**GIT使用文档**

[1 Git 与SVN的区别&优势 3](#_Toc11791)

[2 git的结构 3](#_Toc21155)

[3 git命令行操作 5](#_Toc19899)

[3.1本地库初始化 5](#_Toc22502)

[3.2 设置签名 5](#_Toc25921)

[3.3状态查看 6](#_Toc7952)

[3.4 添加 6](#_Toc23472)

[3.5 提交 7](#_Toc30866)

[3.6 查看历史记录 7](#_Toc14039)

[3.7 前进后退 9](#_Toc21943)

[3.8 删除文件并找回 10](#_Toc25279)

[3.9 比较文件差异 13](#_Toc23204)

[4 分支管理 14](#_Toc16941)

[4.1 什么是分支？ 14](#_Toc22872)

[4.2 分支操作 15](#_Toc14384)

## 1 Git 与SVN的区别&优势

1.SVN是集中式版本控制工具，如图1所示，集中式的版本控制系统都有一个单一的集中管理的服务器，保存所有文件的修订版本，而协调工作的人们都通过客户端连接到这台服务器，取出最新的文件或者提交更新。

1. Git是分布式版本控制工具，Git记录版本历史只关心文件数据的整体是否发生变化，而不保存文件内容前后变化的差异数据。如图2所示，在分布式版本控制系统中，客户端不只提取最新版本的文件快照，而是把原始的代码仓库完整地镜像下来，这样的好处是避免了单点故障。
2. 在版本管理上与SVN不同的是，git采取了文件系统快照的方式，而SVN采用的是增量式管理的方式。

4.在权限控制方面，Git不仅可以对团队中参与开发的人员进行权限控制，还可对团队外开发者贡献的代码进行审核。

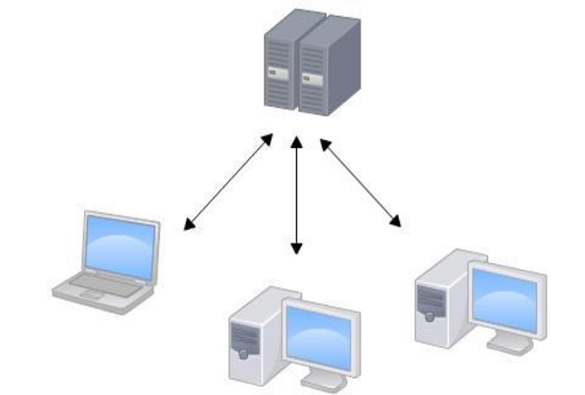
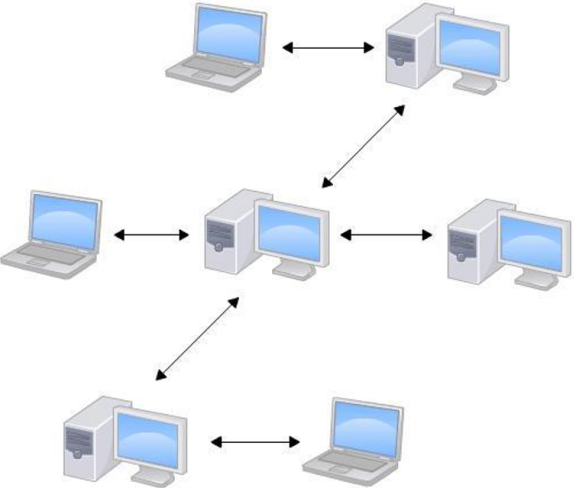
 

图 1 SVN版本控制 图2 Git 版本控制图

5.Git的优势

1大部分操作在本地完成，不需要联网

1. 分支操作非常快捷流畅
2. 每一次提取操作，实际上都是一次对代码仓库的完整备份
3. 与linux命令全面兼容
4. 可对团队外的代码审核

等

## 2 git的结构

Git管理项目时，文件在三个工作区域中流转：Git的本地数据目录Repository，工作目录Working Directory以及暂存区域stage，如图3所示。这里的仓库，即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”。

那么，要将一个文件纳入版本管理的范畴，首先是要用git add将文件纳入stage的监控范围，只有更新到stage中的内容才会在commit的时候被提交。

总结以下几点：

1. 工作区的文件改动（新增、修改、删除文件），必须用git add或者git rm命令标识，使得改动进入stage。另外commit可以一次提交很多文件，所以你可以多次add不同的文件。
2. 提交只对加入stage的改动进行提交
3. 如果一个文件改动加入stage后再次改动，则后续改动不改变stage。即该文件的改动有两个状态，一个是标记到stage中并将在下次提交时入库的改动，另外的后续改动则不被提交，除非再次使用git add命令将改动加入到stage

