

- 1.git init //本地仓库初始化
- 2.git add . //添加所有文件
- 3.git commit -m "初始化" //提交
- 4.git remote add origin <https://github.com/githubusername/demo.git> //绑定到 name origin 的远程连接
- 5.git pull --rebase origin master //拉取远程代码并合并
- 6.git push -u origin master //推送到远程分支

git remote rm origin //删除原来的连接

git status -s //查看简洁的当前文件状态

git status 列出当前目录所有还没有被 git 管理的文件和被 git 管理且被修改但还未提交 (git commit)的文件.

**Git一般有很多分支，我们clone到本地的一般都是master分支，如何进行分支的切换呢？那么下面带大家简单的看看如何通过命令来切换：**

### 1、查看远程仓库及本地的所有分支

命令: git branch -a

```
1 qinjiaxi:~$ git branch -a
2 * master
3 remotes/origin/HEAD -> origin/master
4 remotes/origin/Release_20190311
5 remotes/origin/Release_20190811
6 remotes/origin/develop
7 remotes/origin/feature/TLS_1363
8 remotes/origin/feature/download
9 remotes/origin/master
```

**可看到我们现在master分支**

### 2、查看本地分支

命令: git branch

```
1 qinjiaxi:~$ git branch
2 * master
```

Git 新建本地分支与删除

删除本地分支：git branch -d 分支名字（首先切换到别的分支上面）。

**git push origin master 会报错：**

1:

```
$ git push -f
```

加上 -f，强制推送上去，这时你的 GitHub 上的库会以本地同步。

2:

```
$ git pull --rebase origin master
```

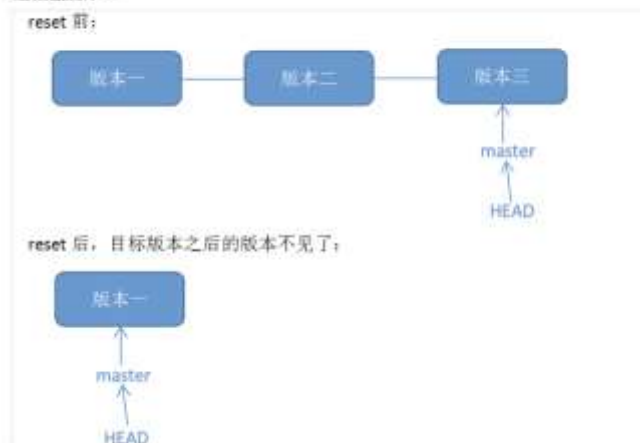
```
$ git push origin master
```

使用 git pull --rebase origin master 合并 GitHub 和 本地 的库，本地会多出之前不同步的文件，在使用 git push -u origin master 推送到 GitHub 库中。

**Git 版本恢复：**

方法一：git reset

原理：git reset的作用是修改HEAD的位置。即将HEAD指向的位置改变为之前存在的某个版本，如下图所示。假设我们要回到版本一：



### 1. 查看版本号:

可以使用命令"git log"查看:

```
MINGW64/c/MyUniversity/android_project/Android_midterm_project
f42aa0e..2df8325 master -> master
773540DE5KTOP-RVQ7LDP MINGW64 /c/MyUniversity/android_project/Android_midterm_project (master)
$ git log
commit 2df8325909977b96d62448df2ff1e17a26df1210 (HEAD -> master, origin/master)
Merge: 370750b f42aa0e
Author: yxty <773544953@qq.com>
Date: Sat Apr 14 20:07:51 2018 +0800

    Merge branch 'master' of https://github.com/yxlshk/Android_midterm_project

commit 370750b9aa61bb7533cd40640d98cf747cd58392
Author: yxty <773544953@qq.com>
Date: Sat Apr 14 20:07:36 2018 +0800

    try

commit f42aa0e173d665fee1f0c1f7401e6d7adb605eae
Merge: 66d936f c9f2aad
Author: ZhangZekun <13829062426@163.com>
Date: Fri Jan 5 14:09:34 2018 +0800

    Merge branch 'master' of https://github.com/yxlshk/Android_midterm_project

commit c9f2aad75af7feb5fcdc6c9285f4857a74da7db2
Author: ZhangZekun <13829062426@163.com>
Date: Sun Nov 26 16:45:07 2017 +0800

    编辑页面和详情页面增加了'字'

commit 43a6f7a2380db3dec2c5803ec43a73c033ccfd80
http://https://blog.csdn.net/yxtykds
```

