

git 的安装与使用

git 提交 全部文件

1、git add .

`git add xx` 命令可以将 xx 文件添加到暂存区，如果有很多改动可以通过 `git add -A .` 来一次添加所有改变的文件。注意 `-A` 选项后面还有一个句点。`git add -A` 表示添加所有内容，`git add .` 表示添加新文件和编辑过的文件不包括删除的文件；`git add -u` 表示添加编辑或者删除的文件，不包括新添加的文件。

2、git commit -m “提交注释”

3、git push origin 分支名称，一般使用： git push origin master

4、正常来说，这三步就够了。

git 介绍

1. **分布式：**Git 版本控制系统是一个分布式的系统，是用来保存工程源代码历史状态的命令行工具。
2. **保存点：**Git 的保存点可以追踪源码中的文件，并能得到某一个时间点上的整个工程项目的状态；可以在该保存点将多人提交的源码合并，也可以回退到某一个保存点上。
3. **Git 离线操作性：**Git 可以离线进行代码提交，因此它称得上是完全的分布式处理，Git 所有的操作不需要在线进行；这意味着 Git 的速度要比 SVN 等工具快得多，因为 SVN 等工具需要在线时才能操作，如果网络环境不好，提交代码会变得非常缓慢。
4. **Git 基于快照：**SVN 等老式版本控制工具是将提交点保存成补丁文件，Git 提交是将提交点指向提交时的项目快照，提交的东西包含一些元数据（作者，日期，GPG 等）。
5. **Git 的分支和合并：**分支模型是 Git 最显著的特点，因为这改变了开发者的开发模式，SVN 等版本控制工具将每个分支都要放在不同的目录中，Git 可以在同一个目录中切换不同的分支。
6. **分支即时性：**创建和切换分支几乎是同时进行的，用户可以上传一部分分支，另外一部分分支可以隐藏在本地，不必将所有的分支都上传到 GitHub 中去。
7. **分支灵活性：**用户可以随时创建、合并、删除分支，多人实现不同的功能，可以创建多个分支进行开发，之后进行分支合并，这种方式使开发变得快速、简单、安全。

Git 工作流程

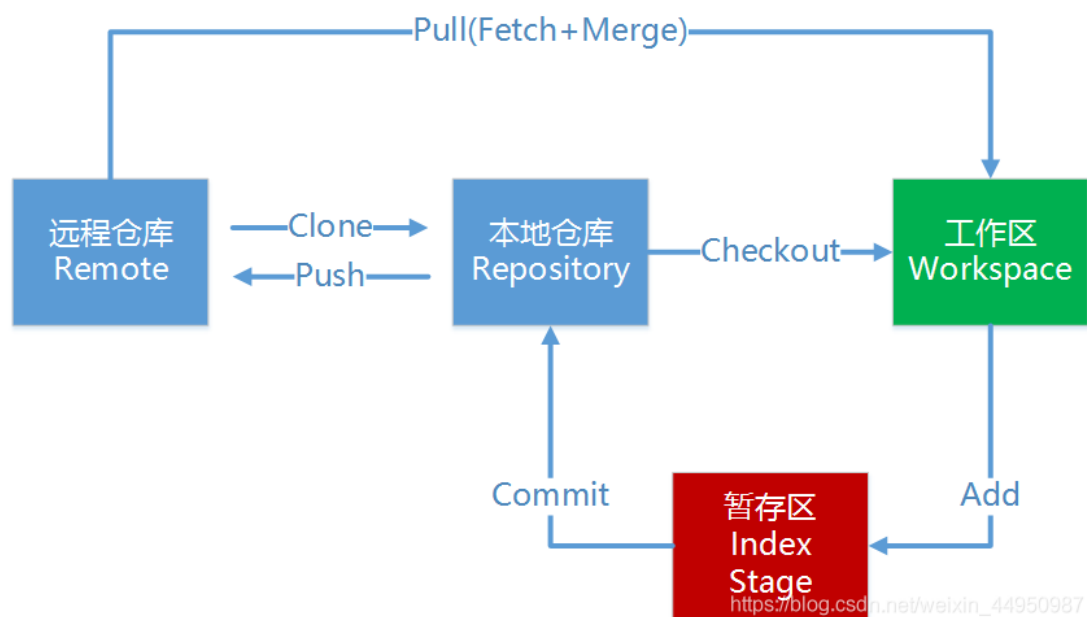
一般工作流程如下：

1. 从远程仓库中克隆 Git 资源作为本地仓库。
2. 从本地仓库中 checkout 代码然后进行代码修改

3. 在提交前先将代码提交到暂存区。
4. 提交修改。提交到本地仓库。本地仓库中保存修改的各个历史版本。
5. 在修改完成后，需要和团队成员共享代码时，可以将代码 push 到远程仓库。

下面展示了 Git 的工作流程：

Git常用命令流程图



Git 的安装

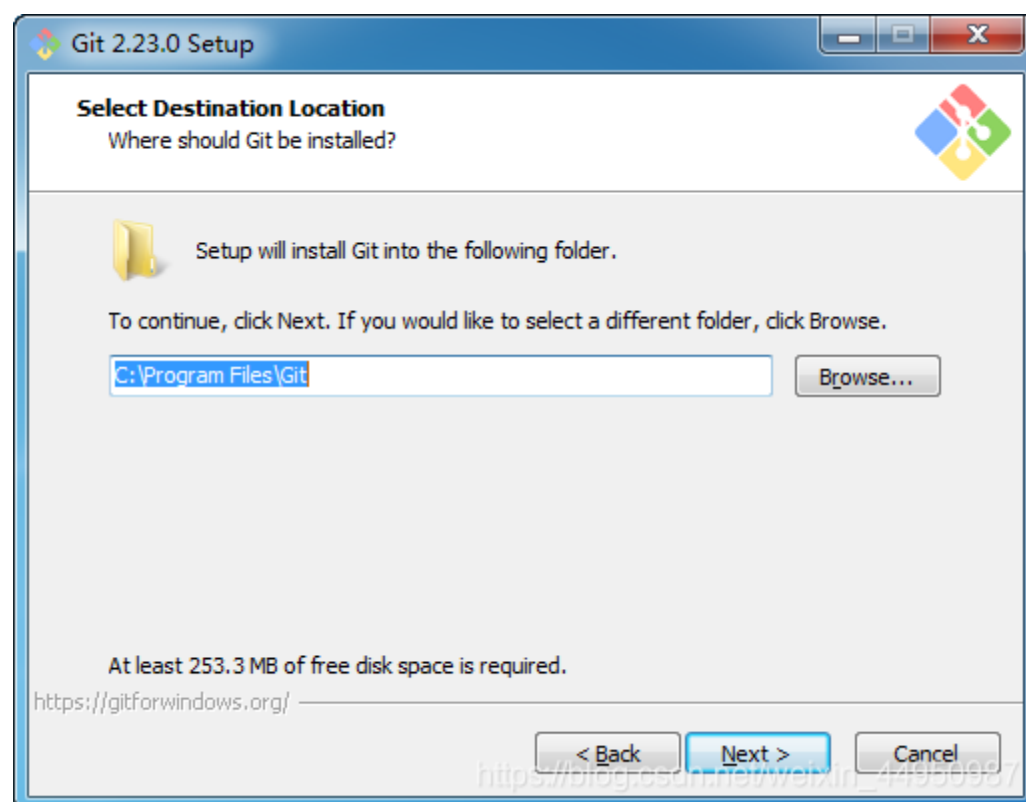
官网下载地址：<https://git-scm.com/download>

Git 客户端安装过程

1、双击安装程序“Git-2.23.0-64-bit.exe”，显示截图如下：

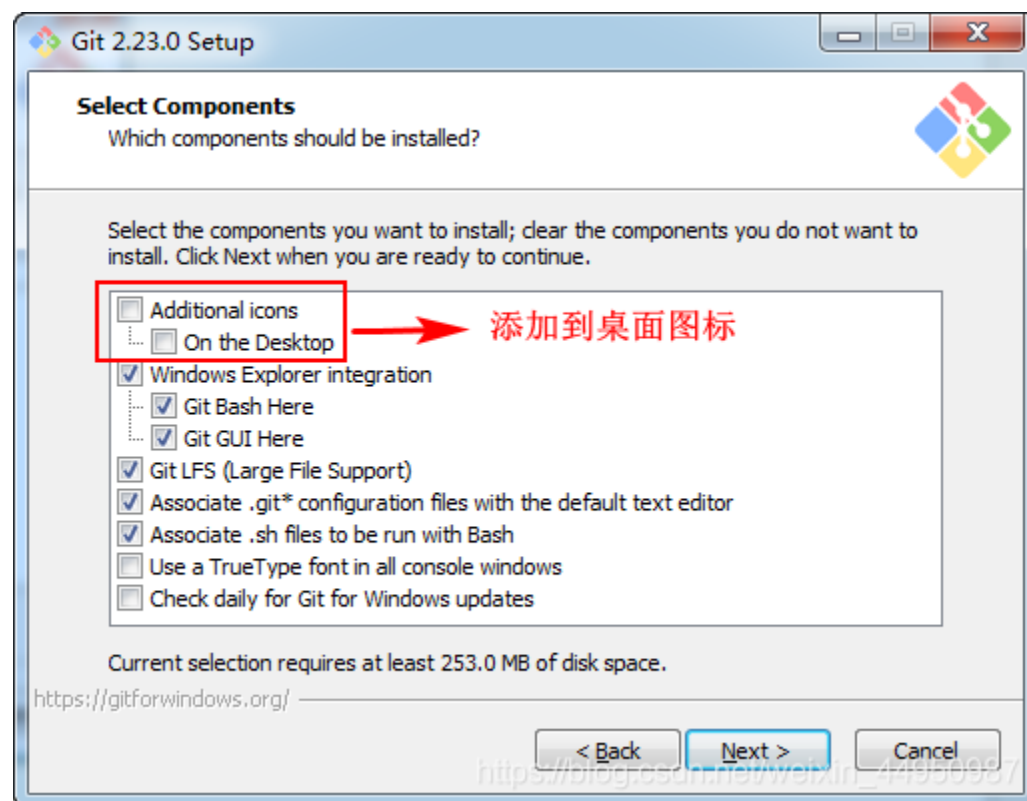


2、点击“Next”，显示截图如下：



根据自己的情况，选择程序的安装目录。

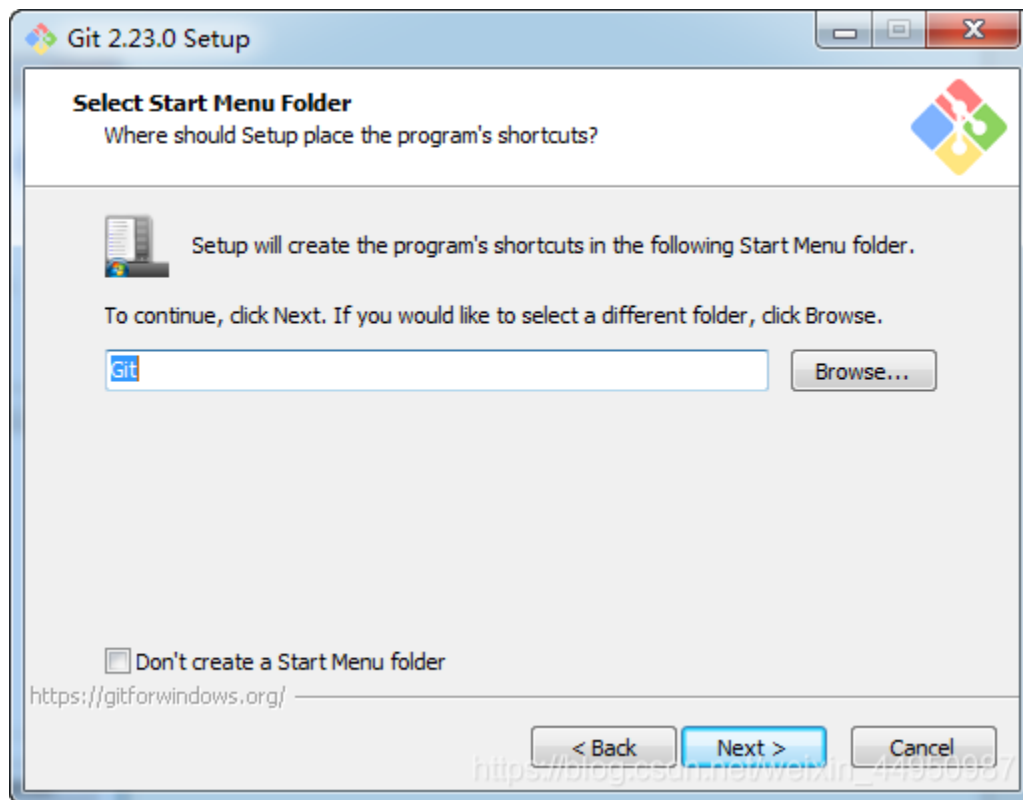
3、继续点击“Next”，显示截图如下：



说明：

- (1) 图标组件(Addition icons)：选择是否创建桌面快捷方式。
- (2) 桌面浏览(Windows Explorer integration)：浏览源码的方法，使用 bash 或者 使用 Git GUI 工具。
- (3) 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。
- (4) 关联 shell 脚本文件：是否关联 Bash 命令行执行的脚本文件。
- (5) 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。

