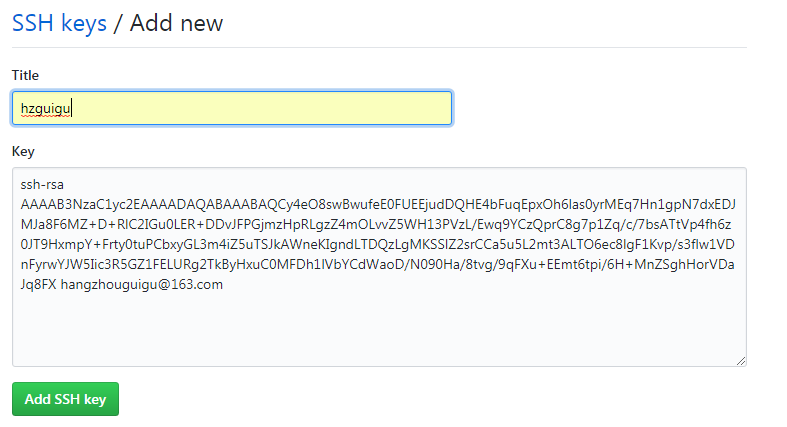
**第1步：创建SSH Key。**在用户主目录下，看看有没有.ssh目录，如果有，再看看这个目录下有没有id\_rsa和id\_rsa.pub这两个文件，如果已经有了，可直接跳到下一步。如果没有，打开Shell（Windows下打开Git Bash），创建SSH Key：

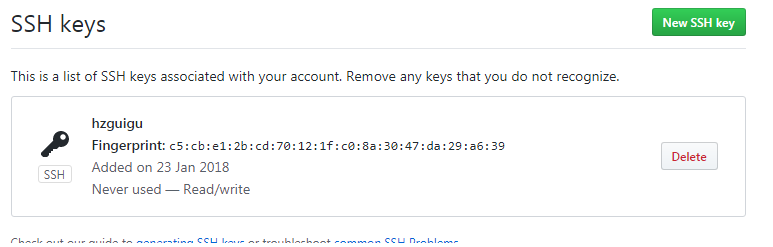
$ ssh-keygen -t rsa -C "youremail@example.com"

你需要把邮件地址换成你自己的邮件地址，然后一路回车，使用默认值即可之后会要求确认路径(默认即可)和输入密码。成功的话会在~/下生成.ssh文件夹。



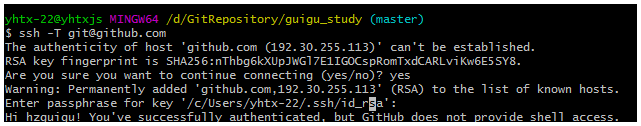
**第2步：登陆GitHub，**打开“Account settings”，“SSH Keys”页面：然后，点“New SSH Key”，填上任意Title，在Key文本框里粘贴id\_rsa.pub文件的内容：



