

Git

Git

Linux(CentOS)环境下载&安装Git

Windows环境下载&安装Git

下载

安装

Git基本命令

设置用户属性

创建版本库

添加到版本库

提交到版本库

查看版本库当前状态

查看文件内容具体的区别

再次查看版本库状态

查看提交历史记录

简化显示

版本回退

回退到上一个版本

查看回退后的文件内容

回退到指定版本

查看命令历史记录

工作区&暂存区

工作区 (Working Directory)

版本库 (Repository)

详解

Git撤销修改&删除文件

撤销修改

撤销工作区修改

撤销暂存区修改

删除文件

远程仓库

创建SSH Key

设置新 SSH key

新建远程仓库

关联远程仓库

Github推荐的关联方式

1. 在命令行上创建一个新的存储库

2. 从命令行推送现有存储库

关联到Github

克隆远程仓库

分支管理

创建分支与合并分支

解决冲突

模拟场景

分支管理策略

Stash存储修改

保存修改

查看所有储存修改

恢复修改

删除修改

恢复并删除修改

清除所有修改

复制一个特定的提交到当前分支

- 特征(Feature)分支
 - 丢弃未合并过的分支
- 多人协作
 - 查看远程库信息
 - 推送分支
 - 拷贝仓库
 - 抓取仓库
 - 创建远程仓库的分支到本地
 - 多人协作模式
 - Rebase
- 标签管理
 - 创建标签
 - 查看标签信息
 - 操作标签
- 自定义Git
 - 修改UI
 - 配置命令别名
- .gitignore文件

Linux(CentOS)环境下载&安装Git

安装git:

```
yum install git
```

查看yum源仓库Git信息

```
yum info git
```

安装依赖库

```
yum install curl-devel expat-devel gettext-devel openssl-devel zlib-devel  
yum install gcc-c++ perl-ExtUtils-MakeMaker
```

查看git版本，如果默认安装的版本过低，移除默认安装的git

查看版本信息

```
git --version
```

移除默认安装的git

```
yum remove git
```

Windows环境下载&安装Git

下载

<https://git-scm.com/download/win>

Downloading Git



Your download is starting...

You are downloading the latest (2.28.0) 64-bit version of Git for Windows. This is the most recent **maintained build**. It was released **about 1 month ago**, on 2020-07-28.

[Click here to download manually](#), if your download hasn't started.

Other Git for Windows downloads

Git for Windows Setup

[32-bit Git for Windows Setup.](#)

[64-bit Git for Windows Setup.](#)

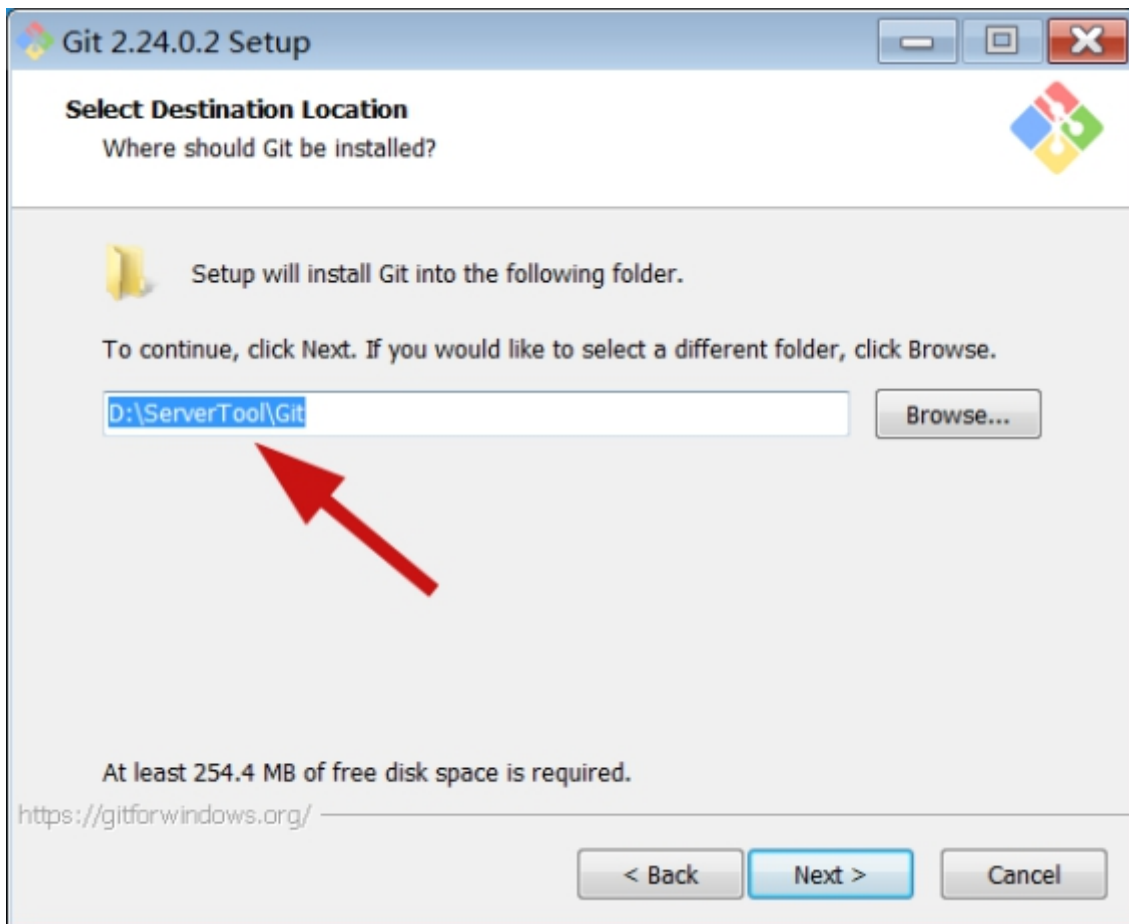
安装

双击安装程序

01、使用许可声明

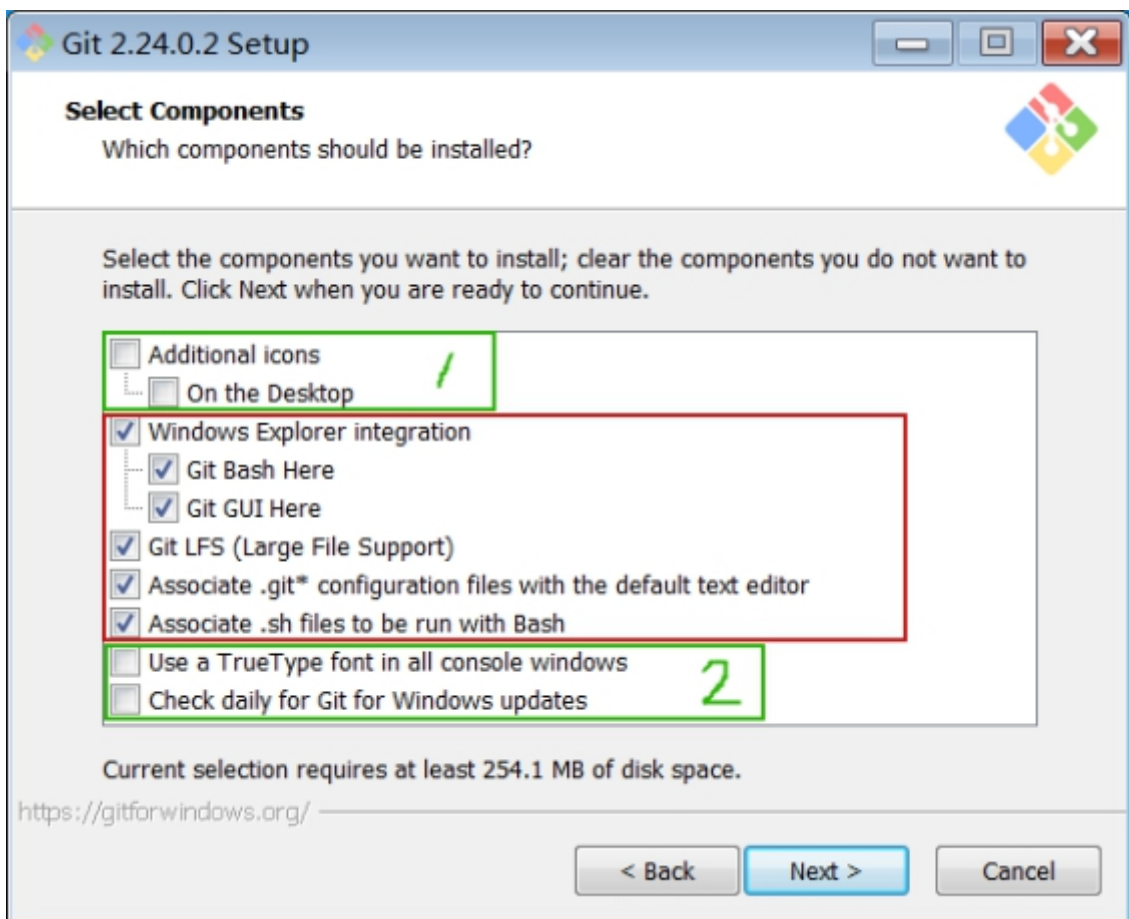


02、选择安装路径



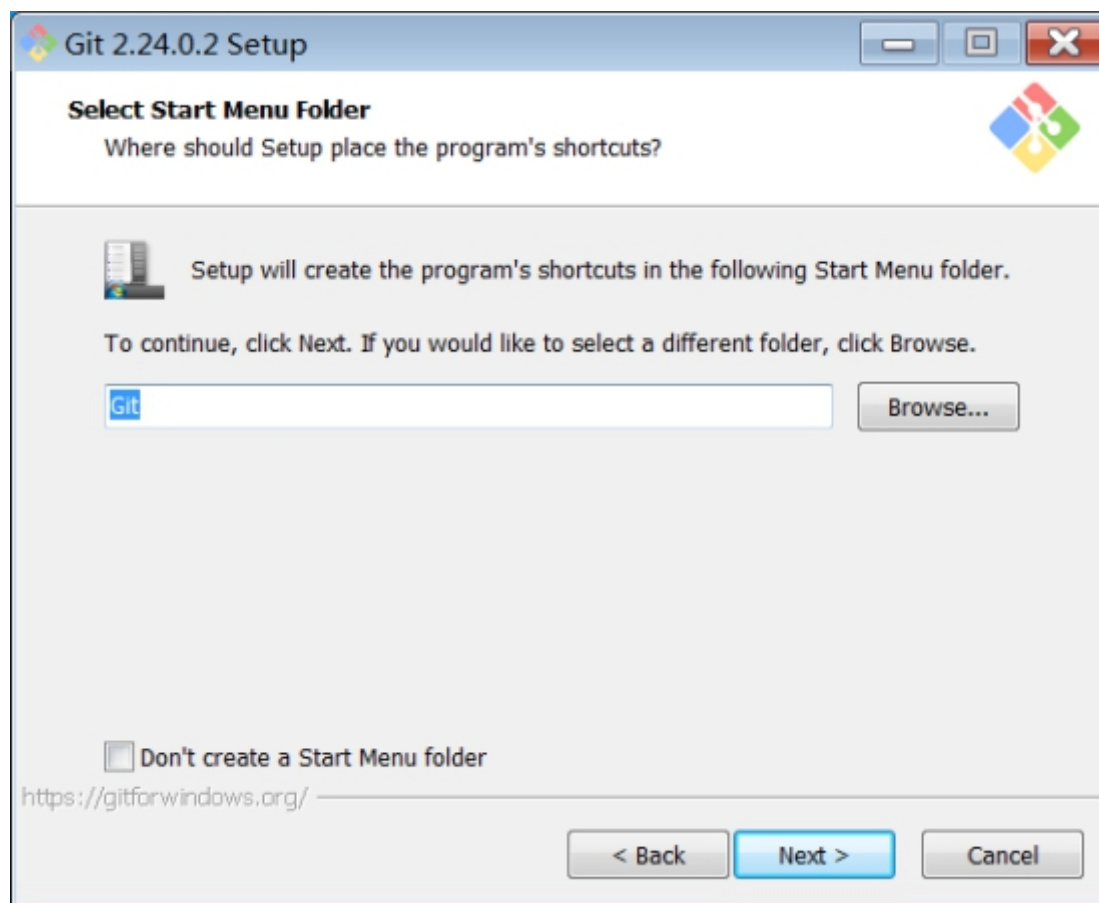
在输入框内输入想要安装到的本机路径，也就是实际文件夹位置，或点击“Browse...”选择已经存在的文件夹，然后点击“Next”按钮继续