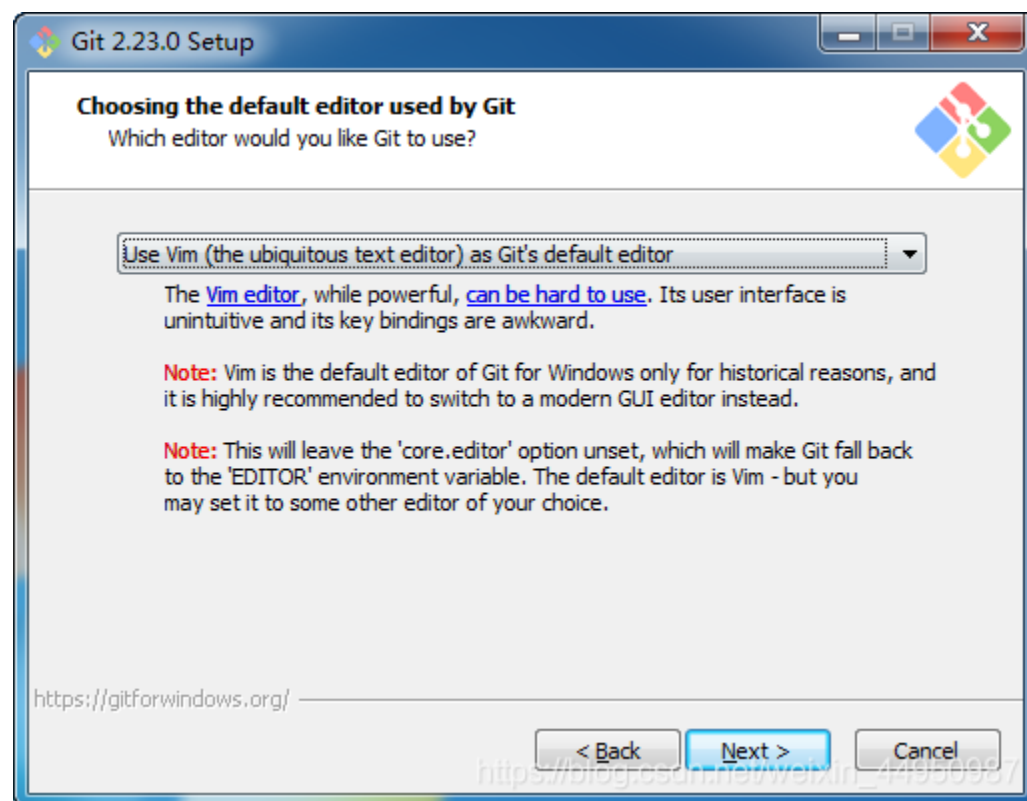
4、选择完之后，点击“Next”，显示截图如下：



开始菜单快捷方式目录：设置开始菜单中快捷方式的目录名称，也可以选择不在开始菜单中创建快捷方式。

5、点击“Next”，显示截图如下：

选择编辑器，可以选 vim，练练指令



6、点击“Next”，显示截图如下：



设置环境变量

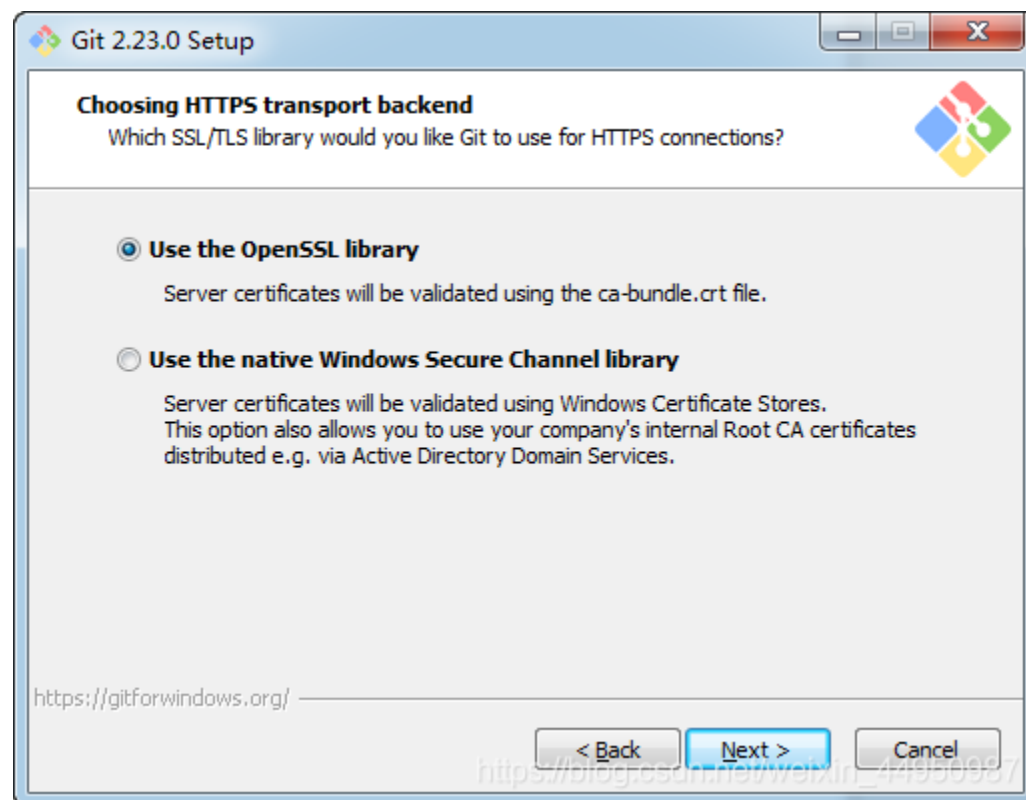
选择使用什么样的命令行工具，一般情况下我们默认使用 Git Bash 即可：

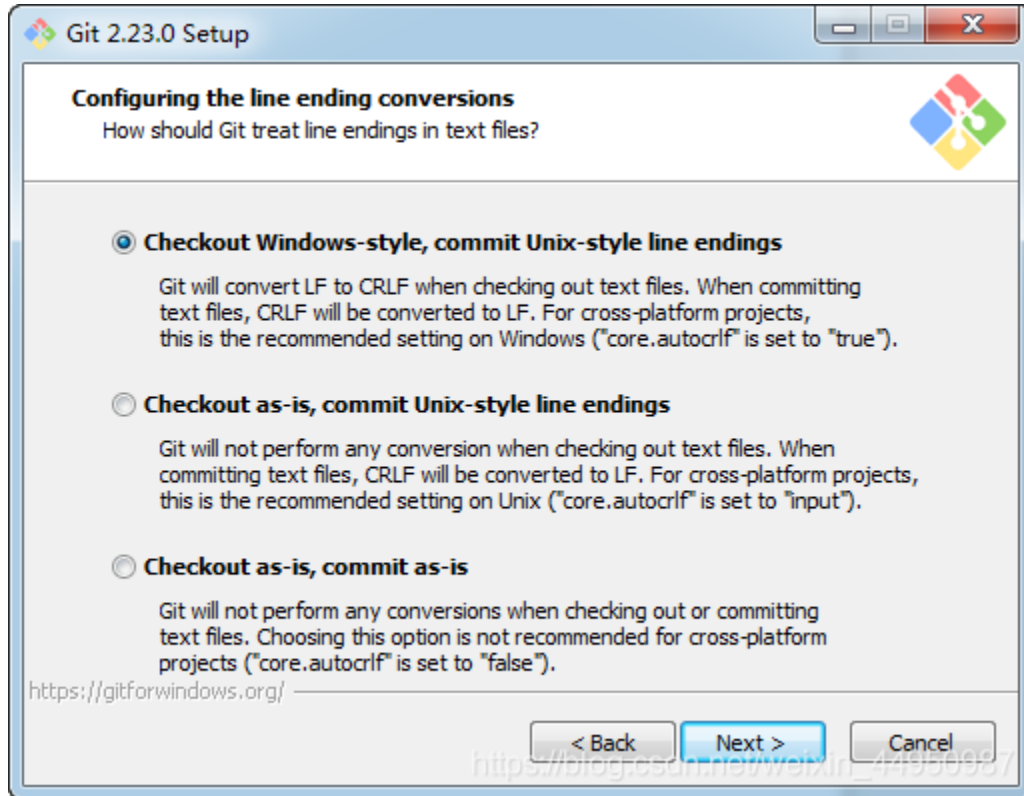
(1) Git 自带：使用 Git 自带的 Git Bash 命令行工具。

(2) 系统自带 CMD：使用 Windows 系统的命令行工具。

(3) 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。

7、选择之后，继续点击“Next”，显示如下：





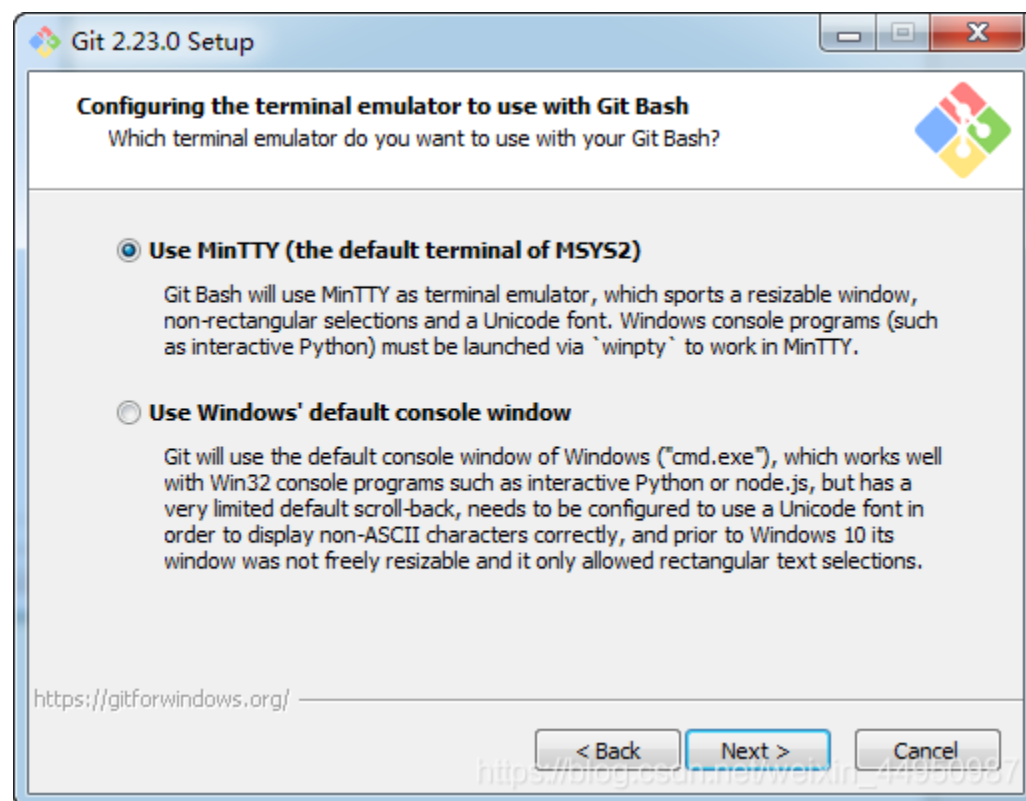
选择提交的时候换行格式

(1) 检查出 windows 格式转换为 unix 格式：将 windows 格式的换行转为 unix 格式的换行再进行提交。

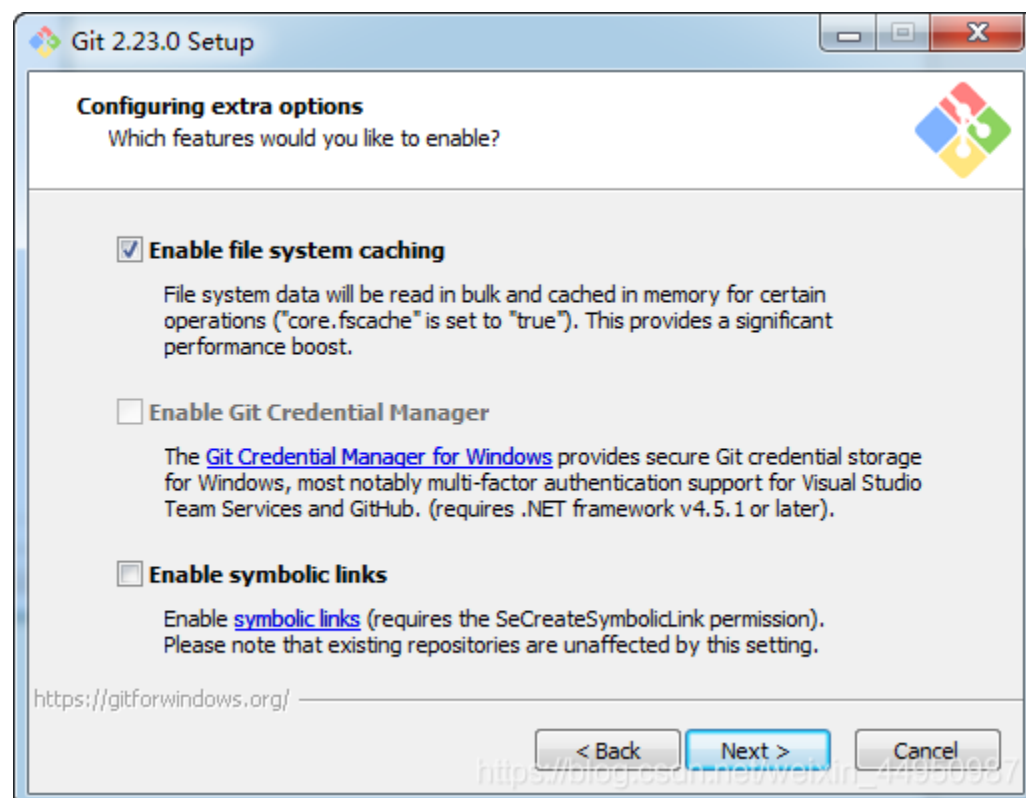
(2) 检查出原来格式转为 unix 格式：不管什么格式的，一律转为 unix 格式的换行再进行提交。

(3) 不进行格式转换：不进行转换，检查出什么，就提交什么。

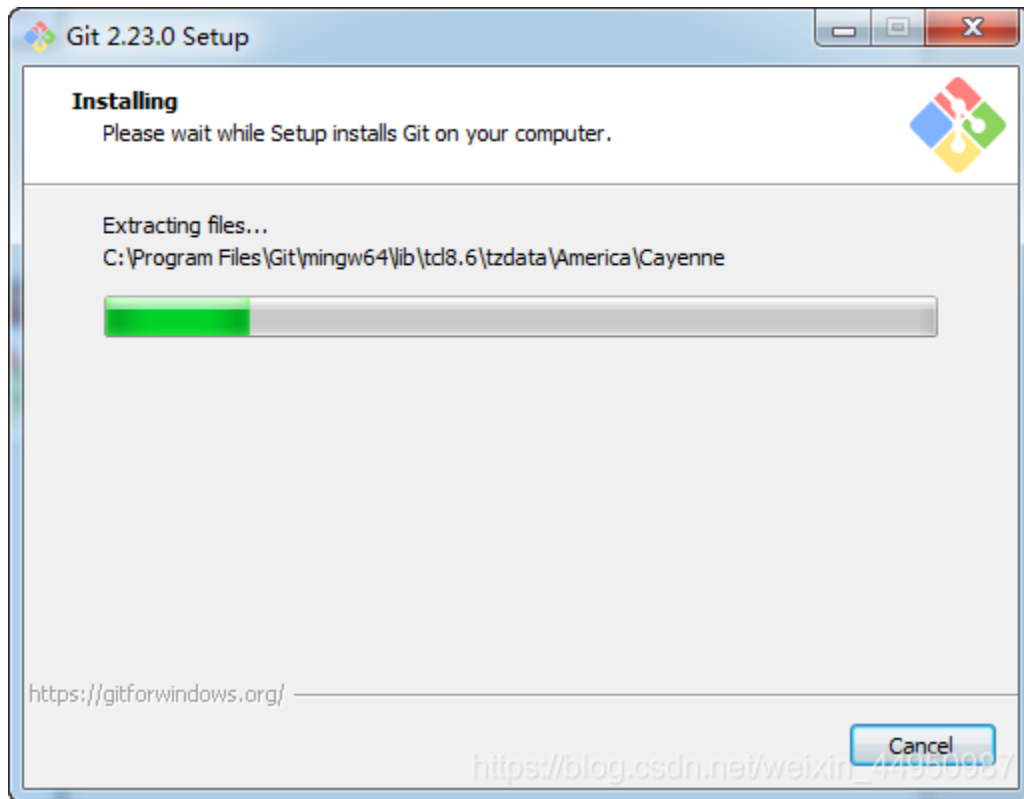
8、选择之后，点击“Next”，显示截图如下：



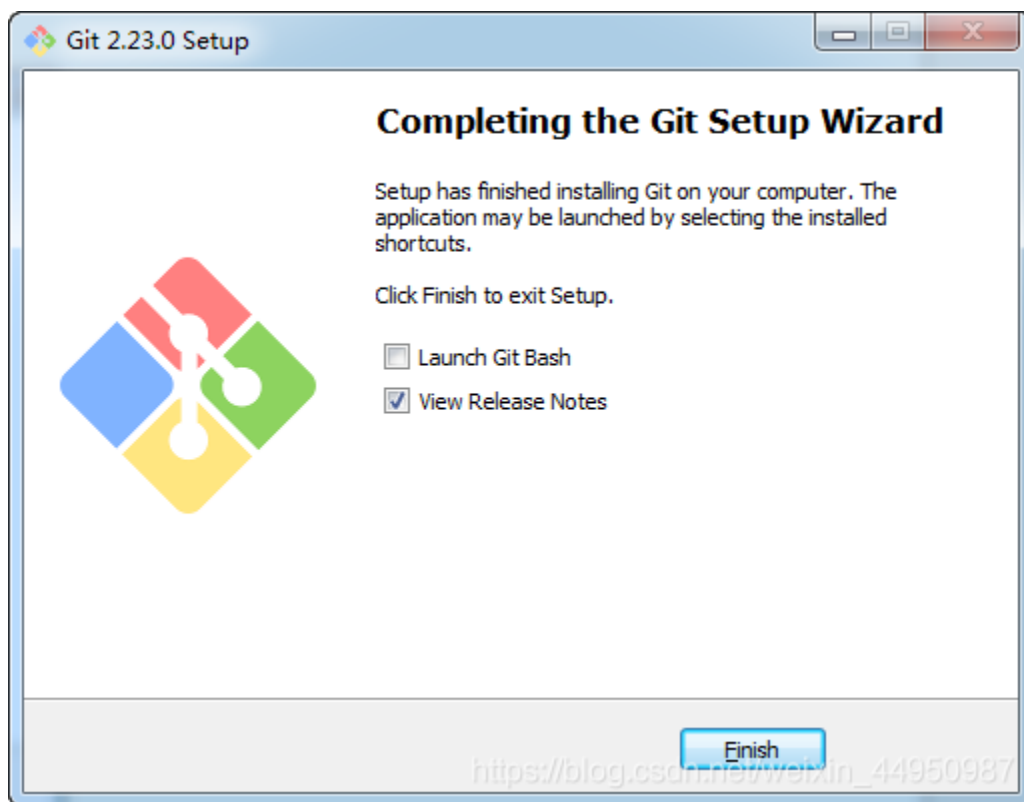
9、选择之后，点击“Next”，显示截图如下：



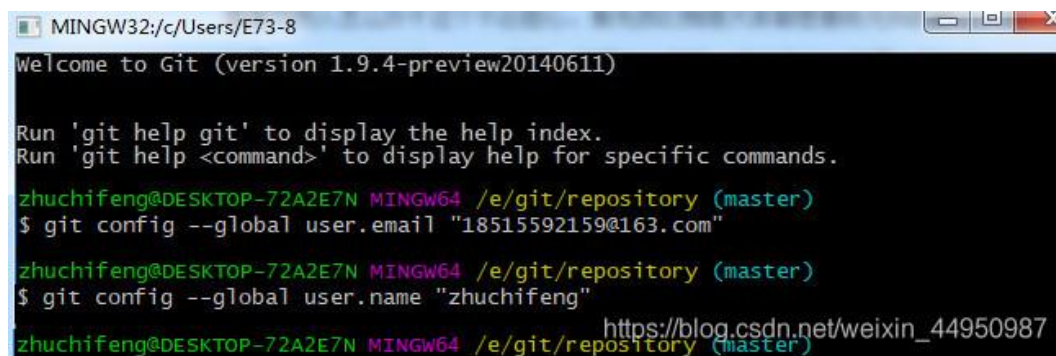
10、选择之后，点击“Install”，开始安装，截图显示如下：



11、安装完成之后，显示截图如下：



12、安装完成后，还需要最后一步设置，在命令行输入如下：

A screenshot of a Windows command prompt window titled 'MINGW32:/c/Users/E73-8'. The window shows the following text: 'Welcome to Git (version 1.9.4-preview20140611)', 'Run 'git help git' to display the help index.', 'Run 'git help <command>' to display help for specific commands.', 'zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/repository (master)', '\$ git config --global user.email "18515592159@163.com"', 'zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/repository (master)', '\$ git config --global user.name "zhuchifeng"', and 'zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/repository (master)'. A URL 'https://blog.csdn.net/weixin_44950987' is visible in the bottom right corner of the terminal window.

```
MINGW32:/c/Users/E73-8
Welcome to Git (version 1.9.4-preview20140611)

Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/repository (master)
$ git config --global user.email "18515592159@163.com"

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/repository (master)