（后面会具体举例说明）

历史版本

本地库

暂存区

工作区

Git commit

临时存储

Git add

写代码

图 3 git管理项目流程图

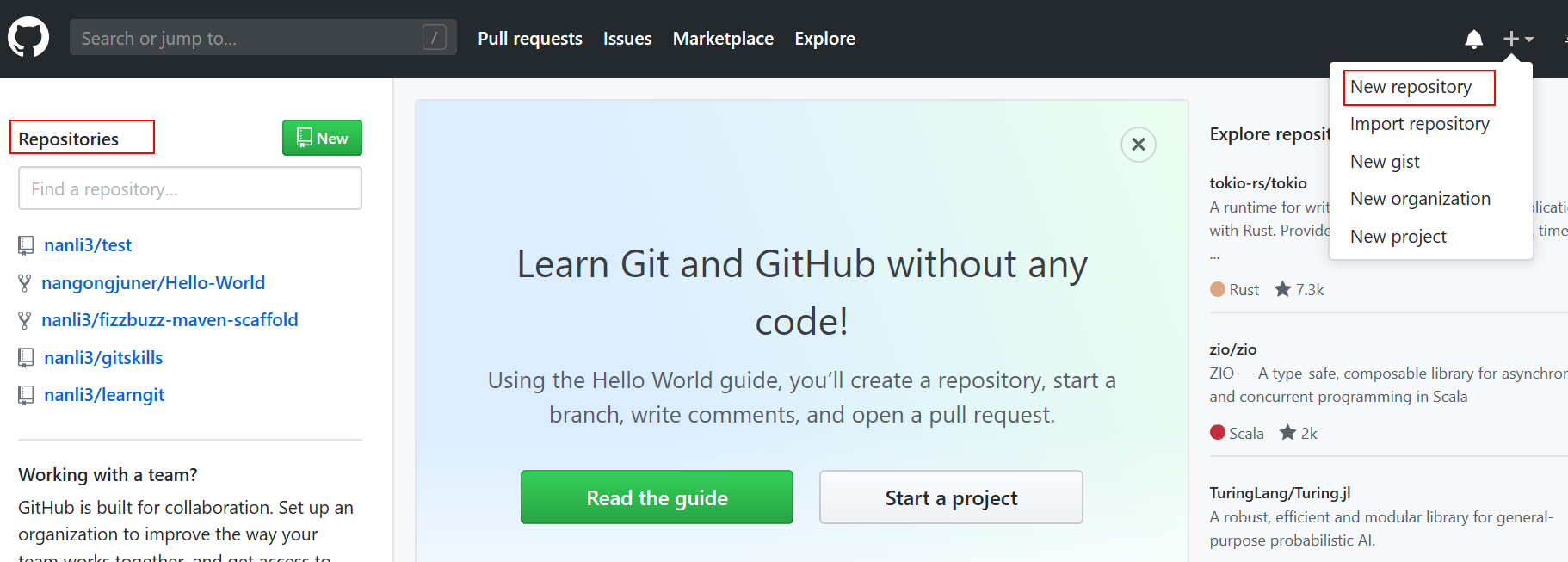
1. 本地库和远程库

除了本地的版本控制，与SVN不同的是git还有远程仓库，前面了解到git是分布式版本控制系统，同一个git仓库可以分布到不同的机器上，那么如何分布呢？最早肯定只有一台机器有一个原始版本库，此后，别的机器可以克隆这个原始版本库，而且每台机器的版本库其实都是一样的，没有主次之分。

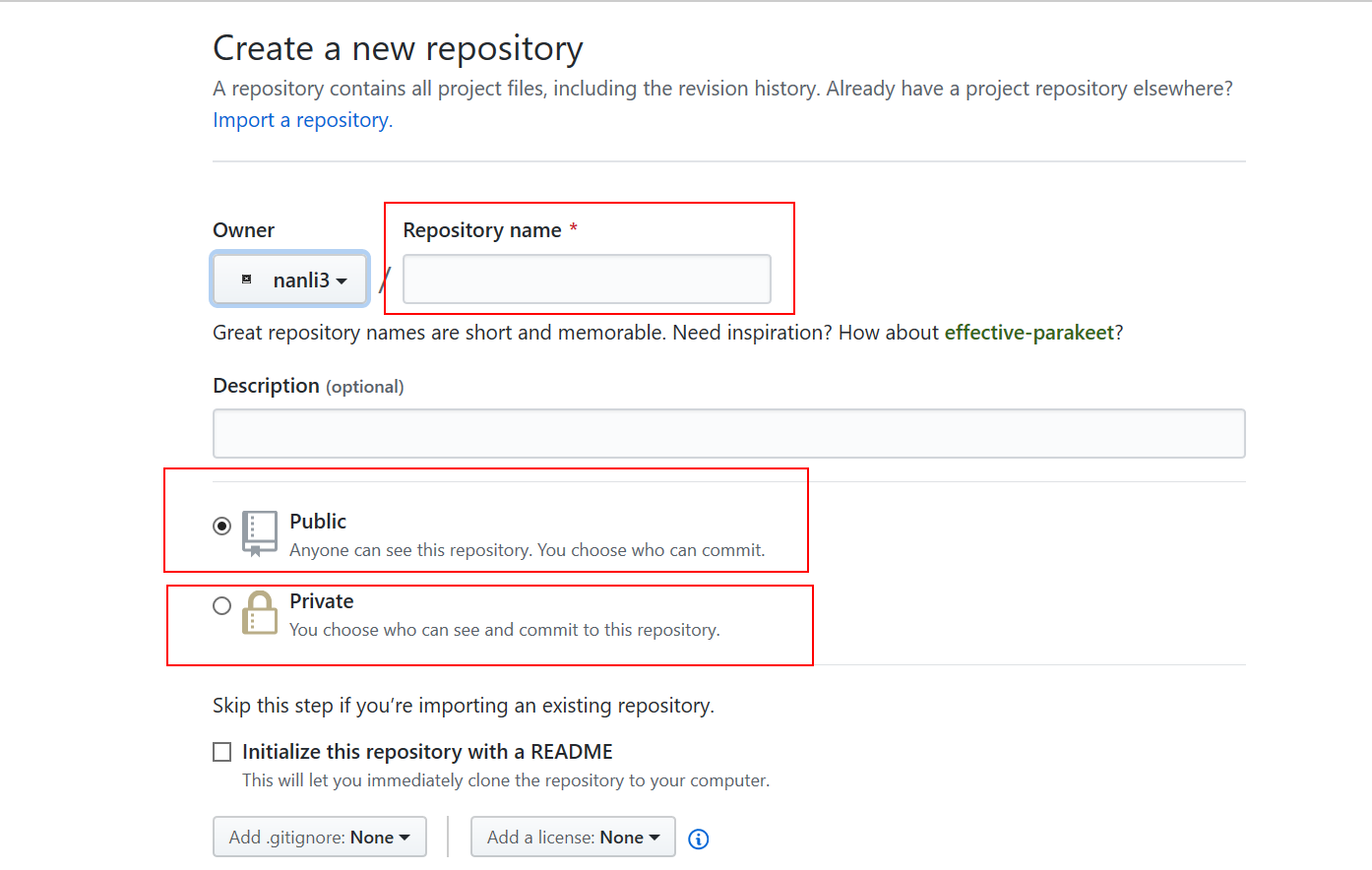
假设你已经在本地创建了一个git仓库，又想在github创建一个git仓库，并且让两个仓库进行远程同步，这样GitHub上的仓库既可以作为备份，又可以让其他人通过该仓库来协作。

