## 下载安装msysgit

1、下载git-for-windows

（1）常用的GitHub客户端msysgit，也就是git-for-windows。

（2）登录官网 https://git-for-windows.github.io/，如下，单击Download按钮下载。

（3）我下载了最新版64位：Git-2.11.0-64-bit.exe

2、安装git-for-windows

**选择安装组件** :**也可以默认选择**;

-- **图标组件**(Addition icons) : 选择是否创建快速启动栏图标 或者 是否创建桌面快捷方式;

-- **桌面浏览**(Windows Explorer integration) : 浏览源码的方法, 单独的上下文浏览 只使用bash 或者 只用Git GUI工具; 高级的上下文浏览方法 使用git-cheetah plugin插件;

-- **关联配置文件** : 是否关联git配置文件, 该配置文件主要显示文本编辑器的样式;

-- **关联shell脚本文件** : 是否关联Bash命令行执行的脚本文件;

-- **使用TrueType编码** : 在命令行中是否使用TruthType编码, 该编码是微软和苹果公司制定的通用编码;

**设置环境变量** : 选择使用什么样的命令行工具, 一般情况下我们**默认使用Git Bash即可, 默认选择**;

-- **Git自带** : 使用Git自带的Git Bash命令行工具;

-- **系统自带CMD** : 使用Windows系统的命令行工具;

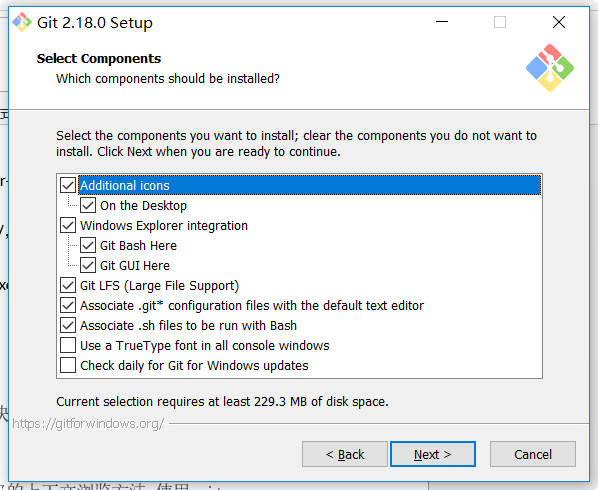
-- **二者都有** : 上面二者同时配置, 但是注意, 这样会将windows中的find.exe 和 sort.exe工具覆盖, 如果不懂这些尽量不要选择;

**选择换行格式** :

-- **检查出windows格式转换为unix格式** : 将windows格式的换行转为unix格式的换行在进行提交;

-- **检查出原来格式转为unix格式** : 不管什么格式的, 一律转为unix格式的换行在进行提交;

-- **不进行格式转换** : 不进行转换, 检查出什么, 就提交什么;



Additional icons 附加图标

​ On the Desktop 在桌面上

Windows Explorer integration Windows资源管理器集成鼠标右键菜单

​ Git Bash Here

​ Git GUI Here

Git LFS (Large File Support)

大文件支持

Associate .git\* configuration files with the default text editor

将 .git 配置文件与默认文本编辑器相关联

Associate .sh files to be run with Bash

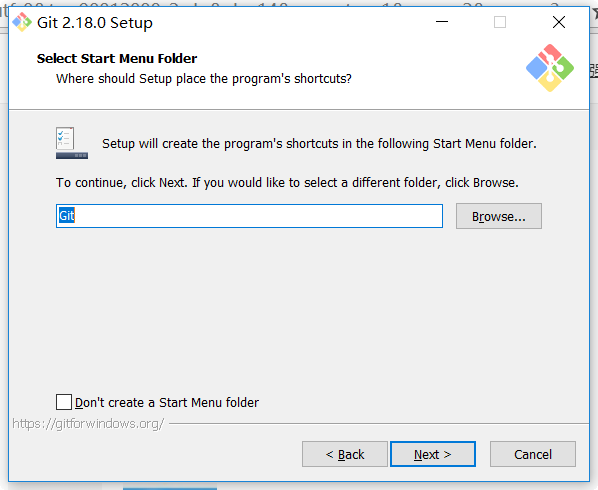
将.sh文件关联到Bash运行

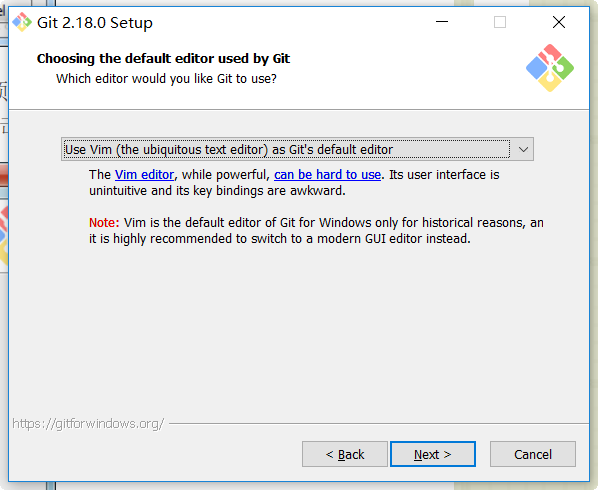
Use a TrueType font in all console windows