$ git config --global user.name "zhuchifeng"

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/repository (master)
```

因为 Git 是分布式版本控制系统，所以需要填写用户名和邮箱作为一个标识。
注意：`git config --global` 参数，有了这个参数，表示你这台机器上所有的 Git 仓库都会使用这个配置，当然你也可以对某个仓库指定的不同的用户名和邮箱。

这样，我们的 Git 客户端就下载并安装完成了。

一、Git 是什么

Git 是目前世界上最先进的分布式版本控制系统。

二、SVN 与 Git 的最主要的区别

SVN 是集中式版本控制系统，版本库是集中放在中央服务器的，而干活的时候，用的都是自己的电脑，所以首先要从中央服务器哪里得到最新的版本，然后干活，干完后，需要把自己做完的活推送到中央服务器。集中式版本控制系统是必须联网才能工作，如果在局域网还可以，带宽够大，速度够快，如果在互联网下，如果网速慢的话，就纳闷了。

Git 是分布式版本控制系统，那么它就没有中央服务器的，每个人的电脑就是一个完整的版本库，这样，工作的时候就不需要联网了，因为版本都是在自己的电脑上。既然每个人的电脑都有一个完整的版本库，那多个人如何协作呢？比如说自己在电脑上改了文件 A，其他人也在电脑上改了文件 A，这时，你们两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。

三、如何操作

1、创建版本库。

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”。

所以创建一个版本库也非常简单，如下我是 E 盘 -> git 目录下新建一个 testgit 版本库。

```
MINGW64:/e/git/testgit
zhuchifeng@DESKTOP-72A2E7N MINGW64 ~/Desktop
$ cd E:
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e
$ cd git
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git
$ mkdir testgit
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git
$ cd testgit/
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit
$ pwd
/e/git/testgit
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit
$ |
```

pwd 命令是用于显示当前的目录。

(1) 通过命令 git init 把这个目录变成 git 可以管理的仓库，如下：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit
$ git init
Initialized empty Git repository in E:/git/testgit/.git/
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$
```

这时候你当前 testgit 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。如下：

电脑 > 本地磁盘 (E:) > git > testgit >

| 名称 | 修改日期 | 类型 |
|------|------------------|-----|
| .git | 2019/10/18 11:22 | 文件夹 |

(2) 把文件添加到版本库中。

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。

下面先看下 demo 如下演示：

我在版本库 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。如下：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git add readme.txt

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
```