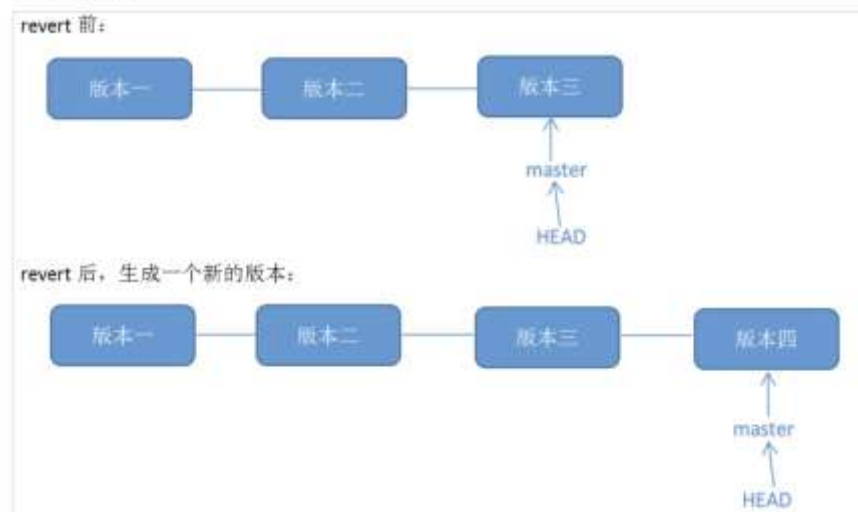
### 2.使用"git reset --hard 目标版本号"命令将版本回退:

### 3. 使用"git push -f"提交更改。

#### 方法 2:

##### 方法二: git revert

原理: git revert是用于"反做"某一个版本,以达到撤销该版本的修改的目的。比如,我们commits了三个版本(版本一、版本二、版本三),突然发现版本二不行(如:有bug),想要撤销版本二,但又不想影响撤销版本三的提交,就可以用 git revert 命令来反做版本二,生成新的版本四,这个版本四里会保留版本三的东西,但撤销了版本二的东西。如下图所示:



## 1. 查看版本号:

可以通过命令行查看 (输入git log) :

如图, 最近的两个版本分别叫: "add text.txt" (即新增了文件text.txt) 、"add text2.txt" (新增了文件text2.txt) 。这个时候我们不需要text.txt这个文件了, 那就是说不想要"add text.txt"那个版本的操作, 那可以通过反做"add text.txt"这个版本来实现。

```
Database_Sync $ git log
commit c3666fca61401bcd3de8bdd9688e7940a551dfde (HEAD -> master)
Author: yxlshk <773544953@qq.com>
Date:   Fri Jul 26 10:43:44 2019 +0800

    add text2.txt

commit 8b89621019c9adc6fc4d242cd41daeb13aeb9861
Author: yxlshk <773544953@qq.com>
Date:   Fri Jul 26 10:42:31 2019 +0800

    add text.txt

commit 82d13f8d2baeff9b3ce7895d3d80909c1dbf6875 (origin/master, origin/HEAD)
Author: yxty <773544953@qq.com>
Date:   Sun Dec 9 15:10:44 2018 +0800

    Revert to AAAA
```

需要反做的版本

## 2. 使用"git revert -n 版本号"反做, 并使用"git commit -m 版本名"提交:

(1) 反做, 使用"git revert -n 版本号"命令。如下命令, 我们反做版本号为8b89621的版本:

```
1 | git revert -n 8b89621019c9adc6fc4d242cd41daeb13aeb9861
```

注意: 这里可能会出现冲突, 那么需要手动修改冲突的文件。而且要git add 文件名。

(2) 提交, 使用"git commit -m 版本名", 如:

```
1 | git commit -m "revert add text.txt"
```

此时可以用"git log"查看本地的版本信息, 可见多生成了一个新的版本, 该版本反做了"add text.txt"版本, 但是保留了"add text2.txt"版本:

```
Database_Sync $ git log
commit 46628d6fda55e04c9a7c290c13420a93ebf1a08a (HEAD -> master, origin/master, origin/HEAD)
Author: yxlshk <773544953@qq.com>
Date:   Fri Jul 26 10:47:09 2019 +0800

    revert add text.txt

commit c3666fca61401bcd3de8bdd9688e7940a551dfde
Author: yxlshk <773544953@qq.com>
Date:   Fri Jul 26 10:43:44 2019 +0800

    add text2.txt
```

新版本

Git 工作原理:

### 3.3 工作目录、暂存区以及版本库概念

为了更好的学习Git，我们需要了解Git相关的一些概念，这些概念在后面的学习中会经常提到。

**版本库：**前面看到的.git隐藏文件夹就是版本库，版本库中存储了很多配置信息、日志信息和文件版本信息等。

**工作目录（工作区）：**包含.git文件夹的目录就是工作目录，主要用于存放开发的代码。

**暂存区：**.git文件夹中有很多文件，其中有一个index文件就是暂存区，也可以叫做stage。暂存区是一个临时保存修改文件的地方。



```
apple@DESKTOP-N226DJA MINGW64 /e/gitRepos/myRepo2 (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   hello.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   README.md

apple@DESKTOP-N226DJA MINGW64 /e/gitRepos/myRepo2 (master)
$ git reset HEAD hello.txt
Unstaged changes after reset:
M   README.md

apple@DESKTOP-N226DJA MINGW64 /e/gitRepos/myRepo2 (master)
$
```

git set head 文件名。将stage暂存的文件更改的未跟踪状态

直接删除与 `git rm` 文件名区别：

第一种方式，没有将删除操作放在暂存区，无法提交删除操作。需要手动添加，才能将删除操作放到暂存区，提交。

第二种方式，将操作放在了暂存区，可以直接提交删除操作。

## 3.5 本地仓库操作

### 将文件添加至忽略列表

一般我们总会有些文件无需纳入Git的管理，也不希望它们总出现在未跟踪文件列表。通常都是些自动生成的文件，比如日志文件，或者编译过程中创建的临时文件等。在这种情况下，我们可以在工作目录中创建一个名为.gitignore的文件（文件名称固定），列出要忽略的文件模式。下面是一个示例：

```
# no .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the TODO file in the current directory, not subdir/TODO
/TODO

# ignore all files in the build/ directory
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

# ignore all .pdf files in the doc/ directory
doc/**/*.pdf
```

### ◆ 查看远程仓库

如果想查看已经配置的远程仓库服务器，可以运行git remote命令。它会列出指定的每一个远程服务器的简写。如果已经克隆了远程仓库，那么至少应该能看到origin，这是Git克隆的仓库服务器的默认名字。

```
zhaogx@zhaogx MINGW64 /d/gitRepos/myGitRepo (master)
$ git remote
origin

zhaogx@zhaogx MINGW64 /d/gitRepos/myGitRepo (master)
$ git remote -v
origin https://gitee.com/ChuanZhiBoKe/myGitRepo.git (fetch)
origin https://gitee.com/ChuanZhiBoKe/myGitRepo.git (push)

zhaogx@zhaogx MINGW64 /d/gitRepos/myGitRepo (master)
$ git remote show origin
* remote origin
Fetch URL: https://gitee.com/ChuanZhiBoKe/myGitRepo.git
Push URL: https://gitee.com/ChuanZhiBoKe/myGitRepo.git
HEAD branch: master
Remote branch:
  master tracked
Local branch configured for 'git pull':
  master merges with remote master
Local ref configured for 'git push':
  master pushes to master (fast-forwardable)
```



## 3.6 远程仓库操作

### ◆从远程仓库中抓取与拉取

git fetch 是从远程仓库获取最新版本到本地仓库，不会自动merge

```
zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2
$ git init
Initialized empty Git repository in D:/gitRepos/repo2/.git/

zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2 (master)
$ git remote add origin https://gitee.com/ChuanZhiBoKe/repo1.git

zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2 (master)
$ git fetch origin master
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 13 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (13/13), done.
From https://gitee.com/ChuanZhiBoKe/repo1
 * branch          master       -> FETCH_HEAD
 * [new branch]     master       -> origin/master

zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2 (master)
$ git merge origin/master
```