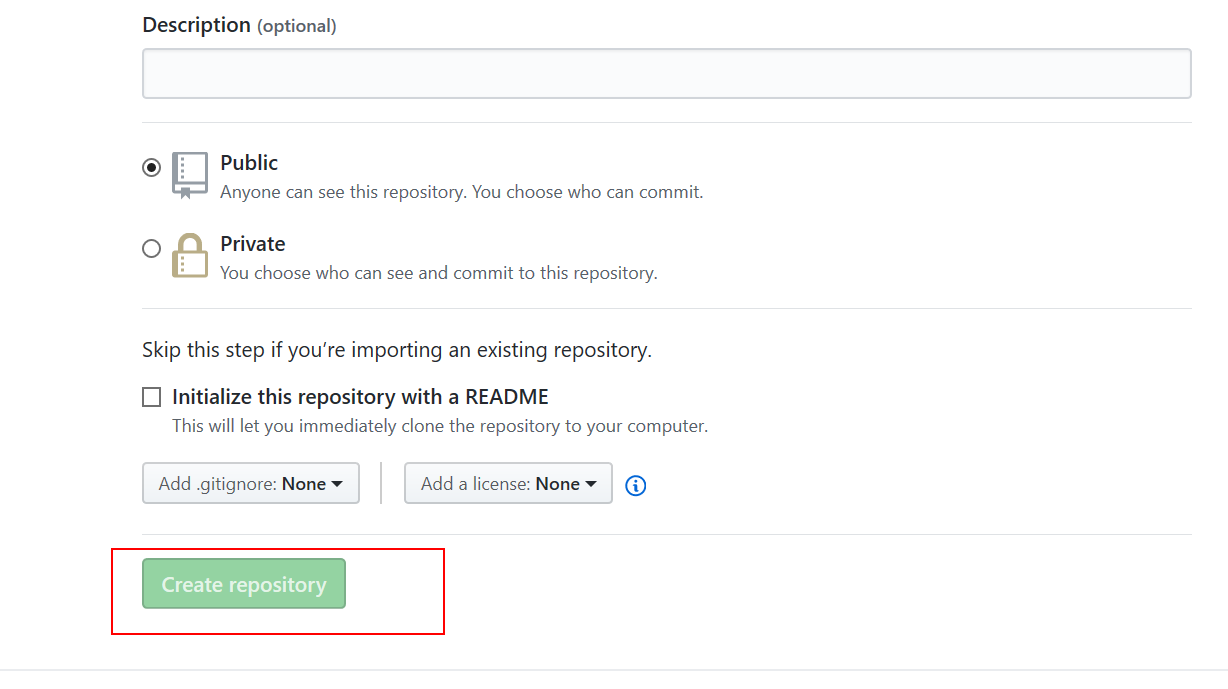
**添加远程库**

首先，登陆GitHub，然后，在右上角找到“Create a new repo”按钮，创建一个新的仓库：



在Repository name填入learngit，其他保持默认设置，点击“Create repository”按钮，就成功地创建了一个新的Git仓库



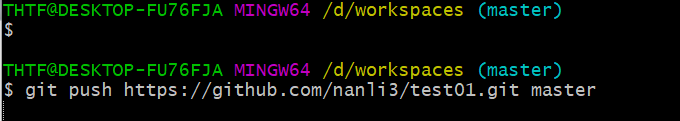


刚创建的仓库还是空的，GitHub会告诉我们，可以从这个仓库克隆出新的仓库，也可以把一个已有的本地仓库与之关联，然后，把本地仓库的内容推送到GitHub仓库。

如下所示http：//gitHub.com/nanli3/test01.git即是你远程仓库的地址，此时用git push命令推送分支，就是把该分支上的所有本地提交提送到远程库。推送时，要指定本地分支。实际上是把当前分支master推送到远程。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。

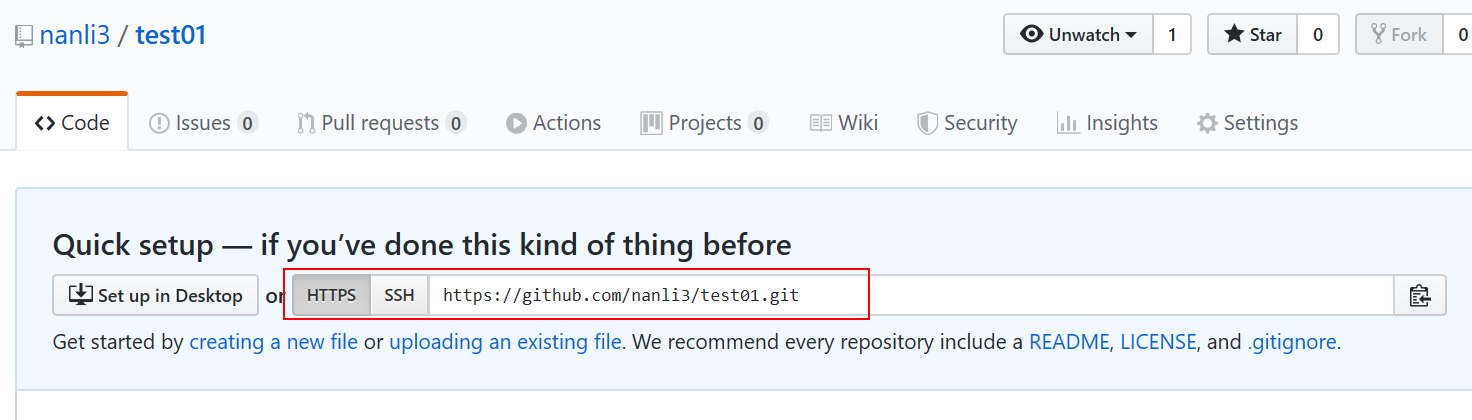
你也许还注意到，GitHub给出的地址不止一个。实际上，Git支持多种协议，默认的git://使用ssh，但也可以使用https等其他协议。

使用https除了速度慢以外，还有个最大的麻烦是每次推送都必须输入口令，但是在某些只开放http端口的公司内部就无法使用ssh协议而只能用https。



但是每次使用地址都不太方便，使用如下命令

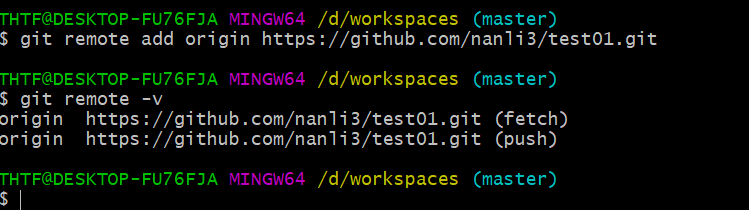
$ git remote add origin http：//gitHub.com/nanli3/test01.git 添加远程仓库