如果和上面一样，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，把文件提交到仓库。

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git commit -m "readme.txt提交"
[master (root-commit) 3871d39] readme.txt提交
1 file changed, 1 insertion(+)
create mode 100644 readme.txt

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
```

是提交的注释

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：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git status
On branch master
nothing to commit, working tree clean

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
```

说明没有任何文件未提交，但是我现在继续来改下 readme.txt 内容，比如我在下面添加一行 2222222222 内容，继续使用 `git status` 来查看下结果，如下：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
```

上面的命令告诉我们 readme.txt 文件已被修改，但是未被提交的修改。

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:

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git diff readme.txt
diff --git a/readme.txt b/readme.txt
index 4d7eaaf..7655a69 100644
--- a/readme.txt
+++ b/readme.txt
@@ -1,2 @@
-11111111
\ No newline at end of file
+11111111
+22222222
\ No newline at end of file

https://blog.csdn.net/weixin_44950987
```

如上可以看到, `readme.txt` 文件内容从一行 `11111111` 改成 二行 添加了一行 `22222222` 内容。

知道了对 `readme.txt` 文件做了什么修改后, 我们可以放心的提交到仓库了, 提交修改和提交文件是一样的 2 步 (第一步是 `git add` 第二步是: `git commit`)。如下:

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git add readme.txt

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   readme.txt

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git commit -m "文件增加222222内容"
[master 01e15a5] 文件增加222222内容
1 file changed, 2 insertions(+), 1 deletion(-)

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git status
On branch master
nothing to commit, working tree clean

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$

https://blog.csdn.net/weixin_44950987
```

2、版本回退

如上, 我们已经学会了修改文件, 现在我继续对 `readme.txt` 文件进行修改, 再增加一行

内容为 33333333333333. 继续执行命令如下:

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git add readme.txt

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git commit -m "添加readme.txt文件内容为33333333"
[master b8a7cf3] 添加readme.txt文件内容为33333333
1 file changed, 2 insertions(+), 1 deletion(-)

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ |
```

现在我已经对 readme.txt 文件做了三次修改了, 那么我现在想查看下历史记录, 如何查呢? 我们现在可以使用命令 `git log` 演示如下所示:

```
1 file changed, 2 insertions(+), 1 deletion(-)

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git log
commit b8a7cf3ecd348b2537511f707b72e1ce47e0d2c6 (HEAD -> master)
Author: zhuchifeng <18515592159@163.com>
Date: Fri Oct 18 11:47:37 2019 +0800

    添加readme.txt文件内容为33333333

commit 01e15a54b59c841ff7e101014dae1747c7f7cf48
Author: zhuchifeng <18515592159@163.com>
Date: Fri Oct 18 11:40:04 2019 +0800

    文件增加222222内容

commit 3871d398fc8b4674cc6e5780da0a20398ddace40
Author: zhuchifeng <18515592159@163.com>
Date: Fri Oct 18 11:29:45 2019 +0800

    readme.txt提交

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ |
```

每次提交的版本号

最近一次增加内容为33333333

上一次提交增加的内容为22222222

https://blog.csdn.net/weixin_44950987

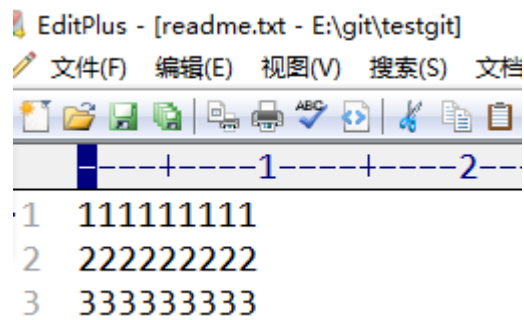
`git log` 命令显示从最近到最远的显示日志, 我们可以看到最近三次提交, 最近的一次是, 增加内容为 333333. 上一次是添加内容 222222, 第一次默认是 111111. 如果嫌上面显示的信息太多的话, 我们可以使用命令 `git log --pretty=oneline` 演示如下:

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git log --pretty=oneline
b8a7cf3ecd348b2537511f707b72e1ce47e0d2c6 (HEAD -> master) 添加readme.txt文件内容为33333333
01e15a54b59c841ff7e101014dae1747c7f7cf48 文件增加222222内容
3871d398fc8b4674cc6e5780da0a20398ddace40 readme.txt提交

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ |
```

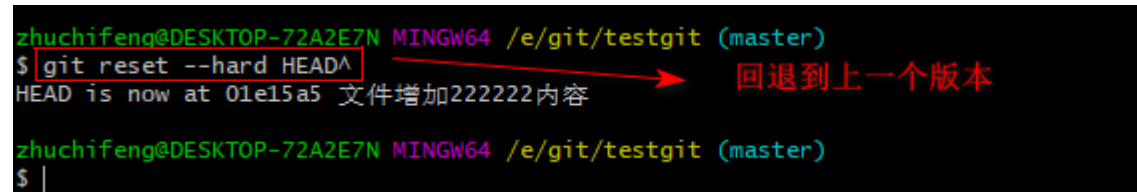
现在我想使用版本回退操作, 我想把当前的版本回退到上一个版本, 要使用什么命令呢? 可以使用如下 2 种命令, 第一种是: `git reset --hard HEAD^` 那么如果要回退到上上个版本只需把 `HEAD^` 改成 `HEAD^^` 以此类推. 那如果要回退到前 100 个版本的话, 使用上面的方法肯定不方便, 我们可以使用下面的简便命令操作: `git reset --hard HEAD~100` 即可. 未回退之前的

readme.txt 内容如下：

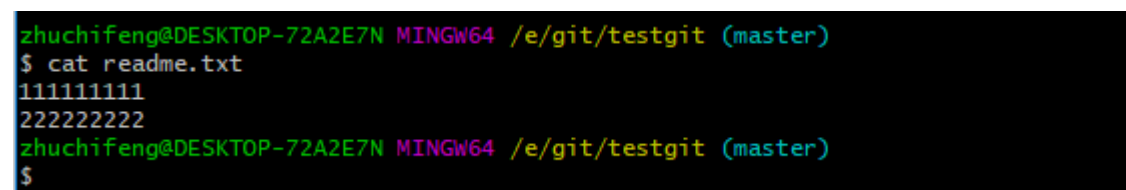
A screenshot of the EditPlus text editor. The title bar reads 'EditPlus - [readme.txt - E:\git\testgit]'. The menu bar includes '文件(F)', '编辑(E)', '视图(V)', '搜索(S)', and '文档'. The toolbar contains icons for file operations like opening, saving, and printing. The main text area shows a file named 'readme.txt' with the following content:

```
1 111111111
2 222222222
3 333333333
```

如果想回退到上一个版本的命令如下操作：

A terminal window screenshot showing a git command. The prompt is 'zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)'. The user enters '\$ git reset --hard HEAD^'. Below the command, a message says 'HEAD is now at 01e15a5 文件增加222222内容'. A red arrow points from the command to the text '回退到上一个版本' (Reset to the previous version). The prompt returns to '\$ |'.

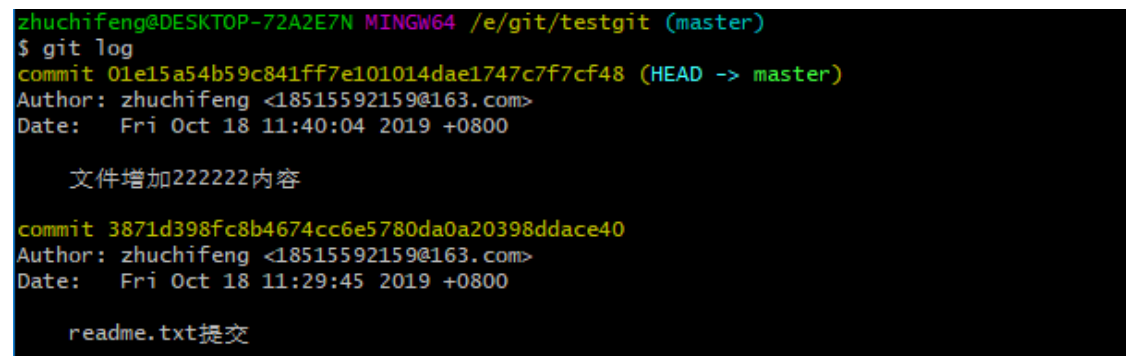
再来查看下 readme.txt 内容如下：通过命令 `cat readme.txt` 查看

A terminal window screenshot showing the output of the 'cat' command. The prompt is 'zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)'. The user enters '\$ cat readme.txt'. The output shows the first two lines of the file:

```
111111111
222222222
```

The prompt returns to '\$'.

可以看到，内容已经回退到上一个版本了。我们可以继续使用 `git log` 来查看下历史记录信息，如下：

A terminal window screenshot showing the output of the 'git log' command. The prompt is 'zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)'. The user enters '\$ git log'. The output shows two commits. The first commit is highlighted in green: 'commit 01e15a54b59c841ff7e101014dae1747c7f7cf48 (HEAD -> master)' with author 'zhuchifeng' and date 'Fri Oct 18 11:40:04 2019 +0800'. Below it, a red annotation says '文件增加222222内容'. The second commit is 'commit 3871d398Fc8b4674cc6e5780da0a20398ddace40' with author 'zhuchifeng' and date 'Fri Oct 18 11:29:45 2019 +0800'. Below it, a red annotation says 'readme.txt提交'.

我们看到 增加 333333 内容我们没有看到了，但是现在我想回退到最新的版本，如：有 333333 的内容要如何恢复呢？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` 演示如下：


```

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git reflog
01e15a5 (HEAD -> master) HEAD@{0}: reset: moving to HEAD^
b8a7cf3 HEAD@{1}: commit: 添加readme.txt文件内容为3333333
01e15a5 (HEAD -> master) HEAD@{2}: commit: 文件增加222222内容
3871d39 HEAD@{3}: commit (initial): readme.txt提交

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ |

```

通过上面的显示我们可以知道，增加内容 3333 的版本号是 b8a7cf3. 我们现在可以使用命令

`git reset --hard b8a7cf3` 来恢复了。演示如下：

```

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git reflog
01e15a5 (HEAD -> master) HEAD@{0}: reset: moving to HEAD^
b8a7cf3 HEAD@{1}: commit: 添加readme.txt文件内容为3333333
01e15a5 (HEAD -> master) HEAD@{2}: commit: 文件增加222222内容
3871d39 HEAD@{3}: commit (initial): readme.txt提交

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git reset --hard b8a7cf3
HEAD is now at b8a7cf3 添加readme.txt文件内容为3333333

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ cat readme.txt
111111111
222222222
333333333

```

查看readme.txt内容如下

https://blog.csdn.net/weixin_44950987

可以看到 目前已经是最新的版本了。

3、理解工作区与暂存区的区别

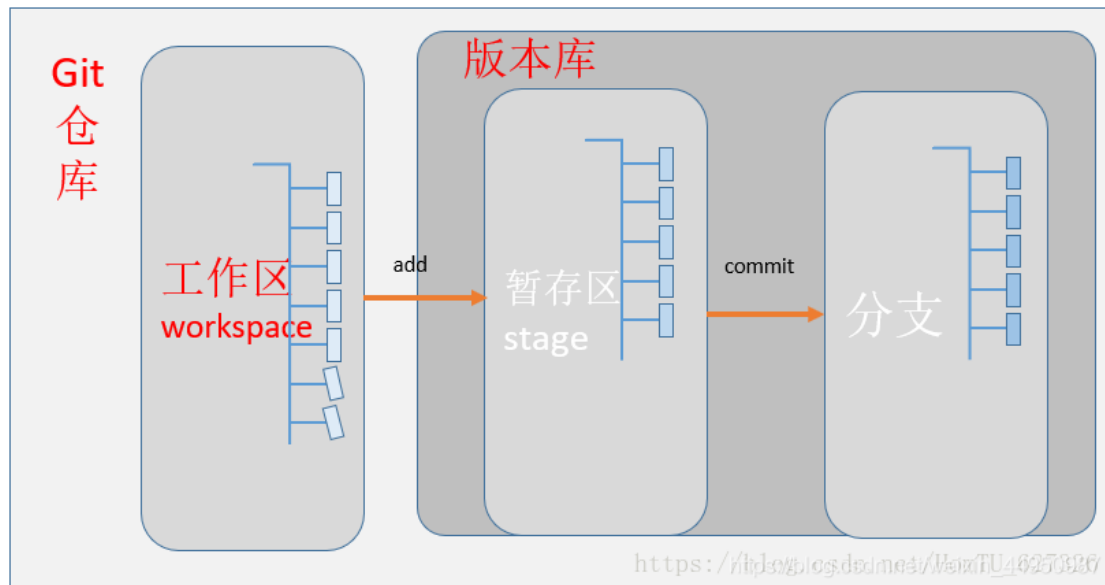
工作区：就是你在电脑上看到的目录，比如目录下 testgit 里的文件(.git 隐藏目录版本库除外)。或者以后需要再新建的目录文件等等都属于工作区范畴。

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)(或者叫 index)，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。

我们前面说过使用 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。



我们继续使用 demo 来演示下：

我们在 readme.txt 再添加一行内容为 4444444，接着在目录下新建一个文件为 test.txt 内容为 test，我们先用命令 `git status` 来查看下状态，如下：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   readme.txt
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    test.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

修改文件
新增文件

https://blog.csdn.net/weixin_44950987

现在我们先使用 `git add` 命令把 2 个文件都添加到暂存区中，再使用 `git status` 来查看下状态，如下：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git add readme.txt

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git add test.txt

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   readme.txt
    new file:   test.txt
```

https://blog.csdn.net/weixin_44950987

接着我们可以使用 `git commit` 一次性提交到分支上，如下：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git commit -m "一次性提交所有文件，包括新建文件test.txt"
[master 99e33c7] 一次性提交所有文件，包括新建文件test.txt
2 files changed, 3 insertions(+), 1 deletion(-)
create mode 100644 test.txt

