git命令

本文章是自己对<https://www.liaoxuefeng.com/wiki/896043488029600>这篇教程的理解与整合，为了更容易理解，可以参考这个教程

# 一，创建版本库（命令控制行要在你的本地库打开，也就是命令控制行的路径要是你的本地库）

**如果第一次使用git，先全局配置一下：用户名和邮箱**



git init |初始化，变为一个本地仓库

说明：进入该仓库,通过文件查找->显示隐藏文件->删除git文件夹即可把本地仓库重新变为普通文件夹。

git add readme.txt |添加文件到暂存区

git commit -m “注释说明” |提交到分支

# 二，时光穿梭机（命令控制行要在你的本地库打开，也就是命令控制行的路径要是你的本地库）

## 1.查看状态和修改说明

git status |查看工作区的文件是什么状态，是否修改，是否添加，是否提交

git diff readme.txt |查看工作区的文件相比较之前具体修改了什么内容

cat readme.txt |查看文件内容

## 2.版本回退

git log |显示从最近到最远的提交日志

git log --pretty=oneline |显示从最近到最远的提交日志（显示每次提交版本关键的信息）

git reset --hard HEAD^ |回退上一个版本

git reset --hard HEAD^^ |回退上一个上一个版本

git reset --hard 1094a |回退到版本号为1094a的那个版本

git reflog 记录每次的命令（查看命令历史，以便确定要回到未来的哪个版本）

## 3.管理修改

git diff HEAD -- readme.txt |查看工作区和版本库里面最新版本的区别

第一次修改 -> git add -> 第二次修改 -> git commit

Git管理的是修改，当你用git add命令后，在工作区的第一次修改被放入暂存区，准备提交，但是，在工作区的第二次修改并没有放入暂存区，所以，git commit只负责把暂存区的修改提交了，也就是第一次的修改被提交了，第二次的修改不会被提交。

## 4.撤销修改

git checkout -- readme.txt

把readme.txt文件在工作区的修改全部撤销，这里有两种情况：

一种是readme.txt自修改后还没有被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态；

一种是readme.txt已经添加到暂存区后，又作了修改，现在，撤销修改就回到添加到暂存区后的状态。（注意：这里指的是，add暂存区以后，不小心又修改了文件，执行命令以后，撤销修改，但是文件还是在暂存区）

总之，就是让这个文件回到最近一次git commit或git add时的状态。

git reset HEAD readme.txt |可以把暂存区的修改撤销掉（unstage），重新放回工作区。

**适用场景**：修改以后，add添加一次，但是这次修改是错误的，不想提交。那么执行

git reset HEAD readme.txt ，把暂存区的修改撤销掉，重新放回工作区。现在暂存区是干净的，工作区有修改。再用git checkout -- readme.txt 撤销工作区的修改。

git reset命令既可以回退版本，也可以把暂存区的修改回退到工作区。当我们用HEAD时，表示最新的版本。

## 5.删除文件——在Git中，删除也是一个修改操作

rm readme.txt |删除了工作区中的文件readme.txt

1. 确实要删除文件

因为删除了工作区的文件，此时工作区和版本库就不一致了，执行git status会告诉你工作区的文件已经删除，所以我们也需要删除版本库中的文件

git rm readme.txt

git commit -m “删除文件”

现在，文件就从版本库中删除了

1. 删错文件，需要恢复 注意：从来没有被添加到版本库就被删除的文件，是无法恢复的！

git checkout –readme.txt

git checkout其实是用版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“一键还原”。

# 三，远程仓库

## 1.添加远程库（命令控制行要在你的本地库打开，也就是命令控制行的路径要是你的本地库）

ssh-keygen -t rsa -C “1871819749@qq.com” |生成密钥，然后进行设置好密钥的步骤

ssh -T [git@github.com](mailto:git@github.com) |验证ssh是否设置关联成功

git remote add origin git@github:zxb008/test.git |把本地库和远程库关联，zxb008是用户名，test是github上的远程库，然后设置一个变量orgin（英文中库的意思）来保存远程库在本地的引用，remote是关键字，英文中是“远程”的意思