03. 选择安装组件



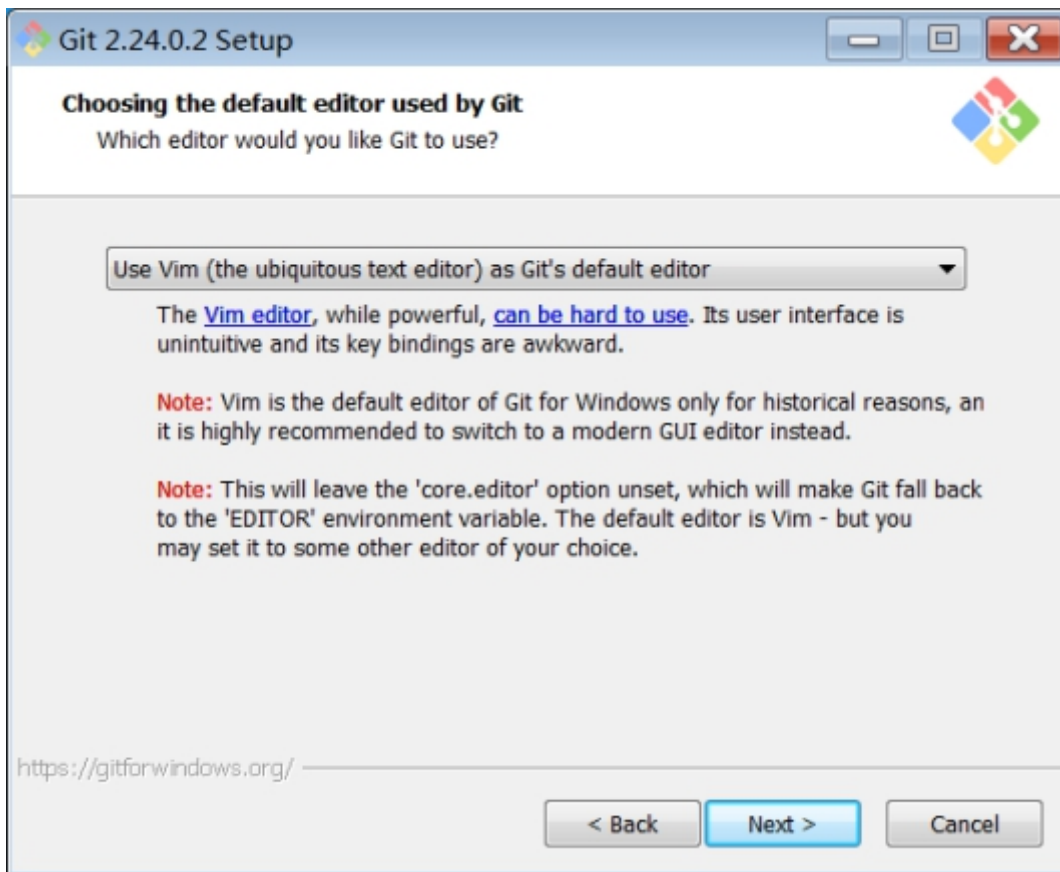
上图红框内的选项是默认勾选的，建议不要动。绿色框1是决定是否在桌面创建快捷方式的。绿色框2是决定在所有控制台窗口中使用TrueType字体和是否每天检查Git是否有Windows更新的。这些根据自己需要选择。

04、选择开始菜单页



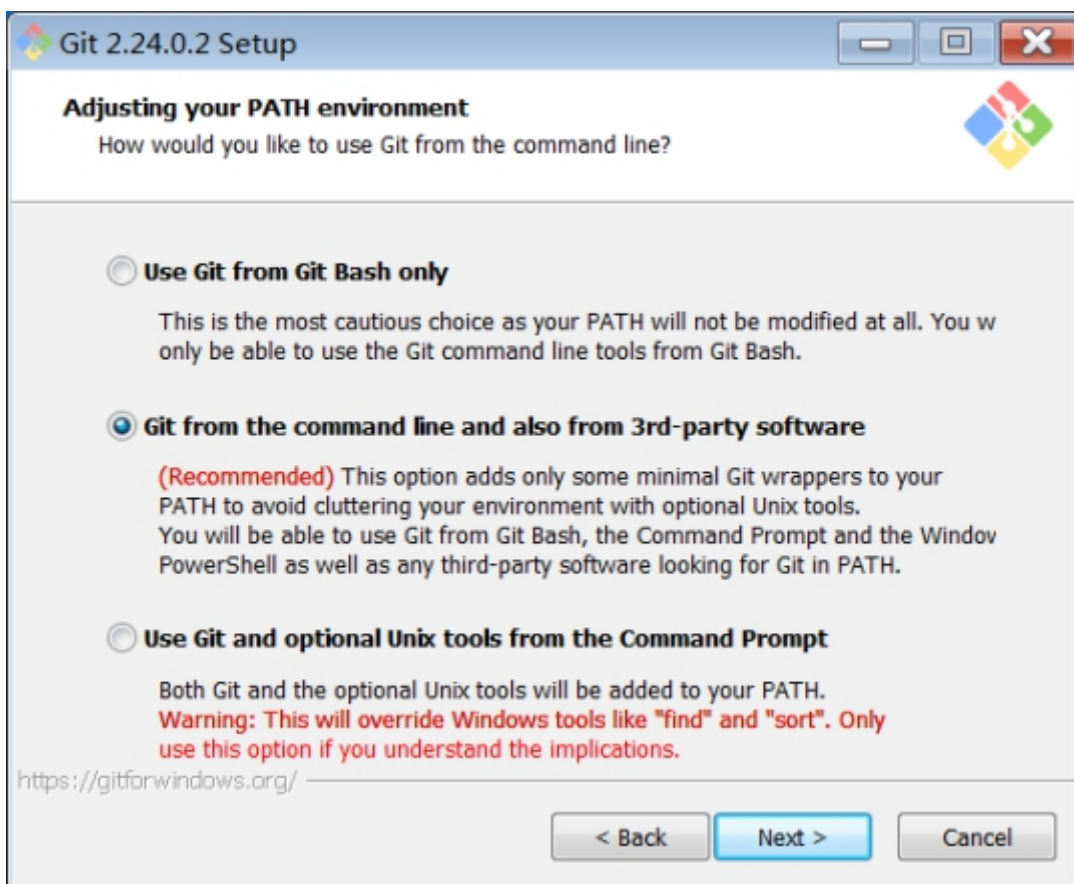
这个界面是创建开始菜单中的名称，不需要修改，直接点“Next”

05、选择Git文件默认的编辑器



这个页面是在选择Git文件默认的编辑器，很少用到，所以默认Vim即可

06. 调整您的PATH环境



这个界面是调整您的PATH环境。

第一种配置是“仅从Git Bash使用Git”。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。但是这将不能通过第三方软件使用。

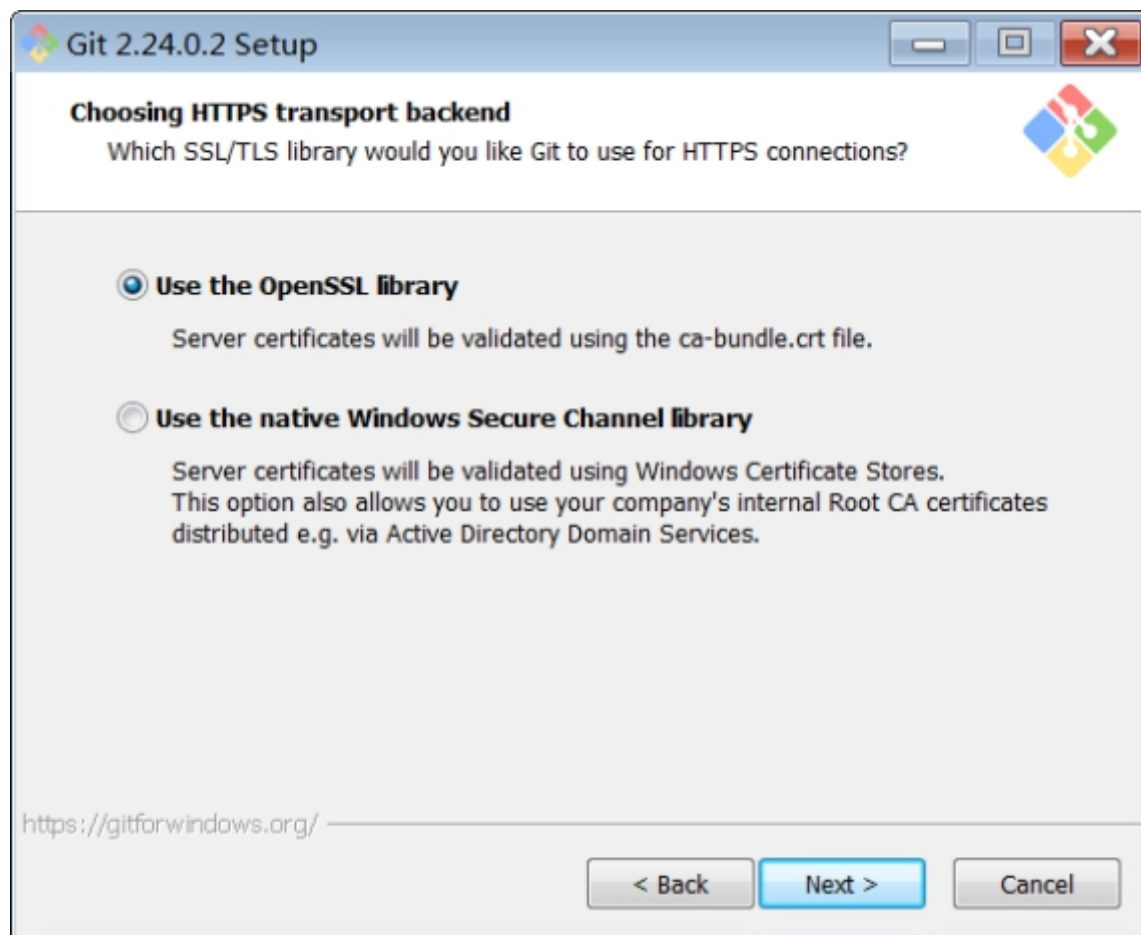
第二种配置是“从命令行以及第三方软件进行Git”。该选项被认为是安全的，因为它仅向PATH添加了一些最小的Git包装器，以避免使用可选的Unix工具造成环境混乱。

您将能够从Git Bash，命令提示符和Windows PowerShell以及在PATH中寻找Git的任何第三方软件中使用Git。这也是推荐的选项。

第三种配置是“从命令提示符使用Git和可选的Unix工具”。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。

我选择推荐的选项第二种配置，点击“Next”

