从廖老师网站上总结的Git笔记，对常见操作进行了总结。

文章目录

一 版本库、文件操作

1. 创建版本库

GitBash中 cd 到需要作为版本库的目录，执行`git init`,创建成功后会生成`.git`文件夹

2. 添加文件到版本库

首先,文件必须放在在版本库对应的目录下,
添加文件到版本库步骤：

1. `git add` 文件.文件类型，将文件放进暂存区(暂存区，工作区概念详见第4小节)
2. `git commit -m “本次提交的说明”`，添加到版本库
   备注:
   `git status` 随时掌握工作状态，提示信息分析：

`Changes to be committed`: 已进入暂存区，但还未提交到版本库，需要commit
`Changes not staged for commit`:更改了，但是还未进入暂存区
`Untracked files`:表示该文件还从来没有被添加进版本库

---

3. 版本回退

1. `git log` 显示从最近到最远的提交日志,`git log --pretty=oneline` 使每个日志单独成行。
2. git中，用`HEAD`表示当前版本
3. 回退到上一个版本：`git reset --hard HEAD^`,一个“^”表示回退1个版本，“~100”回退100个版本。
4. 返回新版本：`git reset --hard 新版本的Id`
5. `git reflog`查看命令历史，以便确定要回到未来的哪个版本

---

4. 工作区和暂存区

工作区：在电脑里能看到的目录； **版本库**：在工作区有一个隐藏目录.git，是Git的版本库。 Git的版本库中存了很多东西，其中最重要的就是称为stage（或者称为index）的暂存区，还有Git自动创建的master，以及指向master的指针HEAD。
在这里插入图片描述

1. `git diff HEAD -- readme.txt`命令可以查看工作区和版本库里面最新版本的区别
2. Git是跟踪修改的,而不是跟踪文件，每次修改，如果不用git add到暂存区，那就不会加入到commit中。

撤消修改

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`。
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。
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。

删除文件

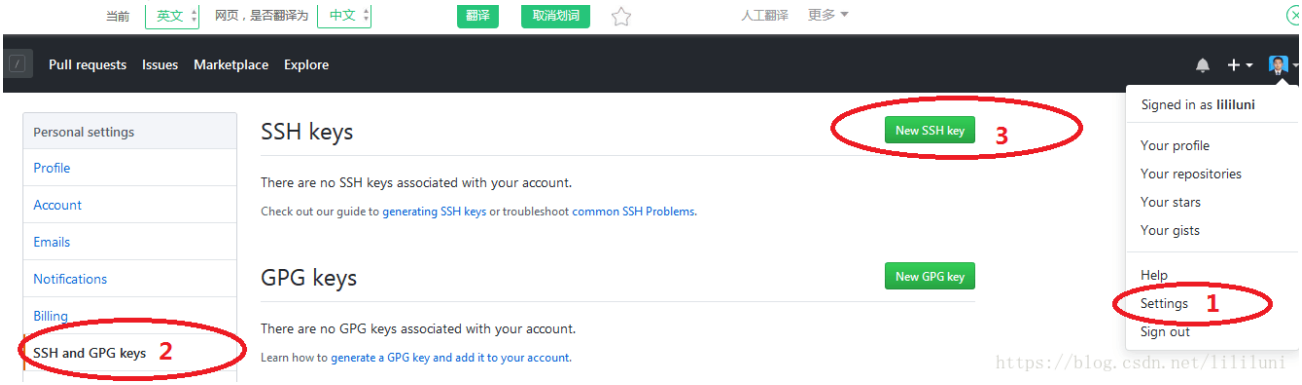
1. 提交到版本库的文件，可以先 `rm file`删除文件
2. 确实要从版本库中删除该文件，那就用命令`git rm file`删掉，并且`git commit`
3. 另一种情况是删错了，因为版本库里还有呢，所以可以很轻松地把误删的文件恢复到最新版本`git checkout -- file`
   备注：`git checkout -- file`就是将版本库的文件替代工作区的文件

二 远程仓库关联、克隆

1: 在GitHub上添加公钥

1. 创建GitHub账号

2. 创建SSH Key，GitBash执行
   `ssh-keygen -t rsa -C "youremail@example.com"`，在用户主目录（在Bash先后中执行`cd ~/.ssh`、`pwd`即可找到主目录路径）下检查是否有`.ssh`目录，里面有`id_rsa`(私钥)和`id_rsa.pub`（公钥）两个文件，这两个就是SSH Key的秘钥对。
3. 打开GitHub，在个人主页找到setting，找到`SSH and GPG keys`，然后，点“New SSH Key”，填上任意Title，在Key文本框里粘贴id_rsa.pub文件的内容。