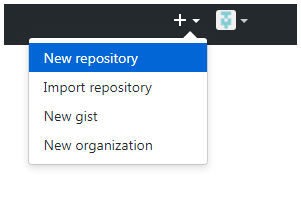


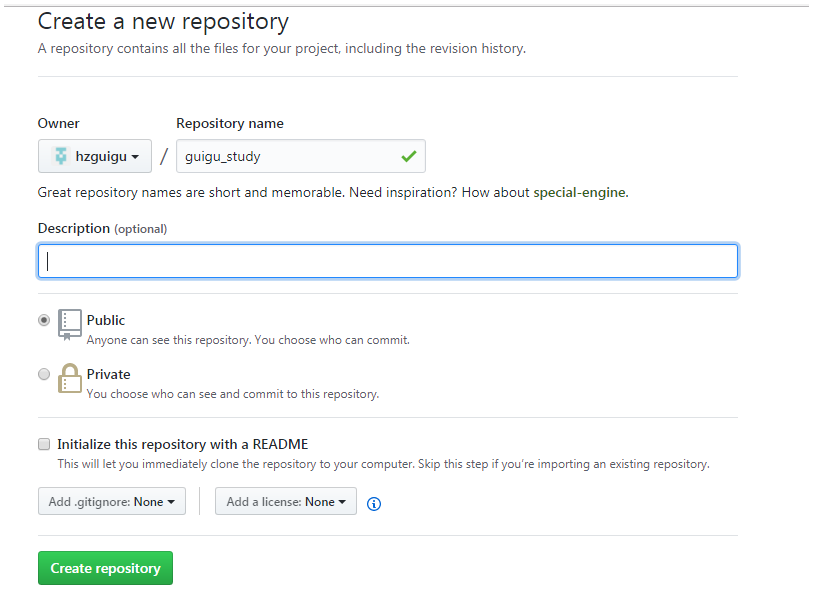
**验证是否成功为了验证是否成功，**输入以下命令：

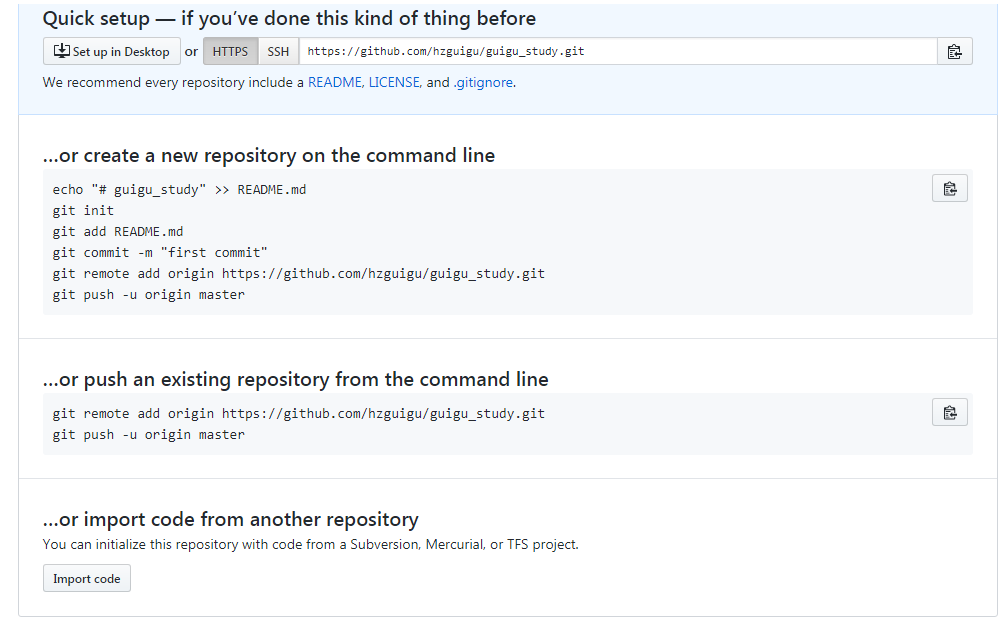
$ ssh -T [git@github.com](mailto:git@github.com)

