Git笔记

前言

> 感谢兵兵学长，在我学习git过程中给予了我很多帮助，三克油
> 感谢廖老师的课程，非常的通俗易懂！强烈推荐！！！
> 感谢网友总结的Git学习笔记，本笔记改编于此，加上了自己的遇到的问题和解决方法！非常感谢！

廖老师Git课程传送门
网友Github传送门
Git官方说明文档

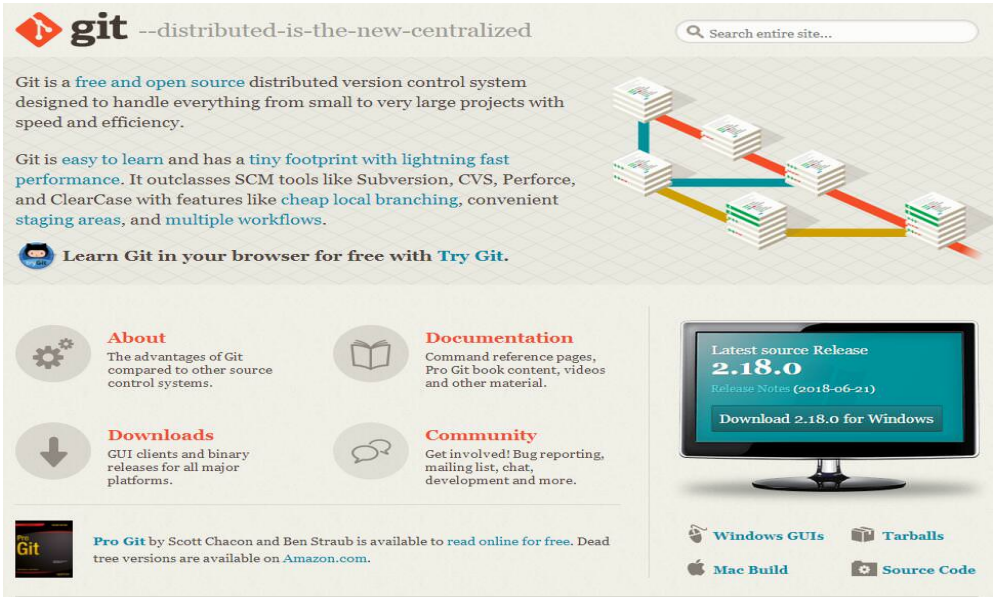
Git介绍

- Git是分布式版本控制系统
- 集中式VS分布式，SVN VS Git
  1. SVN和Git主要的区别在于历史版本维护的位置
  2. Git本地仓库包含代码库还有历史库，在本地的环境开发就可以记录历史而SVN的历史库存在于中央仓库，每次对比与提交代码都必须连接到中央仓库才能进行。
  3. 这样的好处在于：
     - 自己可以在脱机环境查看开发的版本历史。
     - 多人开发时如果充当中央仓库的Git仓库挂了，可以随时创建一个新的中央仓库然后同步就立刻恢复了中央库。

安装Git

Windows系统使用Git，可在Git官网上直接下载安装Git官网地址。

- 进入后，点击图中的电脑屏幕，即可开始下载。



- 具体安装过程可参考window系统Git安装

Git命令

Git配置

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

`git config`命令的`--global`参数，表明这台机器上的所有Git仓库都会使用这个配置，也可以对某个仓库指定不同的用户名和邮箱地址。

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

创建版本库

初始化一个Git仓库

```
$ madir learngit   //在目录上创建learngit文件夹
$ cd    learngit   //跳转进该文件夹
$ pwd               //查看显示当前目录
$ git init         //创建本地空仓库
此时可以去对应目录下查看，文件夹（仓库）已经创建好了。进入仓库后未发现.git项目，请输入Ls -ah，若还未
```

添加文件到Git仓库

包括两步：

```
$ git add <file> //注意添加文件后缀！
$ git commit -m "description"
```

`git add`可以反复多次使用，添加多个文件，`git commit`可以一次提交很多文件，`-m`后面输入的是本次提交的说明，理论上可以输入任意内容，但最好起一个与与修改内容相关的名称，方便开展工作。