07、选择HTTPS后端传输



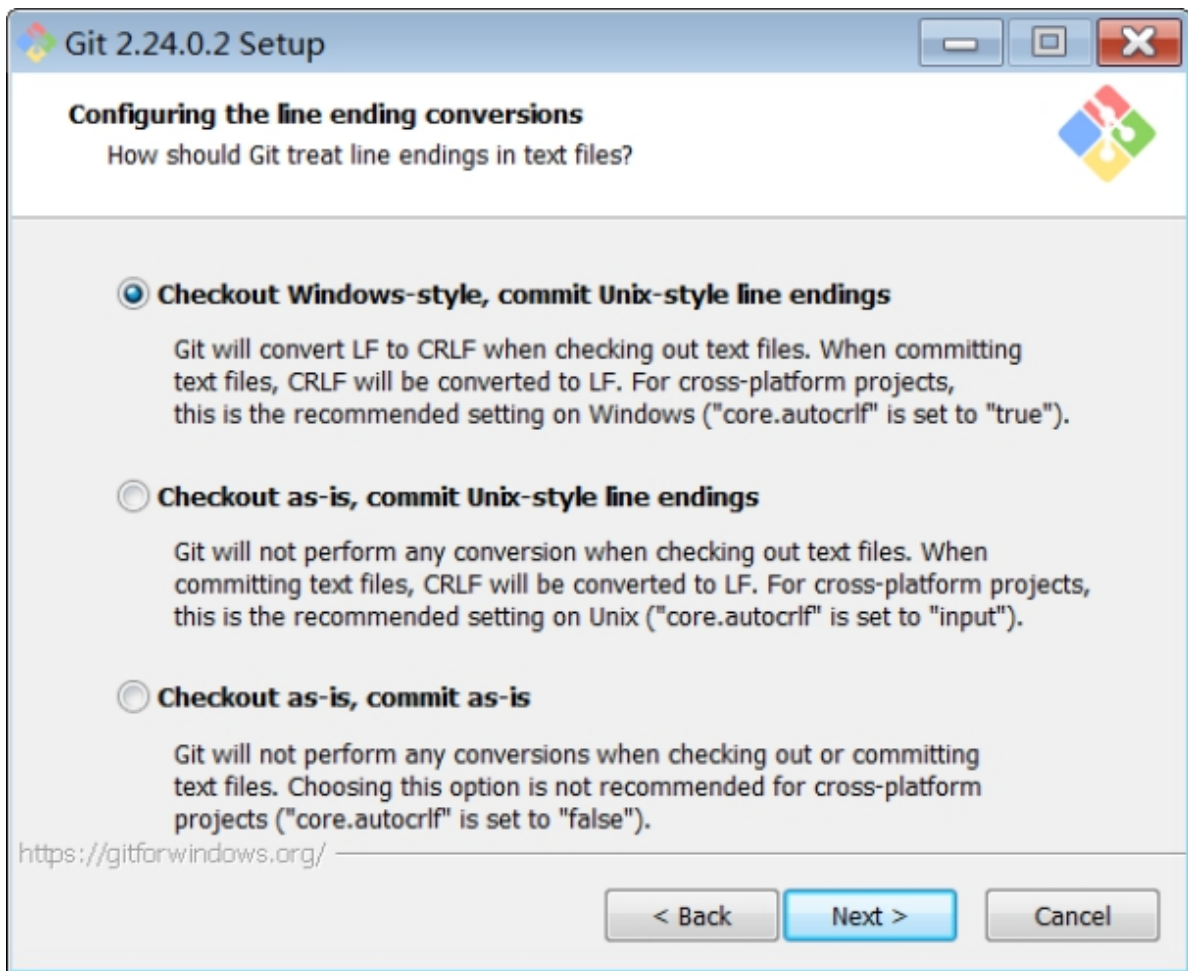
这个界面是选择HTTPS后端传输。

第一个选项是“使用 OpenSSL 库”。服务器证书将使用ca-bundle.crt文件进行验证。这也是我们常用的选项。

第二个选项是“使用本地 Windows 安全通道库”。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。

我使用默认选项第一项

08、配置行尾符号转换



这个界面是配置行尾符号转换。

第一个选项是“签出Windows风格，提交Unix风格的行尾”。签出文本文件时，Git会将LF转换为CRLF。提交文本文件时，CRLF将转换为LF。对于跨平台项目，这是Windows上的推荐设置（“core.autocrlf”设置为“true”）

第二个选项是“按原样签出，提交Unix样式的行尾”。签出文本文件时，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上的建议设置（“core.autocrlf”设置为“input”）

第三种选项是“按原样签出，按原样提交”。当签出或提交文本文件时，Git不会执行任何转换。不建议跨平台项目选择此选项（“core.autocrlf”设置为“false”）

我选择第一种选项，点击“Next”

09、配置终端模拟器以与Git Bash一起使用

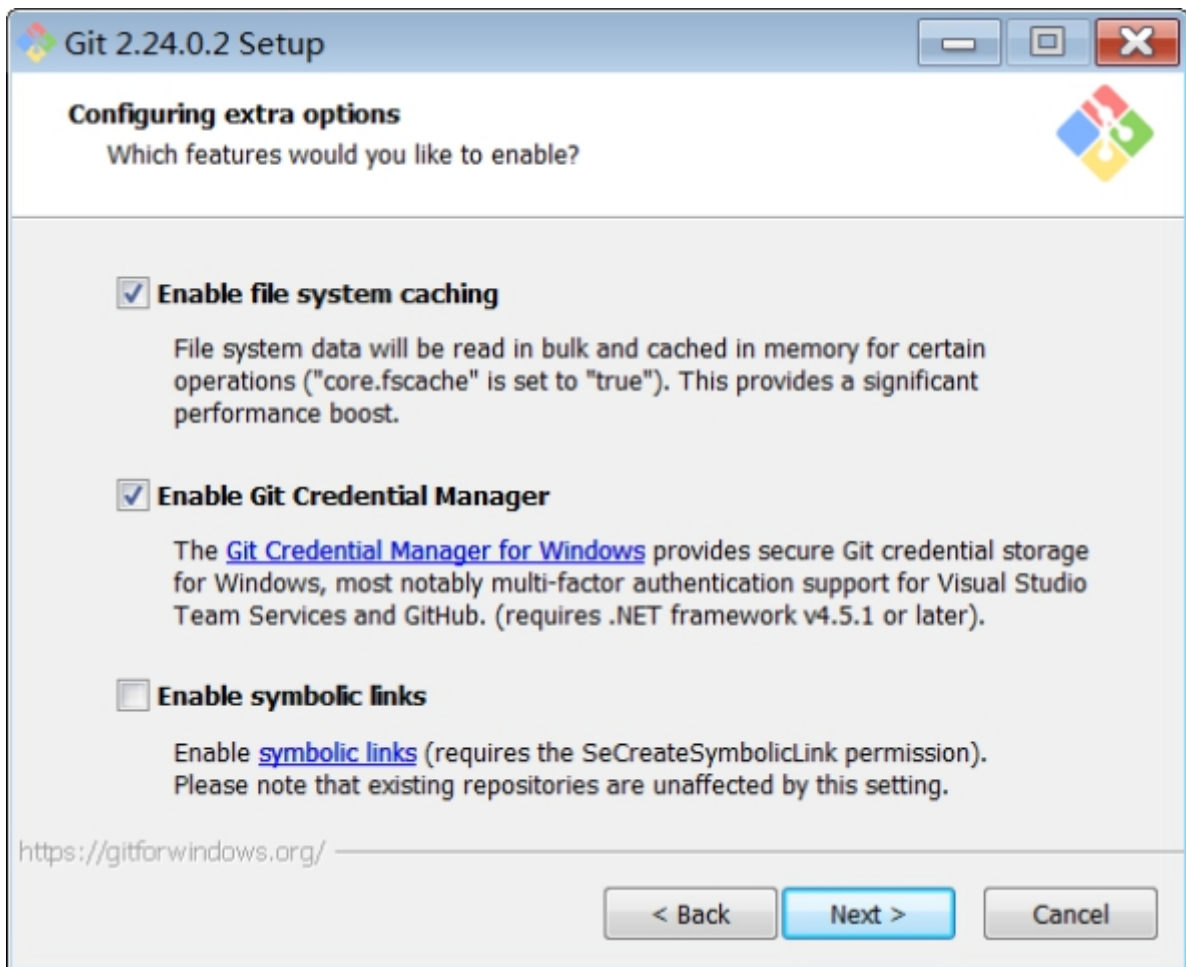


这个界面是配置终端模拟器以与Git Bash一起使用。

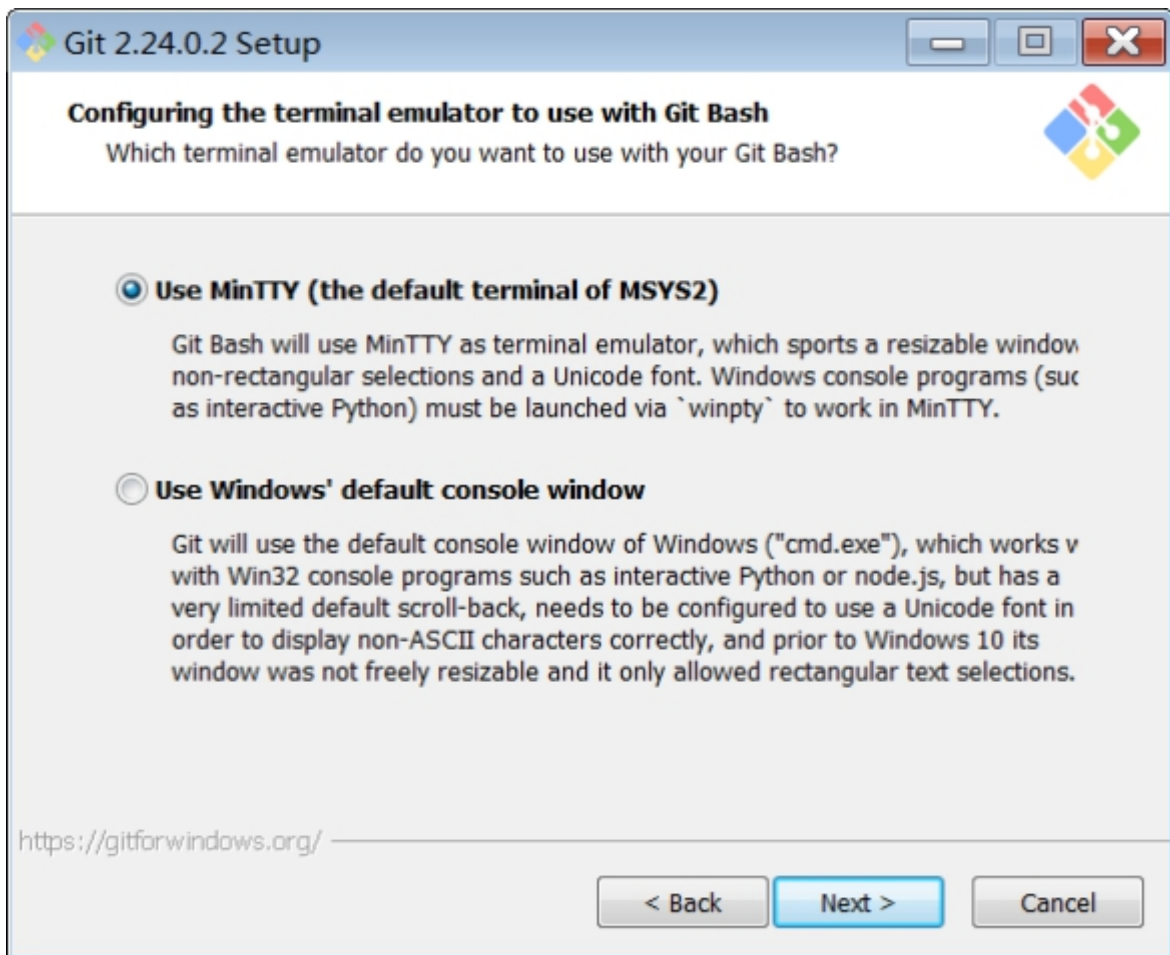
第一个选项是“使用MinTTY（MSYS2的默认终端）”。Git Bash将使用MinTTY作为终端模拟器，该模拟器具有可调整大小的窗口，非矩形选择和Unicode字体。Windows控制台程序（例如交互式Python）必须通过“winpty”启动才能在MinTTY中运行。