添加后，远程库的名字就是origin，这是Git默认的叫法，也可以改成别的。

此时使用git push origin master就可以实现推送。

另外使用git remote -v可以查看到当前所有的远程地址库



团队内部协作

如果有多个人协作开发，那么每个人各自从远程克隆一份就可以了，用命令git clone克隆一个本地库：

代码托管中心

远程库

本地库

加入团队

Pull push clone push

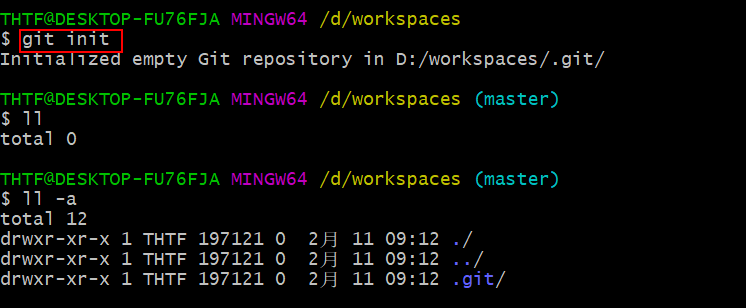
本地库

## 3 git命令行操作

### 3.1本地库初始化

前面我们了解到，git仓库里的所有文件都被git管理起来，因此，创建一个版本库非常简单，首先，选择一个合适的地方，创建一个空目录：如下，我在d盘创建了一个workspaces的空目录。并进入到该目录下，执行git init命令把这个目录变成git可以管理的仓库。

在git版本库中，第一次执行git init时系统会创建一个名为master的分支。Git的版本库和工作区在同一个目录下，工作区的根目录有一个.git的子目录。并且该目录是默认隐藏的，使用参数-a可以看见。



注意：.git 目录中存放的是本地库的子目录和文件，不要删除，也不要胡乱修改

不一定必须在空目录下创建git仓库，选择一个已经有东西的目录也是可以的。

### 3.2 设置签名

git config --global user.name "Your Name"

git config --global user.email "email@example.com"

因为Git是分布式版本控制系统，所以，每个机器都必须自报家门：你的名字和Email地址。

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。此时不需添加参数global。

例如：以下指定d盘的仓库设置签名

用户名：linan

Email地址：[test@linan.com](mailto:test@linan.com)

作用：区分不同开发人员的身份

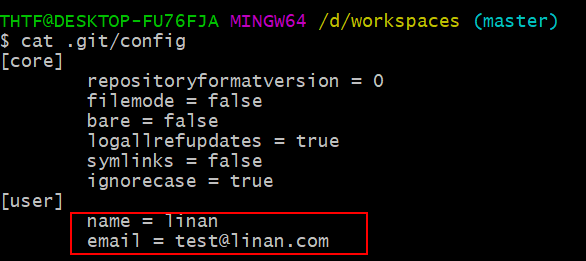
命令：git config user.name linan

git config user.email [test@linan.com](mailto:test@linan.com)

信息保存位置：./.git/config 文件

效果：

45



注意：登陆代码托管中心的账号密码和这里的没有任何关系。

git config 的Global参数可以区分级别。

一个是项目级别/仓库级别：仅在当前本地库范围内有效

一个是系统用户级别：登陆当前操作系统的用户，超越了项目级别。

另外签名级别按就近原则。

如果没有设置项目级别的签名，则按照系统用户级别

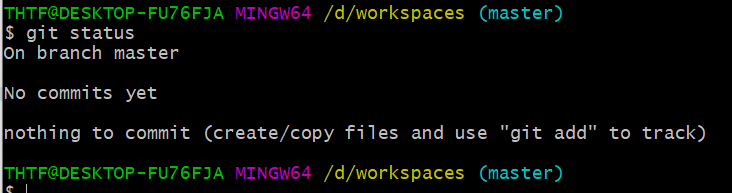
二者都有的则采用项目级别的签名。

二者都没有是不允许的。

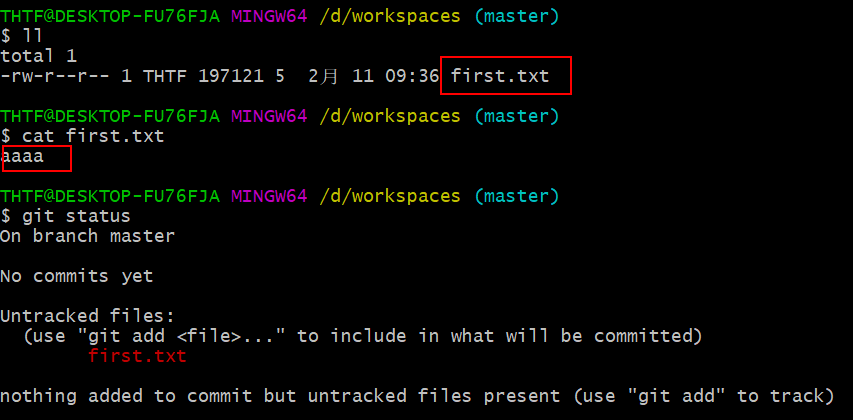
### 3.3状态查看

git status命令可以让我们时刻掌握仓库当前的状态。

例如，没有创建任何文件时候的效果：git会告诉我们现在在master分支，还没有任何提交，并且也没有需要添加到暂存区的进行追踪的文件。



若创建了一个名为first的文本文件，且文件内容为aaaa，此时的状态效果如下：git会提示我们有未被追踪的文件，请使用git add添加到暂存区以便将来进行提交。