查看工作区状态

```
$ git status
```

Window系统下Git创建及编辑文件

```
vi learngit.txt
```

退出编辑

```
先按ESC--然后按两次大写的Z。
Esc+ZZ
```

查看修改内容

```
$ git diff
```

```
$ git diff --cached
```

```
$ git diff HEAD -- <file>
```

```
$ git diff HEAD
```

- `git diff` 可以查看工作区(work dict)和分支(master)的区别，看不懂可以先跳过。总而言之：git diff只在指令git add 发生前有效。
- `git diff --cached` 可以查看暂存区(stage)和分支(master)的区别
- `git diff HEAD -- <file>` 可以查看工作区和版本库里面指定文件最新版本的区别

- `git diff HEAD` 若仓库只有一个文件。想查看工作区与版本库最新版本区别，可省略文件名。

查看提交日志

```
$ git log
```

简化日志输出信息

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

查看命令历史

```
$ git reflog
```

记录每一次命令，穿越到19世纪又想回到21世纪，但是commit_id已经找不到了，可以输入此指令。

版本回退

```
$ git reset --hard HEAD^
```

以上命令是返回上一个版本，在Git中，用`HEAD`表示当前版本，上一个版本就是`HEAD^`，上上一个版本是`HEAD^^`，往上100个版本写成`HEAD~100`。

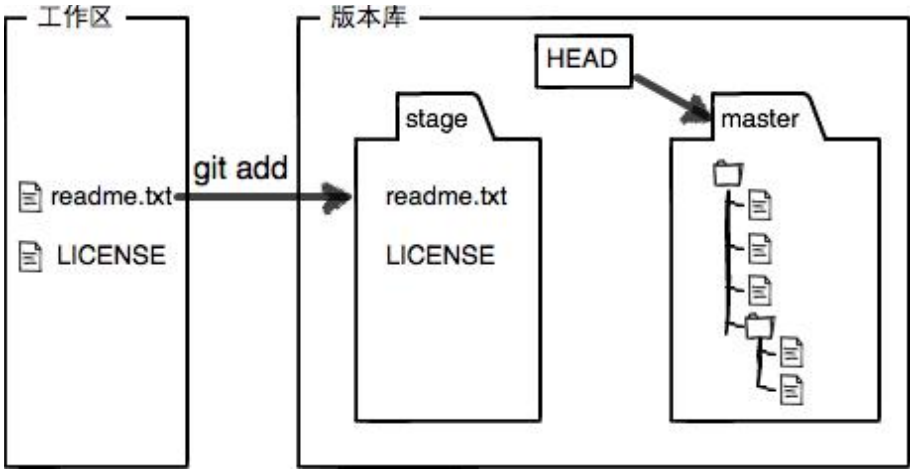
回退指定版本号

```
$ git reset --hard commit_id
```

commit_id是版本号，是一个用SHA1计算出的序列

工作区、暂存区和版本库

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`。



进一步解释一些命令：

- `git add`实际上是把文件添加到暂存区
- `git commit`实际上是把暂存区的所有内容提交到当前分支

撤销修改

丢弃工作区的修改

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

该命令是指将文件在工作区的修改全部撤销，这里有两种情况：

1. 一种是file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；
2. 一种是file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。

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

丢弃暂存区的修改

分两步：
第一步，把暂存区的修改撤销掉(unstage)，重新放回工作区：

```
$ git reset HEAD <file>
```

第二步，撤销工作区的修改

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

小结：

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>`，就回到了第一步，第二步按第一步操作。

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，进行版本回退，前提是没有推送到远程库。

删除文件

```
$ git rm <file>
```

`git rm <file>`相当于执行

```
$ rm <file>
$ git add <file>
```

进一步的解释

Q：比如执行了`rm text.txt`误删了怎么恢复?
A：没关系，只是删除了工作区的文件，执行`git checkout -- text.txt`把版本库的东西重新写回工作区就行了
Q：如果执行了`git rm text.txt`我们会发现工作区的text.txt也删除了，怎么恢复?
A：先撤销暂存区修改，重新放回工作区，然后再从版本库写回到工作区

```
$ git reset HEAD text.txt
$ git checkout -- text.txt
```

Q：如果真的想从版本库里面删除文件怎么做?
A：执行`git commit -m "delete text.txt"`，提交后最新的版本库将不包含这个文件
Q：如何查看目录所有文件（可用来监测文件状态）
A:Ls -ah

