Git学习笔记

2020年6月21日 21:05

- Git配置：

  安装好git后打开Git Bash输入：

  ```
  git config --global user.name "Your Name"
  ```

  ```
  git config --global user.email "email@example.com"
  ```

  让Git显示颜色（可选）：`git config --global color.ui true`

  给命令配置别名（可选）：eg：`git config --global alias.st status`

  每个仓库的Git配置文件都放在`.git/config`文件中，当前用户的Git配置文件放在用户主目录下的一个隐藏文件`.gitconfig`中，一个有意思的配置：`git config --global alias.lg "log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"`

  初始化一个Git仓库，使用`git init`命令。

- Git基本操作：

  添加文件：

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

  查看状态：

  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。

  版本回退：

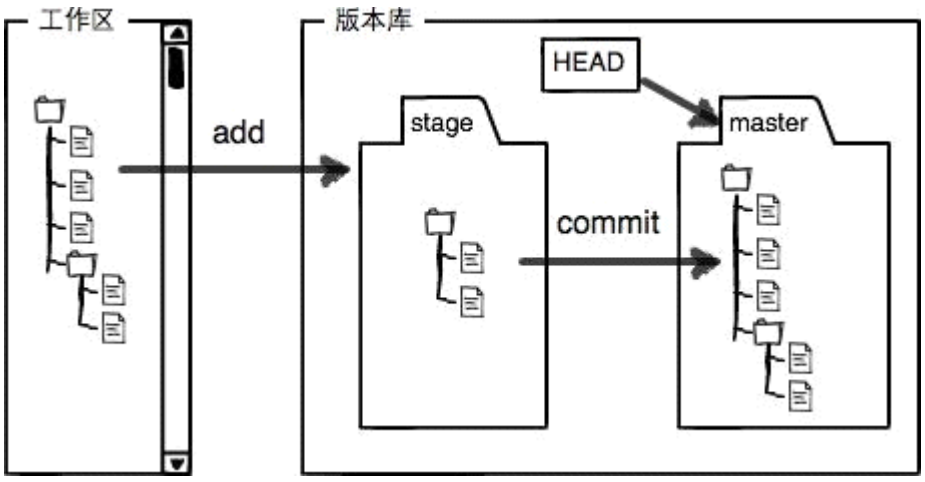
  `HEAD`指向的版本就是当前版本，因此，Git允许我们在版本的历史之间穿梭，使用命令`git reset --hard commit_id`。注：在Git中，用`HEAD`表示当前版本，也就是最新提交的`commit_id`，上一个版本就是`HEAD^`，上上一个版本就是`HEAD^^`，当然往上100个版本写100个^比较容易数不过来，所以写成`HEAD~100`。所以回退到之前的版本可以用`git reset --hard HEAD^`（回退到上一个版本）命令。

  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。

  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。

  工作区、版本库、暂存区关系浅析：

  `master`，以及指向`master`的一个指针叫`HEAD`。关系如下图：



有操作过程如：第一次修改 -> `git add` -> 第二次修改 -> `git commit`。当用`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。提交后，用git diff HEAD -- <filename>命令可以查看工作区和版本库里面最新版本（即git commit提交的版本）的区别。

撤销修改：

1. 没有`git add`时，用`git restore <file>`…

2. 已经`git add`时，先`git restore --staged <file>...`回退到1.，再按1.操作

3. 已经`git commit`时，用`git reset`回退版本

删除文件：

当要删除文件的时候，可以采用命令：`rm <filename>`。这个时候有两种情况：第一种情况：的确要把文件删掉，那么可以执行`git rm <filename>`和git commit -m "remove filename"，文件被删除，且删除记录上传本地库。第二种情况:误删文件，想恢复，这时候还没有commit -m "remove filename"，执行git restore <filename>将文件恢复。如果执行完git commit -m "remove filename"后就不能用restore恢复了，得用git reset --hard commit_id。注：git rm file并且git commit**并不是删除了版本库里的某个版本号，而是对工作目录下的删除操作进行了一个记录，会在仓库里生成一个新的版本号，在该版本下没有该文件。**但是可以用git reset --hard commit_id进行版本回退，回退到有这个文件的版本号。**这一部分内容最能体现git管理的是修改的思想。**

- **远程连接入门：**

添加远程库：

注册GitHub账号。由于本地Git仓库和GitHub仓库之间的传输是通过SSH加密的，所以，需要一点设置：

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：

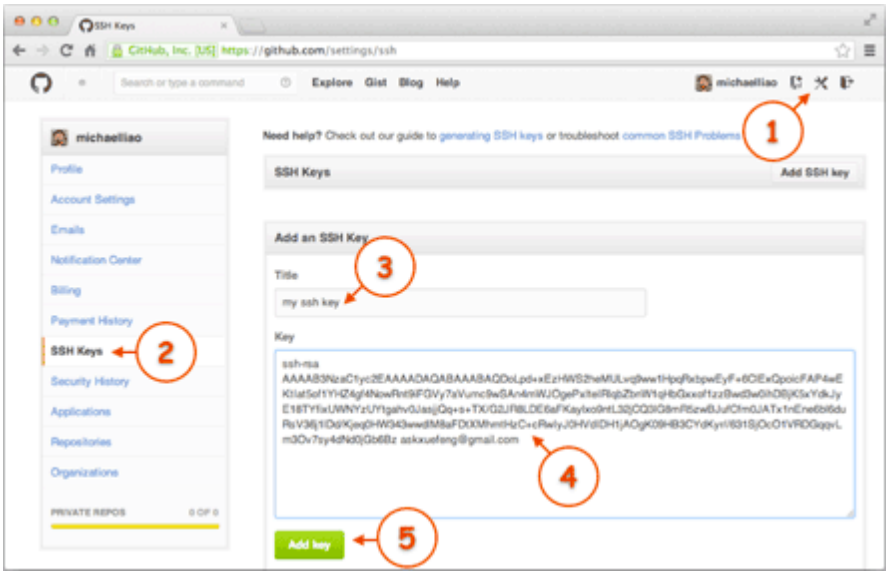
```
$ ssh-keygen -t rsa -C "youremail@example.com"
```