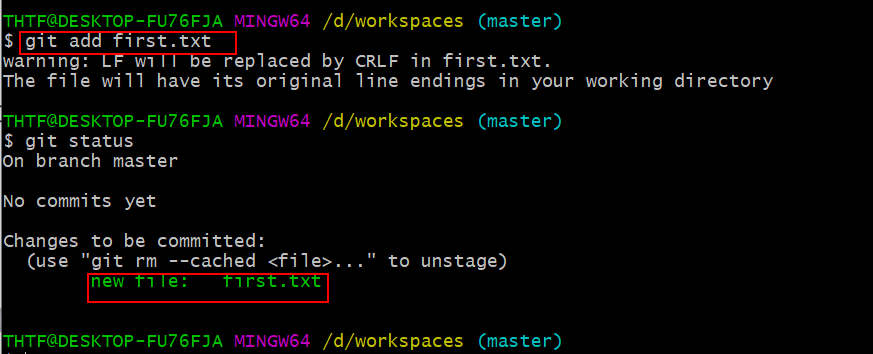


### 3.4 添加

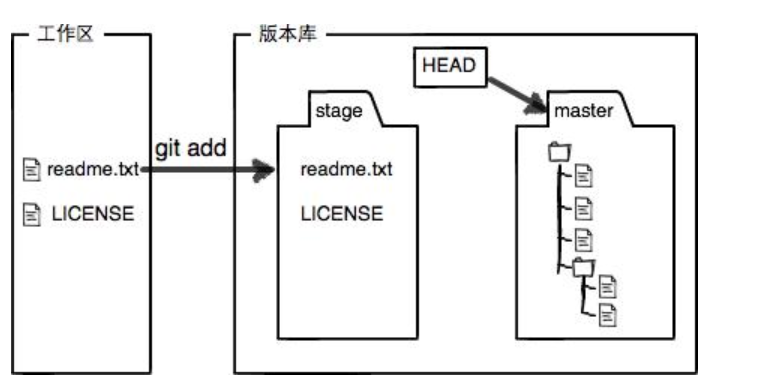
节目接下来我们使用git add 添加到暂存区，以便提交。

git add [file name] 将工作区的“新建/修改”添加到暂存区

此时查看git的状态可以看到git告诉我们有一个新的first.txt将被提交，可使用git rm --cache 文件名 将其从暂存区移除。



这里我们给一个图示来说明，如下

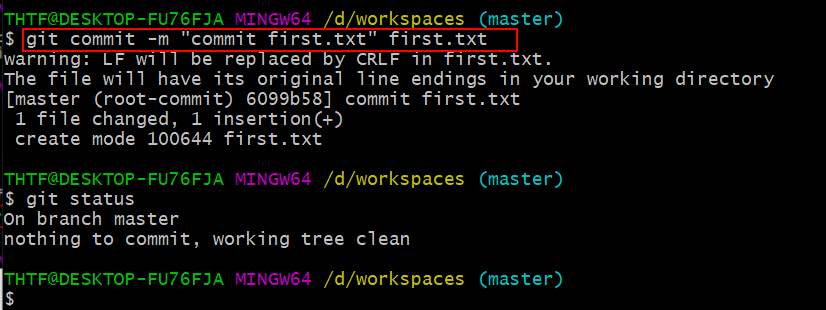


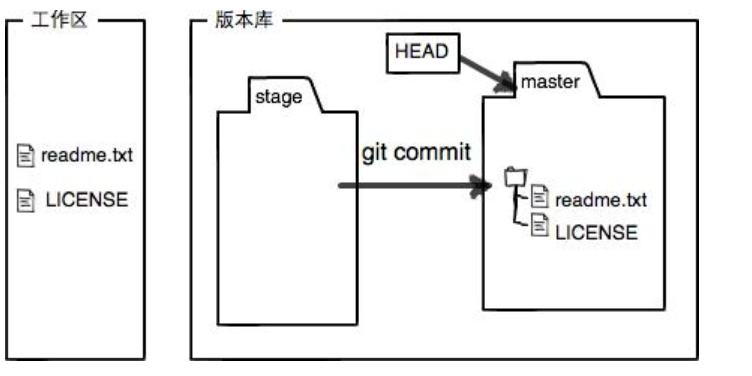
### 3.5 提交

git commit -m “commit message” [file name] 将暂存区的内容提交到本地库。

简单解释一下git commit命令，-m后面输入的是本次提交的说明，可以输入任意内容，当然最好是有意义的，这样你就能从历史记录里方便地找到改动记录。

git commit命令执行成功后会告诉你，1 file changed：1个文件被改动（我们新添加的first.txt文件）；1 insertions：插入了1行内容.

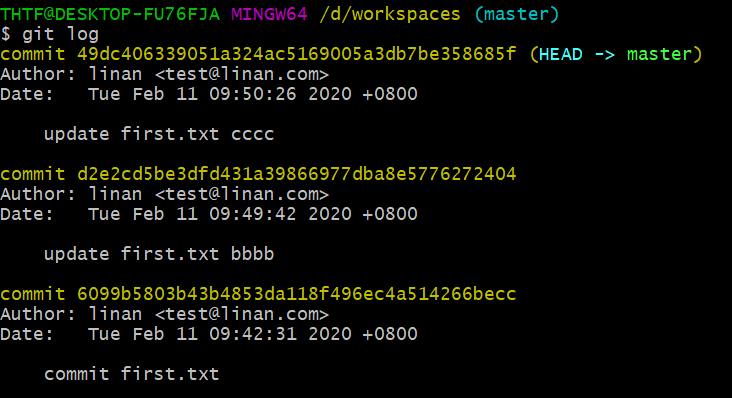




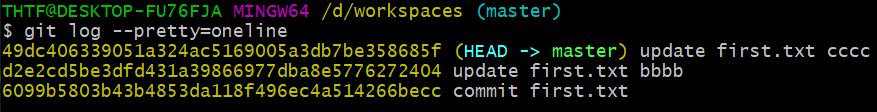
### 3.6 查看历史记录

假如经过三次修改文件first.txt并提交之后的效果

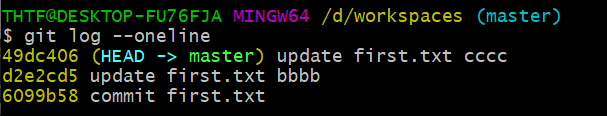
git log命令显示从最近到最远的提交日志，我们可以看到3次提交，最近的一次是update first .txt cccc，上一次是update first .txt bbbb，最早的一次是commit first .txt 。你看到的一大串类似49dc...的是commit id（版本号），和SVN不一样，Git的commit id不是1，2，3……递增的数字，而是一个SHA1计算出来的一个非常大的数字，用十六进制表示。因为Git是分布式的版本控制系统，后面我们还要研究多人在同一个版本库里工作，如果大家都用1，2，3……作为版本号，那肯定就冲突了。