在所有控制台窗口中使用TrueType字体

Check daily for Git for Windows updates

每天检查Git是否有Windows更新





Use the Nano editor by default

默认使用 Nano 编辑器

Use Vim (The ubiquitous text editor) as Git's default editor

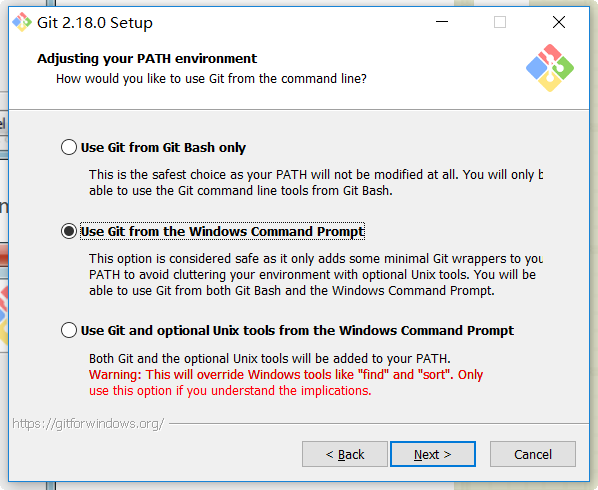
使用 Vim 作为 Git 的默认编辑器

Use Notepad++ as Git's default editor

使用 Notepad++ 作为 Git 的默认编辑器

Use Visual Studio Code as Git's default editor

使用 Visual Studio Code 作为 Git 的默认编辑器



配置PATH环境

Use Git from Git Bash only

This is the safest choice as your PATH will not be modified at all.You will only be able to use the Git command line tools form Git Bash.

这是最安全的选择，因为您的PATH根本不会被修改。您只能使用 Git Bash 的 Git 命令行工具。

Use Git from the Windows Command Prompt

This option is considered safe as it only adds some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools . You will be able to use Git from both Git Bash and the Windows Command Prompt.

这个选项被认为是安全的，因为它只向PATH添加一些最小的 Git包，以避免使用可选的Unix工具混淆环境。 您将能够从 Git Bash 和 Windows 命令提示符中使用 Git。

Use Git and optional Unix tools from the Windows Command Prompt

从Windows命令提示符使用Git和可选的Unix工具

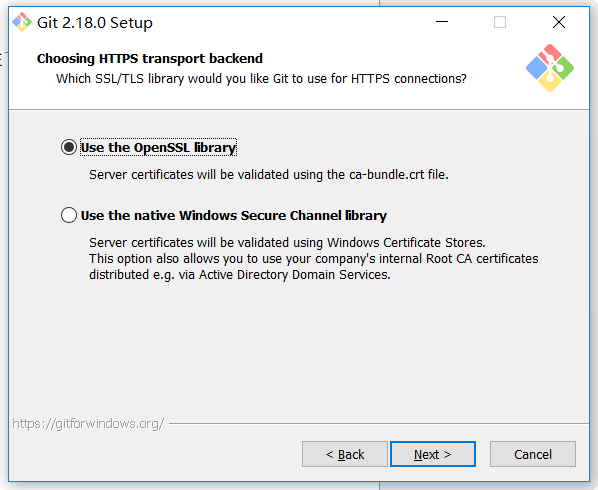
Both Git and the optional Unix tools will be added to you PATH

Git和可选的Unix工具都将添加到您计算机的 PATH 中

Warning:This will override Windows tools like "find and sort".Only use this option if you understand the implications.

警告：这将覆盖Windows工具，如 “ find 和 sort ”。只有在了解其含义后才使用此选项。

**选择HTTPS传输后端**



Use the OpenSSL library

使用 OpenSSL 库

Server certificates will be validated using the ca-bundle.crt file.

服务器证书将使用ca-bundle.crt文件进行验证。

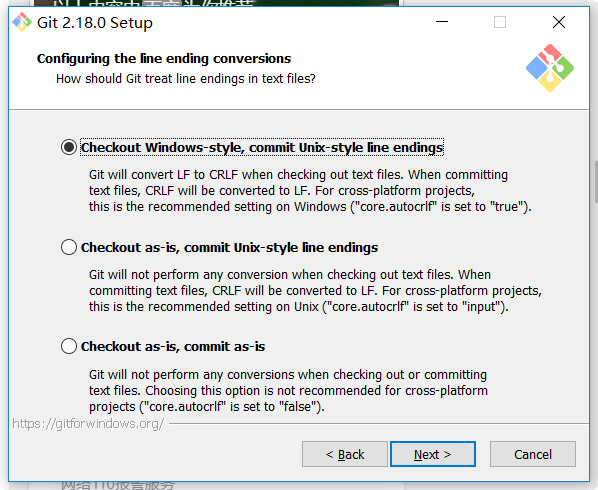
Use the native Windows Secure Channel library

使用本地 Windows 安全通道库

Server certificates will be validated using Windows Certificate Stores.This option also allows you to use your company's internal Root CA certificates distributed e.g. via Active Directory Domain Services.