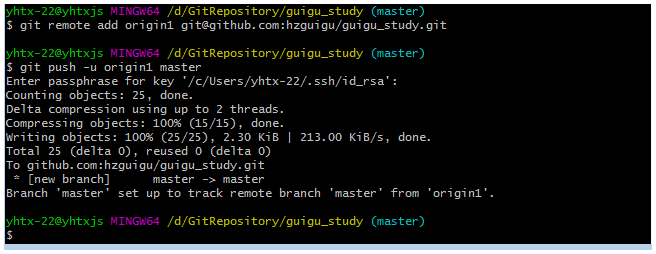


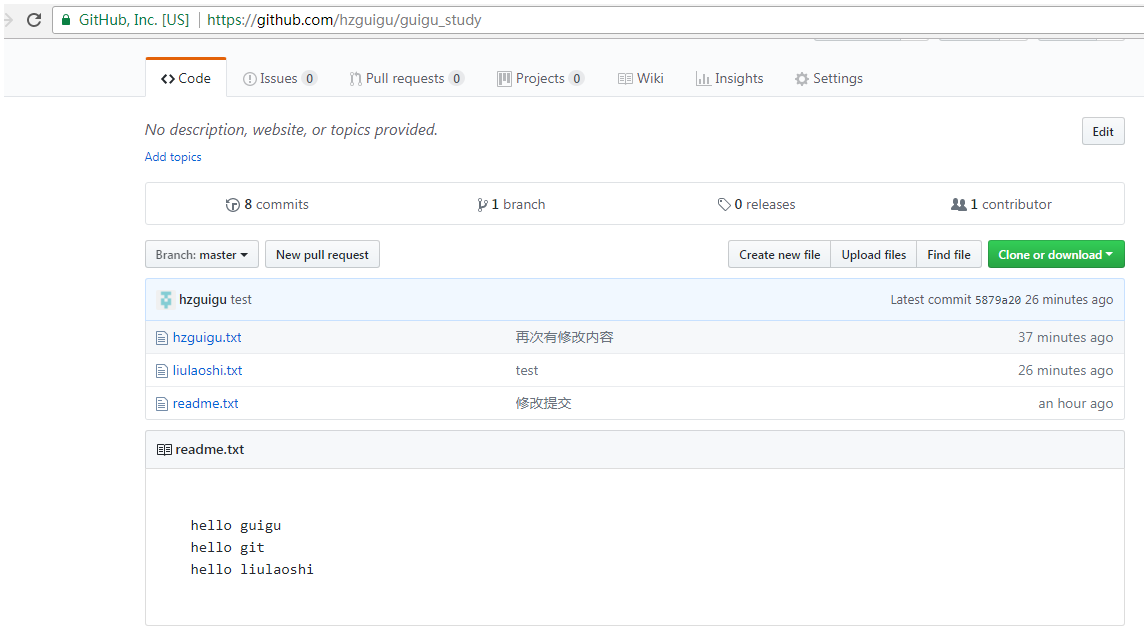
在GitHub创建仓库 使本地仓库和GitHub仓库保持一致











**SSH警告问题**

当你第一次使用Git的clone或者push命令连接GitHub时，会得到一个警告：

The authenticity of host 'github.com (xx.xx.xx.xx)' can't be established.

RSA key fingerprint is xx.xx.xx.xx.xx.

Are you sure you want to continue connecting (yes/no)?

这是因为Git使用SSH连接，而SSH连接在第一次验证GitHub服务器的Key时，需要你确认GitHub的Key的指纹信息是否真的来自GitHub的服务器，输入yes回车即可。

Git会输出一个警告，告诉你已经把GitHub的Key添加到本机的一个信任列表里了：

Warning: Permanently added 'github.com' (RSA) to the list of known hosts.

这个警告只会出现一次，后面的操作就不会有任何警告了。

如果你实在担心有人冒充GitHub服务器，输入yes前可以对照GitHub的RSA Key的指纹信息是否与SSH连接给出的一致。

分布式版本系统的最大好处之一是在本地工作完全不需要考虑远程库的存在，也就是有没有联网都可以正常工作，而SVN在没有联网的时候是拒绝干活的！当有网络的时候，再把本地提交推送一下就完成了同步，真是太方便了！

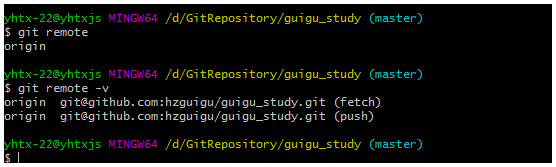
### 2.2.2对远程仓库的操作

**1、查看当前的远程库**

要查看当前配置有哪些远程仓库，可以用命令：

git remote

执行时加上 -v 参数，你还可以看到每个别名的实际链接地址。



**2、提取远程仓库**

Git 有两个命令用来提取远程仓库的更新。

1、从远程仓库下载新分支与数据：

git fetch

该命令执行完后需要执行git merge 远程分支到你所在的分支。

2、从远端仓库提取数据并尝试合并到当前分支：

git merge

该命令就是在执行 git fetch 之后紧接着执行 git merge 远程分支到你所在的任意分支。

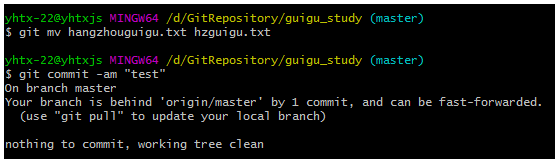
假设你配置好了一个远程仓库，并且你想要提取更新的数据，你可以首先执行 git fetch [alias] 告诉 Git 去获取它有你没有的数据，然后你可以执行 git merge [alias]/[branch] 以将服务器上的任何更新（假设有人这时候推送到服务器了）合并到你的当前分支。

接下来我们在 Github 上点击" guigu.txt" 并在线修改它:

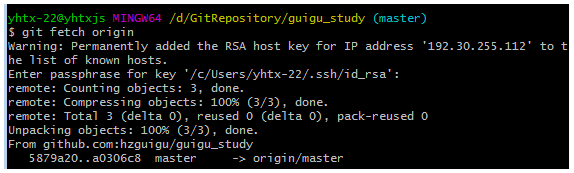
然后我们在本地更新修改。$ git fetch origin

如果本地有修改的文件没有提交， 需要先把本地的修改 提交到本地仓库 。

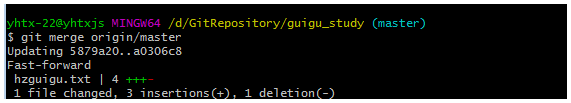
---修改文件的名字 并提交



--抓取远程的数据



--抓取远程的数据合并到本地的文件



**3、推送到远程仓库**

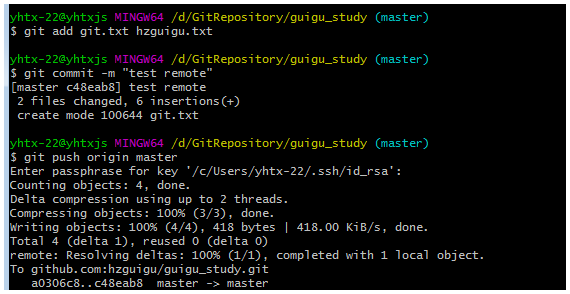
推送你的新分支与数据到某个远端仓库命令:

git push [alias] [branch]

在GitRepository\guigu-git-test目录下创建HelloGit.txt文件上传到git仓库。

新增文件git.txt 并且修改文件hzguigu.txt 然后提交到远程仓库

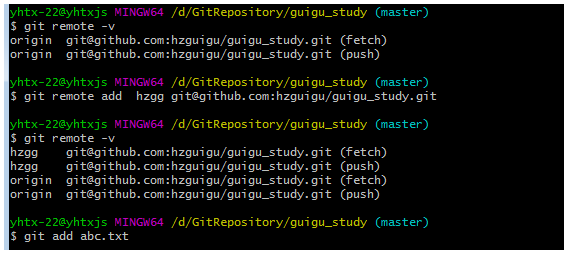
---首先把自己添加的文件 加入到本地仓库

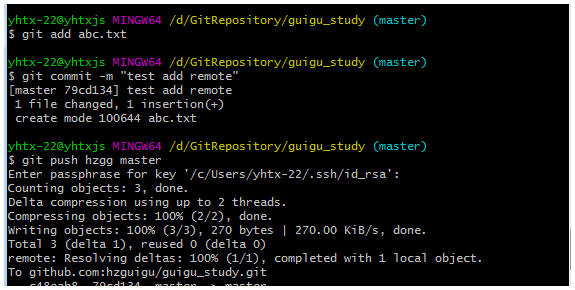


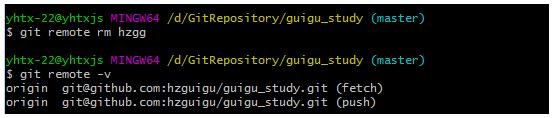
**4、删除远程仓库**

删除远程仓库你可以使用命令：

git remote rm [别名]







# 3GIT基本操作

git 的工作就是创建和保存你项目的快照及与之后的快照进行对比。本章将对有关创建与提交你的项目快照的命令作介绍。

## 3.1 git clone

我们使用git clone从现有 Git 仓库中拷贝项目（类似svn checkout）。

克隆仓库的命令格式为：

git clone <repo>

如果我们需要克隆到指定的目录，可以使用以下命令格式：

git clone <repo> <directory>

参数说明：

• repo:Git 仓库。

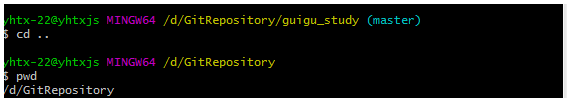
• directory:本地目录。

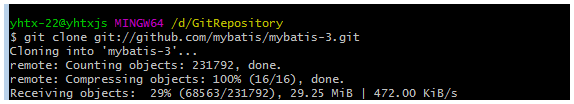
比如，要克隆 MyBatis框架的 Git 代码仓库 ，可以用下面的命令：

$ git clone git://github.com/ mybatis/mybatis-3.git执行该命令后，会在当前目录下创建一个名为mybatis-3的目录，其中包含一个 .git 的目录，用于保存下载下来的所有版本记录。