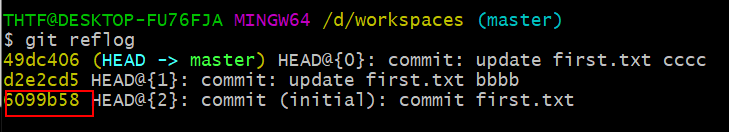
如果嫌输出信息太多，看得眼花缭乱的，可以试试加上--pretty=oneline参数



如果还嫌信息多，则可直接使用--oneline参数，如下版本号只显示前面一部分。



或者使用git reflog 这里可以看到 HEAD@{移动到当前版本需要多少步}

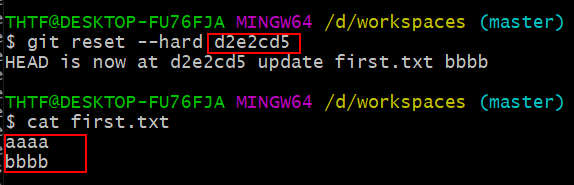


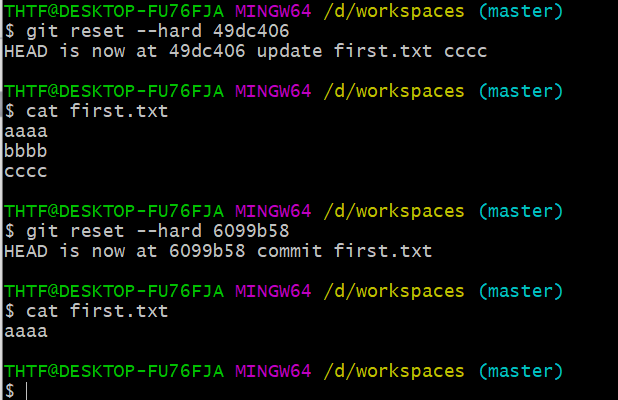
注意到第一次提交时候的哈希值没变。

### 3.7 前进后退

基于索引值操作【推荐】

Git reset --hard [局部索引值]