服务器证书将使用Windows证书存储验证。此选项还允许您使用公司的内部根CA证书，例如， 通过Active Directory Domain Services 。

**配置行结束转换**



Checkout Windows-style,commit Unix-style line endings

Git will convert LF to CRLF when checking out text files.When committing text files,CRLF will be converted to LF .For cross-pltform projects,this is the recommended setting on Windows ("core.autocrlf" is set to "true")

在检出文本文件时，Git会将LF转换为CRLF。当提交文本文件时，CRLF将转换为LF。 对于跨平台项目，这是Windows上推荐的设置（“core.autocrlf”设置为“true”）

Checkout as-is , commit Unix-style line endings

Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects,this is the recommended setting on Unix ("core.autocrlf" is set to "input")

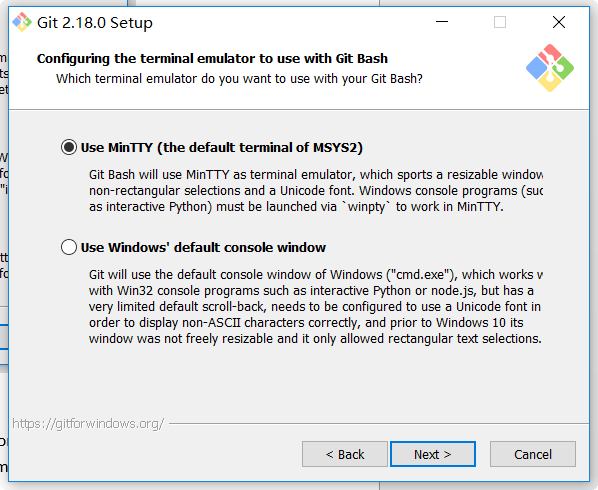
在检出文本文件时，Git不会执行任何转换。 提交文本文件时，CRLF将转换为LF。 对于跨平台项目，这是Unix上的推荐设置 （“core.autocrlf”设置为“input”）

Checkout as-is,commit as-is

Git will not perform any conversions when checking out or committing text files.Choosing this option is not recommended for cross-platform projects ("core.autocrlf"is set to "false")

在检出或提交文本文件时，Git不会执行任何转换。对于跨平台项目，不推荐使用此选项（“core.autocrlf”设置为“false”）

**配置终端模拟器以与 Git Bash 一起使用**



Use MinTTY (the default terminal of MSYS2)

Git Bash will use MinTTY as terminal emulator,which sports a resizable window,non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via 'winpty' to work in MinTTY.

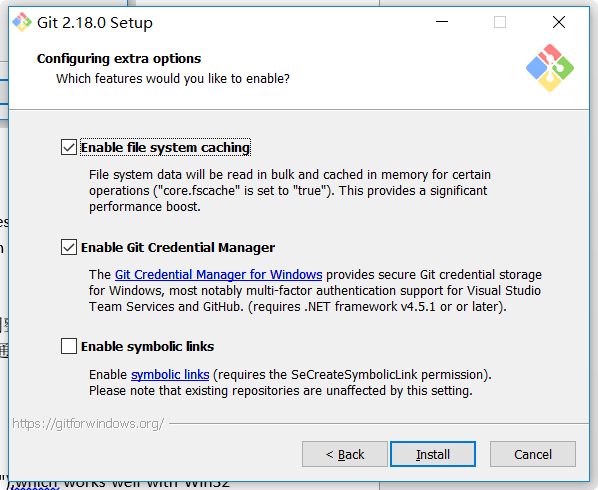
Git Bash将使用MinTTY作为终端模拟器，该模拟器具有可调整大小的窗口，非矩形选区和Unicode字体。 Windows控制台程序（如交互式Python）必须通过'winpty'启动才能在MinTTY中运行。

Use Windows' default console window

Git will use the default console window of Windows ("cmd.exe"),which works well with Win32 console programs such as interactive Python or node.js , but has a very limited default scroll-back,needs to be configured to use aUnicode font in order to display non-ASCII characters correctly,and prior to Windows 10 its windows was not freely resizable and it only allowed rectangular text selections.

Git将使用Windows的默认控制台窗口（“cmd.exe”），该窗口可以与Win32控制台程序（如交互式Python或node.js）一起使用，但默认的回滚非常有限，需要配置为使用unicode 字体以正确显示非ASCII字符，并且在Windows 10之前，其窗口不能自由调整大小，并且只允许矩形文本选择。

**配置额外的选项**



Enable file system caching

启用文件系统缓存

File system data will be read in bulk and cached in memory for certain operations ("core.fscache" is set to "true"). This provides a significant performance boost.

文件系统数据将被批量读取并缓存在内存中用于某些操作（“core.fscache”设置为“true”）。 这提供了显着的性能提升。

Enable Git Credential Manager

启用Git凭证管理器

The Git Credential Manager for Windows provides secure Git credential storage for Windows,most notably multi-factor authentication support for Visual Studio Team Services and GitHub. (requires .NET framework v4.5.1 or or later).