把邮件地址换成自己的邮件地址，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。

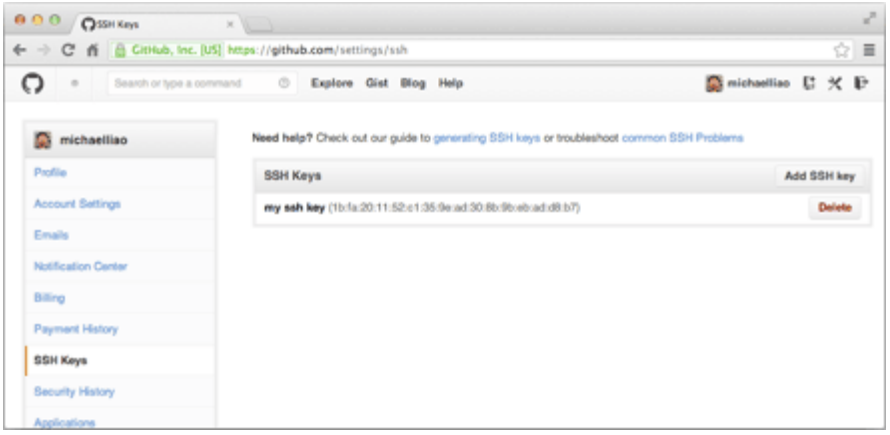
如果一切顺利的话，可以在用户主目录里找到.ssh目录，里面有id_rsa和id_rsa.pub两个文件，这两个就是SSH Key的秘钥对，id_rsa是私钥，不能泄露出去，id_rsa.pub是公钥，可以放心地告诉任何人。

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：



点“Add Key”，应该看到已经添加的Key：



在github创建新的git仓库：点击就送。其中命令git remote add origin git@github.com:michaelliao/learngit.git现在为git remote add origin https://github.com/michaelliao/learngit.git

要关联一个远程库，使用命令git remote add origin https://github.com/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；

从远程库克隆：

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。Git支持多种协议，包括https (eg:https://github.com/michaelliao/gitskills.git)，但ssh (eg:git@github.com:michaelliao/gitskills.git)协议速度最快。

- **分支管理：**

  基本操作：

  查看分支：git branch

  创建分支：git branch <name>

  切换分支：git checkout <name>**或者**git switch <name>

  创建+切换分支：git checkout -b <name>**或者**git switch -c <name>

  合并某分支到当前分支：git merge <name>

  删除分支：git branch -d <name>

  合并冲突的解决方案：

  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（**git log --graph --pretty=oneline --abbrev-commit**）可以看到分支合并图。**

  分支管理策略：

  Git分支十分强大，在团队开发中应该充分应用。合并分支时，加上--no-ff**参数（eg：**git merge --no-ff -m "merge with no-ff" dev**）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，而fast forward合并就看不出来曾经做过合并。**

  Bug分支：

  修复bug时，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**（在此之前可以先用**git stash list**查看**stash**，也可以用如**git stash apply stash@{0}**这样的命令指定恢复哪个**stash**，再用**git stash drop stash@{0}**删除**stash**内容），回到工作现场；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（此时**dev**的工作区需要是**clean**的），把bug提交的修改“复制”到当前分支，避免重复劳动。**

  如果要丢弃一个没有被合并过的分支，可以通过git branch -D <name>**强行删除。**

- **远程连接进阶：**

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；

在本地创建和远程分支对应的分支，使用git switch -c branch-name origin/branch-name，本地和远程分支的名称最好一致；

多人协作的工作模式通常是这样：

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

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；

如果合并有冲突，则解决冲突，并在本地提交；

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>。

Rebase操作：

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：

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

命令git tag -a <tagname> -m "blablabla..." <commit_id>可以指定标签信息；

命令git tag可以查看所有标签。

命令git show <tagname>可以看指定tag的信息

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。

建立多个远程库：

将仓库地址告知本地Git：git remote add [-t <branch>] [-m <master>] [-f] [--[no-]tags] [--mirror=<fetch|push>] <name> <url>
重命名远程库：git remote rename <old> <new>
删除远程库：git remote rm <name>

- 其它：

忽略特殊文件：

配置.gitignore文件以忽略某些文件

文件被忽略的解决方案：git add -f <filename>

检查.gitignore文件：git check-ignore -v <filename>

注：.gitignore文件本身要放到版本库里，并且可以对.gitignore做版本管理！

搭建git服务器：前提得有服务器

Git图形界面工具SourceTree介绍：戳我

一张Git Cheat Sheet