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git status
On branch master
nothing to commit, working tree clean

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
```

四、Git 撤销修改和删除文件操作

1、撤销修改

比如我现在在 `readme.txt` 文件里面增加一行 内容为 555555555555，我们先通过命令查看如下：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ cat readme.txt
111111111
222222222
333333333
444444444
555555555
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
```

在我未提交之前，我发现添加 555555555555 内容有误，所以我得马上恢复以前的版本，现在我可以有如下几种方法可以做修改：

第一：如果我知道要删掉那些内容的话，直接手动更改去掉那些需要的文件，然后 `add` 添加到暂存区，最后 `commit` 掉。

第二：我可以按以前的方法直接恢复到上一个版本。使用 `git reset --hard HEAD^`

但是现在我不想使用上面的 2 种方法，我想直接使用撤销命令该如何操作呢？首先在做撤销之前，我们可以先用 `git status` 查看下当前的状态。如下所示：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
```

可以发现，Git 会告诉你，`git restore - file` 可以丢弃工作区的修改，如下命令：

`git restore -- readme.txt`, 如下所示:

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git restore -- readme.txt

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ cat readme.txt
111111111
222222222
333333333
444444444
—————> 内容55555555没有了

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
```

命令 `git restore -- 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:

1. `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。
2. 另外一种 `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。

对于第二种情况, 我想我们继续做 demo 来看下, 假如现在我对 `readme.txt` 添加一行 内容为 `66666666666666`, 我 `git add` 增加到暂存区后, 接着添加内容 `77777777`, 我想通过撤销命令让其回到暂存区后的状态。如下所示:

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ cat readme.txt
111111111
222222222
333333333
444444444
666666666
—————> 添加一条内容为666666666

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git add readme.txt
—————> 先放到暂存区

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ cat readme.txt
111111111
222222222
333333333
444444444
666666666

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ cat readme.txt
111111111
222222222
333333333
444444444
666666666
777777777
—————> 接着添加内容777777777, 但是没有添加到暂存区

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git restore -- readme.txt
—————> 直接使用撤销命令, 把未添加到暂存区的内容撤销掉

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ cat readme.txt
111111111
222222222
333333333
444444444
666666666
—————> 继续查看下内容, 发现内容77777777已经撤销掉了

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (https://blog.csdn.net/weixin_44950987)
$
```

注意: 命令 `git restore-- readme.txt` 中的 `--` 很重要, 如果没有 `--` 的话, 那么命令变成创建分支了。

2、删除文件

假如我现在版本库 testgit 目录添加一个文件 b.txt, 然后提交。如下:

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git add b.txt
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git commit -m "添加b.txt文件"
[master 7ab9b84] 添加b.txt文件
2 files changed, 2 insertions(+), 1 deletion(-)
create mode 100644 b.txt
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ rm b.txt
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    b.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

添加b.txt文件
接着, 提交b.txt文件
我们可以直接在目录下删除文件或者使用命令rm b.txt
继续查看 b.txt文件已经删除了, 此时有两个选择,
一: 直接commit掉。
二: 从版本库中恢复被删掉的文件

https://blog.csdn.net/weixin_44950987

如上: 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,

电脑 > 本地磁盘 (E:) > git > testgit

| 名称 | 修改日期 | 类型 | 大小 |
|------------|------------------|------|------|
| .git | 2019/10/18 15:39 | 文件夹 | |
| readme.txt | 2019/10/18 15:30 | 文本文档 | 1 KB |
| test.txt | 2019/10/18 14:30 | 文本文档 | 1 KB |

只要没有 `commit` 之前, 如果我想在版本库中恢复此文件如何操作呢?

可以使用如下命令 `git restore -- b.txt`, 如下所示:

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git restore -- b.txt
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
```

恢复b.txt文件

再来看看我们 testgit 目录, 添加了 1 个文件了。如下所示:

电脑 > 本地磁盘 (E:) > git > testgit

| 名称 | 修改日期 | 类型 | 大小 |
|------------|------------------|------|------|
| .git | 2019/10/18 15:48 | 文件夹 | |
| b.txt | 2019/10/18 15:48 | 文本文档 | 0 KB |
| readme.txt | 2019/10/18 15:30 | 文本文档 | 1 KB |
| test.txt | 2019/10/18 14:30 | 文本文档 | 1 KB |

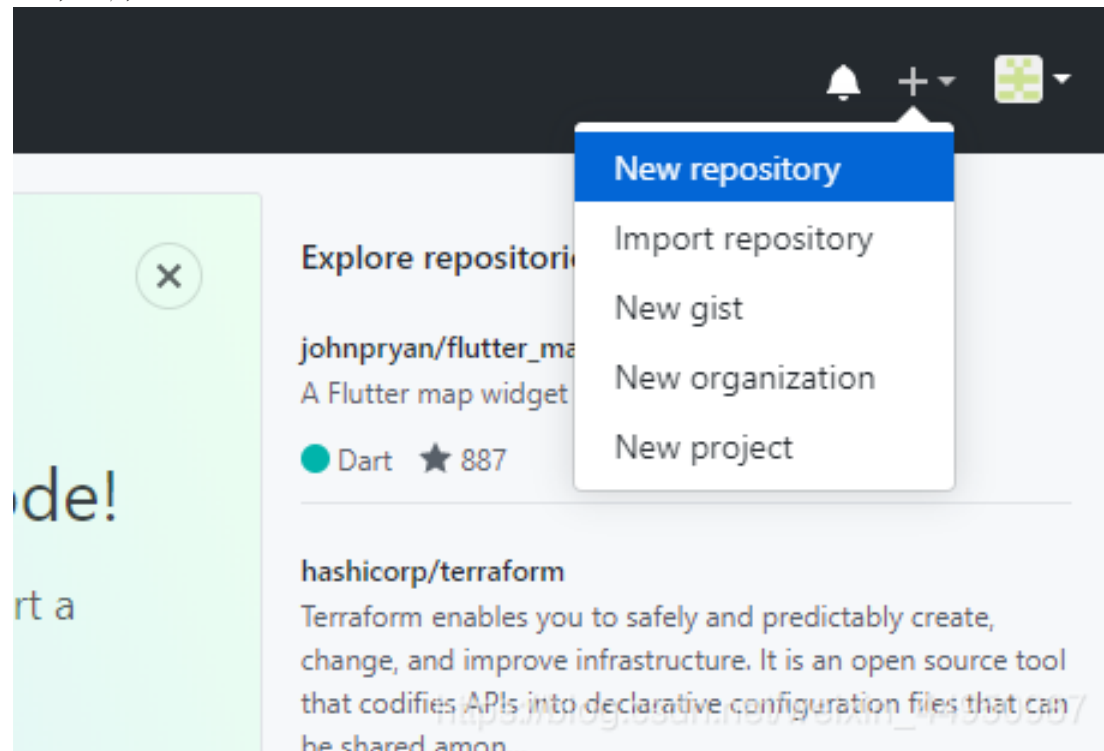
五、远程仓库

现在我们已经在本地创建了一个 Git 仓库, 又想让其他人来协作开发, 此时就可以把本地仓库同步到远程仓库, 同时还增加了本地仓库的一个备份。

常用的远程仓库就是 github: <https://github.com/>，接下来我们演示如何将本地代码同步到 github。

在 github 上创建仓库

首先你得在 github 上创建一个账号，这个就不演示了。然后在 github 上创建一个仓库：



Create a new repository

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

Owner

Repository name *



18515592159zhu ▾

/

testgit



Great repository names are short and memorable. Need inspiration? How about **fuzzy-spork**?

Description (optional)



Public

Anyone can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

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

☐ Initialize this repository with a README

This will let you immediately clone the repository to your computer.

Add .gitignore: None ▾

Add a license: None ▾



Create repository

https://blog.csdn.net/weixin_44950987

18515592159zhu / testgit

Watch 0 Star 0 Fork 0

[Code](#) [Issues 0](#) [Pull requests 0](#) [Projects 0](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

Quick setup — if you've done this kind of thing before

[Set up in Desktop](#) or [HTTPS](#) [SSH](#) <https://github.com/18515592159zhu/testgit.git>

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# testgit" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/18515592159zhu/testgit.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/18515592159zhu/testgit.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)

https://blog.csdn.net/weixin_44950987

点击“create repository”按钮仓库就创建成功了。

Github 支持两种同步方式“https”和“ssh”。

如果使用 https 很简单基本不需要配置就可以使用，但是每次提交代码和下载代码时都需要输入用户名和密码。

如果使用 ssh 方式就需要客户端先生成一个密钥对，即一个公钥一个私钥。然后还需要把公钥放到 github 的服务器上。

这两种方式在实际开发中都应用，所以我們都需要掌握。接下来我们先看 ssh 方式。

什么是 SSH

SSH 为 Secure Shell（安全外壳协议）的缩写，由 IETF 的网络小组（Network Working Group）所制定。

SSH 是目前较可靠，专为远程登录会话和其他网络服务提供安全性的协议。利用 SSH 协议可以有效防止远程管理过程中的信息泄露问题。

- 1
- 2
- 3

基于密钥的安全验证

使用 ssh 协议通信时，推荐使用基于密钥的验证方式。你必须为自己创建一对密钥，并把公用密钥放在需要访问的服务器上。

如果你要连接到 SSH 服务器上，客户端软件就会向服务器发出请求，请求用你的密钥进行安全验证。服务器收到请求之后，先在该服务器上你的主目录下寻找你的公用密钥，然后把它和你发送过来的公用密钥进行比较。