说明：git remote remove origin |取消本地仓库关联远程仓库

git push -u origin master |把本地库的所有内容推送到远程库上面，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。 （origin代表远程库）

git push origin master |此后推送最新的修改，就可以简化了

## 2.从远程库克隆（命令控制行在一个非本地库的目录中打开，克隆的远程仓库将会出现在当前这个目录下面）

git clone git@github.com:zxb008/test2.git |克隆远程库上面的一个仓库test2仓库到本地

cd test2 |切换到test2文件中

ls |显示当前文件目录

# 四，分支管理

## 1.创建与合并分支

git checkout -b dev |创建dev分支，并切换到dev分支

git checkout命令加上-b参数表示创建并切换，相当于以下两条命令：

git branch dev

git checkout dev

git branch |查看当前分支，会列出所有的分支（这里会显示dev和master两个分支），当前分支前面会有一个 \*

注意：现在再添加并提交就是把工作区的文件提交到了当前分支dev上面。此时最近一次修改后的文件保存到了dev分支上，上上次的修改到最早的修改则保存在master分支上。

git checkout master |dev分支的工作已经完成，切换回master分支， 此时再看工作区的文件发现刚才的修改内容不见了，因为那次的最近的修改是保存在dev分支上面，但是我们已经切换到了masster分支上面，只能看见上上次的修改后的文件。

git merge dev |把dev分支合并到当前分支master上面，相当于master分支跑到了dev分支上了（执行"快进式合并"：fast-farward merge），这时候我们就能在工作区看见最近一次修改后的文件了。

git branch -d dev |删除dev分支，此时再git branch，就只剩下master分支了

## 2.解决冲突

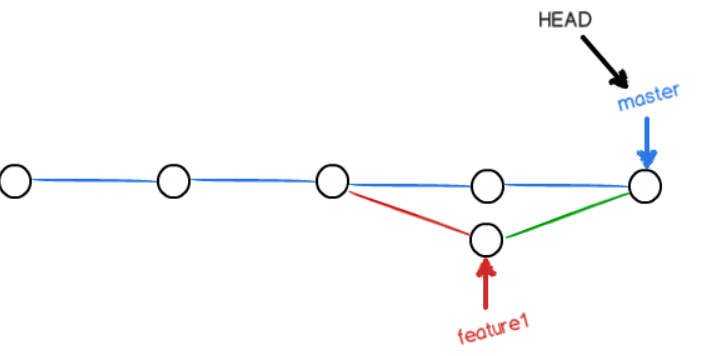
在上面的过程中，我们发现在新建的dev分支上上面添加并提交最新一次修改，

然后再切换到分支master上面，再把dev分支合并到当前分支master上面的前提是**：切换到master分支后，对文件没有进行新的修改同时添加提交。**

**注意：如果切换到master分支后，又对文件进行新的修改同时添加并提交，执行：git merge dev，那么这样是无法快速合并的，会发生冲突**

**解决方法：**执行git merge dev进行快速合并以后，发生冲突，先cat readme.txt 查看文件的内容，Git用<<<<<<标记冲突开始，后面跟的是当前分支中的内容。HEAD指向当前分支末梢的提交。=======之后，>>>>>>>之前是要merge过来的另一条分支上的代码。>>>>>>>之后的dev是该分支的名字。然后借用vim或者IDE或者直接找到冲突文件，修改成为我们希望的内容（比如：删除切换到master分支上所做的修改，也要删除git自动生成的冲突代码分隔符），然后再添加并提交就可以了。

**注意：**手动编辑以后，此时仍在master分支上面，只需要add添加并commit提交，不再需要执行：git merge dev 这个命令。因为git的合并已经完成了，只是造成了冲突，git给我们生成了冲突文件,我们需要手动解决文件中的冲突（确定哪些行删除，哪些行保留），冲突文件处理完后我们需要重新添加该冲突文件到git版本控制中。再执行git branch -d dev删除dev分支。