Windows的Git凭证管理器为Windows提供安全的Git凭证存储，最显着的是对Visual Studio Team Services和GitHub的多因素身份验证支持。 （需要.NET Framework v4.5.1或更高版本）。

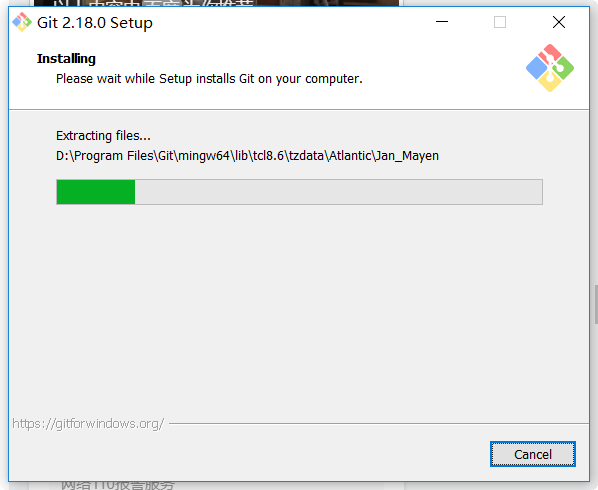
Enable symbolic links

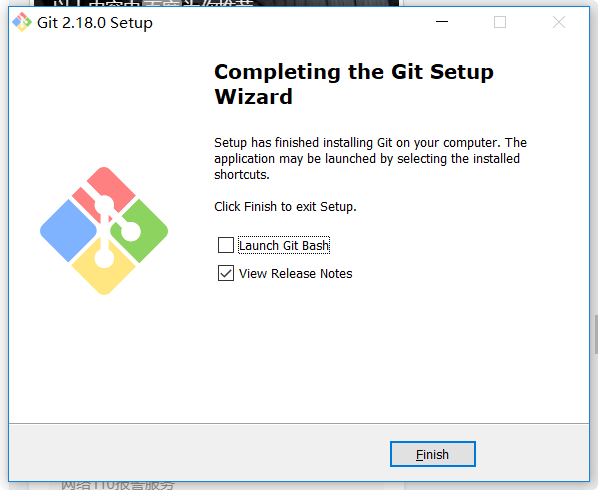
启用符号链接

Enable symbolic links (requires the SeCreateSymbolicLink permission).Please note that existing repositories are unaffected by this setting.

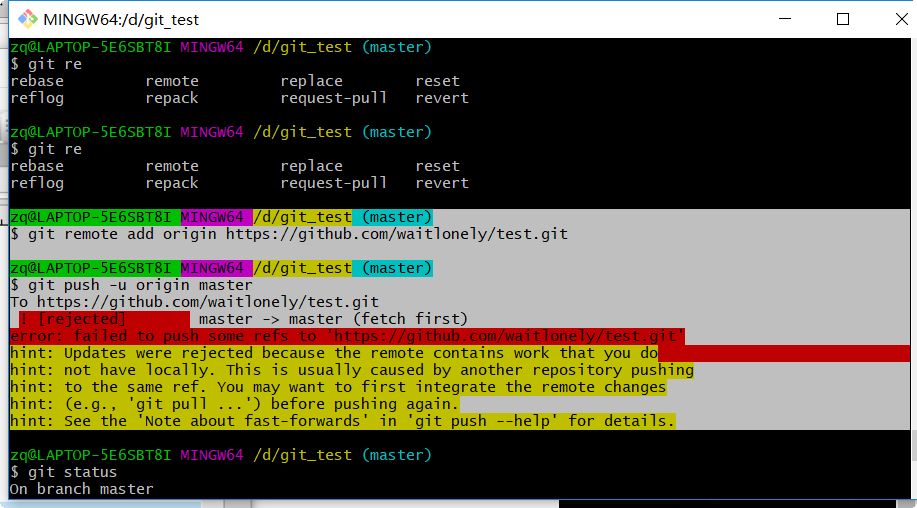
启用符号链接（需要SeCreateSymbolicLink权限）。请注意，现有存储库不受此设置的影响

Installing





## 经历的报错



zq@LAPTOP-5E6SBT8I MINGW64 /d/git\_test (master)

$ git remote add origin https://github.com/waitlonely/test.git

zq@LAPTOP-5E6SBT8I MINGW64 /d/git\_test (master)

$ git push -u origin master

To https://github.com/waitlonely/test.git

! [rejected] master -> master (fetch first)

error: failed to push some refs to 'https://github.com/waitlonely/test.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.