第二个选项是“使用Windows的默认控制台窗口”。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。

我选择默认的第一种选项，点击“Next”按钮继续到下图的界面：



10、**配置配置额外的选项**



这个界面是配置配置额外的选项。

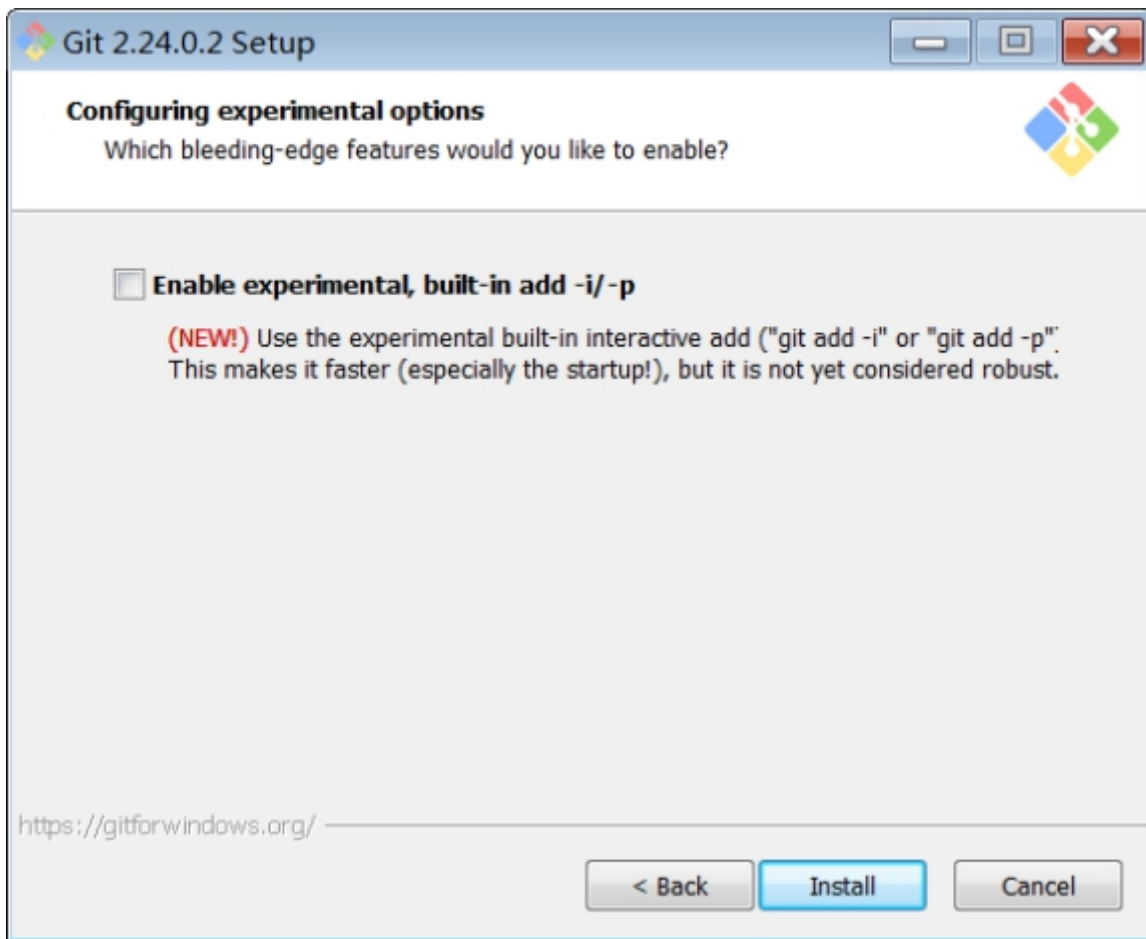
第一个选项是“启用文件系统缓存”。文件系统数据将被批量读取并缓存在内存中用于某些操作（“core.fscache”设置为“true”）。这提供了显著的性能提升。

第二个选项是“启用Git凭证管理器”。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）。

第三个选项是“启用符号链接”。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。

我勾选默认的第一、第二选项，点击“Next”

11、**配置实验选项**



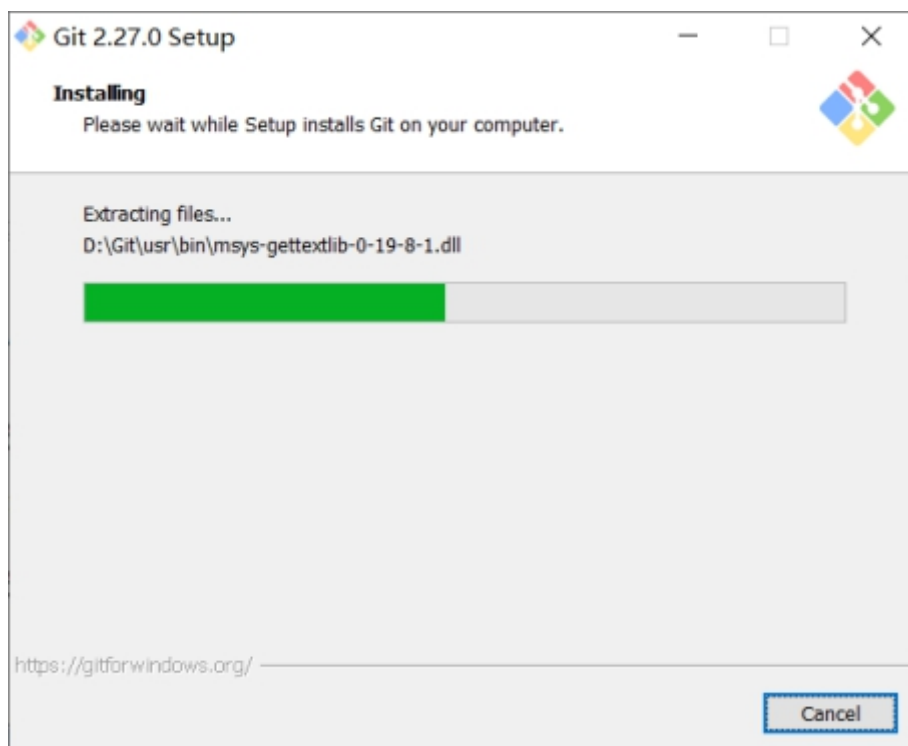
这个界面是配置实验选项。

启用实验性的内置添加 -i / -p。（新！）使用实验性的内置交互式add（“ git add -i”或“ git add -p”）。这使其速度更快（尤其是启动！），但尚未被认为是可靠的。

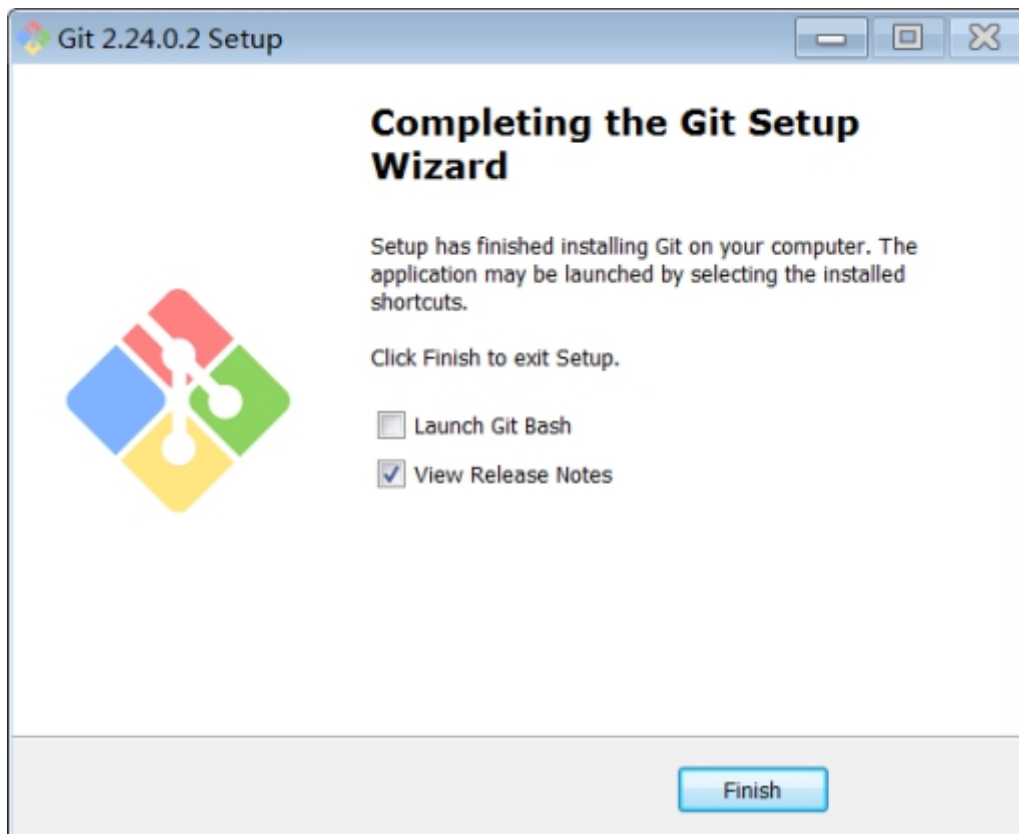
默认不勾选，直接点击“Next”

12、安装进度指示

安装进度结束之后，会出现下图的完成Git安装向导界面：



13、安装完成

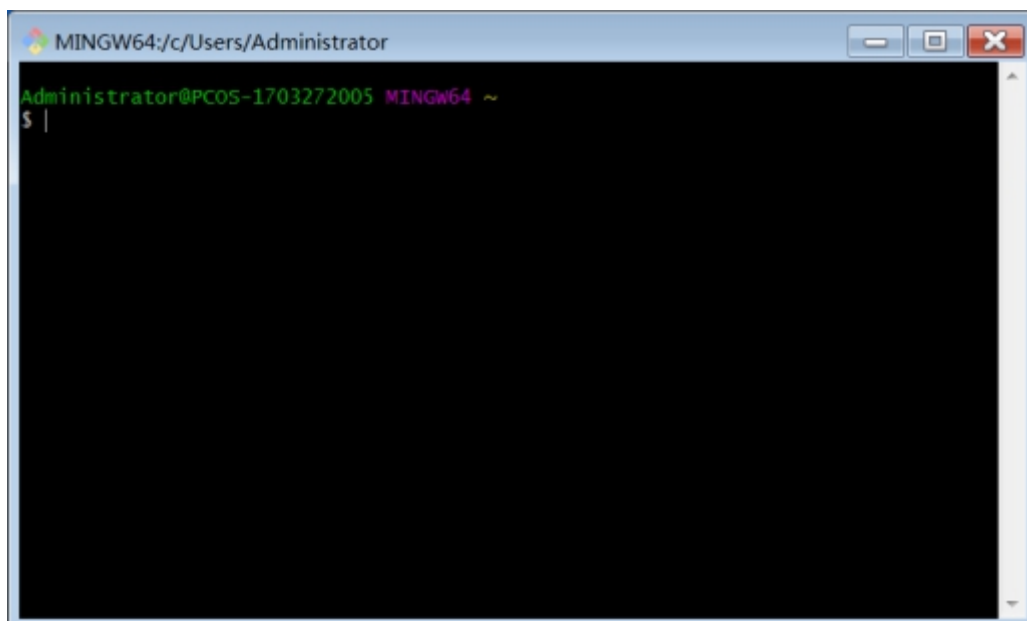


在这个界面，可以勾选是否启动Git Bash和是否查看发行说明，然后点“Finish”按钮退出安装界面。

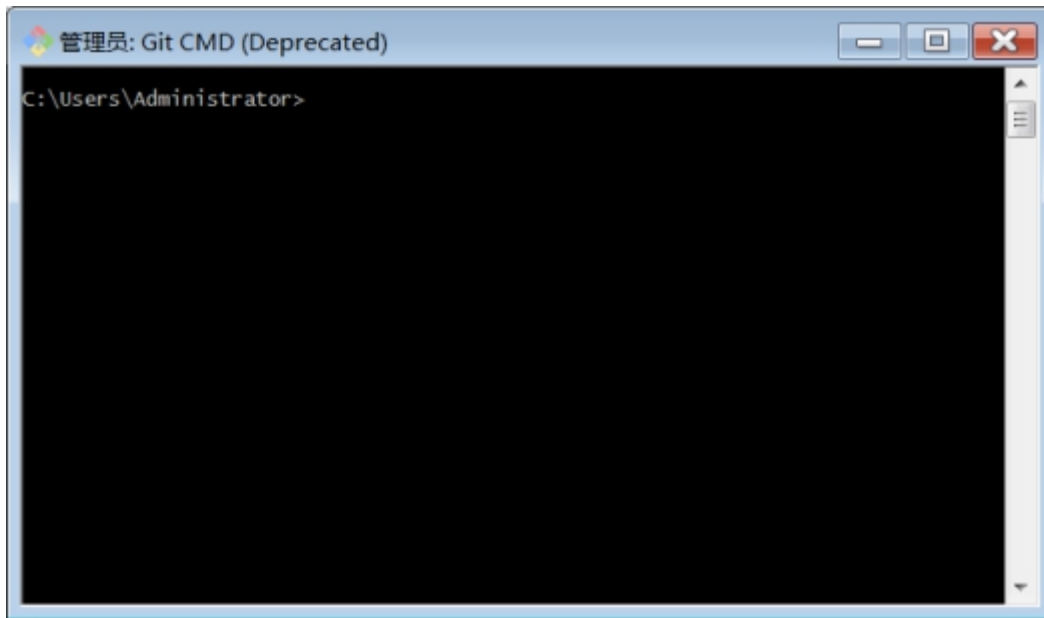
14、启动测试

到此，Git的安装完成，可以在开始菜单中看到Git的三个启动图标（Git Bash、Git CMD(Deprecated)、Git GUI）。

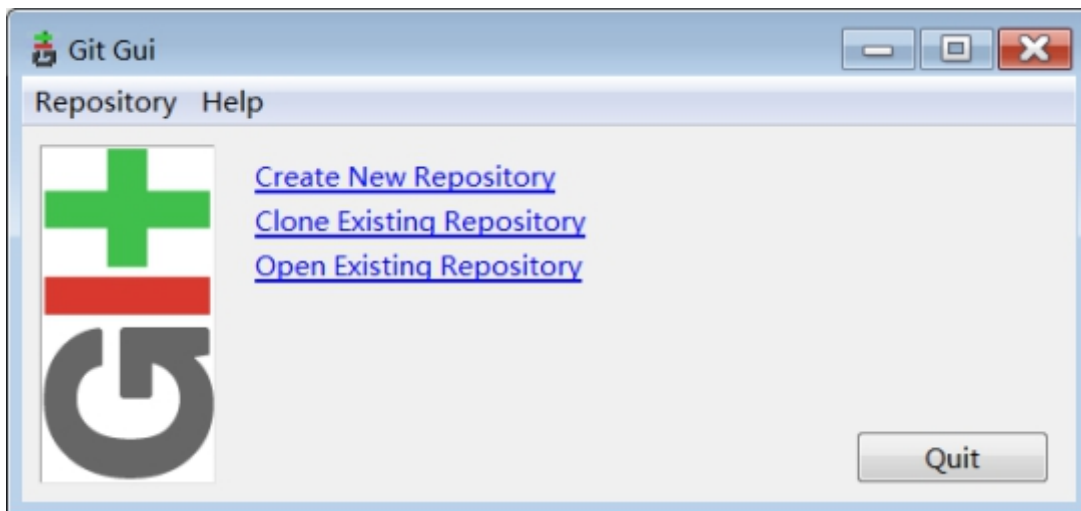
Git Bash，是Git配套的一个控制台，点击打开如下图：



Git CMD(Deprecated)，是通过CMD使用Git（不推荐使用），点击打开如下图：



Git GUI, 是Git的可视化操作工具, 点击打开如下图:



Git基本命令

Windows环境下

开始菜单里找到“Git”->“Git Bash”,打开Git命令行窗口

Linux环境下

直接调用命令

设置用户属性

```
[root@localhost testGit]# git config --global user.name "java"
[root@localhost testGit]# git config --global user.email "testGit@gmail.com"
```

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。

创建版本库

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”。

创建个文件夹

```
[root@localhost testGit]# mkdir /usr/local/git/repository/testGit/
[root@localhost testGit]# cd /usr/local/git/repository/testGit/
[root@localhost testGit]# pwd
/usr/local/git/repository/testGit/
```

将文件夹变成Git可以管理的仓库

```
git init
```

```
[root@localhost testGit]# git init
已初始化空的 Git 仓库于 /usr/local/git/repository/testGit/.git/
```

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。

最好不要用Windows自带的记事本编辑文件，会出现乱码。

创建一个 `test.txt` 文件，编写内容

```
[root@localhost testGit]# touch test.txt
```

编辑 `test.txt`

```
[root@localhost testGit]# vim test.txt
```

内容是

```
Git 是一个版本控制系统
它是一个免费的服务
```

保存在版本库文件夹及其子目录下

添加到版本库

```
git add 文件
```

将文件添加到版本库

```
[root@localhost testGit]# git add test.txt
```


提交到版本库

```
git commit
```

add 只是添加到待提交区, commit 才会正在提交到版本库

-m 后面跟的是提交说明,

```
[root@localhost testGit]# git commit -m "wrote a test file"
[master (根提交) 39d8146] wrote a test file
1 file changed, 2 insertions(+)
create mode 100644 test.txt
```

1 file changed: 1个文件被改动(我们新添加的test.txt文件);

2 insertions: 插入了两行内容(test.txt有两行内容)。

查看版本库当前状态

如果我们修改了 test.txt,修改内容为

Git 是一个分布式版本控制系统
它是一个免费的服务

我们现在查看版本库当前状态

```
git status
```

```
[root@localhost testGit]# git status
位于分支 master
尚未暂存以备提交的变更:
  (使用 "git add <文件>..." 更新要提交的内容)
  (使用 "git checkout -- <文件>..." 丢弃工作区的改动)

    修改:      test.txt

修改尚未加入提交 (使用 "git add" 和/或 "git commit -a")
```

查看文件内容具体的区别

```
git diff 文件
```

```
[root@localhost testGit]# git diff test.txt
diff --git a/test.txt b/test.txt
index cb06441..28abdf5 100644
--- a/test.txt
+++ b/test.txt
@@ -1,2 +1,2 @@
-Git 是一个版本控制系统
+Git 是一个分布式版本控制系统
 它是一个免费的服务
```

重新提交修改后的文件

```
[root@localhost testGit]# git add test.txt
[root@localhost testGit]# git status
位于分支 master
要提交的变更:
(使用 "git reset HEAD <文件>..." 以取消暂存)
```

修改: test.txt

```
[root@localhost testGit]# git commit -m "add distributed"
[master 70e893b] add distributed
1 file changed, 1 insertion(+), 1 deletion(-)
```

再次查看版本库状态

查看状态显示没什么可以提交的

```
[root@localhost testGit]# git status
位于分支 master
无文件要提交，干净的工作区
```

多修改提交一次

内容为

Git 是一个分布式版本控制系统
Git 是根据GPL协议发行的免费软件。

提交

```
[root@localhost testGit]# git add test.txt
[root@localhost testGit]# git commit -m "append GPL"
[master 5c9335e] append GPL
1 file changed, 1 insertion(+), 1 deletion(-)
```

查看提交历史记录

`git log`

```
[root@localhost testGit]# git log
commit 5c9335ee7b4adf73b4215e3591a0a7c6288e7a30 (HEAD -> master)
Author: java <testGit@gmail.com>
Date: Mon Sep 7 10:43:43 2020 +0800

    append GPL

commit 70e893b841d6ff1ccccec0a506a8aed16137e7ce
Author: java <testGit@gmail.com>
Date: Mon Sep 7 10:30:34 2020 +0800

    add distributed
```

```
commit 39d81468aa86277e80e338cab1df686d9ad362c4
Author: java <testGit@gmail.com>
Date: Mon Sep 7 10:20:26 2020 +0800
```

```
wrote a test file
```

简化显示

```
git log --pretty=oneline
```

```
[root@localhost testGit]# git log --pretty=oneline
5c9335ee7b4adf73b4215e3591a0a7c6288e7a30 (HEAD -> master) append GPL
70e893b841d6ff1ccccec0a506a8aed16137e7ce add distributed
39d81468aa86277e80e338cab1df686d9ad362c4 wrote a test file
```

版本回退

```
git reset --hard 提交版本号(只写前几位)或特殊指令
```

回退到上一个版本

```
#表示上个版本,HEAD^^:表示上上个版本
[root@localhost testGit]# git reset --hard 5c9335e
HEAD 现在位于 5c9335e append GPL
```

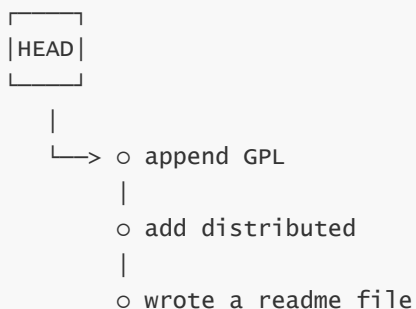
查看回退后的文件内容

```
[root@localhost testGit]# cat test.txt
Git 是一个分布式版本控制系统
它是一个免费的服务
```

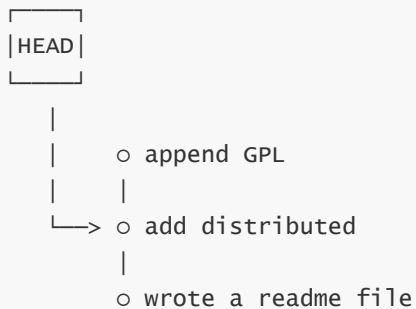
回退到指定版本

```
[root@localhost testGit]# git reset --hard 5c9335e
HEAD 现在位于 5c9335e append GPL
```

Git的版本回退速度非常快，因为Git在内部有个指向当前版本的 `HEAD` 指针，当你回退版本的时候，Git仅仅是把HEAD从指向 `append GPL`：



改为指向 `add distributed`:



然后顺便把工作区的文件更新了。所以你让 `HEAD` 指向哪个版本号，你就把当前版本定位在哪。

查看命令历史记录

如果不知道已经被回退前的版本号，可以使用 `git reflog` 查看命令历史记录

```
git reflog
```

```
[root@localhost testGit]# git reflog
5c9335e (HEAD -> master) HEAD@{0}: reset: moving to 5c9335e
70e893b HEAD@{1}: reset: moving to HEAD^
5c9335e (HEAD -> master) HEAD@{2}: reset: moving to 5c9335e
70e893b HEAD@{3}: reset: moving to HEAD^
5c9335e (HEAD -> master) HEAD@{4}: commit: append GPL
70e893b HEAD@{5}: commit: add distributed
39d8146 HEAD@{6}: commit (initial): wrote a test file
```

工作区&暂存区

工作区 (Working Directory)

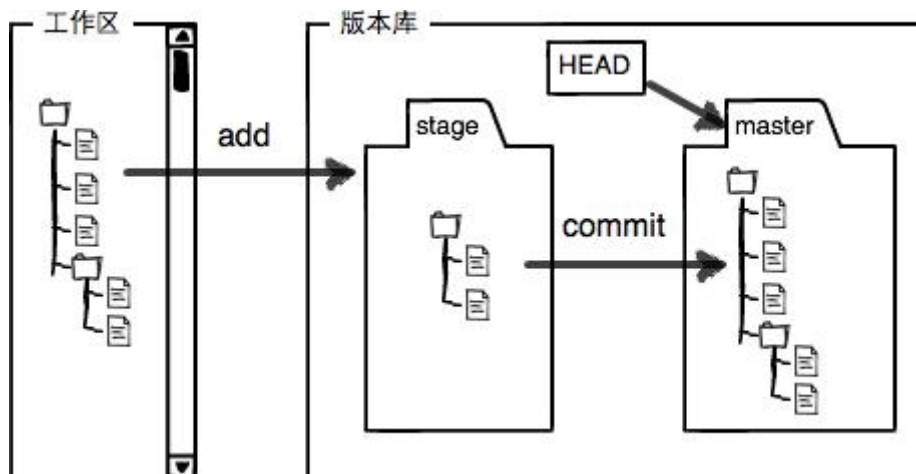
就是创建的本地目录，比如我的 `testGit` 文件夹就是一个工作区：

```
[root@localhost repository]# ll
总用量 0
drwxr-xr-x. 3 root root 34 9月  7 10:50 testGit
```

版本库 (Repository)

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。

Git的版本库里存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自动创建的第一个分支 `master`，以及指向 `master` 的一个指针叫 `HEAD`。



`git add` 把文件添加进去，实际上就是把文件修改添加到暂存区；

`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。

详解

修改 test.txt 文件

Git 是一个分布式版本控制系统
Git 是根据GPL协议发行的免费软件。
Git有一个可变的索引，称为stage。

新建一个 test2.txt 文件,文件内容随意

查看 版本库状态

```
[root@localhost testGit]# git status
位于分支 master
尚未暂存以备提交的变更：
  （使用 "git add <文件>..." 更新要提交的内容）
  （使用 "git checkout -- <文件>..." 丢弃工作区的改动）

    修改：      test.txt

未跟踪的文件：
  （使用 "git add <文件>..." 以包含要提交的内容）

    text2.txt

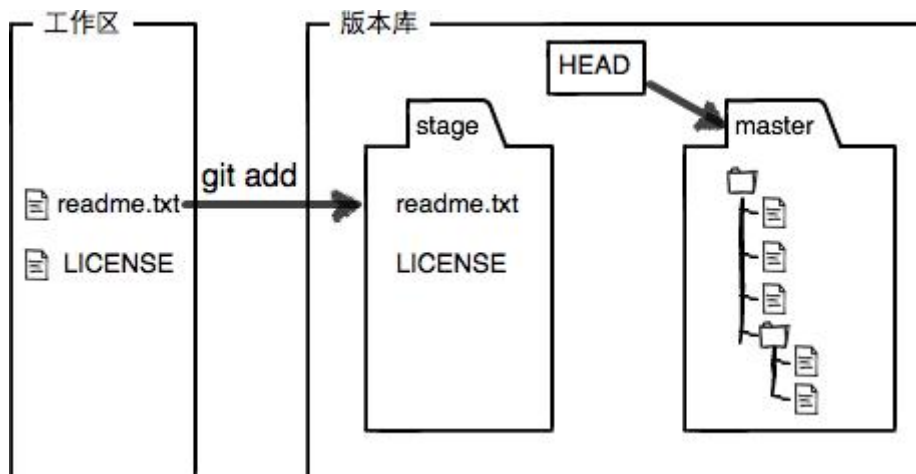
修改尚未加入提交（使用 "git add" 和/或 "git commit -a"）
```