git log –graph |可以看到分支合并图。

**参考链接**：<https://www.cnblogs.com/gavincoder/p/9071959.html>

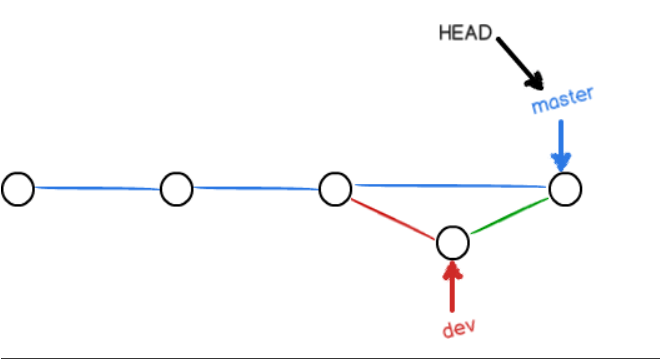
<https://blog.csdn.net/heibao111728/article/details/82879036>

## 3.分支管理策略

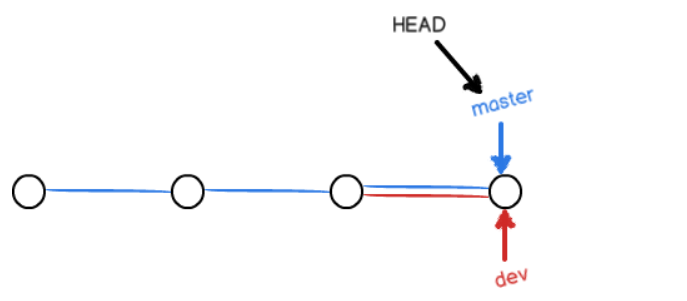
git merge -no-ff -m”注释说明” dev |非快进式合并分支的方式

区别：快进式会把当前分支master直接覆盖在dev分支上面，而-no-ff 则会把当前分支前进一步，再连接dev分支上面。-no-ff的方式更好，合并后的历史有分支，能看出来曾经做过合并，可以看出分支信息；而快进式fast forward合并就看不出来曾经做过合并，删除分支后，会丢掉分支信息。。另外，假如有多个不同的分支，快进式会出问题，-no-ff则一直保持一个主线，从而避免问题

**-no-off ：----------**



**快进式：------------**



## 4.Bug分支

在Git中，由于分支是如此的强大，所以，每个bug都可以通过一个新的临时分支来修复，修复后，合并分支，然后将临时分支删除。

当你接到一个修复一个代号101的bug的任务时，很自然地，你想创建一个分支issue-101来修复它，但是，等等，当前正在dev上进行的工作还没有提交。并不是你不想提交，而是工作只进行到一半，还没法提交，预计完成还需1天时间。但是，必须在两个小时内修复该bug，怎么办？

git stash |可以把当前dev分支工作现场“储藏”起来，等以后恢复现场后继续工作,此时再git status 会发现工作区是干净的。

然后从dev分支上切换到相应的master分支上，再创建bug分支，进行修复。修复完成以后，合并该bug分支，再删除bug分支，就完成了bug的修复。再切换到dev分支上，继续完成dev的工作。

git stash list |查看储存的工作现场

git stash apply |恢复dev的工作现场，但是stash内容并没有删除

git stash drop |删除stash内容

git stash pop |恢复的同时把stash的内容也删除了，此时再执行git stash list就看不到stash内容了

## 5.Feature分支

软件开发中，总有无穷无尽的新的功能要不断添加进来。添加一个新功能时，你肯定不希望因为一些实验性质的代码，把主分支搞乱了，所以，每添加一个新功能，最好新建一个feature分支，在上面开发，完成后，合并，最后，删除该feature分支。但是，假如你在feature分支上已经开发完成，但是还没来得及合并到主分支上面，老板突然说取消这个新的功能，这时候我们应该执行git branch -d feature 来删除这个分支，但是会删除失败，因为这个分支没有合并到主分支上面。所以应该执行git branch -D feature强行删除

git branch -d feature |强行删除分支feature，-D表示强行

## 6.多人协作

### 1.查看远程库的信息

当你从远程仓库克隆时，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，远程仓库的默认名称是origin。

git remote |查看远程库的信息

git remote -v |显示远程库的更加详细的信息

### 2.推送分支

把本地库和远程库关联（git remote add origin git@github:zxb008/test.git）以后，需要推送分支到远程库分支上面，例如：  
git push origin master（推送本地主分支到远程库的主分支上）；git push origin dev（推送本地开发分支到远程库的开发分支上）