正确步骤：

1. git init //初始化仓库

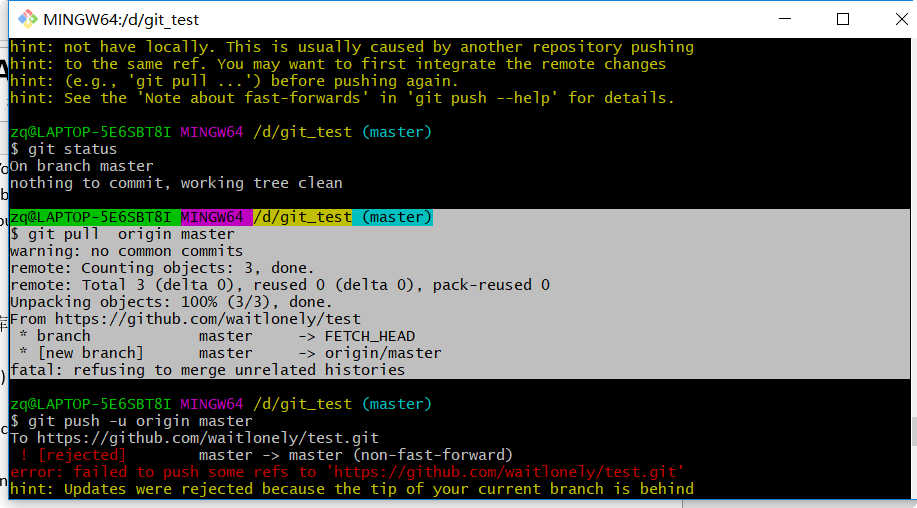
2. git add .(文件name) //添加文件到本地仓库

3. git commit -m "first commit" //添加文件描述信息

4. git remote add origin + 远程仓库地址 //链接远程仓库，创建主分支

5. git pull origin master // 把本地仓库的变化连接到远程仓库主分支

6. git push -u origin master //把本地仓库的文件推送到远程仓库



zq@LAPTOP-5E6SBT8I MINGW64 /d/git\_test (master)

$ git pull origin master

warning: no common commits

remote: Counting objects: 3, done.

remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0

Unpacking objects: 100% (3/3), done.

From https://github.com/waitlonely/test

\* branch master -> FETCH\_HEAD

\* [new branch] master -> origin/master

fatal: refusing to merge unrelated histories

先pull，因为两个仓库不同，发现refusing to merge unrelated histories，无法pull

因为他们是两个不同的项目，要把两个不同的项目合并，git需要添加一句代码，在git pull，

这句代码是在git 2.9.2版本发生的，最新的版本需要添加--allow-unrelated-histories

git pull origin master --allow-unrelated-histories

## 第一次使用的步骤

理解工作区与暂存区的区别？

工作区：就是你在电脑上看到的目录，比如目录下testgit里的文件(.git隐藏目录版本库除外)。或者以后需要再新建的目录文件等等都属于工作区范畴。

　　版本库(Repository)：工作区有一个隐藏目录.git,这个不属于工作区，这是版本库。其中版本库里面存了很多东西，其中最重要的就是stage(暂存区)，还有Git为我们自动创建了第一个分支master,以及指向master的一个指针HEAD。

　　使用Git提交文件到版本库有两步：

　　第一步：是使用 git add 把文件添加进去，实际上就是把文件添加到暂存区。

第二步：使用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支上。

### 一：创建版本库

1.初始化本地仓库

什么是版本库？版本库又名仓库，英文名repository,你可以简单的理解一个目录，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还可以将文件”还原”。

选择一个目录当做本地仓库，在当前目录下执行git init，目录下会多了一个.git的目录，这个目录是Git来跟踪管理版本的，没事千万不要手动乱改这个目录里面的文件，否则，会把git仓库给破坏了。

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。

2. 把文件添加到版本库中

在版本库testgit目录下新建一个记事本文件 readme.txt 内容如下：11111111

第一步：使用命令 git add readme.txt添加到暂存区里面去。没有任何提示，说明已经添加成功了。

第二步：用命令 git commit告诉Git，git commit –m ‘注释’，把文件提交到仓库。可以通过命令git status来查看是否还有文件未提交。使用git diff readme.txt命令可以查看文件修改了什么内容。

3.版本回退

git log命令显示从最近到最远的显示日志，如果嫌上面显示的信息太多的话，我们可以使用命令 git log --pretty=oneline。

**现在我想使用版本回退操作，我想把当前的版本回退到上一个版本，要使用什么命令呢？可以使用如下2种命令：**

**第一种是：git reset --hard HEAD^ 那么如果要回退到上上个版本只需把HEAD^ 改成 HEAD^^ 以此类推。那如果要回退到前100个版本的话，使用上面的方法肯定不方便，我们可以使用下面的简便命令操作：git reset --hard HEAD~100 即可。**

**第二种：**我们可以通过版本号回退，使用命令方法如下：

git reset --hard 版本号 ，但是现在的问题假如我已经关掉过一次命令行或者333内容的版本号我并不知道呢？要如何知道增加3333内容的版本号呢？可以通过如下命令即可获取到版本号：git reflog