将两个文件都 git add 后 再查看版本库状态

```
[root@localhost testGit]# git status
位于分支 master
要提交的变更：
  （使用 "git reset HEAD <文件>..." 以取消暂存）

    修改：      test.txt
    新文件：    text2.txt
```

此时暂存区变成



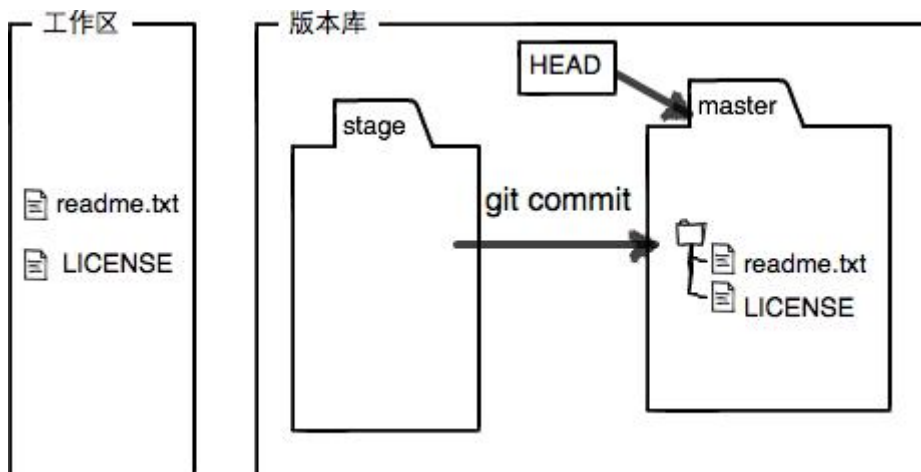
提交

```
[root@localhost testGit]# git commit -m "understand how stage works"
[master 9661e39] understand how stage works
 2 files changed, 2 insertions(+)
 create mode 100644 text2.txt
```

再次查看版本库状态

```
[root@localhost testGit]# git status
位于分支 master
无文件要提交，干净的工作区
```

现在版本库变成



Git撤销修改&删除文件

撤销修改

撤销工作区修改

```
#将版本库中的文件替换工作区文件
git checkout -- 文件
```

修改 test.txt 文件内容为


```
[root@localhost testGit]# cat test.txt
Git 是一个分布式版本控制系统
Git 是根据GPL协议发行的免费软件。
Git有一个可变的索引，称为stage。
Git跟踪变化。
哈哈哈哈哈(需要撤销的内容)
```

撤销工作区修改

```
[root@localhost testGit]# git checkout -- test.txt
[root@localhost testGit]# cat test.txt
Git 是一个分布式版本控制系统
Git 是根据GPL协议发行的免费软件。
Git有一个可变的索引，称为stage。
```

撤销暂存区修改

```
#1. 拉取最近一次提交到版本库的文件到暂存区 该操作不影响工作区
git reset HEAD 文件
#2. 将版本库中的文件替换工作区文件
git checkout -- 文件
```

删除文件

```
#1. 工作区和暂存区删除文件
git rm 文件
#2. 提交修改
git commit -m "提交删除说明"
```

如果已经提交过的文件，误删可以用 `git checkout -- 文件` 来恢复

远程仓库

创建SSH Key

SSH Key :远程仓库需要识别出你推送的提交确实是你推送的，而不是别人冒充的，而Git支持SSH协议，所以，远程只要知道了你的公钥，就可以确认只有你自己才能推送。

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

用户主目录

```
#windows默认(新建后看不到,开启显示隐藏文件夹)
C:\Users\用户名\.ssh\
#Linux中root用户(查看目录,ls -a 显示隐藏文件夹)
/root/.ssh/
#Linux中非root用户
/home/用户名/.ssh/
```

```
[root@localhost ~]# ssh-keygen -t rsa -C "邮箱地址@email.com"
Generating public/private rsa key pair.
```

```

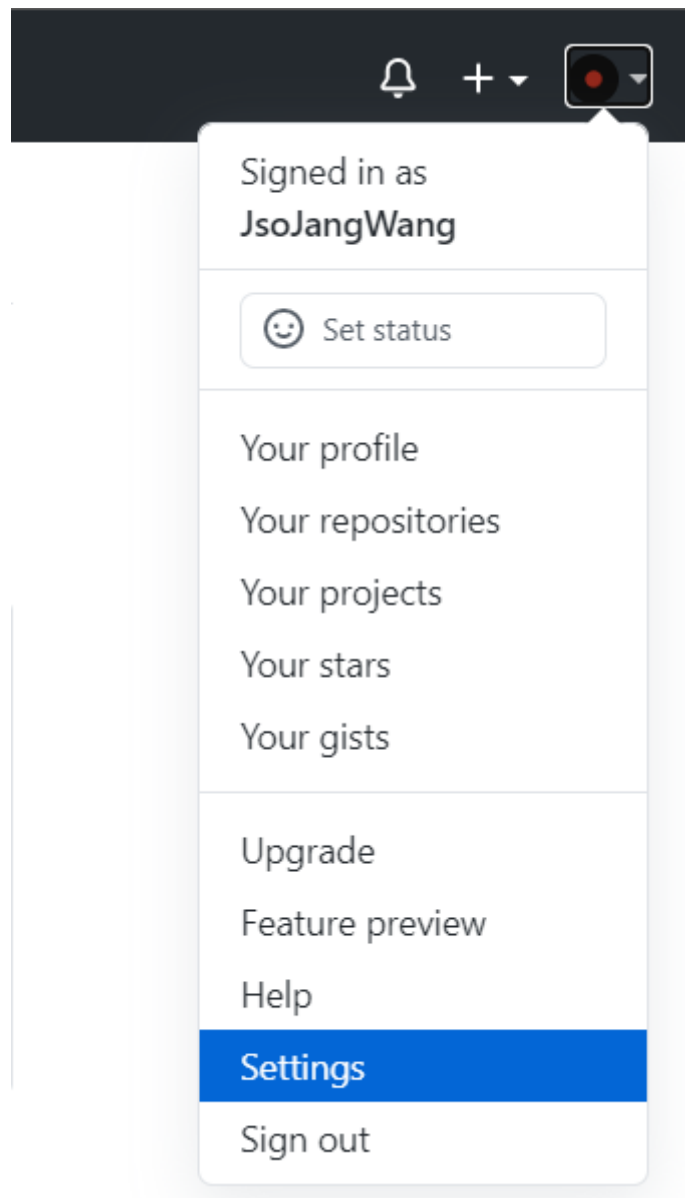
#设置保存密钥文件 ， 直接回车就好
Enter file in which to save the key (/root/.ssh/id_rsa):
Created directory '/root/.ssh'.
#设置 ssh 密码 ， 一般可以不设置 ， 直接回车
Enter passphrase (empty for no passphrase):
#确认 ssh 密码 ， 直接回车
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:OodRx//WScuwjw53aJqVebqrY5jNGAMtRsL28Se9Ivs 707546728@qq.com
The key's randomart image is:
+---[RSA 3072]-----+
|      .                |
|      + o .            |
|      . + = +          |
|      * = +            |
|      o S o o . .      |
|      = + . .O.o       |
|      + + X. Oo*.      |
|      + + =B.*         |
|      .E.++*o.        |
+---[SHA256]-----+

```


在用户主目录里找到 `.ssh` 目录，里面有 `id_rsa` 和 `id_rsa.pub` 两个文件，这两个就是SSH Key的密钥对，`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。

设置新 SSH key

登录 Github 后，点击用户头像，选择 `Settings`，选择 `SSH and GPG keys`



,点击 New SSH key 按钮

image-20200907150207553

获取公钥内容

```
[root@localhost .ssh]# cat id_rsa.pub
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQGDQFIo4CiPYD/B4xFwBARVWYRVA6QvmJe+OZGYRY+v+vvF6dw9FB
+kpEyP9ZnHC76+20VXFtxBUuNK3x6qHLTraou3m45Zhr0vbhCnkyEKkk2Zm2b4mU/BK/sQJQbED1wzWf
fexvQsbqzwOhqfc41kv8JhTudxMA/a4wcuJ4J+NHg1uXRqHGc4wQUqrZd7PdpuQbo0RGryDNvj6JukRQ
62dnXwsilmq8IdnQp2TbjDmTj7iFks/xIWVFZedsIc8xJ2B2U1Z205e4PBjnAeOrNppIrgVssh7G41m
ePrAp1aVfp+afbxcpRTHAT3jjeUPR2rQveU31ebFegnqQN2BHjQAwFr7d+HMnqw0InB00TENh67N5sf
PAxgw2H5Szfxb8vE8nJ3JPn90fZIQwDoadnKL2Mhrb5RestCwhhBS22ZUTUgwxTq1T9v7SpM+WR1Y4Z
Eg+um0IDTwVHyd7w5mIH9XeguOnHACOD0brjAHZrWODAEu3sQVKOM7F5z3EBfm8= 你的邮箱地址
```

在Github上粘贴

Title


linux的SSH key

Key

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQgQDFlo4CiPYD/B4xFwBARVWYRVA6QvmJe+OZGYRY+v+vF6dW9FB+kp
EyP9ZnHC76+20VXFtBUuNK3x6qHLTraOu3m45ZhR0vbhCnkyEKk2Zm2b4mU/BK/sQJQbED1WzWffexvQSbqzw
Ohqfc4lkV8JhTUdxMA/a4wcUj4J+NHg1uXRqHGc4WQUqrZd7PdpuQbo0RGryDNvj6JukRQ62dnXWsilmq8ldnQp
2TbjDmTj7iFks/xlWVFZedslc8xJ2B2UIZ205e4PBjnAeOrNPplrgVsshW7G41mePrAp1aVFp+afbxcpRTHAT3jjeUPR2r
QveU3lebFegngQN2BHjQAwFr7d+HMnqw0InB00TENh67N5sfPAxgw2H5SzfXb8vE8nJ3JPn90fZIQwDOadDnKL2
MHrb5RestCwhhBS22ZUTUgwXtQIT9v7SpM+WRlY4ZEg+uM0IDtwVHyd7W5mIH9XeguOnHACOD0brjAHZrWOD
AeU3sQVK0M7F5z3EBfm8= 707546728@qq.com

Add SSH key

添加成功



linux的SSH key
9c:e7:85:f7:ca:4a:87:f8:b7:da:11:b2:f2:42:ea:3a
Added on 7 Sep 2020
Never used — Read/write

Delete

新建远程仓库

回到主页面,点击 new 按钮

Repositories

New

Find a repository...

设置仓库名

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner *

 JsoJangWang ▾

Repository name *

/

testGit



Great repository names are short and memorable. Need inspiration? How about [symmetrical-chainsaw?](#)

Description (optional)



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☐ **Add a README file**

This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**

Choose which files not to track from a list of templates. [Learn more.](#)

☐ **Choose a license**

A license tells others what they can and can't do with your code. [Learn more.](#)

关联远程仓库

Github推荐的关联方式

1. 在命令行上创建一个新的存储库

```
echo "# testGit" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M master
git remote add origin git@github.com:JsoJangWang/testGit.git
git push -u origin master
```

2. 从命令行推送现有存储库

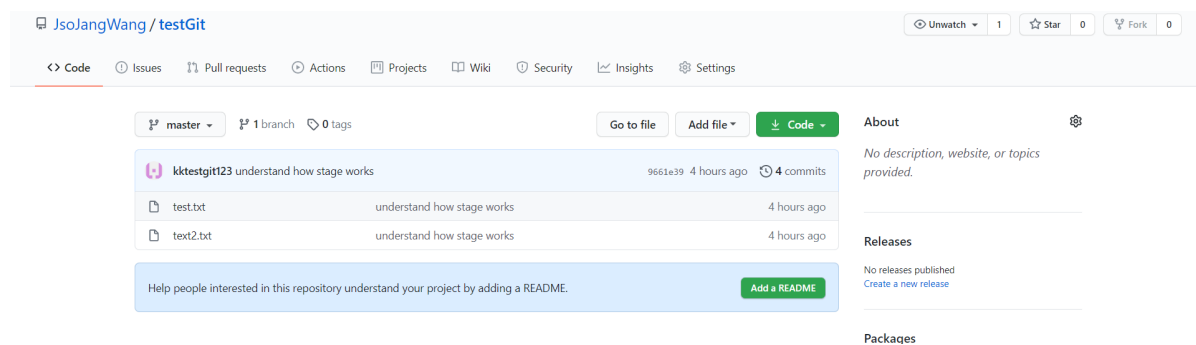
```
git remote add origin git@github.com:JsoJangWang/testGit.git
git branch -M master
git push -u origin master
```