2： 将本地仓库与远程仓库关联

1. 本地仓库为git，在GitHub上同样新建一个仓库为`git`，此时远程git仓库为空.
2. 创建远程仓库后，在GitBash中，在本地的`git`m目录里，执行命令`git remote add origin git@github.com:用户名/仓库名.git`实现关联
3. 执行`git push -u origin master`本地库的内容推送到远程，用git push命令，实际上是把当前分支master推送到远程（执行完后在GitHub上就可看到内容）。以后每次本地提交后，使用命令`git push origin master`推送最新修改

3： 从远程仓库克隆

4. 首先确保远程仓库有需要克隆的仓库，然后cd到本地仓库目录的上一级目录，执行：`git clone git@github.com:用户名/仓库名.git`,就会发现克隆到本地了。
5. 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

三 分支管理

1： 创建与合并分支

1. 创建分支dev： `git branch dev`；切换至div分支：`git checkout dev`,一句话：`git checkout -b dev`
2. 用`git branch`命令查看当前分支,当前分支前面会标一个`*`号
3. 合并分支 ，先`git checkout master`切换到主分支，然后执行`git merge dev`进行合并。然后可以再执行`git branch -d dev`删除分支
4. 查看分支：git branch
   创建分支：git branch name
   切换分支：git checkout name
   创建+切换分支：git checkout -b name

合并某分支到当前分支：git merge name
删除分支：git branch -d name

2： 解决分支冲突

当Git无法自动合并分支时，就必须首先解决冲突。解决冲突后，再提交，合并完成。解决冲突就是把Git合并失败的文件手动编辑为我们希望的内容，再提交。用`git log --graph`命令可以看到分支合并图。
备注： 进入log后直接按`Q`就可以退出。

3： 分支管理策略

合并分支时，默认采取的为`Fast farword`模式，这种合并看不到合并历史，但是这种模式下，删除分支会丢掉分支信息。
1.禁用`Fast farword`模式时，合并分支会产生一个commit， `git merge --no-ff -m "merge with no-ff" dev` 其中，`--no-ff`参数，表示禁用Fast forward。
采用`git log --graph --pretty=oneline --abbrev-commit`查看分支历史。

备注：不用–no-ff，实际上只是将master的指针update成dev分支而已，用的还是dev的commit ID，而使用之后，则是重新commit了一哈，有了新的commit ID
在这里插入图片描述

4： Bug分支

我个人觉得场景是这样的。设A为游戏软件

1. master 上面发布的是A的1.0版本
2. dev 上开发的是A的2.0版本
3. 这时，用户反映 1.0版本存在漏洞，有人利用这个漏洞开外挂
4. 需要从dev切换到master去填这个漏洞，正常必须先提交dev目前的工作，才能切换。
5. 而dev的工作还未完成，不想提交，所以先把dev的工作stash一下`git stash`。然后切换到master
6. 如果是在master修复bug，就在master建立分支issue101并切换. `git checkout -b issue101`
7. 在issue101上修复漏洞，并`git commit`。
8. 修复后，在master上合并issue101 ,`git merge --no-ff -m "merged bug fix 101" issue-101`
9. 切回dev，`git stash list`命令看看stash内容，然后`git stash pop`，恢复的同时把stash内容也删了继续工作。

5：强制分支删除

发一个新feature，最好新建一个分支；如果要丢弃一个没有被合并过的分支，可以通过`git branch -D branchname`强行删除。

6：多人协作

多人协作的工作模式通常是这样

1. 查看远程库信息，使用`git remote -v`；

2. 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；
3. 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；
4. 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；
5. 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`；
6. 从远程抓取分支，使用`git pull`，如果有冲突，要先处理冲突。

四 标签管理

1： 前言

Git的标签就是是版本库的快照，它其实就是指向某个commit的指针（跟分支很像，但是分支可以移动，标签不能移动），所以，创建和删除标签都是瞬间完成的。

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。

2：创建标签

1. 切换到需要打标签的分支 ，然后 `git tag tagname`。
   备注： 可以用`git tag`查看所有标签,
   可以用`git show tagname`查看标签信息
2. 标签默认是打在最新的commit上，如果需要打在之前的commit上，可以使用`git log --pretty=oneline --abbrev-commit`查看commit id,然后用`git tag tagname commitId`打上就可以了。
3. 创建带有说明的标签，用-a指定标签名，-m指定说明文字,`git tag -a tagname -m "tag说明" commitId`

3：操作标签

1. 命令`git push origin tagname`可以推送一个本地标签到远程；
2. 命令`git push origin --tags`可以推送全部未推送过的本地标签；
3. 命令`git tag -d tagname`可以删除一个本地标签；
4. 如果标签已经推送到远程，但又需要删除的话，可以先用命令`git push origin :refs/tags/tagname`删除一个远程标签，然后再用`git tag -d tagname`删除本地标签。

```
        </div>
                    <link href="https://csdnimg.cn/release/phoenix/mdeditor/markdown_views-778f64ae39
```