4.删除文件

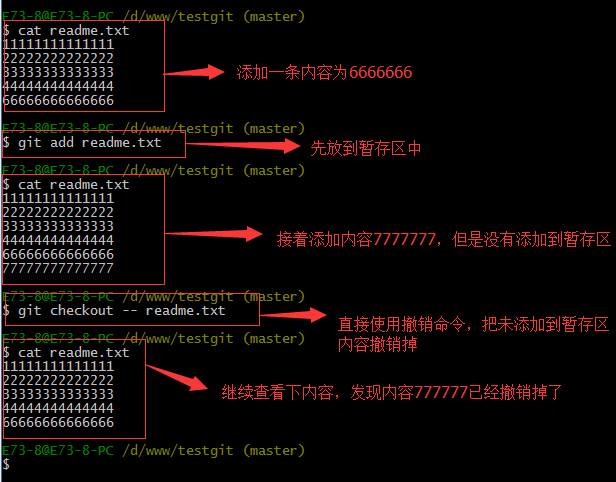
一般情况下，可以直接在文件目录中把文件删了，或者使用如上rm命令：rm b.txt ，如果我想彻底从版本库中删掉了此文件的话，可以再执行commit命令 提交掉，现在目录是这样的。

5. Git撤销修改

命令 git checkout -- readme.txt 意思就是，把readme.txt文件在工作区做的修改全部撤销，这里有2种情况，如下：

readme.txt自动修改后，还没有放到暂存区，使用 撤销修改就回到和版本库一模一样的状态。

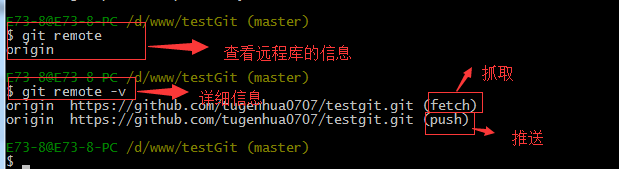
另外一种是readme.txt已经放入暂存区了，接着又作了修改，撤销修改就回到添加暂存区后的状态。



s

### 二：连接远程仓库

**当你从远程库克隆时候，实际上Git自动把本地的master分支和远程的master分支对应起来了，并且远程库的默认名称是origin。要查看远程库的信息 使用 git remote，要查看远程库的详细信息 使用 git remote –v**



本地Git仓库和github仓库之间的传输是通过SSH加密的，所以需要一点设置

第一步：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，如果有的话，直接跳过此如下命令，如果没有的话，打开命令行，输入如下命令： ssh-keygen -t rsa–C [youremail@example.com](mailto:youremail@example.com)

一路按回车健就可以不用输用户名和密码。id\_rsa是私钥，不能泄露出去，id\_rsa.pub是公钥，可以放心地告诉任何人。

第二步：登录github,打开” settings”中的SSH Keys页面，然后点击“Add SSH Key”,填上任意title，在Key文本框里黏贴id\_rsa.pub文件的内容。

在GitHub上Create repository，　目前，在GitHub上的这个testgit仓库还是空的，GitHub告诉我们，可以从这个仓库克隆出新的仓库，也可以把一个已有的本地仓库与之关联，然后，把本地仓库的内容推送到GitHub仓库。

　　现在，我们根据GitHub的提示，在本地的testgit仓库下运行命令：

git remote add origin [git@github.com:waitlonely/test.git](mailto:git@github.com:waitlonely/test.git)

git push -u origin master

**如果修改远程库url：**

git remote -v 查看现有远程仓库的地址url

三种方式都可以，这里介绍两种

1. 修改命令

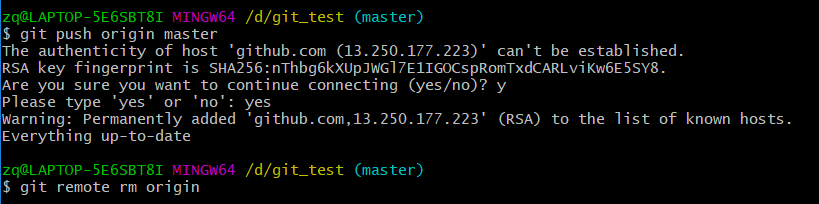
git remote set-url origin <URL> 更换远程仓库地址。把<URL>更换为新的url地址。

2.先删后加

git remote rm origin

git remote add origin [git@github.com:Liutos/foobar.git](mailto:git@github.com:Liutos/foobar.git)

3. 直接修改config文件



　　由于远程库是空的，我们第一次推送master分支时，加上了 –u参数，Git不但会把本地的master分支内容推送的远程新的master分支，还会把本地的master分支和远程的master分支关联起来，在以后的推送或者拉取时就可以简化命令。推送成功后，可以立刻在github页面中看到远程库的内容已经和本地一模一样了，上面的要输入github的用户名和密码如下所示：

正确步骤：

1. git init //初始化仓库

2. git add .(文件name) //添加文件到本地仓库

3. git commit -m "first commit" //添加文件描述信息

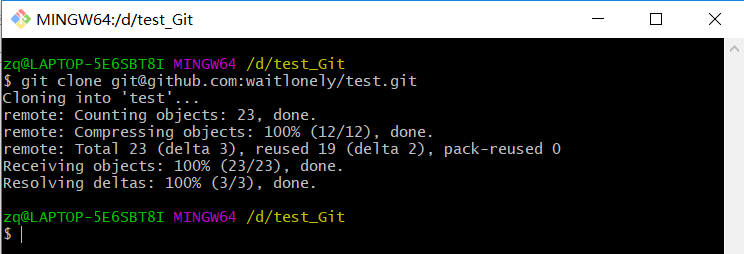
4. git remote add origin + 远程仓库地址 //链接远程仓库，创建主分支

