Git 学习笔记

- 作者：lu KONG
- 此md为作者学习Git时的笔记，用于日后查阅以及可能的分享，文章主题内容以及主要的学习资料来自廖雪峰的官方网站，如有任何建议和指导，欢迎联系作者
- 感谢廖雪峰老师
- v2.0 --*2020年2月15日*

安装git的初始设置

```
$ git config --global user.name "Your Name"
$ git config --global user.email "email@example.com"
```

创建版本库

```
$ mkdir learngit
$ cd learngit
$ pwd
/Users/michael/learngit

$ git init
>>>Initialized empty Git repository in /Users/michael/learngit/.git/
```

初始化一个Git仓库，使用git init命令。

添加文件到Git仓库，分两步：

1. 使用命令git add ，注意，可反复多次使用，添加多个文件；
2. 使用命令git commit -m ，完成。

要随时掌握工作区的状态，使用git status命令。

如果git status告诉你有文件被修改过，用git diff可以查看修改内容。

时间机器

版本回退

HEAD指向的版本就是当前版本，

```
$ git reset --hard HEAD^ #HEAD^^ ...HEAD~100
```

因此，Git允许我们在版本的历史之间穿梭，使用命令`git reset --hard commit_id`。

穿梭前，用`git log`可以查看提交历史，以便确定要回退到哪个版本。

```
**$ git log --pretty=oneline
```

要重返未来，用`git reflog`查看命令历史，以便确定要回到未来的哪个版本。

```
$ cat readme.txt #to read the txt
----------------------------------------
$ git diff HEAD -- readme.txt
```

撤下修改

- 场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令

  ```
  git    checkout -- file
  or
  $git restore <file>
  ```

- 场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令`git reset HEAD <file>`，就回到了场景1，第二步按场景1操作。

  ```
  $git reset HEAD <file>
  or
  $git restore --staged <file>
  ```

- 场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考版本回退一节，不过前提是没有推送到远程库。

删除

```
$ git rm test.txt

git remote add origin https://github.com/lu-kong/learngit.git
git push -u origin master
```

远程仓库

Github 远程库的准备

1. 创建SHH—key
   第1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，如果已经有了，可直接跳到下一步。如果没有，打开Shell（Windows下打开Git Bash），创建SSH Key：

   ```
   $ ssh-keygen -t rsa -C "youremail@example.com"
   ```

2. 你需要把邮件地址换成你自己的邮件地址，然后一路回车，使用默认值即可，由于这个Key也不是用于军事目的，所以也无需设置密码。

3. 如果一切顺利的话，可以在用户主目录里找到.ssh目录，里面有id_rsa和id_rsa.pub两个文件，这两个就是SSH Key的秘钥对，id_rsa是私钥，不能泄露出去，id_rsa.pub是公钥，可以放心地告诉任何人。

4. 第2步：登陆GitHub，打开“Account settings”，“SSH Keys”页面:
   然后，点“Add SSH Key”，填上任意Title，在Key文本框里粘贴id_rsa.pub文件的内容可以直接txt打开

关联远程库

要关联一个远程库，使用命令

```
git remote add origin git@server-name:path/repo-name.git;
```

关联后，使用命令`git push -u origin master`第一次推送master分支的所有内容；

此后，每次本地提交后，只要有必要，就可以使用命令`git push origin master`推送最新修改；

分布式版本系统的最大好处之一是在本地工作完全不需要考虑远程库的存在， 也就是有没有联网都可以正常工作，而SVN在没有联网的时候是拒绝干活的！ 当有网络的时候，再把本地提交推送一下就完成了同步，真是太方便了！

从远程库克隆

1. 要克隆一个仓库，首先必须知道仓库的地址，然后使用git clone命令克隆。

2. 进到要存放库的目录上层`\Documents\Github\`

3. 使用`git clone` 命令， 加上远程库的地址

4. Git支持多种协议，包括https，但通过ssh支持的原生git协议速度最快。

分支管理

Git鼓励大量使用分支：

- 查看分支：git branch
- 创建分支：git branch
- 切换分支：git checkout 或者git switch
- 创建+切换分支：git checkout -b 或者git switch -c
- 合并某分支到当前分支：git merge
- 删除分支：git branch -d