## 3.6 远程仓库操作

### ◆从远程仓库中抓取与拉取

git pull 是从远程仓库获取最新版本并merge到本地仓库

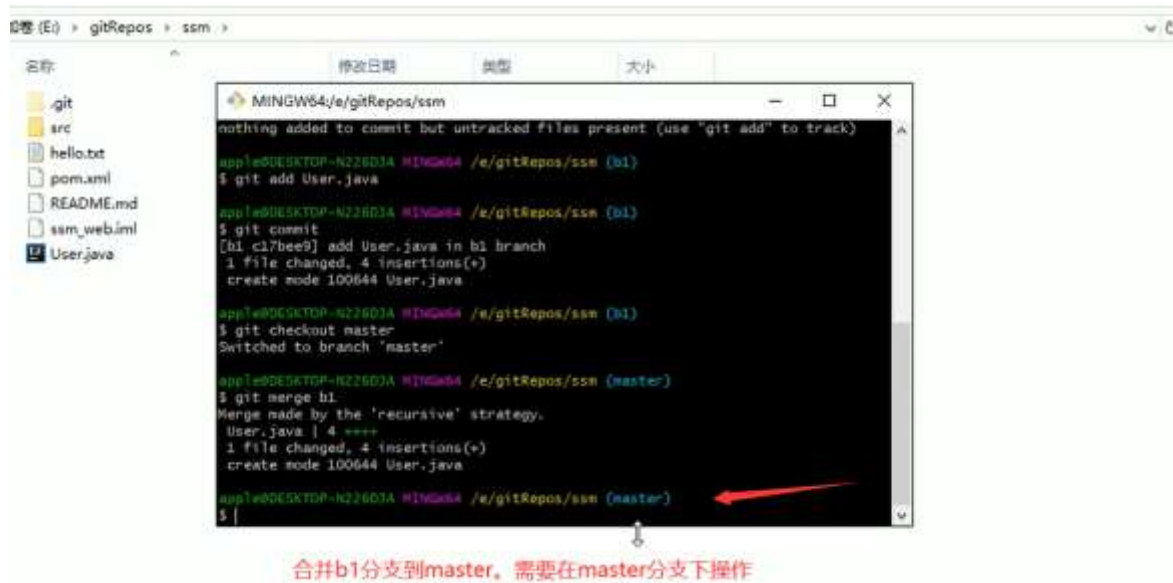
```
zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2
$ git init
Initialized empty Git repository in D:/gitRepos/repo2/.git/

zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2 (master)
$ git remote add origin https://gitee.com/ChuanZhiBoKe/repo1.git

zhaoqx@zhaoqx MINGW64 /d/gitRepos/repo2 (master)
$ git pull origin master
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 13 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (13/13), done.
From https://gitee.com/ChuanZhiBoKe/repo1
 * branch          master       -> FETCH_HEAD
 * [new branch]     master       -> origin/master
```

注意：如果当前本地仓库不是从远程仓库克隆，而是本地创建的仓库，并且仓库中存在文件，此时再从远程仓库拉取文件的时候会报错（fatal: refusing to merge unrelated histories），解决问题可以在git pull命令后加入参数--allow-unrelated-histories

Git remote -v//查看远程仓库的绑定地址



### 3.7 Git分支

#### ◆删除分支

```
chaogx@zhaogx MINGW64 /e/gitRepos/ssm
$ git checkout master
Switched to branch 'master'

chaogx@zhaogx MINGW64 /e/gitRepos/ssm (master)
$ git branch -d b1
Deleted branch b1 (was 897f100).

chaogx@zhaogx MINGW64 /e/gitRepos/ssm (master)
$ git branch
* master

chaogx@zhaogx MINGW64 /d/gitRepos/repo2 (master)
$ git branch -d b2
error: The branch 'b2' is not fully merged.
If you are sure you want to delete it, run 'git branch -D b2'.

chaogx@zhaogx MINGW64 /d/gitRepos/repo2 (master)
$ git branch -D b2
Deleted branch b2 (was 075ab59).

chaogx@zhaogx MINGW64 /d/gitRepos/repo2 (master)
$ git branch
* master
```

如果要删除的分支中进行  
中的-d参数改为-D

删除此分支，可以将命令

git删除远程分支

如果要删除远程仓库中的分支，可以使用  
命令git push origin -d branchName

### 3.7 Git分支

#### ◆综合应用

前面我们已经学习完成了Git分支相关的

工作场景如下：

开发某个网站。

为实现某个新的需求，创建一个分支(dev)