5. git pull origin master // 把本地仓库的变化连接到远程仓库主分支

6. git push -u origin master //把本地仓库的文件推送到远程仓库

上面我们了解了先有本地库，后有远程库时候，如何关联远程库。现在我们想，假如远程库有新的内容了，我想克隆到本地来

git clone [git@github.com:waitlonely/test.git](mailto:git@github.com:waitlonely/test.git)



### 三：创建与合并分支

分支策略：首先master主分支应该是非常稳定的，也就是用来发布新版本，一般情况下不允许在上面干活，干活一般情况下在新建的dev分支上干活，干完后，比如上要发布，或者说dev分支代码稳定后可以合并到主分支master上来。

总结创建与合并分支命令如下：

查看分支：git branch

　　创建分支：git branch name

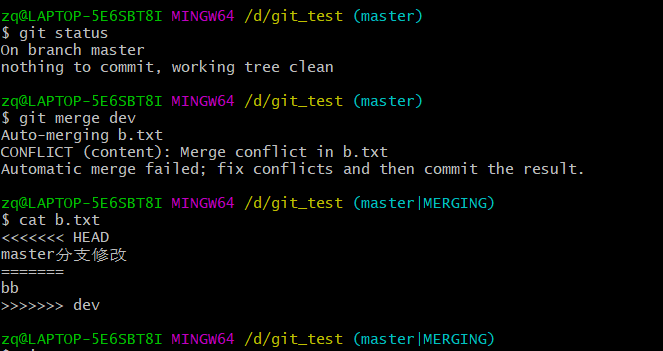
　　切换分支：git checkout name

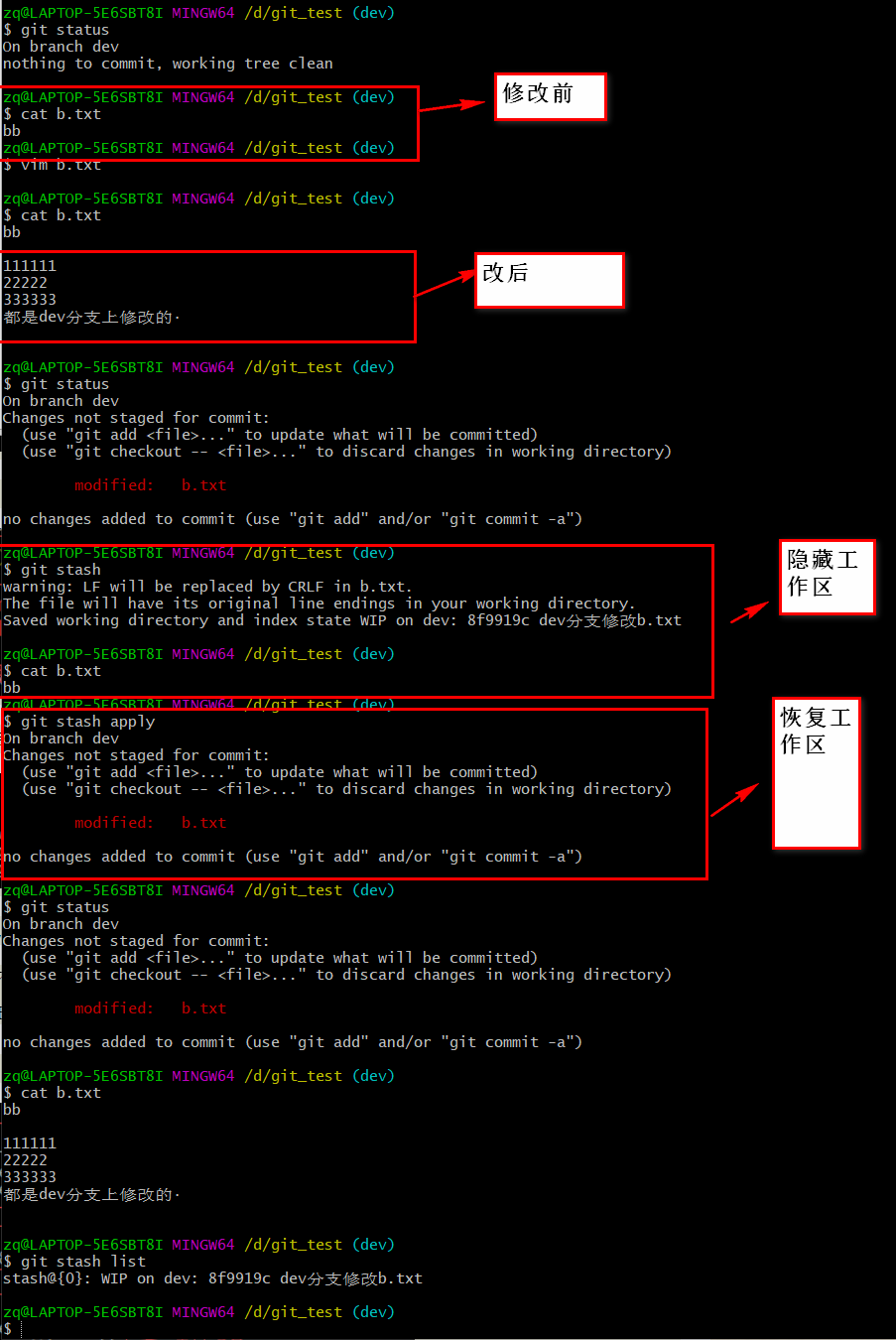
　　创建+切换分支：git checkout –b name

　　合并某分支到当前分支：git merge name

　　删除分支：git branch –d name

解决冲突





工作区是干净的，那么我们工作现场去哪里呢？我们可以使用命令 git stash list来查看

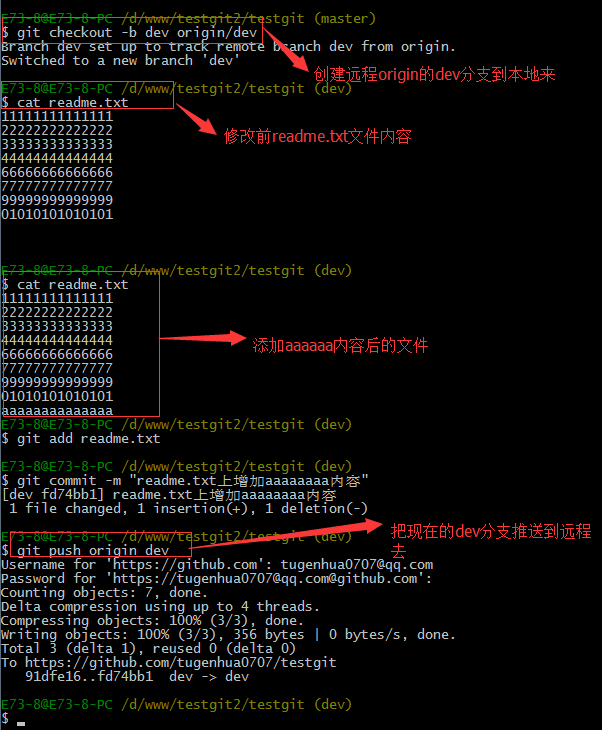
工作现场还在，Git把stash内容存在某个地方了，但是需要恢复一下，可以使用如下2个方法：

git stash apply恢复，恢复后，stash内容并不删除，你需要使用命令git stash drop来删除。

另一种方式是使用git stash pop,恢复的同时把stash内容也删除了。。

　　现在我们的小伙伴要在dev分支上做开发，就必须把远程的origin的dev分支到本地来，于是可以使用命令创建本地dev分支：git checkout –b dev origin/dev

　　现在小伙伴们就可以在dev分支上做开发了，开发完成后把dev分支推送到远程库时。如下：



多人协作工作模式一般是这样的：

首先，可以试图用git push origin branch-name推送自己的修改。如果推送失败，则因为远程分支比你的本地更新早，需要先用git pull试图合并。如果合并有冲突，则需要解决冲突，并在本地提交。再用git push origin branch-name推送。

