git使用记录

笔记本:	笔记		
创建时间:	2019/12/5 15:24	**更新时间:**	2020/11/24 9:59
作者:	ming71		
URL:	https://www.liaoxuefeng.com/wiki/896043488029600/896954074659008		

基础入门

基础教程:
https://www.liaoxuefeng.com/wiki/896043488029600/896954074659008
命令:

```
git status    // 查看暂存区状态
git diff        // 查看暂存区和当前版本差别
```

1. 创建版本库和简单版本提交

1.1 从本地创建

新建文件夹，放入代码，执行下面代码，可以创建一个.git隐藏文件夹，内部实现了git的版本控制文件

```
git init
```

1.2 从远程创建

克隆repo的地址到指定路径。注意不能直接下载必须用clone，下载的文件中不含.git文件夹

```
git clone https://github.com/ming71/CV_PaperDaily.git
```

1.3 添加文件到暂存区

```
git add * // 添加当前目录所有文件
git add name // 添加文指定文件或文件夹下所有文件
```

1.4 提交到版本库

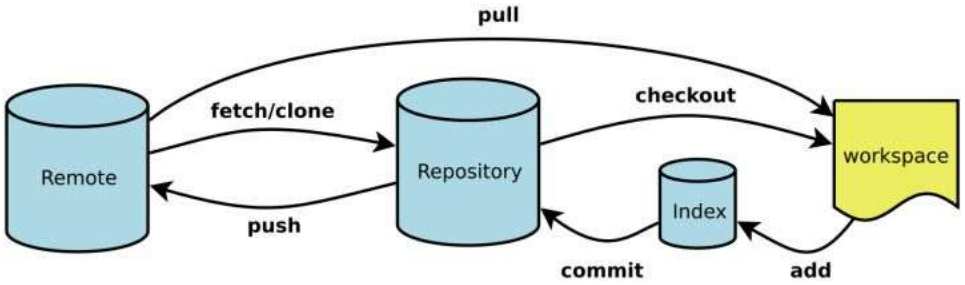
注意:

（1）提交的注释一定要认真写，不然版本回退的时候都是‘update’无从查起。

（2）所有的更改必须先add到暂存区才能在commit后在版本库的master分支进行修改。

```
git commit -m 'the first committing.'
```

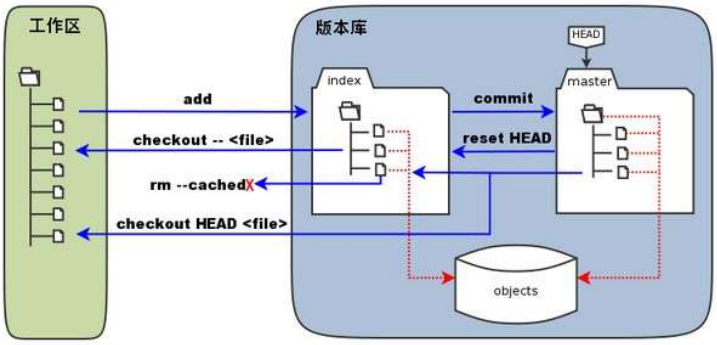
2. Git工作结构



上图将结构和命令展示地十分清楚。可以看到有四个部分：

- Remote：远程仓库，发布到github的远程仓库。
- Repository：本地仓库（版本库），位于.git下，最终修改提交都会存储到这里，和远程发布的仓库是相对的，用于本地的版本控制。
- workspace：工作区，可以认为是当前的实际代码部分，也就是电脑的工作文件目录。
- Index：暂存区，存储在.git/Index文件，用于记录提交暂存的文件修改记录。

工作大致流产是：本地进行源码的修改操作，add添加到暂存区，然后确认无误后commit提交到本地仓库；如果要发布到远程仓库可以push到远程即可。具体而言：



git的版本控制就是一次次commit的时间节点连成的时间线，其中master分支是当前版本，有一个HEAD指针指向它，每当有新的提交更新master分支时，修改HEAD指针位置移动指向新节点就是新的master。 当对工作区修改（或新增）的文件执行 "git add" 命令时，暂存区的目录树被更新，使用"git commit" 命令时，暂存区的目录会被清空，内容被推到版本库，master 分支会做相应的更新。