如果要自己定义要新建的项目目录名称，可以在上面的命令末尾指定新的名字：

$ git clone git://github.com/ mybatis/mybatis-3.git mybatis

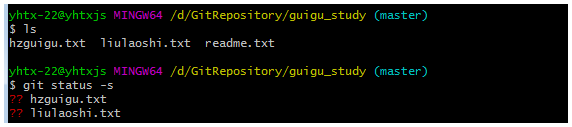




## 3.2 git add

git add 命令可将该文件添加到缓存，我们添加以下两个文件：



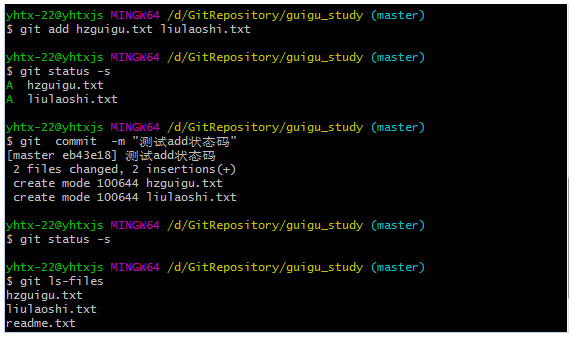


git status 命令用于查看项目的当前状态。演示该命令的时候加了 -s 参数，以获得简短的结果输出。常见的状态码如下：

* A: 你本地新增的文件（服务器上没有）.
* C: 文件的一个新拷贝.
* D: 你本地删除的文件（服务器上还在）.
* M: 文件的内容或者mode被修改了.
* R: 文件名被修改了。
* T: 文件的类型被修改了。
* U: 文件没有被合并(你需要完成合并才能进行提交)。
* X: 未知状态(很可能是遇到git的bug了，你可以向git提交bug report)。

接下来我们执行 **git add** 命令来添加文件：

查看当前哪些文件在仓库下



## 3.3 git diff

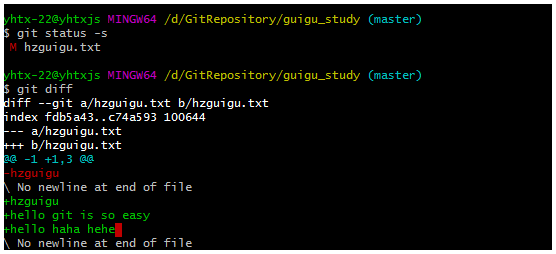
接下来修改hzguigu.txt文件的内容添加如下内容：

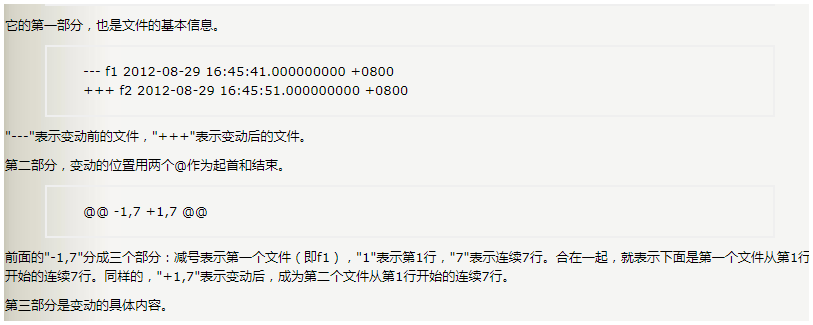
hello guigu

hello git

然后继续通过git status –s查看状态，然后执行 git diff 来查看执行 git status 的结果的详细信息。。

git diff 命令显示已写入缓存与已修改但尚未写入缓存的改动的区别。





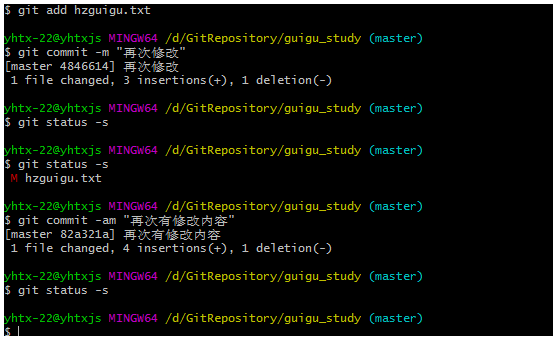
常见的diff的参数：

* 尚未缓存的改动：git diff
* 查看已缓存的改动： git diff –cached
* 查看已缓存的与未缓存的所有改动：git diff HEAD
* 显示摘要而非整个 diff：git diff –stat

