# 资源

* <https://git-scm.com/>
* <https://git-scm.com/book/zh/v2>
* <https://backlog.com/git-tutorial/cn/>
* <https://learngitbranching.js.org/?locale=zh_CN>

# 案例1 - 本地仓库

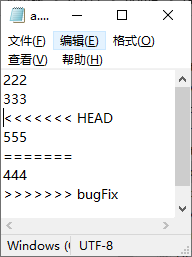
# 初始化仓库   
git init   
   
# 写一个文件   
echo 111>>a.txt   
   
# 添加文件到暂存区   
git add \*   
# 或者 git add .   
# 或者 git add a.txt   
   
# 提交到本地库   
git commit -m "初始提交"   
   
# 修改文件，再提交两次   
echo 222>a.txt   
git add \*   
git commit -m "提交2"   
   
echo 333>>a.txt   
git add \*   
git commit -m "提交3"   
   
# 查看提交树   
git log --graph   
git log --graph --pretty=oneline   
   
# 为了可以正常显示中文，做如下设置：   
# 提交的编码   
git config --global i18n.commitencoding utf-8   
# 日志显示的编码   
git config --global i18n.logoutputencoding utf-8   
# 系统环境变量，linux中把set换成export   
set LESSCHARSET=utf-8   
# 再看日志，中文可以正常显示   
git log --graph --pretty=oneline



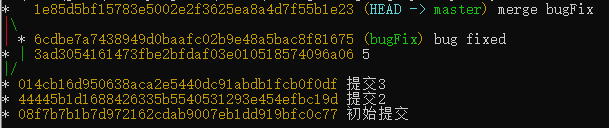
# 案例2 - 分支、merge

# 创建新的分支bugFix并检出它   
git branch bugFix   
git checkout bugFix   
# 或者可以使用简化的命令来合并上面两步： git checkout -b bugFix   
   
# 提交一次bugFix   
echo 444>>a.txt   
git add a.txt   
git commit -m "bug fixed"   
   
# 检出master分支并提交一次   
git checkout master   
echo 555>>a.txt   
git add a.txt   
git commit -m "5"   
   
# 将bugFix分支合并到master分支   
git merge bugFix

接下来，对自动合并后的a.txt进行手动修正，修正后再提交master分支



git add a.txt   
git commit -m "merge bugFix"   
   
# 查看提交树   
git log --graph   
git log --graph --pretty=oneline   
   
# 合并后，bugFix如果不再需要使用，可以将其删除   
git branch -d bugFix



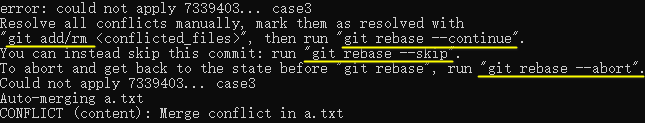
# 案例3 - rebase

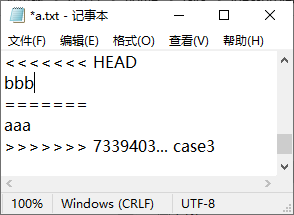
代码合并可以选择merge，也可以选择rebase，rebase简单的将当前分支的多次新提交复制到指定分支上，形成一条干净的提交树

# 创建新的分支case3并检出它   
git checkout -b case3   
   
# 提交一次case3   
echo aaa>>a.txt   
git add a.txt   
git commit -m "case3"   
   
# 检出master分支并提交一次   
git checkout master   
echo bbb>>a.txt   
git add a.txt   
git commit -m "bbb"   
   
# 回到case3分支   
git checkout case3   
   
# 将case3分支新的提交记录，复制到master分支   
git rebase master

rebase时可能出现两个分支文件修改冲突的情况，这时需要按提示选择其中一项适合的操作：

* 将自动合并的文件手动进行修正后，添加到暂存区再执行git rebase --continue
* 丢弃当前分支的修改，使用目标分支的修改，执行git rebase --skip
* 取消执行rebase，回到执行git rebase之前的状态，执行git rebase --abort