如果两个密钥一致，服务器就用公用密钥加密“质询”（challenge）并把它发送给客户端软件。

客户端软件收到“质询”之后就可以用你的私人密钥解密再把它发送给服务器。

- 1
- 2
- 3
- 4

ssh 密钥生成

在了解之前，先注册 github 账号，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：

`git bash` 执行命令, 生成公钥和私钥, 命令: `ssh-keygen -t rsa`

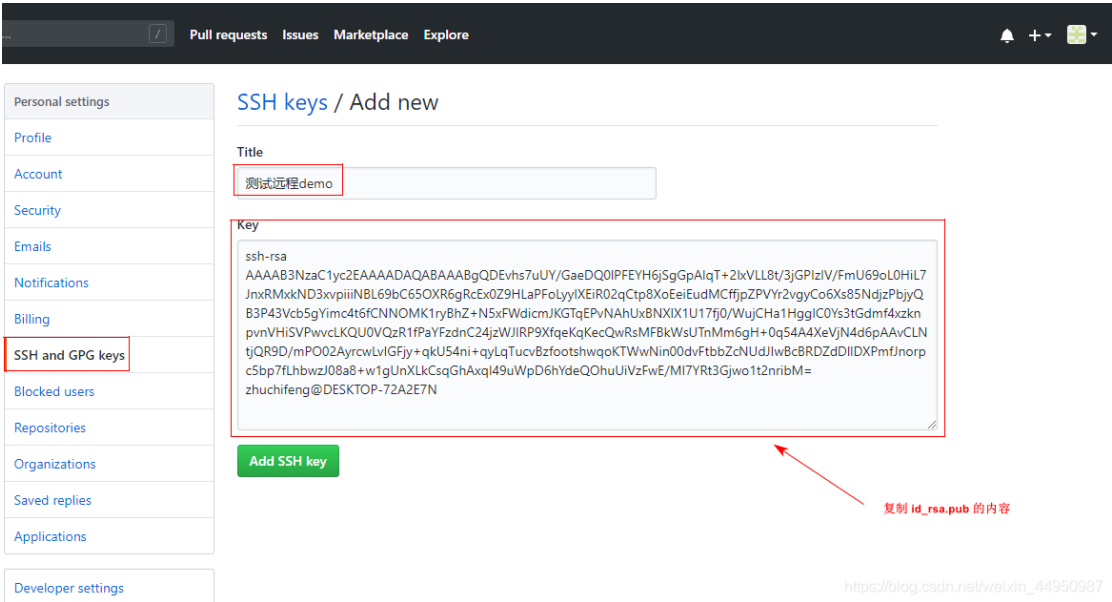
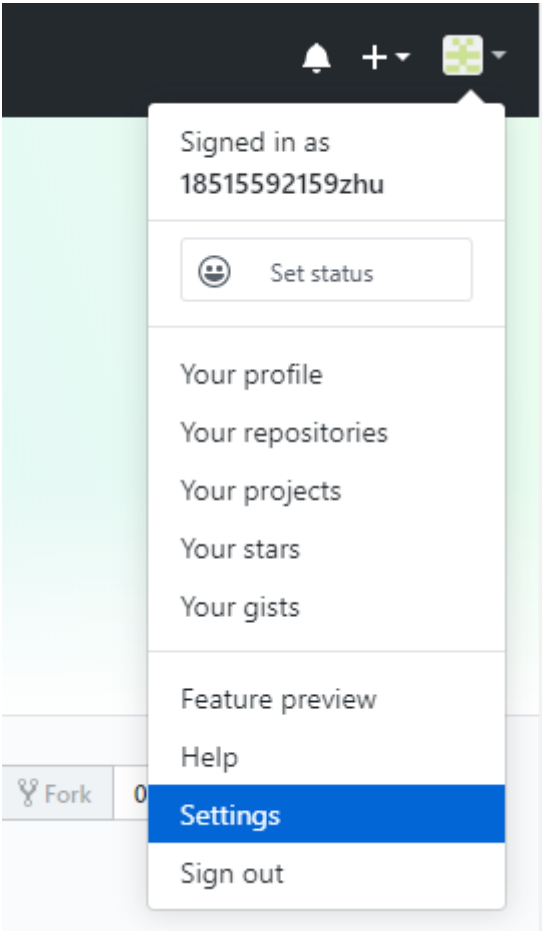
```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/zhuchifeng/.ssh/id_rsa):
Created directory '/c/Users/zhuchifeng/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/zhuchifeng/.ssh/id_rsa.
Your public key has been saved in /c/Users/zhuchifeng/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:0c95HZw4dbyRw42SkNVhc6j11K80m1g5y5n6JfSzBBM zhuchifeng@DESKTOP-72A2E7N
The key's randomart image is:
+---[RSA 3072]-----+
|      .o..+oo|
|      .. o===|
|      E oooB=|
|      . ..= o*|
|      S +.o +o|
|      +=+.o..|
|      +=B*o..|
|      . B= o..|
|      .o. .  |
+---[SHA256]-----+
https://blog.csdn.net/weixin_44950987
```

执行命令完成后, 在 window 本地用户 .ssh 目录 `C:\Users\用户名\.ssh` 下面生成如下名称的公钥和私钥:

| 电脑 > 本地磁盘 (C:) > 用户 > zhuchifeng > .ssh | | | | |
|---|------------------|---------------------|------|--|
| 名称 | 修改日期 | 类型 | 大小 | |
|  id_rsa 私钥 | 2019/10/18 16:04 | 文件 | 3 KB | |
|  id_rsa.pub 公钥 | 2019/10/18 16:04 | Microsoft Publis... | 1 KB | |

ssh 密钥配置

密钥生成后需要在 github 上配置密钥本地才可以顺利访问。



在 key 部分将 `id_rsa.pub` 文件内容添加进去，然后点击 “Add SSH key” 按钮

完成配置。

SSH keys

[New SSH key](#)

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

**测试远程demo**
ff:4a:0e:fd:97:8b:78:31:23:fc:f0:8b:f2:02:17:39
Added on 18 Oct 2019
Never used — Read/write

Delete

Check out our guide to [generating SSH keys](#) or [troubleshoot common SSH Problems](#).

GPG keys

[New GPG key](#)

There are no GPG keys associated with your account.

Learn how to [generate a GPG key and add it to your account](#).

https://blog.csdn.net/weixin_44950987

同步到远程仓库

同步到远程仓库可以使用 `git bash`。

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 (`E:\git\testgit`) 运行命令：

```
git remote add origin https://github.com/18515592159zhu/testgit.git
git push -u origin master
```

所有的如下：

如果出现如下错误：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git remote add origin git@github.com:18515592159zhu/testgit.git
fatal: remote origin already exists.

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
```

可以先执行如下命令，然后再执行上面的命令

```
git remote rm origin
```

```
$ git remote rm origin

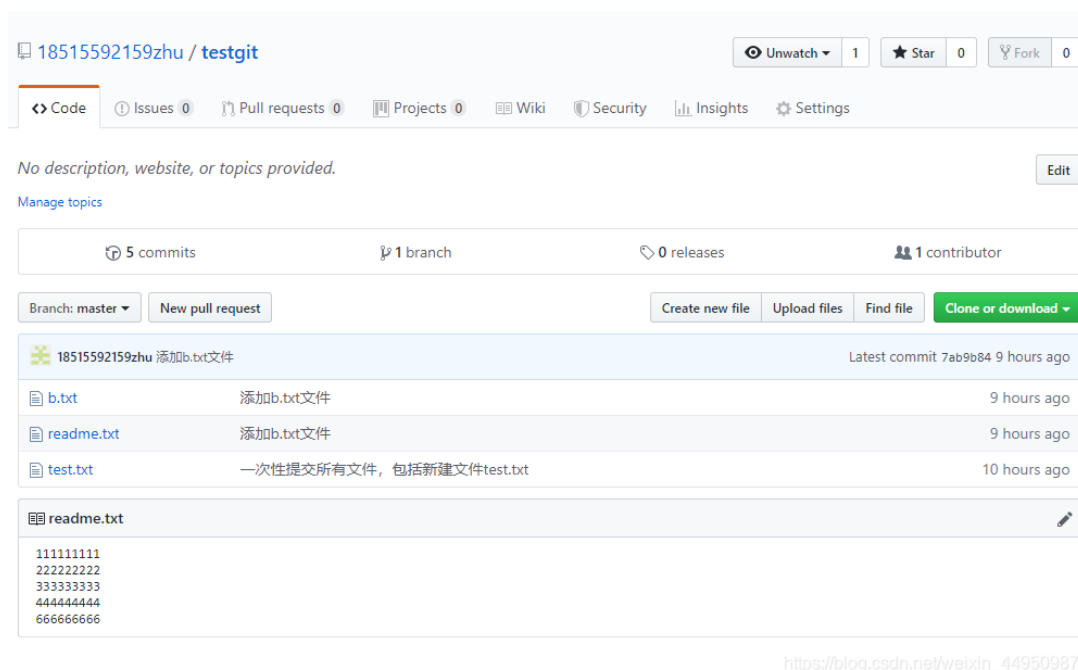
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git remote add origin git@github.com:18515592159zhu/testgit.git

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git push -u origin master
The authenticity of host 'github.com (13.250.177.223)' can't be established.
RSA key fingerprint is SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com,13.250.177.223' (RSA) to the list of known hosts.
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 4 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (17/17), 1.33 KiB | 195.00 KiB/s, done.
Total 17 (delta 0), reused 0 (delta 0)
To github.com:18515592159zhu/testgit.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ |
```

把本地库的内容推送到远程，使用 `git push` 命令，实际上是把当前分支 `master` 推送到远程。

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：



从现在起，只要本地作了提交，就可以通过如下命令：`git push origin master`

把本地 master 分支的最新修改推送到 github 上了，现在你就拥有了真正的分布式版本库了。

如何从远程库克隆

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 如何克隆呢？

首先，登录 github，创建一个新的仓库，名字叫 `testgit2`。如下：

Issues Marketplace Explore

第一步 New repository

Create a new repository

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

Owner 18515592159zhu / Repository name * testgit2 第二步，填入仓库名字

Great repository names are short and memorable. Need inspiration? How about solid-bassoon?

Description (optional)

Public Anyone can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

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

☒ Initialize this repository with a README This will let you immediately clone the repository to your computer. 勾选此按钮，会自动生成README.md文件

Add .gitignore: None Add a license: None

Create repository 第四步，点击此按钮

https://blog.csdn.net/weixin_44950987

如下，我们看到：

18515592159zhu / testgit2

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Security Insights Settings

No description, website, or topics provided. Edit

Manage topics

1 commit 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

18515592159zhu Initial commit Latest commit a03eac3 now

README.md Initial commit now

README.md

testgit2

https://blog.csdn.net/weixin_44950987

现在，远程库已经准备好了，下一步是使用命令 `git clone` 克隆一个本地库

了。如下所示：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git clone git@github.com:18515592159zhu/testgit2.git
Cloning into 'testgit2'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$
```

接着在我本地目录下 生成 testgit2 目录了，如下所示：

电脑 > 本地磁盘 (E:) > git > testgit > testgit2 >

| 名称 | 修改日期 | 类型 | 大小 |
|-----------|-----------------|--------------|------|
| .git | 2019/10/19 0:21 | 文件夹 | |
| README.md | 2019/10/19 0:21 | Markdown 源文件 | 1 KB |

六、创建与合并分支

在版本回填退里，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。

首先，我们来创建 dev 分支，然后切换到 dev 分支上。如下操作：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git checkout -b dev
Switched to a new branch 'dev'

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git branch
* dev
  master

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$
```

`git checkout` 命令加上 `-b` 参数表示创建并切换，相当于如下 2 条命令：

```
git branch dev
```

```
git checkout dev
```

`git branch` 查看分支，会列出所有的分支，当前分支前面会添加一个星号。然后在 dev 分支上继续做 demo，比如我们现在在 `readme.txt` 再增加一行
7777777777777777

首先我们先来查看下 readme.txt 内容，接着添加内容 77777777，如下：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ cat readme.txt
111111111
222222222
333333333
444444444
666666666
dev分支上未添加内容之前

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ cat readme.txt
111111111
222222222
333333333
444444444
666666666
777777777
添加内容之后

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git add readme.txt

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git commit -m "dev分支上增加内容777777"
[dev d1f8e4a] dev分支上增加内容777777
1 file changed, 2 insertions(+), 1 deletion(-)

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ |
```

现在 dev 分支工作已完成，现在我们切换到主分支 master 上，继续查看 readme.txt 内容如下：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ cat readme.txt
111111111
222222222
333333333
444444444
666666666
查看内容777777不见了，因为不是分支dev

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ |
```

现在我们可以把 dev 分支上的内容合并到分支 master 上了，可以在 master 分支上，使用如下命令 `git merge dev` 如下所示：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git merge dev
Updating 7ab9b84..d1f8e4a
Fast-forward
 readme.txt | 3 ++-
1 file changed, 2 insertions(+), 1 deletion(-)

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ cat readme.txt
111111111
222222222
333333333
444444444
666666666
777777777
继续查看内容，多了一条77777777

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ |
```

git merge 命令用于合并指定分支到当前分支上，合并后，再查看 readme.txt 内容，可以看到，和 dev 分支最新提交的是完全一样的。

注意到上面的 Fast-forward 信息，Git 告诉我们，这次合并是“快进模式”，也就是直接把 master 指向 dev 的当前提交，所以合并速度非常快。

合并完成后，我们可以接着删除 dev 分支了，操作如下：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git branch -d dev
Deleted branch dev (was d1f8e4a).