## 3.4 git commit

使用 git add 命令将想要快照的内容写入缓存区，而执行 git commit 将缓存区内容添加到仓库中。

如果你觉得 git add 提交缓存的流程太过繁琐，Git 也允许你用 -a 选项跳过这一步。



## 3.5 git rm

如果只是简单地从工作目录中手工删除文件，运行 git status 时就会在 Changes not staged for commit 的提示。

要从 Git 中移除某个文件，就必须要从已跟踪文件清单中移除，然后提交。可以用以下命令完成此项工作 git rm <file>

如果删除之前修改过并且已经放到暂存区域的话，则必须要用强制删除选项 –f

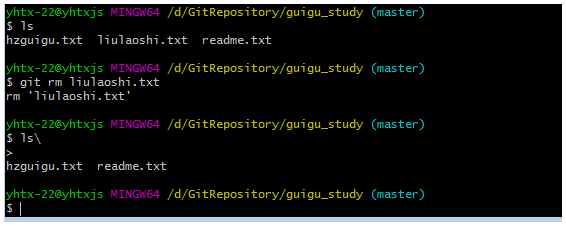
git rm –f <file>

如果把文件从暂存区域移除，但仍然希望保留在当前工作目录中，换句话说，仅是从跟踪清单中删除，使用 --cached 选项即可

git rm –cached <file>

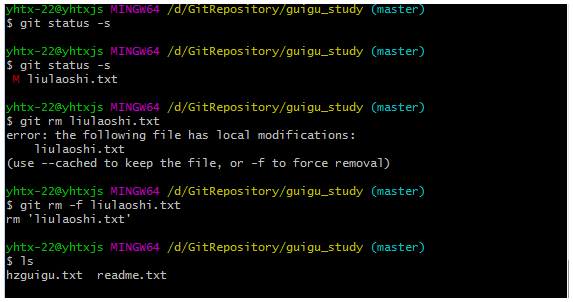
比如我们删除test.txt文件：

**直接从仓库删除文件 本地文件也一并删除**

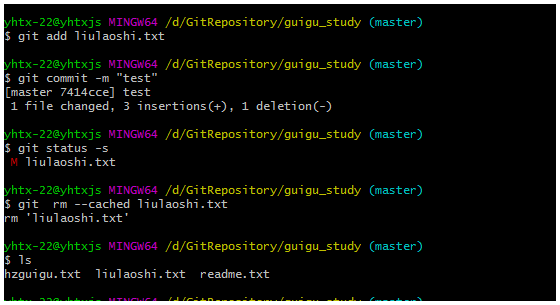


如果文件有修改 但是没有提及到版本库是无法直接删除的 需要通过-f 或者 –cache进行删除

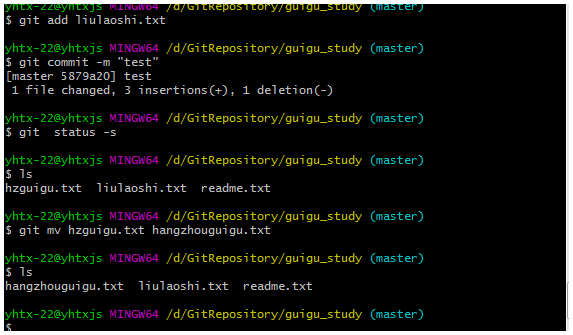
首先添加文件liulaoshi.txt 到仓库 然后再次修改内容 不提交 ，然后直接删除 是无法删除的 可以通过 –f强制删除 本地的和暂存区



首先添加文件liulaoshi.txt 到仓库 然后再次修改内容 不提交 ，然后直接删除 是无法删除的 可以通过 –cached 删除暂存区的数据 本地数据和仓库的数据依然存在