远程仓库

请先注册Github账号！！！

- Git仓库里可以对一个文件进行管理，再也不用担心文件备份和丢失问题，**But**你电脑挂了咋整~因此我们还需要远程舱来进行合作保管，Github就是提供Git仓库远程保管功能的，你可以随时把自己的提交推送到服务器端，也可以从服务器端拉取别人的提交。
  Github官网
  Github注册及使用

创建SSH Key

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

关联远程仓库

```
$ git remote add origin https://github.com/username/repositoryname.git
```

推送到远程仓库

```
$ git push -u origin master
```

`-u` 表示第一次推送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。

从远程克隆

```
$ git clone https://github.com/usern/repositoryname.git
```

分支

创建分支

```
$ git branch <branchname>
```

查看分支

```
$ git branch
```

`git branch`命令会列出所有分支，当前分支前面会标一个*号。

切换分支

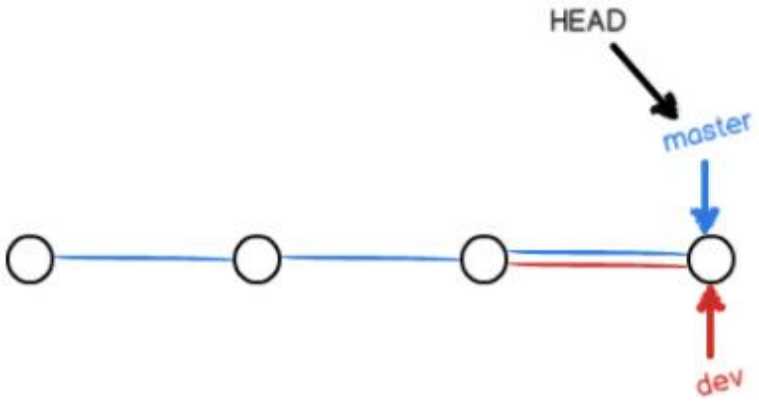
```
$ git checkout <branchname>
```

创建+切换分支

```
$ git checkout -b <branchname>
```

合并某分支到当前分支

```
$ git merge <branchname>
```



删除分支

```
$ git branch -d <branchname>
```

- $ git merge使用的是快速合并，删除分支后，不可恢复。
- $ git merge --nno-ff使用的是普通模式合并，删除分支后，可恢复。**（推荐使用本方式合并）**

查看分支合并图

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

解决冲突

当Git无法自动合并分支时，就必须首先解决冲突(建立了另一个分支dev，提交了修改，master分支也提交了一次修改，此时合并会产生冲突)。解决冲突后，再提交，合并完成。用`git log --graph`命令可以看到分支合并图。

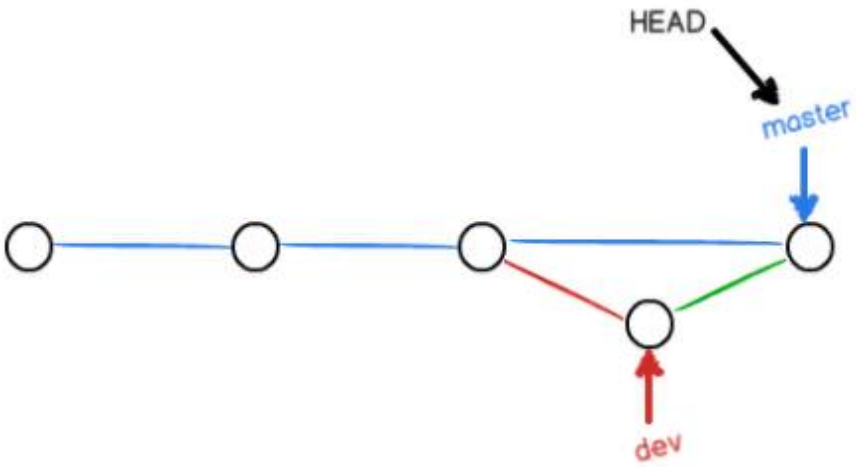
```
$ git merge readme.txt   //合并文件，显示冲突
$ git status            //查看具体问题
$ git cat readme.txt    //显示文件内容
$ vi readme.txt         //修改合并后的文件
$ git add readme.txt    //提交到暂存区
$ git commit -m "add and simple"  //提交至master
$ git log --graph --pretty=oneline --abbrev-commit  //查看合并分支
```