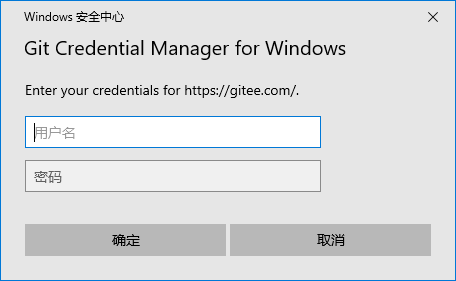
# 解决冲突，手动修正文件后，继续完成 rebase

git rebase --continue

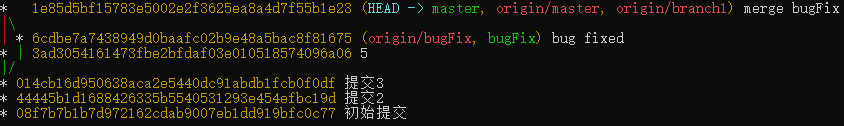
# 案例4 - 远程仓库

# 设置开发者个人信息   
git config --global user.name "王海涛"   
git config --global user.email "wht@wht.com"   
# 查看   
git config --global user.name   
git config --global user.email   
   
# 当前本地仓库添加关联的远程仓库   
# origin是为远程仓库命名，习惯使用origin   
git remote add origin https://gitee.com/benwang6/demo.git   
   
# 本地的master分支，推送到远程库的master分支   
# 这里实际上是 master:master 格式的缩写   
git push origin master   
# 或者推送全部分支   
git push origin --all   
# 也可以将本地分支推送到远程的另一个分支   
# 这个例子演示将本地master分支推送到远程branch1分支，该分支不存在会被自动创建   
git push origin master:branch1   
   
# 通过-u参数，让本地master分支关联到远程的master分支，以后推送可以不用再指定分支   
git push origin -u master   
   
git push

推送时，会弹出windows的添加凭据对话框



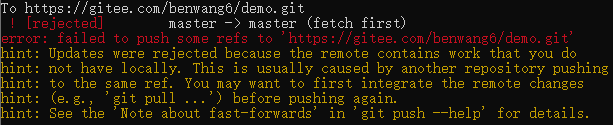
# 提交完成后，查看提交树   
# 红色为远程分支的名称，绿色为本地分支的名称   
git log --graph --pretty=oneline



# 案例5

首先在远程仓库中做几次提交,来模拟团队开发中，其他开发者向远程分支提交了代码。

# 远程仓库有新的提交，而本地仓库没有时，远程仓库会阻止推送   
git push



# 这时你要做的是，把远程库的代码下载回来，然后将下载的代码与本地的代码进行合并   
   
# 先下载远程最新代码   
git fetch   
# 再将下载的代码合并到本地代码   
# 经过自动合并的代码可能需要进行手动修正   
git merge origin/master   
   
# 或者，也可以用简化的命令，将上面两步合并为一步来执行   
# git pull命令实际上是 fetch+merge   
git pull   
   
# 代码在本地合并后，可以向远程推送   
git add .   
git commit -m "完成合并，解决了冲突"   
git push

# 常用命令

### 1、初始化仓库

git init

### 2、将文件添加到仓库

git add 文件名 # 将工作区的某个文件添加到暂存区，空格隔开可写多个文件   
git add -u # 添加所有被tracked文件中被修改或删除的文件信息到暂存区，不处理untracked的文件   
git add -A # 添加所有被tracked文件中被修改或删除的文件信息到暂存区，包括untracked的文件   
git add . # 将当前工作区的所有文件都加入暂存区   
git add -i # 进入交互界面模式，按需添加文件到缓存区

### 3、将暂存区文件提交到本地仓库

git commit -m "提交说明" # 将暂存区内容提交到本地仓库   
git commit -a -m "提交说明" # 跳过缓存区操作，直接把工作区内容提交到本地仓库

### 4、查看仓库当前状态

git status

### 5、比较文件异同

git diff # 工作区与暂存区的差异   
git diff 分支名 #工作区与某分支的差异，远程分支这样写：remotes/origin/分支名   
git diff HEAD # 工作区与HEAD指针指向的内容差异   
git diff 提交id 文件路径 # 工作区某文件当前版本与历史版本的差异   
git diff --stage # 工作区文件与上次提交的差异(1.6 版本前用 --cached)   
git diff 版本TAG # 查看从某个版本后都改动内容   
git diff 分支A 分支B # 比较从分支A和分支B的差异(也支持比较两个TAG)   
git diff 分支A...分支B # 比较两分支在分开后各自的改动   
   