## 3.7 git mv

git mv 命令用于移动或重命名一个文件、目录、软连接。  


## 3.8 git log

在使用 Git 提交了若干更新之后，又或者克隆了某个项目，想回顾下提交历史，我们可以使用 git log 命令查看。

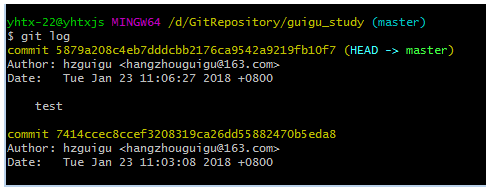
针对我们前一章节的操作，使用 git log 命令列出历史提交记录如下：

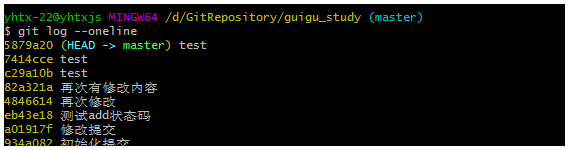
我们可以用 --oneline 选项来查看历史记录的简洁的版本。

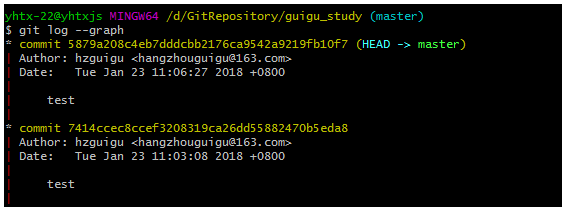
这告诉我们的是，此项目的开发历史。

我们还可以用 --graph 选项，查看历史中什么时候出现了分支、合并。以下为相同的命令，开启了拓扑图选项：

如果只想查找指定用户的提交日志可以使用命令：git log --author , 例如，比方说我们要找 Git 源码中 Linus 提交的部分：







**12.5GIT分支管理**

**12.5.1分支管理的概念**

几乎每一种版本控制系统都以某种形式支持分支。使用分支意味着你可以从开发主线上分离开来，然后在不影响主线的同时继续工作。

有人把 Git 的分支模型称为"必杀技特性"，而正是因为它，将 Git 从版本控制系统家族里区分出来。分支就是科幻电影里面的平行宇宙，当你正在电脑前努力学习Git的时候，另一个你正在另一个平行宇宙里努力学习SVN。如果两个平行宇宙互不干扰，那对现在的你也没啥影响。不过，在某个时间点，两个平行宇宙合并了，结果，你既学会了Git又学会了SVN！



图12-61分支管理的概念

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。

现在有了分支，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分支上，这样，既安全，又不影响别人工作。

其他版本控制系统如SVN等都有分支管理，但是用过之后你会发现，这些版本控制系统创建和切换分支比蜗牛还慢，简直让人无法忍受，结果分支功能成了摆设，大家都不去用。

但Git的分支是与众不同的，无论创建、切换和删除分支，Git在1秒钟之内就能完成！无论你的版本库是1个文件还是1万个文件。

**12.5.2 创建切换与合并分支**

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：



图12-62分支图解

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：



图12-63分支图解

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：



图12-64分支图解

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：



图12-65分支图解

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：



图12-66分支图解

创建分支命令：

git branch (branchname)

切换分支命令:

git checkout (branchname)

当你切换分支的时候，Git 会用该分支的最后提交的快照替换你的工作目录的内容， 所以多个分支不需要多个目录。

合并分支命令:

git merge

**实例**

首先，我们创建dev分支，然后切换到dev分支：

Git鼓励大量使用分支：

查看分支：git branch

创建分支：git branch <name>

切换分支：git checkout <name>

创建+切换分支：git checkout -b <name>

合并某分支到当前分支：git merge <name>

删除分支：git branch -d <name>

**12.5.3解决冲突**

人生不如意之事十之八九，合并分支往往也不是一帆风顺的。准备新的guigu分支，继续我们的新分支开发：

现在master分支和guigu分支各自都分别有新的提交，变成了这样：



图12-75 master分支和guigu分支的图解

Git用<<<<<<<，=======，>>>>>>>标记出不同分支的内容，我们修改如下后保存：

hello liulaoshi hen shuai

hello git is just so so

hello git is so easy

现在，master分支和guigu分支变成了下图所示



图12-77 在master分支和guigufenzhi合并后图解

用带参数的git log也可以看到分支的合并情况

# 5 GIT可视化工具

TortoiseGit软件是windows下的git图形界面工具，有了它不需要再记各种git操作指令，非常方便。

**12.6.1安装TortoiseGit**

1、双击TortoiseGit-2.5.0.0-64bit，出现如下画面：

2、选择next按钮，进入End-User License Agreement页面，如下图选择选项：

3、选择next按钮，进入Choose SSH Client页面，选择TortoisePLink选项：

4、选择next按钮，进入Custom Setup页面，使用默认或者指定路径安装；

5、选择next按钮，进入Ready to Install页面

6、选择Install按钮，开始安装；

**12.6.2使用TortoiseGit**

1、创建一个版本库

创建一个文件夹guigu-redis-study，然后右键--Git Create repository here...