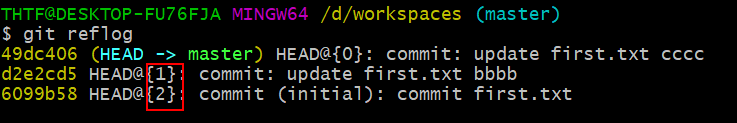


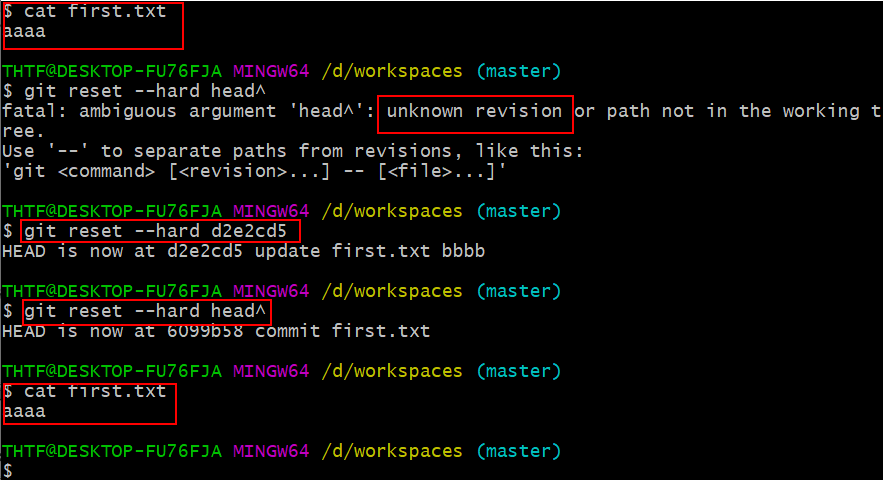


使用^符号：只能后退

Git reset --hard HEAD^

注：一个^表示后退一步，n个表示后退n步





使用~符号：只能后退

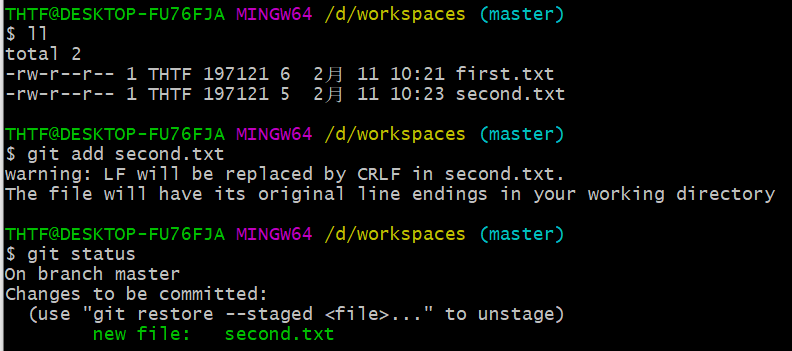
Git reset --hard HEAD~n

注：表示后退n步

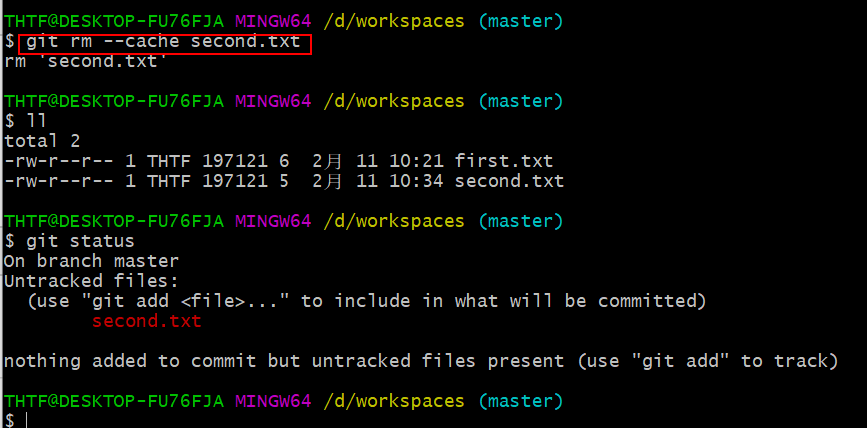
### 3.8 撤销修改

### 3.8 删除文件并找回

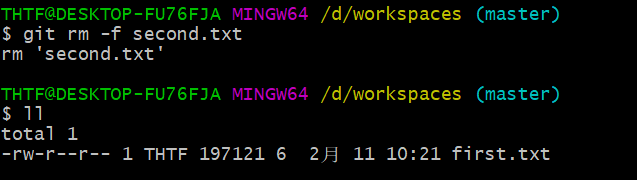
情况一：提交到暂存区，尚未提交到本地库，如下所示



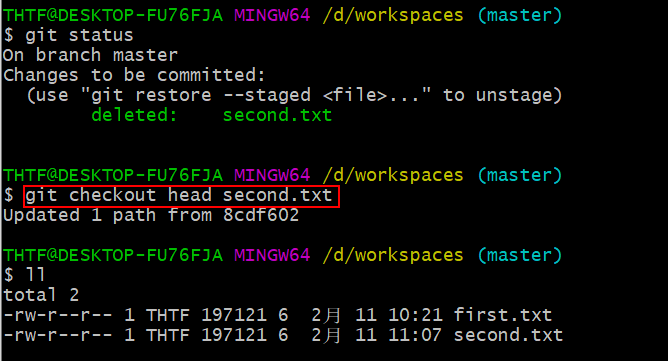
使用git rm --cache 文件名，仅仅删除暂存区的文件，如下图所示，此时工作区仍有此文件。



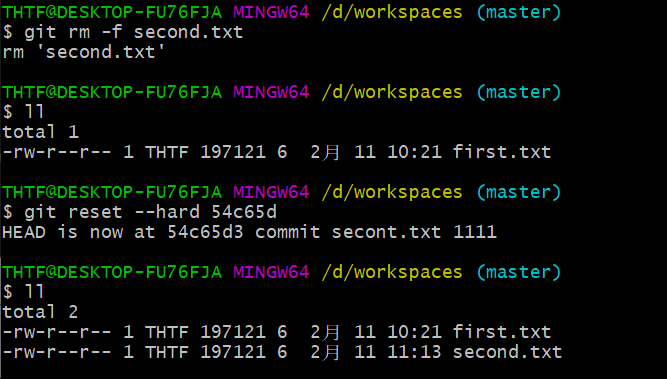
删除使用 git rm -f 文件名，删除暂存区和工作区的文件，如图所示，工作区已经没有此文件了。



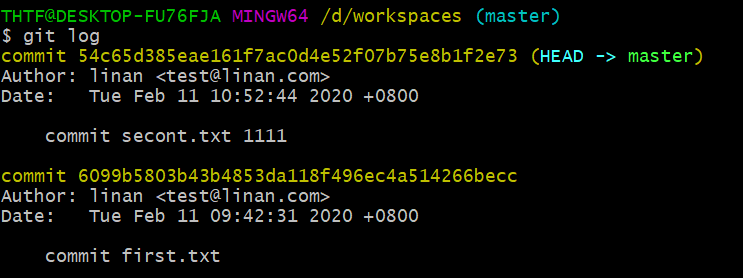
找回：git checkout head 文件名



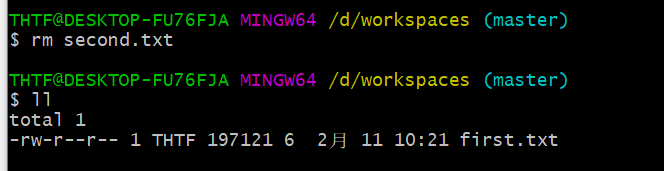
或者：git reset --hard[指针位置]



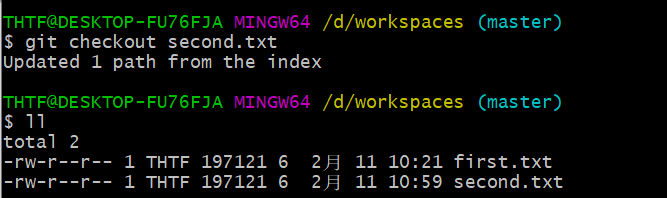
情况二：已经提交到本地库的文件，



使用rm只是将工作区中的文件删除，暂存区相应的文件依然存在，只要将暂存区内容恢复到工作区即可。



命令 git checkout 文件名



### 3.9 比较文件差异

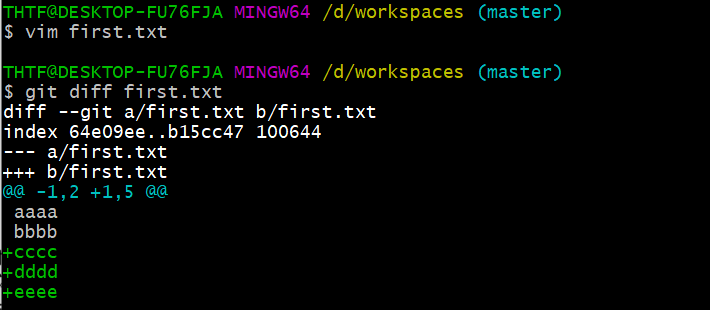
虽然Git告诉我们readme.txt被修改了，但如果能看看具体修改了什么内容，自然是很好的。比如你休假两周从国外回来，第一天上班时，已经记不清上次怎么修改的readme.txt，所以，需要用git diff这个命令看看：

命令：git status 查看工作区、暂存区状态

Git diff [文件名]

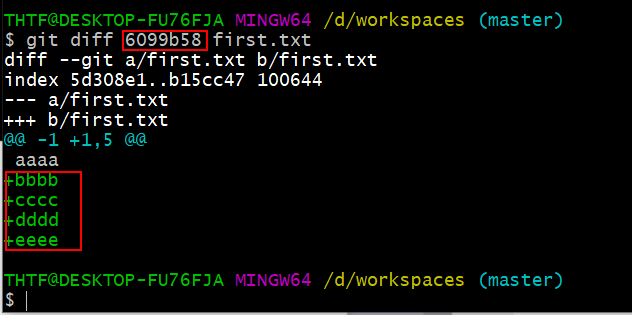
将工作区的文件和暂存区进行比较

例如，重新在工作区编辑了文件，但未提交到暂存区



Git diff [本地库中历史版本][文件名]

将工作区中的文件和本地库历史记录比较



## 4 分支管理

### 4.1 什么是分支？

在版本控制过程中，使用多条线同时推进多个任务。git的分支实际上仅是一个包含所指对象校验和（40个字符长度SHA-1哈希值）的文件。所以创建分支/里程碑的复杂度是O（1），这里称轻量级分支/里程碑，不会因为版本库的愈加庞大而变得缓慢。另外git的分支是完全隔离的。