在这个分支上开展工作。

正在此时，你突然接到一个电话说有个bug

切换到你的线上分支(master)。

为这个紧急任务新建一个分支(fix)，并在

在测试通过之后，切换回线上分支(master)，然后合并这个修补分支(fix)，最后将改动推送到线上分支(master)。

切换回你最初工作的分支上(dev)，继续工作。

```
MINGW64/e/gitRepos/ssm
$ git branch dev
$ git checkout dev
Switched to branch 'dev'
$ git add UserDao.java
$ git commit -m "add UserDao.java in dev branch"
[dev 4a4da32] add UserDao.java in dev branch
1 file changed, 1 insertion(+)
create mode 100644 UserDao.java

MINGW64/e/gitRepos/ssm (dev)
$ git checkout master
Switched to branch 'master'

MINGW64/e/gitRepos/ssm (master)
$ git branch fix
$ git checkout fix
Switched to branch 'fix'
$ git add fix.java
$ git commit -m "add fix.java in fix branch"
[fix 1a1a1a1] add fix.java in fix branch
1 file changed, 1 insertion(+)
create mode 100644 fix.java

MINGW64/e/gitRepos/ssm (fix)
$ git merge master
Merge made by the 'recursive' strategy.
master...fix: 1 file changed, 1 insertion(+)
create mode 100644 fix.java

MINGW64/e/gitRepos/ssm (master)
$ git push origin master
Counting objects: 1, done.
Writing objects: 1 (100%) | 100 bytes | 0.00 MB | 0.00 MB/s
Compressing objects: 0 (0%) | 0 bytes | 0.00 MB | 0.00 MB/s
Compressing and sending objects: 0 (0%) | 0 bytes | 0.00 MB | 0.00 MB/s
Total: 1 (100%) | 100 bytes | 0.00 MB | 0.00 MB/s
remote: Compressing...
remote: Total 1 (100%) | 100 bytes | 0.00 MB | 0.00 MB/s
remote: Resolving deltas: 0 (0%) | 0 bytes | 0.00 MB | 0.00 MB/s
remote: Success: Repository updated.
To https://github.com:chaogx/ssm.git
   master
- [deleted]
+ [deleted]
```

在master分支的基础上创建分支



## 3.8 Git标签

像其他版本控制系统（VCS）一样，Git可以给历史中的某一个提交打上标签，以示重要。比较有代表性的是人们会使用这个功能来标记发布结点（v1.0、v1.2等）。标签指的是某个分支某个特定时间点的状态。通过标签，可以很方便的切换到标记时的状态。

在本节中，我们将学习：

- ◆ 列出已有的标签
- ◆ 创建新标签
- ◆ 将标签推送至远程仓库
- ◆ 检出标签
- ◆ 删除标签

## 3.8 Git标签

- ◆ 列出已有的标签

# 列出所有tag

\$ git tag

# 查看tag信息

\$ git show [tag]

```
MINGW64/e/gitRepos/ssm
v1.0
appleDESKTOP-N226D3A MINGW64 /e/gitRepos/ssm (dev)
$ git show v0.1
commit 4a4da32c827ff269cdf8b8087ab17c374ce50cc0 (HEAD -> dev, tag: v1.0, tag: v0.1)
Author: itcast <hello@itcast.com>
Date: Thu Feb 21 17:41:10 2019 +0800

    add UserDao.java in dev branch

diff --git a/UserDao.java b/UserDao.java
new file mode 100644
index 0000000..08072c1
--- /dev/null
+++ b/UserDao.java
@@ -0,0 +1,3 @@
+// 0000000..08072c1 = 08072c1
+public class UserDao{
+}
\ No newline at end of file
appleDESKTOP-N226D3A MINGW64 /e/gitRepos/ssm (dev)
$
```