解决冲突

需要手动修改，把当前分支下出现conflict的文件手动合并修改。 Git用`<<<<<<<`，`=======`，`>>>>>>>`标记出不同分支的内容，我们修改如下后保存： 然后提交， 则当前分支merge到修改后的分支，另一个冲突分支留在原处。 *用带参数的`git log`也可以看到分支的合并情况：*

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

通常，合并分支时，如果可能，Git会用`Fast forward`模式，但这种模式下，删除分支后，会丢掉分支信息。 如果要强制禁用Fast forward模式，Git就会在merge时生成一个新的commit， 这样，从分支历史上就可以看出分支信息。

准备合并dev分支，请注意--no-ff参数，表示禁用Fast forward：

```
$ git merge --no-ff -m "merge with no-ff" dev
```

分支策略

在实际开发中，我们应该按照几个基本原则进行分支管理：

- 首先，master分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；
- 那在哪干活呢？干活都在dev分支上，也就是说，dev分支是不稳定的，到某个时候，比如1.0版本发布时，再把dev分支合并到master上，在master分支发布1.0版本；
- 你和你的小伙伴们每个人都在dev分支上干活，每个人都有自己的分支，时不时地往dev分支上合并就可以了。

bug 分支

对于bug处理， 可以把当前分支`stash` 起来 修复bug时，我们会通过创建新的`bug`分支进行修复，然后合并，最后删除；

- 当手头工作没有完成时，先把工作现场`git stash`一下，然后去修复bug，修复后，再`git stash pop`，回到工作现场；

```
Or,
$git stash list
$git stash apply
$git stash drop
```

- 在master分支上修复的bug，想要合并到当前dev分支， 可以用`git cherry-pick <commit>`命令，把bug提交的修改“复制”到当前分支，避免重复劳动。

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。

和bug分支的处理方式一样，正常的话，我们在创建一个feature分支，在这个分支上面完成需要的工作，git commit -m "add feature xxx". 然后再把这个分支合并到dev上，最后删除feature分支。

如果我们还没有把feather分支merge到dev分支上，此时我们放弃原先修改，假设需要立即删除feature分支，这时我们会看到下面的情况： $ git branch -d feature-vulcan error: The branch 'feature-vulcan' is not fully merged. If you are sure you want to delete it, run 'git branch -D feature-vulcan'.

提示我们如果一定要删除这个还没有被合并的分支的话，我们可以使用： $git branch -D feature-xxx 来强行删除

多人协作

1. 推送分支

- `git remote`可以让我们查看远程库的状态
- 如果要看远程库更为详细的信息，可以使用`git remote -v`
- 推送分支，就是把该分支上的所有本地提交推送到远程库。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:

  ```
  $ git push origin master
  ```

  如果要推送到其他分支，比如`dev`则相应的改成

  ```
  $ git push origin dev
  ```

- 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！

2.抓取分支

- 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。

```
$ git branch
* master
```

- 如果想要在dev分支上开发：
- ```
  $git checkout -b dev origin/dev
  ```
- 现在，队友视角就可以在本地的dev库进行修改，并且时不时的推送到远程库

```
$ git add xxx.xx
$ git commit -m "XXX'
$ git push origin dev
```

3.解决远程冲突

- 当其他队友已经对文档进行过修改之后，我们再次推送自己版本时，会失败。此时，git提示我们要用`git pull`把最新的提交从`origin/dev`上抓下来，然后在本地合并，再推送。
- `git pull`也可能会失败，提示

> 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
> If you wish to set tracking information for this branch you can do so with:
> git branch --set-upstream-to=origin/ dev

- 根据提示，使用`git branch --set-upstream-to=origin/dev dev`来设定链接。
- 然后`pull`会自动合并，如果合并失败，则转到之前的合并冲突。
- 等到手动合并完了修改，就可以正常提交然后再推送到远程库了。