Gitflow工作流详解

* 主干分支 master

主要负责管理正在运行的生产环境代码。永远保持与正在运行的生产代码保持一致。

* 开发分支Develop

主要负责管理正在开发过程中的代码。一般情况下应该是最新的代码。

* Bug 修理分支hotfix

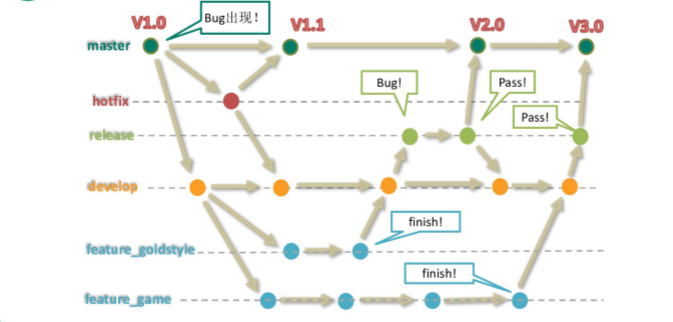
主要负责管理生产环境下出现的紧急修复的代码。从主干分支分出，修理完毕并测试上线后，并回主干分支。并回后，视情况可以删除该分支。

* 准生产分支/预发布分支 release

较大的版本上线前，会从开发分支中分出准生产分支，进行最后阶段的集成测试。该版本上线后，会合并到主干分支。生产环境运行一段阶稳定后可以视情况删除。

* 功能分支 feature

为了不影响较短期的开发工作，一般把中长期开发模块，会从开发分支中独立出来。开发完成后会合并到开发分支



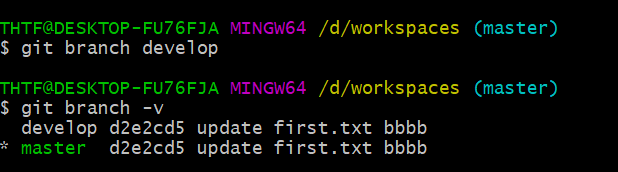
### 4.2 分支操作

创建分支

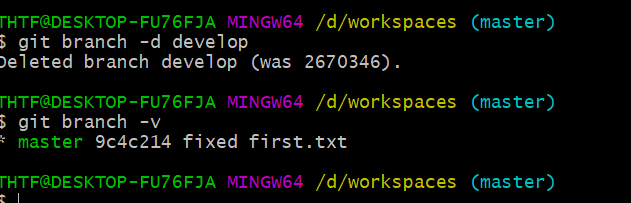
git branch [分支名]

查看分支

Git branch -v

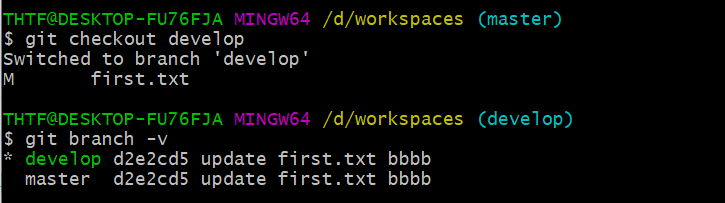


删除分支 git branch -d 分支名



切换分支

Git checkout [分支名]



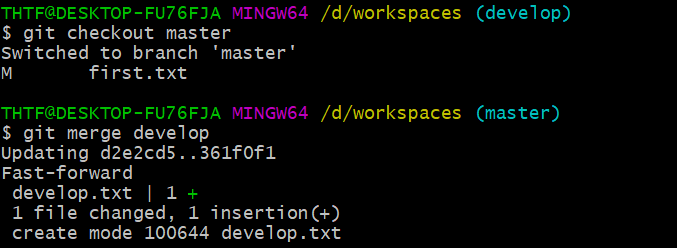
合并分支

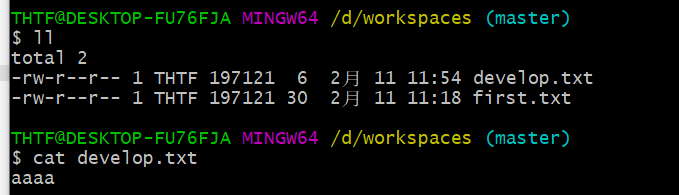
第一步：切换到接受修改的分支（被合并的分支）上

git checkout [被合并分支名]

第二步 ：执行merge命令

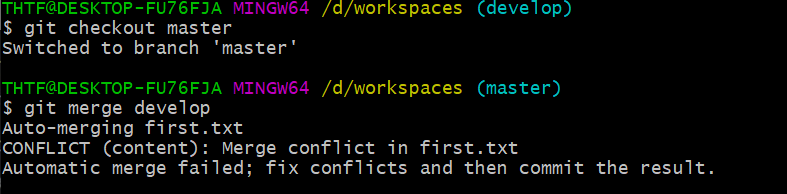
git merge [有新内容分支名]

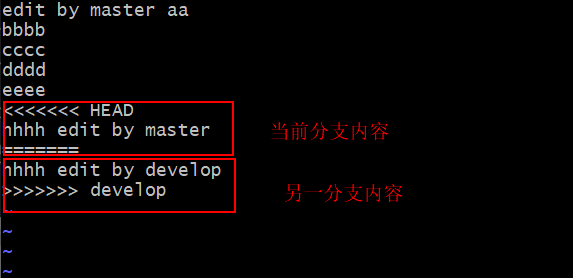




解决冲突

冲突的表现





冲突解决

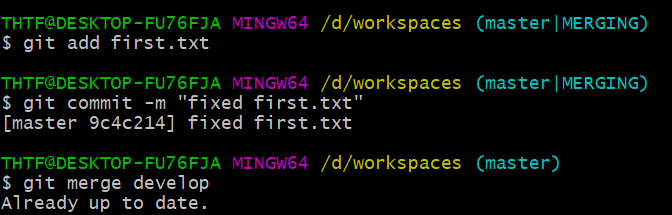
第一步：编辑文件，删除特殊符号

第二步：把文件修改到满意程度，保存退出

第三步：git add[文件名]

第四步: git commit -m “日志信息”

此时commit 一定不能带具体的文件名



## 5 GitHub