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git branch
* master

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ |
```

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

1、如何解决冲突

下面我们还是一步一步来，先新建一个新分支，比如名字叫 fenzhi1，在 readme.txt 添加一行内容 8888888，然后提交，如下所示：


```

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git checkout -b fenzhi1
Switched to a new branch 'fenzhi1'

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (fenzhi1)
$ cat readme.txt
1111111111
2222222222
3333333333
4444444444
6666666666
7777777777

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (fenzhi1)
$ cat readme.txt
1111111111
2222222222
3333333333
4444444444
6666666666
7777777777
8888888888

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (fenzhi1)
$ git add readme.txt

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (fenzhi1)
$ git commit -m "添加内容888888"
[fenzhi1 e5dd439] 添加内容888888
1 file changed, 2 insertions(+), 1 deletion(-)

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (fenzhi1)
$

```

新建并切换分支

内容为添加之前的内容

添加8888内容之后

https://blog.csdn.net/weixin_44950987

同样，我们现在切换到 master 分支上来，也在最后一行添加内容，内容为 99999999，如下所示：

```

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (fenzhi1)
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ cat readme.txt
1111111111
2222222222
3333333333
4444444444
6666666666
7777777777

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ cat readme.txt
1111111111
2222222222
3333333333
4444444444
6666666666
7777777777
9999999999

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git add readme.txt

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git commit -m "在master分支上新增内容9999"
[master 2337606] 在master分支上新增内容9999
1 file changed, 2 insertions(+), 1 deletion(-)

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$

```

切换到master分支上

未添加内容之前

添加内容9999之后

https://blog.csdn.net/weixin_44950987

现在我们需要在 master 分支上来合并 fenzhi1，如下操作：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git merge fenzhil
Auto-merging readme.txt
CONFLICT (content): Merge conflict in readme.txt
Automatic merge failed; fix conflicts and then commit the result.

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master|MERGING)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
(use "git push" to publish your local commits)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:   readme.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    testgit2/

no changes added to commit (use "git add" and/or "git commit -a")

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master|MERGING)
$ cat readme.txt
111111111
222222222
333333333
444444444
666666666
777777777
<<<<<< HEAD
999999999
=====
888888888
>>>>>> fenzhil
```

在master分支上合并fenzhil

产生冲突

查看状态

查看readme.txt内容

冲突代码

https://blog.csdn.net/weixin_44950987

Git 用<<<<<<, =====, >>>>>>标记出不同分支的内容, 其中<<<HEAD 是指主分支修改的内容, >>>>>>fenzhil 是指 fenzhil 上修改的内容, 我们可以修改下如下后保存:

```
readme.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
111111111
222222222
333333333
444444444
666666666
777777777
<<<<<< HEAD
999999999
=====
888888888
>>>>>> fenzhil
```

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master|MERGING)
$ cat readme.txt
111111111
222222222
333333333
444444444
666666666
777777777
999999999

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master|MERGING)
$ git add readme.txt

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master|MERGING)
$ git commit -m "conflict fixed"
[master edf8530] conflict fixed

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ |
```

查看内容，修改成和主干上代码一样，如上述文本文档

https://blog.csdn.net/weixin_44950987

如果我想查看分支合并的情况的话，需要使用命令 `git log`. 命令行演示下：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git log
commit edf853021cddac74d05fcbe09555f7016fd79633 (HEAD -> master)
Merge: 2337606 e5dd439
Author: zhuchifeng <18515592159@163.com>
Date: Sat Oct 19 01:28:14 2019 +0800
```

conflict fixed

```
commit 2337606c734d9926b2d26c6ed24859e9583a38d1
Author: zhuchifeng <18515592159@163.com>
Date: Sat Oct 19 01:10:51 2019 +0800
```

在master分支上新增内容9999

```
commit e5dd439d5c8bd71a09fca7eb39b0da11d3f86d7a (fenzhi1)
Author: zhuchifeng <18515592159@163.com>
Date: Sat Oct 19 01:07:03 2019 +0800
```

添加内容888888

```
commit d1f8e4a716f2665105bcbcef118c3ee6073c9de2
Author: zhuchifeng <18515592159@163.com>
Date: Sat Oct 19 00:29:45 2019 +0800
```

dev分支上增加内容777777

```
commit 7ab9b844610dbffa7b19bb6a46f2224a6218d006 (origin/master)
Author: zhuchifeng <18515592159@163.com>
Date: Fri Oct 18 15:39:12 2019 +0800
```

添加b.txt文件

```
commit 99e33c772b6a619d138f7be2ef1b109b31634973
Author: zhuchifeng <18515592159@163.com>
Date: Fri Oct 18 14:34:44 2019 +0800
```

一次性提交所有文件，包括新建文件test.txt

```
commit b8a7cf3ecd348b2537511f707b72e1ce47e0d2c6
Author: zhuchifeng <18515592159@163.com>
Date: Fri Oct 18 11:47:37 2019 +0800
```

添加readme.txt文件内容为3333333

```
commit 01e15a54b59c841ff7e101014dae1747c7f7cf48
...skipping...
commit edf853021cddac74d05fcbe09555f7016fd79633 (HEAD -> master)
Merge: 2337606 e5dd439
Author: zhuchifeng <18515592159@163.com>
Date: Sat Oct 19 01:28:14 2019 +0800
```

conflict fixed

```
commit 2337606c734d9926b2d26c6ed24859e9583a38d1
Author: zhuchifeng <18515592159@163.com>
Date: Sat Oct 19 01:10:51 2019 +0800
```

在master分支上新增内容9999

```
commit e5dd439d5c8bd71a09fca7eb39b0da11d3f86d7a (fenzhi1)
Author: zhuchifeng <18515592159@163.com>
Date: Sat Oct 19 01:07:03 2019 +0800
```

添加内容888888

```
commit d1f8e4a716f2665105bcbcef118c3ee6073c9de2
Author: zhuchifeng <18515592159@163.com>
Date: Sat Oct 19 00:29:45 2019 +0800
```

dev分支上增加内容777777

```
commit 7ab9b844610dbffa7b19bb6a46f2224a6218d006 (origin/master)
Author: zhuchifeng <18515592159@163.com>
Date: Fri Oct 18 15:39:12 2019 +0800
```

添加b.txt文件

```
commit 99e33c772b6a619d138f7be2ef1b109b31634973
```

https://blog.csdn.net/weixin_44950987

3、分支管理策略

通常合并分支时，git 一般使用“Fast forward”模式，在这种模式下，删除分支后，会丢掉分支信息，现在我们来使用带参数 `-no-ff` 来禁用“Fast forward”模式。首先我们来 demo 演示下：

1. 创建一个 dev 分支。
2. 修改 readme.txt 内容。
3. 添加到暂存区。
4. 切换回主分支(master)。
5. 合并 dev 分支，使用命令 `git merge --no-ff -m “注释” dev`
6. 查看历史记录

截图如下：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git checkout -b dev
Switched to a new branch 'dev'
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git add readme.txt
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git commit -m "add merge"
[dev 9b8eb36] add merge
1 file changed, 1 insertion(+), 1 deletion(-)
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 4 commits.
(use "git push" to publish your local commits)
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git merge --no-ff -m "merge with no-ff" dev
Merge made by the 'recursive' strategy.
readme.txt | 1 +
1 file changed, 1 insertion(+)
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git branch -d dev
Deleted branch dev (was 9b8eb36).
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git branch
* master
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git log --graph --pretty=oneline --abbrev-commit
* 1d5174e (HEAD -> master) merge with no-ff
|
| * 9b8eb36 add merge
| * 31d0e79 add merge
|/
* edf8530 conflict fixed
|
| * e5dd439 (fenzhi1) 添加内容888888
| * 2337606 在master分支上新增内容9999
|/
* d1f8e4a dev分支上增加内容777777
* 7ab9b84 (origin/master) 添加b.txt文件
* 99e33c7 一次性提交所有文件，包括新建文件test.txt
* b8a7cf3 添加readme.txt文件内容为3333333
* 01e15a5 文件增加222222内容
* 3871d39 readme.txt提交
```

https://blog.csdn.net/weixin_44950987

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。

七、bug 分支

在开发中，会经常碰到 bug 问题，那么有了 bug 就需要修复，在 Git 中，分支是很强大的，每个 bug 都可以通过一个临时分支来修复，修复完成后，合并分支，然后将临时的分支删除掉。

比如我在开发中接到一个 404 bug 时候，我们可以创建一个 404 分支来修复它，但是，当前的 dev 分支上的工作还没有提交。比如如下：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git status
On branch dev
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

并不是我不想提交，而是工作进行到一半时候，我们还无法提交，比如我这个分支 bug 要 2 天完成，但是我 issue-404 bug 需要 5 个小时内完成。怎么办呢？还好，Git 还提供了一个 stash 功能，可以把当前工作现场 ” 隐藏起来 ” ，等以后恢复现场后继续工作。如下：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git status
On branch dev
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git stash → 将当前的工作现场隐藏起来
Saved working directory and index state WIP on dev: 1d5174e merge with no-ff

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git status
On branch dev
nothing to commit, working tree clean → 查看状态，是干净的

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ |
```

所以现在我可以通过创建 issue-404 分支来修复 bug 了。

首先我们要确定在那个分支上修复 bug，比如我现在是在主分支 master 上来修复的，现在我要在 master 分支上创建一个临时分支，演示如下：

```

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 7 commits.
(use "git push" to publish your local commits)

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git checkout -b issue-404
Switched to a new branch 'issue-404'
在master分支上创建临时分支issue-404

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (issue-404)
$ cat readme.txt
111111111
222222222
333333333
444444444
666666666
777777777
999999999
000000000000000
未修改前查看readme.txt内容

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (issue-404)
$ cat readme.txt
111111111
222222222
333333333
444444444
666666666
777777777
999999999
aaaaaaaaaaa
修改后把readme.txt内容最后一行0000000改成aaaaa

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (issue-404)
$ git add readme.txt

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (issue-404)
$ git commit -m "fix bug 404"
[issue-404 9d802c5] fix bug 404
1 file changed, 1 insertion(+), 1 deletion(-)

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (issue-404)
$ |

```

修复完成后，切换到 master 分支上，并完成合并，最后删除 issue-404 分支。
演示如下：


```

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (issue-404)
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 7 commits.
(use "git push" to publish your local commits)

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git merge --no-ff -m "merge bug fix 404" issue-404
Merge made by the 'recursive' strategy.
 readme.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ cat readme.txt
1111111111
222222222
333333333
444444444
666666666
777777777
999999999
aaaaaaaaa

```

切换到master分支上

合并分支issue-404内容

合并分支后查看内容如下，和issue-404内容一致

```

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git branch -d issue-404
Deleted branch issue-404 (was 9d802c5).

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ |

```

在master分支上删除临时分支issue-404

https://blog.csdn.net/weixin_44950987

现在，我们回到 dev 分支上干活了。

```

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (master)
$ git checkout dev
Switched to branch 'dev'

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git status
On branch dev
nothing to commit, working tree clean

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ |

```

从master分支切换到dev分支上

目前干净的

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` 来查看下。如下：

```

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git stash list
stash@{0}: WIP on dev: 1d5174e merge with no-ff
stash@{1}: WIP on master: 1d5174e merge with no-ff

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ |

```

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。

演示如下：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git stash list
stash@{0}: WIP on dev: 1d5174e merge with no-ff
stash@{1}: WIP on master: 1d5174e merge with no-ff
删除前

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git stash drop
Dropped refs/stash@{0} (85c036a8b7025357a92553d3447f881a68abfe4e)
删除一条

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git stash list
stash@{0}: WIP on master: 1d5174e merge with no-ff
剩下一条

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git stash drop
Dropped refs/stash@{0} (050a823e27e8f3bc47ff94352c44851b68d7f221)
继续删