   
   
# 另外：如果只想统计哪些文件被改动，多少行被改动，可以添加 --stat 参数

### 6、查看历史记录

git log # 查看所有commit记录(SHA-A校验和，作者名称，邮箱，提交时间，提交说明)  
git log -p -次数 # 查看最近多少次的提交记录  
git log --stat # 简略显示每次提交的内容更改  
git log --name-only # 仅显示已修改的文件清单  
git log --name-status # 显示新增，修改，删除的文件清单  
git log --oneline # 让提交记录以精简的一行输出  
git log –graph –all --online # 图形展示分支的合并历史  
git log --author=作者 # 查询作者的提交记录(和grep同时使用要加一个--all--match参数)  
git log --grep=过滤信息 # 列出提交信息中包含过滤信息的提交记录  
git log -S查询内容 # 和--grep类似，S和查询内容间没有空格  
  
git log fileName # 查看某文件的修改记录，找背锅专用

### 7、代码回滚

git reset HEAD^ # 恢复成上次提交的版本   
git reset HEAD^^ # 恢复成上上次提交的版本，就是多个^，以此类推或用~次数   
   
git reflog   
   
git reset --hard 版本号   
   
--soft：只是改变HEAD指针指向，缓存区和工作区不变；   
--mixed：修改HEAD指针指向，暂存区内容丢失，工作区不变；   
--hard：修改HEAD指针指向，暂存区内容丢失，工作区恢复以前状态；

### 8、同步远程仓库

git push -u origin master

### 9、删除版本库文件

git rm 文件名

### 10、版本库里的版本替换工作区的版本

git checkout -- test.txt

### 11、本地仓库内容推送到远程仓库

git remote add origin git@github.com:帐号名/仓库名.git

### 12、从远程仓库克隆项目到本地

git clone git@github.com:git帐号名/仓库名.git

### 13、创建分支

git checkout -b dev  
-b表示创建并切换分支  
上面一条命令相当于一面的二条：  
git branch dev //创建分支  
git checkout dev //切换分支

### 14、查看分支

git branch

### 15、合并分支

git merge dev  
//用于合并指定分支到当前分支  
  
git merge --no-ff -m "merge with no-ff" dev  
//加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并

### 16、删除分支

git branch -d dev

### 17、查看分支合并图

git log --graph --pretty=oneline --abbrev-commit

### 18、查看远程库信息

git remote  
// -v 显示更详细的信息

### 19、git相关配置

# 安装完Git后第一件要做的事，设置用户信息(global可换成local在单独项目生效)：   
git config --global user.name "用户名" # 设置用户名   
git config --global user.email "用户邮箱" #设置邮箱   
git config --global user.name # 查看用户名是否配置成功   
git config --global user.email # 查看邮箱是否配置

# 其他查看配置相关  
git config --global --list # 查看全局设置相关参数列表  
git config --local --list # 查看本地设置相关参数列表  
git config --system --list # 查看系统配置参数列表  
git config --list # 查看所有Git的配置(全局+本地+系统)  
git config --global color.ui true //显示git相关颜色

### 20、撤消某次提交

git revert HEAD # 撤销最近的一个提交  
git revert 版本号 # 撤销某次commit

### 21、拉取远程分支到本地仓库

git checkout -b 本地分支 远程分支 # 会在本地新建分支，并自动切换到该分支  
git fetch origin 远程分支:本地分支 # 会在本地新建分支，但不会自动切换，还需checkout  
git branch --set-upstream 本地分支 远程分支 # 建立本地分支与远程分支的链接

### 22、标签命令

git tag 标签 //打标签命令，默认为HEAD  
git tag //显示所有标签  
git tag 标签 版本号 //给某个commit版本添加标签  
git show 标签 //显示某个标签的详细信息

### 23、同步远程仓库更新

git fetch origin master   
 //从远程获取最新的到本地，首先从远程的origin的master主分支下载最新的版本到origin/master分支上，然后比较本地的master分支和origin/master分支的差别，最后进行合并。