3. 版本管理

- 查看历史版本

通过下面命令可以查看历史提交版本，commit后面有一长串码是版本号commit id，通过该ID实现回溯。

```
git log      // 查看历史commit提交的版本，根据每次的注释可以判断修改内容
git log --pretty=oneline   // 不显示提交时间，更加简洁
```

- 倒退版本

注意版本的倒退，不只是退回本地版本库，连带工作区（文件）也撤回更改了；
不仅可以退回，也可以向后，只要是有commit的ID，任何版本都可以抵达；
所以需要获取commit的ID号才行，也有命令可获取。

```
git reset --hard HEAD^      // 退回上个修改的版本
git reset --hard 661c90b113f0bfd688425a44dbf305122eeea149  // 根据版本id，到达指定版本
git reflog  // 记录每个更改版本的id号和提交注释
```

git log和git reflog区别：
　　git log只能查看历史版本，但是如果版本撤销回溯到以前，那么不能查看撤销的这个操作id和历史；而git reflog会真正记录所有的操作版本，包括撤销。

- 文件删除
- 文件目录上误删

很简单，直接回溯到上一个提交的版本就行，只要版本库还有就可以。

- 文件目录删除

但是版本库还有文件，导致二者不同步，这时只能在版本库中也删除文件。

```
git rm test.txt   // 在版本库中删除目标文件
```

4. 远程仓库

- 本地仓库和github仓库关联

参见：https://www.liaoxuefeng.com/wiki/896043488029600/896954117292416

- 添加远程仓库

（1）Github网页新建repo并获取地址。
（2）添加远程仓库
远程仓库的名字默认是origin，一般都用这个，就可以成功地将本地库关联到一个远程库。

```
git remote add origin  git@github.com:ming71/test.git
```

- 本地库内容推送到远程库

下面命令是将本地库的master分支推送到远程库中，第一次推送时远程库加上-u，不仅会推送而且将本地master与远程库master分支联系起来，以后就可以不用带参数地push了。

```
git push -u origin master    // 第一次推送
git push origin master    // 以后的推送
```

如果push失败，且提醒远程仓库已经存在，可以移除重新建立：

```
git remote rm origin
```

- 常用操作

一般而言我就是管理一下自己的代码版本，不涉及多人合作，顶多新建分支管理不同的发行版本，不用那么多的功能，大多数任务也是本地工作区完成并且能保证是最新的，因此可以采用强制上传。
先新建repo（如果是现有仓库或者远程仓库的更新更为频繁，则git clone下来，注意用SSH），然后如下操作即可：

```
git remote add origin  git@github.com:ming71/test.git
// 有gitignore文件先git add加入
// 工作区提交到本地版本库
git push -u origin master   // 第一次推送
git push origin master  -f       // 以后推送

------------ 如果强制push失败，可以将网址换成https协议--------
git remote rm origin
```

5. 分支管理

一般而言，master用于发布正式版本，而每个人要在上面工作是不能修改发行版本的，可以创建自己的分支branch在平行分支上工作互不影响。严格来说，Git的master指向的是最新提交，而HEAD指向的是当前分支。所以切换分支只需要修改HEAD指针就可以快速实现。比如新建一个分支Beta，会创建一个Beta指针指向最新提交，每次修改提交该分支项目时Beta就会向前移动指向最新提交（和主分支的master指向最新提交一样）；如果从master分支切换到Beta分支，实际就是HEAD指针从指向master改为指向Beta线，所以切换分支十分快捷。Git鼓励多分支工作

在新分支上进行工作提交，只会改变新分支的进程，master分支不会改变；如果在子分支上完成新任务的发行需要正式发布，直接将master指向当前分支的最新提交即可可以将该分支合并到master上。合并完分支后，甚至可以删除dev分支。删除dev分支就是把dev指针给删掉，删掉后，就剩下一条master分支。

- 分支建立切换和删除

注意，在新分支下，文件的所有操作会记录到当前的新分支，原来master不会记录。

```
git branch  // 查看当前分支(带*是当前分支)
git branch  -a // 查看所有分支(带-a是查看包括远程分支)
git branch  dev  // 新建叫dev的分支
git branch -d dev  //  删除dev分支
git checkout dev   //  切换到dev分支
git checkout -b dev //  创建并切换到dev分支
```