## 常用命令

　　Git基本常用命令如下：

　　mkdir： XX (创建一个空目录 XX指目录名)

　　pwd： 显示当前目录的路径。

　　git init 把当前的目录变成可以管理的git仓库，生成隐藏.git文件。

　　git add XX 把xx文件添加到暂存区去。

　　git commit –m “XX” 提交文件 –m 后面的是注释。

　　git status 查看仓库状态

　　git diff XX 查看XX文件修改了那些内容

　　git log 查看历史记录

　　git reset --hard HEAD^ 或者 git reset --hard HEAD~ 回退到上一个版本

　　(如果想回退到100个版本，使用git reset –hard HEAD~100 )

　　cat XX 查看XX文件内容

　　git reflog 查看历史记录的版本号id

　　git checkout -- XX 把XX文件在工作区的修改全部撤销。

　　git rm XX 删除XX文件

　　git remote add origin https://github.com/tugenhua0707/testgit 关联一个远程库

　　git push –u(第一次要用-u 以后不需要) origin master 把当前master分支推送到远程库

　　git clone https://github.com/tugenhua0707/testgit 从远程库中克隆

　　git checkout –b dev 创建dev分支 并切换到dev分支上

　　git branch 查看当前所有的分支

　　git checkout master 切换回master分支

　　git merge dev 在当前的分支上合并dev分支

　　git branch –d dev 删除dev分支

　　git branch name 创建分支

　　git stash 把当前的工作隐藏起来 等以后恢复现场后继续工作

　　git stash list 查看所有被隐藏的文件列表

　　git stash apply 恢复被隐藏的文件，但是内容不删除

　　git stash drop 删除文件

　　git stash pop 恢复文件的同时 也删除文件

　　git remote 查看远程库的信息

　　git remote –v 查看远程库的详细信息

　　git push origin master Git会把master分支推送到远程库对应的远程分支上