4.多人协作模式

- 最开始可以回顾多人版本库分支策略

1. 首先，可以试图用`git push origin <branch-name>`推送自己的修改；
```

2. 如果推送失败，则因为远程分支比你的本地更新，需要先用`git pull`试图合并；
3. 如果合并有冲突，则解决冲突，并在本地提交；
4. 没有冲突或者解决掉冲突后，再用`git push origin <branch-name>`推送就能成功！

- 如果`git pull`提示`no tracking information`，则说明本地分支和远程分支的链接关系没有创建，用命令`git branch --set-upstream-to <branch-name> origin/<branch-name>`。

这就是多人协作的工作模式，一旦熟悉了，就非常简单。

5. 小结

- 查看远程库信息，使用`git remote -v`；
- 本地新建的分支如果不推送到远程，对其他人就是不可见的；
- 从本地推送分支，使用`git push origin branch-name`，如果推送失败，先用`git pull`抓取远程的新提交；
- 在本地创建和远程分支对应的分支，使用`git checkout -b branch-name origin/branch-name`，本地和远程分支的名称最好一致；
- 建立本地分支和远程分支的关联，使用`git branch --set-upstream branch-name origin/branch-name`；
- 从远程抓取分支，使用`git pull`，如果有冲突，要先处理冲突。

Rebase

- `git rebase`操作的特点：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。
- 最后，通过push操作把本地分支推送到远程：远程分支的提交历史也是一条直线。

小结

- rebase操作可以把本地未push的分叉提交历史整理成直线；
- rebase的目的是使得我们在查看历史提交的变化时更容易，因为分叉的提交需要三方对比。

标签管理

发布版本时，我们通常会在版本库中打一个标签`tag`，这样将来无论什么时候，都可以直接回取到标签时刻的版本。

- 所以说，标签是版本库的一个快照
- Git的标签虽然是版本库的快照，但其实它就是指向某个commit的指针
- 和分支的区别是：分支可以移动，标签不能移动
- 创建和删除标签都是瞬间完成的。

创建标签

- 命令`git tag <tagname>`用于新建一个标签，默认为HEAD，也可以指定一个commit id；

```
$ git tag -a v0.1 -m "version 0.1 released" 1094adb
```

- 命令`git tag -a <tagname> -m "blablabla..."`可以指定标签信息；
- 命令`git tag`可以查看所有标签。
    - 注意，标签不是按时间顺序列出，而是按字母排序的。
    - 注意，标签总是和某个commit挂钩。如果这个commit既出现在master分支，又出现在dev分支，那么在这两个分支上都可以看到这个标签。

管理标签

- 命令`git push origin <tagname>`可以推送一个本地标签；
- 命令`git push origin --tags`可以推送全部未推送过的本地标签；
- 命令`git tag -d <tagname>`可以删除一个本地标签；
- 命令`git push origin :refs/tags/<tagname>`可以删除一个远程标签。
    - 如果需要删除已经提交的远程标签，则需要先删除本地标签，然后再进行此步骤。

自定义Git

.gitignore

- 忽略某些文件时，需要编写.gitignore；
    - 使用Windows, 在文本编辑器里"保存"或者"另存为"就可以把文件保存为.gitignore
- 有些时候，你想添加一个文件到Git，但发现添加不了，原因是这个文件被.gitignore忽略了
    - 可以使用`git add -f <nameigonred>`来强制添加被忽略的文件；
    - 可以用`git check-ignore`命令检查为什么此文件被忽略

```
$ git check-ignore -v App.class
.gitignore:3:*.class    App.class
```

    - git告诉我们，`.gitignore`的第3行规则忽略了该文件，于是我们就可以知道应该修订哪个规则。
- `.gitignore`文件本身要放到版本库里，并且可以对.gitignore做版本管理！

配置别名

如果敲git st就表示git status那就简单多了，当然这种偷懒的办法我们是极力赞成的。

我们只需要敲一行命令，告诉Git，以后st就表示status：

```
$ git config --global alias.st status
```

another example:

```
$ git config --global alias.last 'log -1'
```

其实是简单字符串的替换协议

查看配置文件

- 对于本地的文件，在当前目录下`cat .git/config`即可查看到。
- 对于用户的全局配置，则在用户主目录的隐藏文件`.gitconfig`中保存。

出现vi界面

It's not a Git error message, it's the editor git uses your default editor.

To solve this:

- press "i"
- write your merge message
- press "esc"
- write ":wq"
- then press enter

> If it helps anyone, the way you remember this is that "i" is for "insert", "esc" is the exit the insertion, and ":wq" is just "write" and "quit".