- 远程分支

```
git remote -v  // 获取远程分支信息，得到抓取和推送的地址，没有权限则看不到
git push origin master  // 推送主分支
git push origin dev  //  推送dev分支
```

如果是多人合作维护，可能出现远程仓库比本地更新的情况，这时需要先把最新的推送拉下来，逐个比较解决冲突然后推送。

```
git pull  // 抓取远程仓库最新提交
```

6. 自定义部分

- 忽略文件

如密码文件、随时更新的临时文件等，不便或者没必要加到暂存区，可以通过一个.gitignore文件忽略这些文件。就用的mmdetection的文件：

注意：由于.gitignore文件隐藏，直接add不到暂存区；首次push必须先添加该文件，否则ignore无法生效；万一未生效就上传到仓库了，可以移除本地版本库，不过之前的历史版本就没有了，需要慎重。

```
git add -f .gitignore    //  一定要先add这个文件
git add *  // 再添加其他文件
```

万一忘了顺序添加失败，后面想补：

```
git rm -r --cached .  //  慎重！会删掉版本库
git add .
git commit -m 'Don't  track this anymore!'
```

- 创建标签

发布重要版本的提交时，打上标签，方便后续查找

```
git tag <name>   // 如：git tag v1.0  默认会打在最后一个commit版本上
git tag   // 查看所有标签
git tag v0.9 661c90b113f0bfd688425a44dbf305122eeea149  // 对特定的提交版本打标签

git tag -a <Tag名字> -m <注释文字> <SHA-1 Code>  # <SHA-1 Code>即git log的代码
例如：
git tag -a v1.0 -m "V1.0 released" 123456  // 创建带有说明的标签:-a指定标签名，-m指定说明文字，123456为commit id
git tag -d <tagname> // 删除
```

7. 其他命令

- **比较差别**

```
git diff  // 工作区和暂存区
git diff HEAD  // 工作区和本地仓库
git diff --cache  // 暂存区和本地仓库
git commit <commit id1>  <commit id2>
```

参数：
-- file1 file2 这比较这两个文件差异
--stat 仅仅显示哪些文件被改动（删了多少行、新增多少行）

- 连接失败，端口22被拒绝

(配置文件)
https://blog.csdn.net/MBuger/article/details/70226712

注意：上述方案是让SSH走443端口，如果443端口被拒绝，删掉上面的config文件即可。

- **合并提交**

rebase操作，没有成功过....尴尬....
目前简单的处理方式就是版本倒退reset就能消除历史了

- **其他**

```
git ls-files  // 查看当前版本库文件
git reset --soft HEAD^  //丢弃commit修改（但是保留本地的代码修改，很重要）
git reset HEAD  //进一步丢弃add修改
```

- **PR操作**

靠谱的教程：知乎入口

- **公钥无权限访问**

问题：git clone或者git push失败
方法：参考解决方案

1. eval `ssh-agent`
2. ssh-add ~/.ssh/id_rsa # 这一步的最后这个文件根据自己的名字而定，可能是id_rsa
3. ssh-add -l
4. 检查连接是否成功： ssh -v git@github.com

快捷复制：

```
eval `ssh-agent`
ssh-add ~/.ssh/id_rsa
```

- **.ignore禁止追踪失败**

可能是已经追踪上了,不管怎么撤回倒退都没用,需要删掉被追踪的文件(只能是文件).但是往往误追踪的是文件夹,也只能手动在目录下删除:

```
git rm --cached  exapmle.py    # 删除文件
git rm --cached folder/*       #删除文件夹下所有内容
```

- **commit了但是未push想撤回**

两种模式hard和非hard注意谨慎选择，一般非hard比较好（hard改代码的）：入口

- **强制将远程pull并覆盖本地**

```
git fetch --all && git reset --hard origin/master
```

- **新仓库的操作**

```
git init
git remote add origin  git@github.com:ming71/test.git
// 有gitignore文件先git add加入
// 工作区提交到本地版本库
git push -u origin master   // 第一次推送
git push origin master  -f       // 以后推送


------------ 如果强制push失败，可以将网址换成https协议--------
git remote rm origin
```

- 重大更新！！

以后新建的repo全都是main分支了，以前操作的所有master替换为main。这仅限于你新建repo并且git c lone下来的，如果直接init默认分支还是master。总之看本地分支名叫什么，然后git push origin master 或git push origin main就行了