## 3.8 Git标签

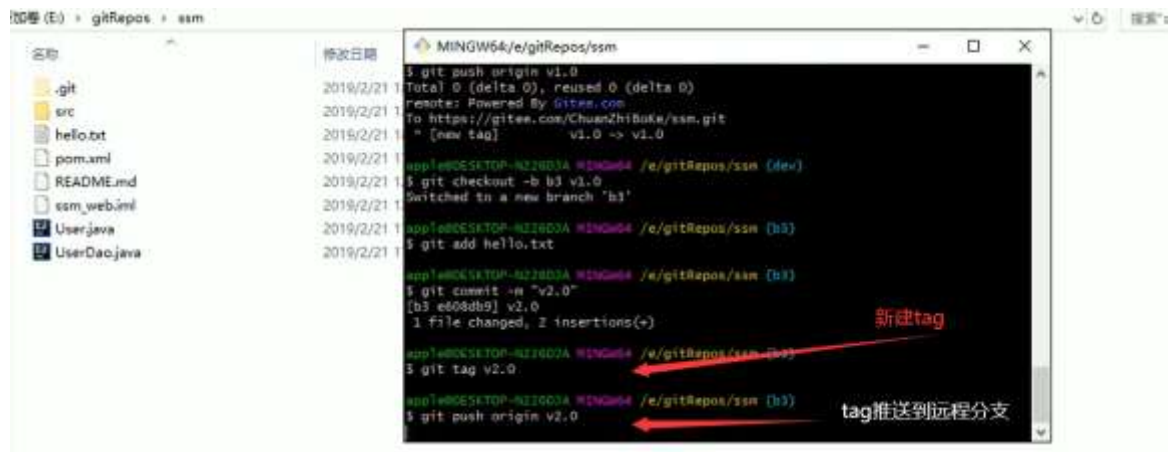
- ◆ 检出标签

# 新建一个分支，指向某个tag

\$ git checkout -b [branch] [tag]

```
zhaoqx@zhaoqx MINGW64 /d/gitRe
$ git checkout -b b3 v0.3
Switched to a new branch 'b3'
M       .idea/workspace.xml

zhaoqx@zhaoqx MINGW64 /d/gitRe
$ git checkout -b b3 v1.0
Switched to a new branch 'b3'
appleDESKTOP-N226D3A MINGW64 /e/gitRepos/ssm (b3)
$
```



### 3.8 Git标签

#### ◆删除标签

# 删除本地tag

\$ git tag -d [tag]

# 删除远程tag

\$ git push origin :refs/tags/[tag]

```

chuangzhiboke MINGW64 /d/gitRepos/myproject (master)
$ git tag
v0.1
v0.2
v0.3

chuangzhiboke MINGW64 /d/gitRepos/myproject (master)
$ git tag -d v0.3
Deleted tag 'v0.3' (was ff11ada)

chuangzhiboke MINGW64 /d/gitRepos/myproject (master)
$ git tag
v0.1
v0.2

chuangzhiboke MINGW64 /d/gitRepos/myproject (master)
$ git push origin :refs/tags/v0.3
remote: Powered By Gitee.com
To https://gitee.com/ChuanZhiBoke/myproject.git
- [deleted]

```



```

$ git status
HEAD detached at 796e78f (dirty) 796e78f
Last commands done (5 commands done):
pick 0a78e04 update: 准备接口
pick 3dec38e 暂存移动网页设置
(see more in file .git/rebase-merge/done)
No commands remaining.
You are currently editing a commit while rebasing branch 'feature/xiao' on '796e78f'.
(use "git commit --amend" to amend the current commit)
(use "git rebase --continue" once you are satisfied with your changes)

nothing to commit, working tree clean

chuangzhiboke MINGW64 /d/Projects/dev/web-project/vctravel-web (feature/xiao|REBASE 5/5)
$ git rebase --continue
Successfully rebased and updated refs/heads/feature/xiao

```

查看一番发现,由于之前使用过 `git pull --rebase origin develop` 命令拉取代码,使用过 `git rebase` 执行代码覆盖,但是上一次进程还没有完成导致

原因:

查看 `git` 的提示,大概意思是 你当前正在编辑的提交将要覆盖在 `796e78` commitid 上

两种解决方案

1. 使用 `git commit --amend` 命令修订当前的提交
2. 使用 `git rebase --continue` 命令继续代码的提交(推荐),执行之后,需要重新提交,解决一下当前的代码冲突之后重新提交直至没有 `rebase` 提示,就可以正常提交了

## "--amend" 命令说明

`git commit --amend` 命令用来对最近的提交 (commit) 进行“撤销”或“补充”操作。比如，你在提交的文件中有敏感信息，或者漏提了文件，或者 commit 信息不明确等，都可以使用 `--amend` 命令进行补救。

该命令只是重写最近的提交，并不会在最近的提交上新增一个提交。

(`git commit --amend` 实际上是重新生成了一个提交，因为修改已经提交 (commit) 的任何内容，都会导致其 commit ID 改变。使用 `git log` 查看并没有多一个提交是因为新的提交覆盖了被修改的提交)。

## "--amend" 的适用情况

该命令适用于还没有 `push` 到 Git 服务器的提交 (commit)。使用 `git commit --amend` 后再使用 `git push` 就会将 `--amend` 后的 commit 推送到服务器。

如果提交已经被 `push` 到了服务器，使用该命令后再 `push` 就会出现上面的 [rejected] 问题。当然，你可以使用 `git push -f origin master` 强行上推。但是，这么做有两个前提：

1. 其他人没有 `clone` 你的项目。如果其他人正在使用你的项目，等你把 `--amend` 后的 commit 推送到服务器后，其他人使用 `git pull` 下载时，就会产生 merge 信息，需要他人手动修改。除非他重新 `clone` 你的仓库。
2. 只有你自己向这个项目提交文件。如果其他人也向这个项目提交文件，在你使用了 `git push -f` 后，很可能把其他人的 commit 干掉。