2.创建一个文件，然后提交(commit)到版本库[这个操作可以在离线状态操作]

2.1选择Git Commit->"master

2.2填写message，勾选要commit的文件，然后点击OK。在弹出一个窗口点击close即可。

3.然后在HelloGit.java中添加一个新的方法addGit。当文件代码改变，那么文件会显示为一个红色感叹号如下图所示:

4.将新增的代码，提交到版本库中，参考2.1和2.2步。

5.查看日志右键TortoiseGit--->show log在日志中，文件更改的信息，都很清楚。

6.将代码推送(push)到云端，例如：github、oschina上面。

6.1以github为例讲解在github上新建项目guigu-redis-study。可参考12.4.2GIT创建远程仓库使用Github

6.2将本地的代码push到云端。右键-->tortoiseGit-->push

6.3成功后，我们去github上看看push的项目吧。

7.在TortoiseGit中使用分支

7.1新建一个分支，右键-->TortoiseGit-->Create Branch

右键，你会发现当前的分支为你新建的guigu\_newbranch分支了

7.2接下来，继续增加新功能，我们再次编写HelloGit中不仅增加了showGit方法，还修改了addGit方法

7.3提交我们的代码，此时我们发现这个新功能不行，存在很多漏洞等等问题。我们想回到之前的代码。怎么办呢？

**12.7Eclipse中使用Git**

**12.7.1安装插件Egit**

1、Eclipse官网下载相关插件或者直接通过地址更新。

http://www.eclipse.org/egit/download/

2、根据官网的步骤执行相关的配置和安装

3、在Eclipse中Help--->InstallNew Software然后拷贝刚才的地址栏到Eclipse中并回车，当地址加载完毕后，根据自己需要的特性选择安装插件，可以选择官方推荐的全部安装。

安装完成后提示重新Eclipse，点击重启即可。

**12.7.2Eclipse中使用git**

**12.7.2.1从github克隆工程到Eclipse中**

1、首先切换到git视图页面进行操作

2、在git视图页面可以选择添加一个已经存在的git项目，可以克隆一个项目，可以创建一个新的git项目。

3、在执行项目前必须先设置用户名和email信息

4、选择从github克隆一个已经存在的项目，操作如下图所示:

5、选择克隆这个git项目到本地哪个文件夹下：

6、克隆后还是一个git工程需要导入到工作区间，在当前项目下右键---->Import Projects转换为Java工程。

7、选择具体的导入到本地的位置

8、切换视图到Java视图界面，可以看到已经把github上的工程转换为本地的java工程。

**12.7.2.2从Eclipse把工程上传到GitHub中**

1、新建Java工程

2、编写部分测试代码如下图所示:

3、分享项目到GitHub上，在工程上右键---->Team--->share project---> git

4、在弹出的视窗中，先选中user or create，然后点击选中项目，再次点击下方的create repository。系统会自动在当前项目下创建Repository。

5、在项目上右键Team---->Add to Index 添加索引

6、在项目上右键Team---->Commit 提交项目

7、在提交的过程中需要添加注释和配置信息

8、在提交项目到GitHub中，选中项目右键--->Team--->Push Branch “master”

9、在Github中新建一个远程仓库SpringMVCStudy，然后得到其地址进行如下配置即可。

10.配置完毕点击下一步提示需要输入用户名和密码，这里的用户名和密码是你在github注册的用户名和密码。如图12-125所示，如果不想每次都输入密码可以把Store in secure Store勾选，勾选完毕后系统为了防止用户忘记密码需要设置两个问题提示，如图12-126所示：

11、设置完毕点击ok系统会自动把选中的项目或者内容提交到指定的github仓库中，提交完毕后可以查阅github中的内容。