关联到Github

```
#关联到远程仓库
[root@localhost testGit]# git remote add origin
git@github.com:JsoJangWang/testGit.git
#重命名分支
[root@localhost testGit]# git branch -M master
```

```
#将本地内容 push 到远程仓库 , -u : 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。
[root@localhost testGit]# git push -u origin master
The authenticity of host 'github.com (52.74.223.119)' can't be established.
RSA key fingerprint is SHA256:nThbg6kXupJWG17E1IGOCsPRomTxdCARLviKw6E5SY8.
#第一次连接 , 需要确认是否连接
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
#警告:把GitHub的key添加到本机的一个信任列表里了(第一次连接才会警告)
warning: Permanently added 'github.com,52.74.223.119' (RSA) to the list of known hosts.
枚举对象: 13, 完成.
对象计数中: 100% (13/13), 完成.
Delta compression using up to 2 threads.
压缩对象中: 100% (9/9), 完成.
写入对象中: 100% (13/13), 1.12 KiB | 143.00 KiB/s, 完成.
Total 13 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To github.com:JsoJangWang/testGit.git
 * [new branch]      master -> master
分支 'master' 设置为跟踪来自 'origin' 的远程分支 'master'。
```

关联成功

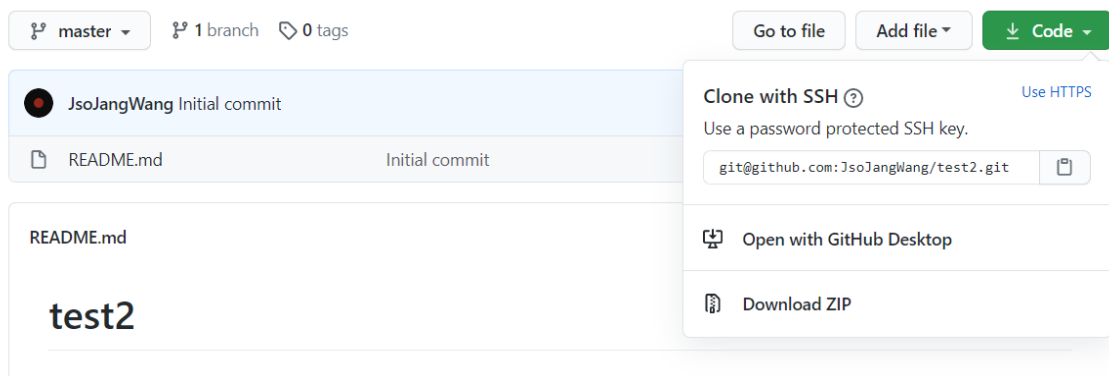


简化后的 push 远程仓库

```
git push origin master
```

克隆远程仓库

```
git clone git@github.com:用户名/仓库名.git
```



```
[root@localhost repository]# git clone git@github.com:JsoJangWang/test2.git
```



```
正克隆到 'test2'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
接收对象中: 100% (3/3), 完成.
[root@localhost repository]# ll
总用量 0
drwxr-xr-x. 3 root root 35 9月  7 16:00 test2
drwxr-xr-x. 3 root root 51 9月  7 13:57 testGit

[root@localhost repository]# cd test2
[root@localhost test2]# ls
README.md
```

分支管理

创建分支与合并分支

```
#查看当前分支
git branch
#创建分支
git branch 分支名
#删除分支
git branch -d 分支名
#切换分支 switch : 2.23版本用来替代 checkout 切换分支的功能,推荐使用;yum安装的版本不够
git checkout 分支名 / git switch 分支名
#创建并切换分支
git checkout -b 分支名 / git switch -c 分支名
#合并分支 ,将其他分支的内容合并到本分支
git merge 其他分支名
```

查看当前分支

```
[root@localhost testGit]# git branch
* master
```

创建并切换分支

```
[root@localhost testGit]# git checkout -b dev
切换到一个新分支 'dev'
[root@localhost testGit]# git branch
* dev
  master
```

修改 test.txt 文件内容为:

```
[root@localhost testGit]# cat test.txt
Git 是一个分布式版本控制系统
Git 是根据GPL协议发行的免费软件。
Git有一个可变的索引，称为stage。
创建新分支很快。
[root@localhost testGit]# git add test.txt
[root@localhost testGit]# git commit -m "branch test"
[master 5dffb4f] branch test
1 file changed, 1 insertion(+)
```

切换分支并合并分支

```
[root@localhost testGit]# git checkout master
切换到分支 'master'
[root@localhost testGit]# git merge dev
更新 9661e39..5dffb4f
Fast-forward
 test.txt | 1 +
1 file changed, 1 insertion(+)
```

删除分支

```
[root@localhost testGit]# git branch -d dev
已删除分支 dev（曾为 5dffb4f）。
```

解决冲突

模拟场景

创建新分支

```
[root@localhost testGit]# git checkout -b feature1
切换到一个新分支 'feature1'
```

修改 test.txt 文件内容

```
[root@localhost testGit]# cat test.txt
Git 是一个分布式版本控制系统
Git 是根据GPL协议发行的免费软件。
Git有一个可变的索引，称为stage。
创建新分支既快捷又简单。
```

在 feature1 分支上提交

```
[root@localhost testGit]# git add test.txt
[root@localhost testGit]# git commit -m "冲突1"
[feature1 91fda45] 冲突1
1 file changed, 1 insertion(+), 1 deletion(-)
```

切换到 master 分支

```
[root@localhost testGit]# git checkout master
```

切换到分支 'master'

您的分支领先 'origin/master' 共 1 个提交。

(使用 "git push" 来发布您的本地提交)

制造冲突,编辑 test.txt 文件内容

```
[root@localhost testGit]# cat test.txt
```

Git 是一个分布式版本控制系统

Git 是根据GPL协议发行的免费软件。

Git有一个可变的索引,称为stage。

冲突内容。

提交

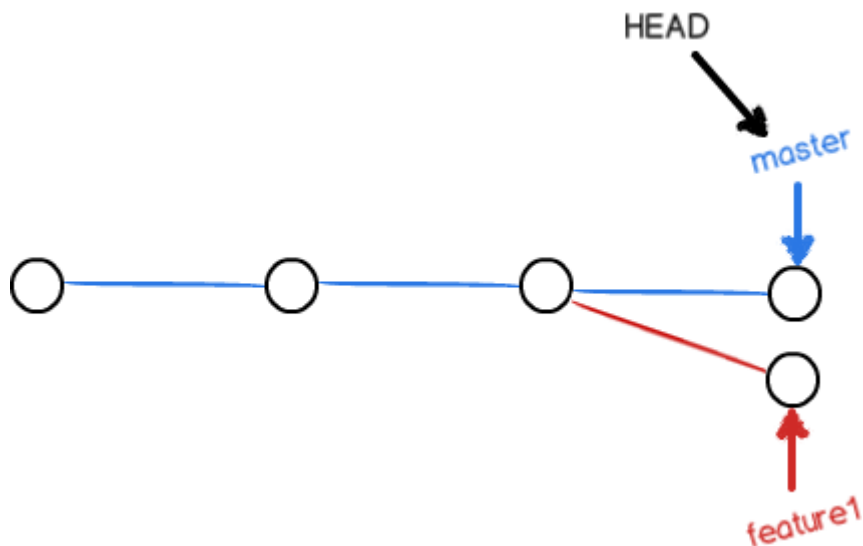
```
[root@localhost testGit]# git add test.txt
```

```
[root@localhost testGit]# git commit -m "冲突2"
```

```
[master d014fcb] 冲突2
```

```
1 file changed, 1 insertion(+), 1 deletion(-)
```

合并内容,出现冲突



```
[root@localhost testGit]# git merge feature1
```

自动合并 test.txt

冲突(内容): 合并冲突于 test.txt

自动合并失败, 修正冲突然后提交修正的结果。

查看冲突内容

Git用 <<<<<<<, =====, >>>>>>> 标记出不同分支的内容

```
[root@localhost testGit]# cat test.txt
Git 是一个分布式版本控制系统
Git 是根据GPL协议发行的免费软件。
Git有一个可变的索引，称为stage。
<<<<<<< HEAD
冲突内容。
=====
创建新分支既快捷又简单。
>>>>>>> feature1
```

解决冲突

```
[root@localhost testGit]# cat test.txt
Git 是一个分布式版本控制系统
Git 是根据GPL协议发行的免费软件。
Git有一个可变的索引，称为stage。
创建新分支既快捷又简单。解决完毕。
[root@localhost testGit]# git add test.txt
[root@localhost testGit]# git commit -m "解决完冲突"
[master 64f51a7] 解决完冲突
```

查看合并情况

```
[root@localhost testGit]# git log --graph --pretty=oneline --abbrev-commit
* 64f51a7 (HEAD -> master) 解决完冲突
|\
| * 91fda45 (feature1) 冲突1
* | d014fcb 冲突2
|/
* 5dffb4f branch test
* 9661e39 (origin/master) understand how stage works
* 5c9335e append GPL
* 70e893b add distributed
* 39d8146 wrote a test file
```

删除 feature1 分支

```
[root@localhost testGit]# git branch -d feature1
已删除分支 feature1（曾为 91fda45）。
```

分支管理策略

合并分支时有两种策略:Fast forward(默认), No Fast forward;

Fast forward :

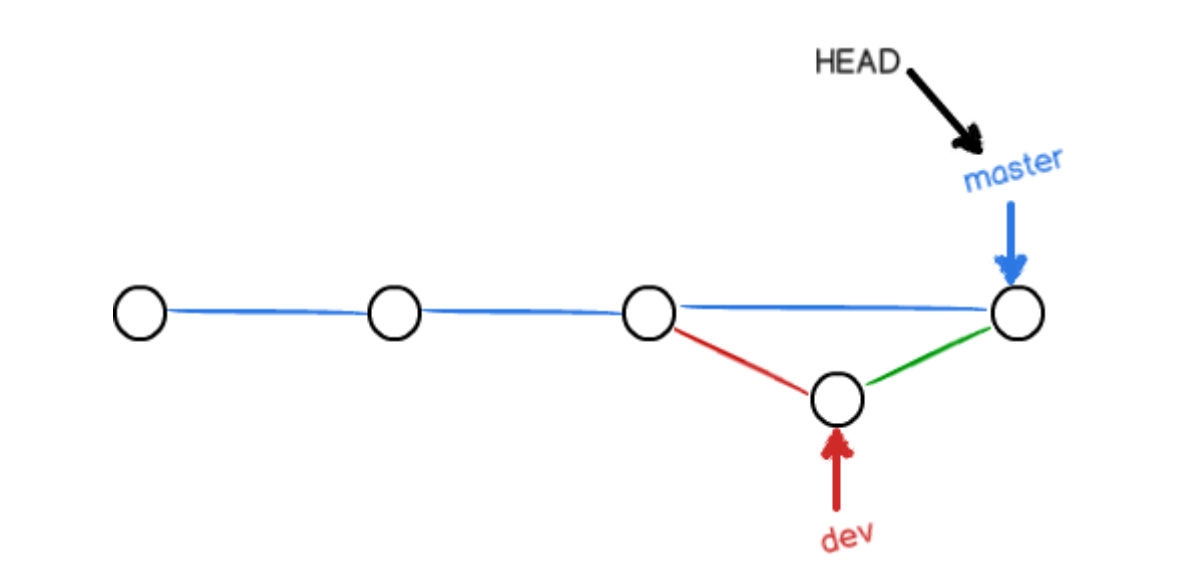
合并后不生产新的commit,删除分支, 会丢掉分支信息

No Fast forward:

合并时生成一个新的commit, 从分支历史上就可以看出分支信息

```
\git merge --no-ff -m "合并说明" 被合并分支
```

```
[root@localhost testGit]# git log --graph --pretty=oneline --abbrev-commit
*   32cd59a (HEAD -> master) Merge branch 'dev' #No Fast forward分支合并,会保留被合并的分支信息
|\
| * ec15cd0 (dev) dev新增内容3
|/
* f49aef6 dev新增内容2 #Fast forward 分支合并,不会保留被合并的分支信息
* c8ece10 dev新增内容 #Fast forward 分支合并
```



Stash存储修改

修复bug时，我们会通过创建新的bug分支进行修复，然后合并，最后删除；

当手头工作没有完成时，先把工作现场保存 `git stash` 一下，然后去修复bug，修复后，再恢复 `git stash pop`，回到工作现场；

在master分支上修复的bug，想要合并到当前dev分支，可以用 `git cherry-pick 提交编号` 命令，把bug提交的修改“复制”到当前分支，避免重复劳动。