但是，并不是一定要把本地分支往远程推送，那么，哪些分支需要推送，哪些不需要呢？

master分支是主分支，因此要时刻与远程同步；

dev分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步；

bug分支只用于在本地修复bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个bug；

feature分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发。

总之，就是在Git中，分支完全可以在本地自己藏着玩，是否推送，视你的心情而定！

### 3.抓取分支

多人协作时，大家都会往master和dev分支上推送各自的修改。现在，模拟一个你的小伙伴，可以在另一台电脑（注意要把SSH Key添加到GitHub）或者同一台电脑的另一个目录下克隆。当你的小伙伴从远程库clone时，默认情况下，你的小伙伴只能看到本地的master分支（其实远程库上面也有dev分支，但是克隆到本地以后，默认只会克隆master分支到本地）。可以用git branch命令查看有那几个分支，会发现只有master分支。

那么如何在dev分支上开发呢？

你的小伙伴要在dev分支上开发，就必须创建远程origin的dev分支到本地。执行git checkout -b dev origin/dev来创建远程origin的dev分支到本地。然后在某个文件上进行开发，开发完毕以后，add添加，commit提交，再远程推送 git push origin dev（注意：这里不要误认为添加并提交以后还需要把dev分支合并到master主分支上面去。其实，应该是所有的功能（不同的开发人员对应不同的功能）在dev分支上开发完毕，再由项目组长来把dev分支合并到主分支master上面）

git checkout -b dev origin/dev |创建远程origin的dev分支到本地

在某个文件上进行相应的开发

git add readme.txt |添加

git commit -m “一个功能” |提交

git push origin dev |推送本地dev到远程库里面的dev

场景：你的小伙伴在本地dev上对一个文件hello进行修改，并推送到了远程库上面的dev，刚好你也在本地dev上对同一个文件hello（如果不是同一个文件，也是一样失败，因为一个项目有好多文件，文件变化，项目就发生变化）进行修改，再add添加并commit提交，然后也进行push推送到远程库上面的dev，但是发现推送失败。

原因：因为你的小伙伴的最新提交和你试图推送的提交有冲突

方法：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。

git branch –-set-upstrem-to=origin/dev dev |指定本地dev分支与远程origin/dev分支的链接

git pull |从远程库中获取某个分支的更新，再与本地指定的分支进行自动merge（合并），合并可能有冲突。也就是把最新的提交从origin/dev抓下来（就是小伙伴那次提交），同时进行合并，合并可能有冲突。

然后利用vim命令对冲突进行修改合并

（注意：这里不需要git add，个人认为，将远程库中的分支和本地分支和并以后，已经在暂存区了）

git commit -m “解决了dev冲突” |提交

git push origin dev |推送到远程库上面的dev分支

说明：

-----------------------------------------------------------------

git fetch |相当于是从远程获取最新到本地，不会自动merge

git pull = git fetch +git merge

-------------------------------------------------------------------

1、将远程指定分支 拉取到 本地指定分支上：

git pull origin <远程分支名>:<本地分支名>

2、将远程指定分支 拉取到 本地当前分支上：

git pull origin <远程分支名>

3、将与本地当前分支同名的远程分支 拉取到 本地当前分支上(需先关联远程分支= git branch –-set-upstrem-to=origin/dev dev)

git pull

## 7.Rebase

**git rebase**  |把分叉的提交历史“整理”成一条直线，看上去更直观。缺点是本地的分叉提交已经被修改过了。

# 五，标签管理

## 1.创建标签

<https://www.liaoxuefeng.com/wiki/896043488029600/902335212905824>

## 2.操作标签

<https://www.liaoxuefeng.com/wiki/896043488029600/902335479936480>

# 六，自定义git

<https://www.liaoxuefeng.com/wiki/896043488029600/900785521032192>

# 七，在Git项目中定义.gitignore文件,可以忽略某些项目文件提交到本地库中

<https://www.cnblogs.com/kevingrace/p/5690241.html>