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git stash list
没有了
https://blog.csdn.net/weixin_44950987
```

八、多人协作

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`

如下演示：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git remote
origin
查看远程库的信息

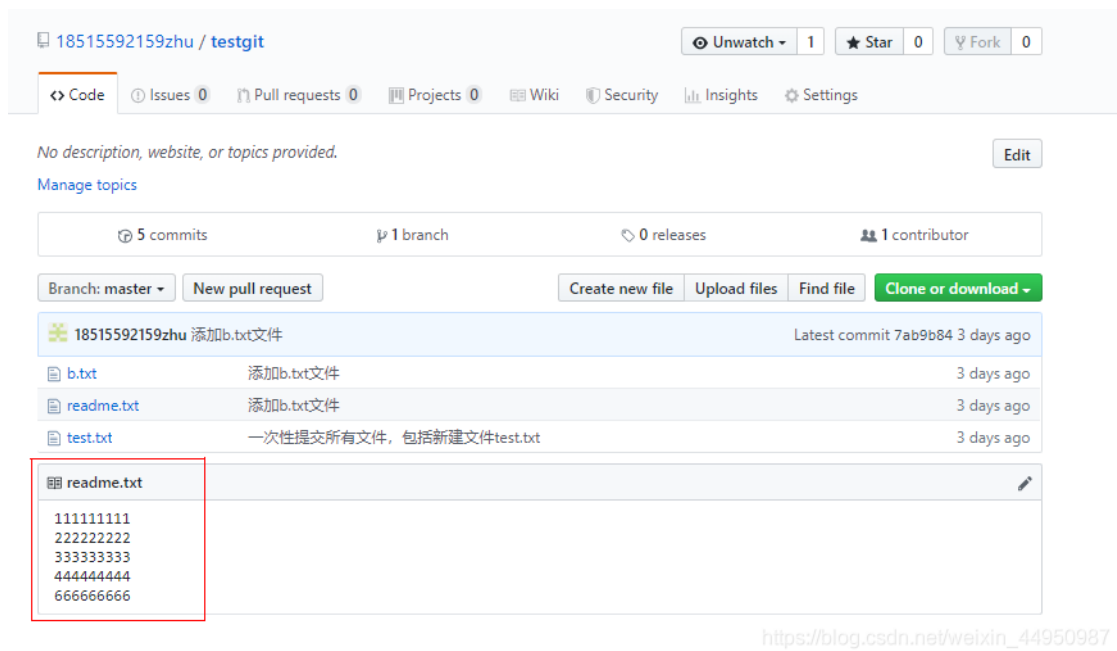
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git remote -v
origin git@github.com:18515592159zhu/testgit.git (fetch)
origin git@github.com:18515592159zhu/testgit.git (push)
详细信息
抓取
推送

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ |
```

1、推送分支：

推送分支就是把该分支上所有本地提交到远程库中，推送时，要指定本地分支，这样，Git 就会把该分支推送到远程库对应的远程分支上：使用命令 `git push origin master`

比如我现在的 github 上的 readme.txt 代码如下：



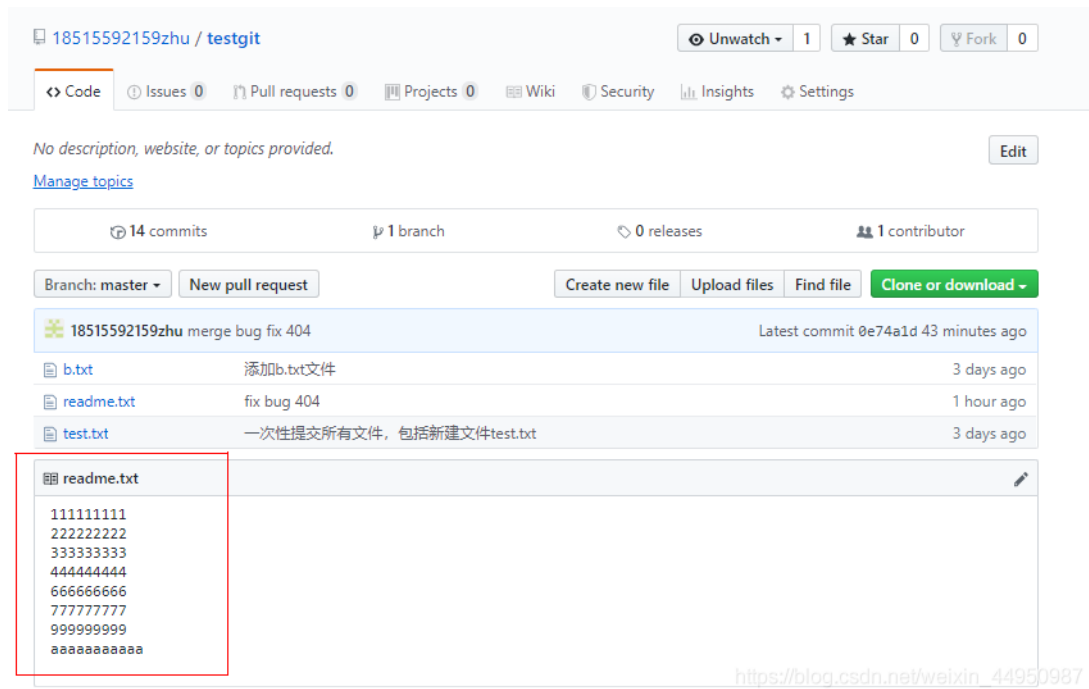
本地的 readme.txt 代码如下：



现在我想把本地更新的 readme.txt 代码推送到远程库中，使用命令如下：



我们可以看到如上，推送成功，我们可以继续来截图 github 上的 readme.txt 内容 如下：



可以看到 推送成功了，如果我们现在要推送到其他分支，比如 dev 分支上，我们还是那个命令 `git push origin dev`，那么一般情况下，那些分支要推送呢？

- master 分支是主分支，因此要时刻与远程同步。
- 一些修复 bug 分支不需要推送到远程去，可以先合并到主分支上，然后把主分支 master 推送到远程去。

2、抓取分支：

多人协作时，大家都会往 master 分支上推送各自的修改。现在我们可以模拟另外一个同事，可以在另一台电脑上（注意要把 SSH key 添加到 github 上）或者同一台电脑上另外一个目录克隆，新建一个目录名字叫 testgit2

但是我首先要把 dev 分支也要推送到远程去，如下：

```
zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git push origin dev
Total 0 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'dev' on GitHub by visiting:
remote:   https://github.com/18515592159zhu/testgit/pull/new/dev
remote:
To github.com:18515592159zhu/testgit.git
 * [new branch]      dev -> dev

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ |
```

接着进入 testgit2 目录，进行克隆远程的库到本地来，如下：

```

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ cd testgit2

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit/testgit2 (master)
$ git clone https://github.com/18515592159zhu/testgit.git
Cloning into 'testgit'...
remote: Enumerating objects: 40, done.
remote: Counting objects: 100% (40/40), done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 40 (delta 10), reused 40 (delta 10), pack-reused 0
Unpacking objects: 100% (40/40), done.

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit/testgit2 (master)
$

```

克隆远程库

https://blog.csdn.net/weixin_44950987

现在目录下生成有如下所示：

| 电脑 > 本地磁盘 (E:) > git > testgit > testgit2 > testgit | | | | 从远程库生成testgit |
|---|------------------|------|------|---------------|
| 名称 | 修改日期 | 类型 | 大小 | |
| .git | 2019/10/21 10:37 | 文件夹 | | |
| b.txt | 2019/10/21 10:37 | 文本文档 | 0 KB | |
| readme.txt | 2019/10/21 10:37 | 文本文档 | 1 KB | |
| test.txt | 2019/10/21 10:37 | 文本文档 | 1 KB | |

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：

```

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit/testgit2 (master)
$ cd testgit

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit/testgit2/testgit (master)
$ git checkout -b dev origin/dev
Switched to a new branch 'dev'
Branch 'dev' set up to track remote branch 'dev' from 'origin'.

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit/testgit2/testgit (dev)
$ cat readme.txt
111111111
222222222
333333333
444444444
666666666
777777777
999999999
000000000000000

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit/testgit2/testgit (dev)
$ cat readme.txt
111111111
222222222
333333333
444444444
666666666
777777777
999999999
000000000000000
aaaaaaaaaaaaa

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit/testgit2/testgit (dev)
$ git add readme.txt

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit/testgit2/testgit (dev)
$ git commit -m "readme.txt上增加aaaaaaa内容"
[dev 4dec1b1] readme.txt上增加aaaaaaa内容
1 file changed, 1 insertion(+)

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit/testgit2/testgit (dev)
$ git push origin dev
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 354 bytes | 354.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/18515592159zhu/testgit.git
1d5174e..4dec1b1 dev -> dev

```

创建远程origin的dev分支到本地来

修改前readme.txt内容

添加aaaaaaaaa内容之后的文件

把现有的dev分支推送到远程去

https://blog.csdn.net/weixin_44950987

小伙伴们已经向 origin/dev 分支上推送了提交，而我在我的目录文件下也对同样的文件同个地方作了修改，也试图推送到远程库时，如下：

```

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ pwd
/e/git/testgit

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git checkout dev
Already on 'dev'
    切换目录到我的dev分支上

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ cat readme.txt
111111111
222222222
333333333
444444444
666666666
777777777
999999999
000000000000000
    修改之前内容

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ cat readme.txt
111111111
222222222
333333333
444444444
666666666
777777777
999999999
000000000000000
aaaaaaaaaaaaa
    给readme.txt文件添加内容aaaaaaa后

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git add readme.txt

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git commit -m "我自己分支dev上同样提交readme.txt文件"
[dev 4fcb398] 我自己分支dev上同样提交readme.txt文件
1 file changed, 1 insertion(+)
    推送到远程库时发生错误，不同的人推同样的文件，修改同一个文件同一个地方报错

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ git push origin dev
Warning: Permanently added the RSA host key for IP address '52.74.223.119' to the list of known hosts.
To github.com:18515592159zhu/testgit.git
! [rejected]        dev -> dev (fetch first)
error: failed to push some refs to 'git@github.com:18515592159zhu/testgit.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.

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$

```

由上面可知：推送失败，因为我的小伙伴最新提交的和我试图推送的有冲突，解决的办法也很简单，上面已经提示我们，先用 `git pull` 把最新的提交从 origin/dev 抓下来，然后在本地合并，解决冲突，再推送。


```

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ 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 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), done.
From github.com:18515592159zhu/testgit
   1d5174e..4dec1b1 dev -> origin/dev
There is no tracking information for the current branch.
Please specify which branch you want to merge with.
See git-pull(1) for details.

    git pull <remote> <branch>

If you wish to set tracking information for this branch you can do so with:

    git branch --set-upstream-to=origin/<branch> dev

zhuchifeng@DESKTOP-72A2E7N MINGW64 /e/git/testgit (dev)
$ |

```

git pull 也失败了，原因是没有指定本地 dev 分支与远程 origin/dev 分支的链接，根据提示，设置 dev 和 origin/dev 的链接：如下：

```

E73-8@E73-8-PC /d/www/testgit (dev)
$ git branch --set-upstream dev origin/dev
The --set-upstream flag is deprecated and will be removed. Consider using --track or --set-upstream-to
Branch dev set up to track remote branch dev from origin.

E73-8@E73-8-PC /d/www/testgit (dev)
$ git pull
Auto-merging readme.txt
CONFLICT (content): Merge conflict in readme.txt
Automatic merge failed; fix conflicts and then commit the result.

E73-8@E73-8-PC /d/www/testgit (dev|MERGING)
$

```

pull成功了，但是有冲突，需要解决，再pull

这回 git pull 成功，但是合并有冲突，需要手动解决，解决的方法和分支管理中的 解决冲突完全一样。解决后，提交，再 push：我们可以先来看看 readme.txt 内容了。

```

E73-8@E73-8-PC /d/www/testgit (dev|MERGING)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
7777777777777777
9999999999999999
0101010101010101
<<<<<<< HEAD
aaaaaaaaaaaaaaaa
=====
aaaaaaaaaaaaaaaa
>>>>>>> fd74bb10291a19708e7c503def84fca4a2481594

E73-8@E73-8-PC /d/www/testgit (dev)
$

```


现在手动已经解决完了，我接在需要再提交，再 push 到远程库里面去。如下所示：

```
E73-8@E73-8-PC /d/www/testgit (dev|MERGING)
$ cat readme.txt
11111111111111
22222222222222
33333333333333
44444444444444
66666666666666
77777777777777
99999999999999
01010101010101
aaaaaaaaaaaaaa

E73-8@E73-8-PC /d/www/testgit (dev|MERGING)
$ git add readme.txt

E73-8@E73-8-PC /d/www/testgit (dev|MERGING)
$ git commit -m "merge & fix readme.txt"
[dev bbaf5ad] merge & fix readme.txt

E73-8@E73-8-PC /d/www/testgit (dev)
$ git push origin dev
Username for 'https://github.com': tughnhua0707@qq.com
Password for 'https://tughnhua0707@qq.com@github.com':
Counting objects: 10, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 567 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
To https://github.com/tughnhua0707/testgit.git
fd74bb1..bbaf5ad dev -> dev
```

手动解决后的文件是这样的

继续推送到远程库中 恭喜你success了

https://blog.csdn.net/weixin_44950987

因此：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。

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/18515592159zhu/testgit.git` 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/18515592159zhu/testgit.git` 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