保存修改

将正在进行的修改储存起来

```
#[] 可选
git stash [save 注释]
```

[查看所有储存修改](#)

```
git stash list
```

恢复修改

```
#不写可选项，默认 stash@{0}
git stash apply [stash@{编号,通过list查看}]
```

删除修改

```
#不写可选项 , 默认 stash@{0}
git stash drop [stash@{编号,通过list查看}]
```

恢复并删除修改

```
#不写可选项 , 默认 stash@{0}
git stash pop [stash@{编号,通过list查看}]
```

清除所有修改

```
git stash clear
```

复制一个特定的提交到当前分支

```
git cherry-pick 提交编号
```

特征(Feature)分支

软件开发中，总有无穷无尽的新的功能要不断添加进来。

添加一个新功能时，你肯定不希望因为一些实验性质的代码，把主分支搞乱了，所以，每添加一个新功能，最好新建一个feature分支，在上面开发，完成后，合并，最后，删除该feature分支。

现在，你终于接到了一个任务：开发代号为Vulcan的新功能，该功能计划用于下一代星际飞船。

开发一个新feature，最好新建一个分支,就在开发完成时，要合并到 master 分支时，接到上级命令，因经费不足，新功能必须取消！

虽然白干了，但是这个包含机密资料的分支还是必须就地销毁：

丢弃未合并过的分支

```
git branch -D 分支名
```

多人协作

查看远程库信息

```
git remote
#更详细的信息
git remote -v
```



```
#远程仓库的默认名称是origin
[root@localhost testGit]# git remote
origin
#显示可以抓取和推送的origin的地址,如果没有推送权限,看不到push的地址
[root@localhost testGit]# git remote -v
origin  git@github.com:JsoJangwang/testGit.git (fetch)
origin  git@github.com:JsoJangwang/testGit.git (push)
```

推送分支

```
git push 远程库名 分支名
```

```
[root@localhost testGit]# git push origin master
Warning: Permanently added the RSA host key for IP address '13.250.177.223' to
the list of known hosts.
枚举对象: 24, 完成.
对象计数中: 100% (24/24), 完成.
Delta compression using up to 2 threads.
压缩对象中: 100% (22/22), 完成.
写入对象中: 100% (22/22), 2.01 KiB | 187.00 KiB/s, 完成.
Total 22 (delta 8), reused 0 (delta 0)
remote: Resolving deltas: 100% (8/8), done.
To github.com:JsoJangwang/testGit.git
    9661e39..32cd59a  master -> master
```

拷贝仓库

多人协作时, 大家都会往 `master` 和 `dev` 分支上推送各自的修改。

现在, 模拟多人协作, 可以在另一台电脑(Win主机) (注意要把SSH Key添加到GitHub) 或者同一台电脑的另一个目录下克隆:

```
git clone 远程仓库地址
```

```
$ git clone git@github.com:JsoJangwang/testGit.git
Cloning into 'testGit'...
remote: Enumerating objects: 35, done.
remote: Counting objects: 100% (35/35), done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 35 (delta 10), reused 34 (delta 9), pack-reused 0
Receiving objects: 100% (35/35), done.
Resolving deltas: 100% (10/10), done.
```

从远程库clone时, 默认情况下只能看到本地的 `master` 分支

```
$ git branch
* master
```

抓取仓库

#如果仓库信息没同步过来或者提交错误,也可以重新拉取数据

```
git pull [远程仓库名] [远程分支名]
```

```
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 280 bytes | 15.00 KiB/s, done.
From github.com:JsoJangwang/testGit
* [new branch]      dev      -> origin/dev
Already up to date.
```

创建远程仓库的分支到本地

```
git checkout -b 本地分支名 远程仓库名/远程分支名
```

```
$ git checkout -b dev origin/dev
Switched to a new branch 'dev'
Branch 'dev' set up to track remote branch 'dev' from 'origin'.
```

多人协作模式

1. 首先,可以试图用 `git push origin <branch-name>` 推送自己的修改;
2. 如果推送失败,则因为远程分支比你的本地更新,需要先用 `git pull` 试图合并;
3. 如果合并有冲突,则解决冲突,并在本地提交;
4. 没有冲突或者解决掉冲突后,再用 `git push origin <branch-name>` 推送就能成功!

如果 `git pull` 提示 `no tracking information`, 则说明本地分支和远程分支的链接关系没有创建,用命令 `git branch --set-upstream-to 本地分支名 远程仓库名/远程分支名`。

这就是多人协作的工作模式,一旦熟悉了,就非常简单。

Rebase

#让提交记录变得更线性

```
git rebase
```

多人协作每次合并再push后,分支变成了这样:

```
$ git log --graph --pretty=oneline --abbrev-commit
* e0ea545 (HEAD -> master) Merge branch 'master' of
github.com:michaelliao/learngit
|\
| * f005ed4 (origin/master) set exit=1
* | 582d922 add author
* | 8875536 add comment
|/
* d1be385 init hello
...
```

显得很杂乱

使用 Rebase 会将我们未push的部分移动到 远程仓库后面, 让提交记录变得线性

```
$ git rebase
First, rewinding head to replay your work on top of it...
Applying: add comment
Using index info to reconstruct a base tree...
M   hello.py
Falling back to patching base and 3-way merge...
Auto-merging hello.py
Applying: add author
Using index info to reconstruct a base tree...
M   hello.py
Falling back to patching base and 3-way merge...
Auto-merging hello.py
```

```
$ git log --graph --pretty=oneline --abbrev-commit
* 7e61ed4 (HEAD -> master) add author
* 3611cfe add comment
* f005ed4 (origin/master) set exit=1
* d1be385 init hello
...
```

标签管理

创建标签

```
#查看标签和打标签
git tag [标签] [提交编号]
#创建有说明的标签
git tag -a 标签 -m "说明" 提交编号
```

切换到需要打标签的分支上

```
[root@localhost testGit]# git tag v1.0
[root@localhost testGit]# git tag
v1.0
```

打标签到指定提交记录上

```
5c9335ee7b4adf73b4215e3591a0a7c6288e7a30 append GPL
70e893b841d6ff1ccccec0a506a8aed16137e7ce add distributed
39d81468aa86277e80e338cab1df686d9ad362c4 wrote a test file
[root@localhost testGit]# git tag v0.2 70e89
[root@localhost testGit]# git tag
v0.1
v0.2
v1.0
```

创建有说明的标签

```
[root@localhost testGit]# git tag -a v0.3 -m "version 0.3 released" f49ae
[root@localhost testGit]# git show v0.3
tag v0.3
Tagger: java <testGit@gmail.com>
Date:   Mon Sep 7 20:49:32 2020 +0800

version 0.3 released

commit f49aef6130d8404a2d72434f579bd15550013635 (tag: v0.3)
Author: java <testGit@gmail.com>
Date:   Mon Sep 7 17:38:54 2020 +0800

    dev新增内容2

diff --git a/test.txt b/test.txt
index 7274cc8..0b674ac 100644
--- a/test.txt
+++ b/test.txt
@@ -3,3 +3,5 @@ Git 是根据GPL协议发行的免费软件。
    Git有一个可变的索引，称为stage。
    创建新分支既快捷又简单。解决完毕。
    Dev分支插入的内容
+Dev分支插入的内容2
+
```

查看标签信息

#按字母排序，不指定默认显示排序后第一个
git show [标签]

```
[root@localhost testGit]# git show
commit 32cd59ac9d8eabd5b4fc360cdcb7b8e42461f22f (HEAD -> master, tag: v1.0,
origin/master)
Merge: f49aef6 ec15cd0
Author: java <testGit@gmail.com>
Date:   Mon Sep 7 17:40:57 2020 +0800

    Merge branch 'dev'

[root@localhost testGit]# git show v0.1
commit 64f51a7a45d6342050f831e6a1b8b73dfcb672f0 (tag: v0.1)
Merge: d014fcb 91fda5
Author: java <testGit@gmail.com>
```

Date: Mon Sep 7 16:57:33 2020 +0800

解决完冲突

```
diff --cc test.txt
index 1195064,1525885..c6bbd86
--- a/test.txt
+++ b/test.txt
@@@ -1,4 -1,4 +1,4 @@@
    Git 是一个分布式版本控制系统
    Git 是根据GPL协议发行的免费软件。
    Git有一个可变的索引,称为stage。
- 冲突内容。
- 创建新分支既快捷又简单。
++ 创建新分支既快捷又简单。解决完毕。
```

操作标签

```
#删除标签(标签删除,不会删除提交记录)
git tag -d 标签
#推送单个标签
git push 远程仓库名 标签
#推送所有标签
git push 远程仓库名 --tags
#删除远程标签
git push 远程仓库名 :refs/tags/标签名
```

删除标签

```
[root@localhost testGit]# git tag -d v0.1
已删除标签 'v0.1' (曾为 64f51a7)
```

推送单个标签

```
[root@localhost testGit]# git push origin v1.0
Total 0 (delta 0), reused 0 (delta 0)
To github.com:JsoJangwang/testGit.git
 * [new tag]          v1.0 -> v1.0
```

自定义Git

修改UI

```
#Git显示颜色,会让命令输出看起来更醒目
git config --global color.ui true
```

配置命令别名

```
git config --global alias.别名 命令名
```

```
[root@localhost ~]# git config --global alias.br branch
[root@localhost testGit]# git br
dev
* master
```

推荐配置 lg 为

```
git config --global alias.lg "log --color --graph --
pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)
<%an>%Creset' --abbrev-commit"
```

.gitignore文件

一般来说每个Git项目中都需要一个“.gitignore”文件，这个文件的作用就是告诉Git哪些文件不需要添加到版本管理中。实际项目中，很多文件都是不需要版本管理的，比如Python的.pyc文件和一些包含密码的配置文件等等。这个文件的内容是一些规则，Git会根据这些规则来判断是否将文件添加到版本控制中。

下面我们看看常用的规则：

/mtk/	过滤整个文件夹
*.zip	过滤所有.zip文件
/mtk/do.c	过滤某个具体文件

很简单吧，被过滤掉的文件就不会出现在git仓库中（gitlab或github）了，当然本地库中还有，只是push的时候不会上传。需要注意的是，gitignore还可以指定要将哪些文件添加到版本管理中：

```
!*.*zip
!/mtk/one.txt
```

唯一的区别就是规则开头多了一个感叹号，Git会将满足这类规则的文件添加到版本管理中。为什么要有两种规则呢？想象一个场景：假如我们只需要管理/mtk/目录中的one.txt文件，这个目录中的其他文件都不需要管理，那么我们就需要使用：

```
/mtk/
!/mtk/one.txt
```

假设我们只有过滤规则，而没有添加规则，那么我们就需要把/mtk/目录下除了one.txt以外的所有文件都写出来！

最后需要强调的一点是，如果你不慎在创建.gitignore文件之前就push了项目，那么即使你在.gitignore文件中写入新的过滤规则，这些规则也不会起作用，Git仍然会对所有文件进行版本管理。简单来说，出现这种问题的原因就是Git已经开始管理这些文件了，所以无法再通过过滤规则过滤它们。因此一定要养成在项目开始就创建.gitignore文件的习惯，否则一旦push，处理起来会非常麻烦。

.gitignore配置文件用于配置不需要加入版本管理的文件，配置好该文件可以为版本管理带来很大的便利，以下是对于配置.gitignore的一些心得记录：

配置语法：

以斜杠“/”开头表示目录；

以星号“*”通配多个字符；

以问号“?”通配单个字符

以方括号“[]”包含单个字符的匹配列表；

以叹号“!”表示不忽略(跟踪)匹配到的文件或目录；

此外，git 对于 .ignore 配置文件是按行从上到下进行规则匹配的，意味着如果前面的规则匹配的范围更大，则后面的规则将不会生效；

规则很简单，不做过多解释，但是有时候在项目开发过程中，突然心血来潮想把某些目录或文件加入忽略规则，按照上述方法定义后发现并未生效，原因是.gitignore只能忽略那些原来没有被track的文件，如果某些文件已经被纳入了版本管理中，则修改.gitignore是无效的。那么解决方法就是先把本地缓存删除（改变成未track状态），然后再提交：

```
git rm -r --cached .  
git add .  
git commit -m 'update .gitignore'  
123
```

注意：不要误解了 .gitignore 文件的用途，该文件只能作用于 Untracked Files，也就是那些从来没有被 Git 记录过的文件（自添加以后，从未 add 及 commit 过的文件）。如果文件曾经被 Git 记录过，那么.gitignore 就对它们完全无效