普通模式合并分支

```
$ git merge --no-ff -m "description" <branchname>
```

因为本次合并要创建一个新的commit，所以加上`-m`参数，把commit描述写进去。合并分支时，加上`--no-ff`参数就可以用普通模式合并，能看出来曾经做过合并，包含作者和时间戳等信息，而fast forward合并就看不出来曾经做过合并。

建议使用这个合并方式！！！不然会丢失之前建立的分支信息



保存工作现场

```
$ git stash
```

查看工作现场

```
$ git stash list
```

恢复工作现场

```
$ git stash pop
$ git stash pop = $git stash apply(恢复工作现场) + $ git stash drop(丢掉stash保存的内容)
```

丢弃一个没有合并过的分支

```
$ git branch -D <branchname>
```

查看远程库详细信息

```
$ git remote -v
```

在本地创建和远程分支对应的分支

```
$ 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

```
$ git branch --set_upstream branch-name origin/branch-name；
```

从本地推送分支

```
$ git push origin branch_name
```

可以推送主分支master和开发分支，取决于个人。

- 推送成功~恭喜你！！推送成功的前提：推送前本地仓库和远程仓库文件相同，不产生冲突。
- 推送失败。当前远程仓库文件领先于本地仓库文件。此时应该先使用git pull指令抓取origin/branch_name，然后在本地合并后提交。

从远程抓取分支

```
$ git pull
git pull也失败了，错误信息提示如下：
If you wish to set tracking information for this branch you can do so with:
         git branch --set-upstream-to=origin/<branch> dev
这是因为本地分支dev没有和远程分支dev建立关联
此时应使用 $ git branch --set_upstream branch-name origin/branch-name 指令
再次git pull
接下来步骤与解决冲突过程类似(假设文件名未learngit.txt)
$ git status
$ git cat learngit.txt
$ vi learngit.git
$ git add learngit.txt
$ git commit -m "fix lerangit conflict"
$ git push origin branch_name
```

Rebase

Git有一种神奇的操作，叫做rebase，为了好记，就把它记成**变基**。

假设活动只发生在master分支上，所有的操作都是你一个人完成的，查看分支合并图 $ git log --graph --pretty=oneline --abbrev-commit应该是一条竖直线，假设此时你领先远程仓库两个提交，你发出了 $ git push origin master 指令。惊奇的发现你哥们先于你提交，他也想修改了master，冲突！！！如何解决冲突咱就不累述了。先git pull一下。这时候请查看分支合并图，你会惊奇的发现你的分支图变清晰了。

```
$ git log --graph --pretty=oneline --abbrev-commit
*   e0ea545 (HEAD -> master) Merge branch 'master' of github.com:michaelliao/learngit
|\
| * f005ed4 (origin/master) set exit=1
* | 582d922 add author
* | 8875536 add comment
|/
* d1be385 init hello
...
```

输入`$ git rebase`指令，再次查看合并分支图

```
$ git log --graph --pretty=oneline --abbrev-commit
* 7e61ed4 (HEAD -> master) add author
* 3611cfe add comment
* f005ed4 (origin/master) set exit=1
* d1be385 init hello
```

- **总结**
  git rebase就是让分支图更清晰易懂。什么？有什么用？大概就在版本回退的时候你能更快的看懂流程图

标签

tag就是一个让人容易记住的有意义的名字，它跟某个commit绑在一起。

新建一个标签

新建一个标签，默认为HEAD

```
$ git tag <tagname>
```

指定commit id。

```
$ git tag <tarname> commit_id
```

查看所有标签

```
$ git tag
```

注意：标签不是按时间顺序列出，而是按照字母排序

指定标签信息

```
$ git tag -a <tagname> -m <description> <branchname> or commit_id
```

`git tag -a <tagname> -m "blablabla..."`可以指定标签信息。

查看标签信息

```
$ git show <tagname>
```

推送一个本地标签

```
$ git push origin <tagname>
```

推送全部未推送过的本地标签

```
$ git push origin --tags
```

删除一个本地标签

```
$ git tag -d <tagname>
```

删除一个远程标签

```
$ git push origin :refs/tags/<tagname>
```

克隆仓库

- 1 进入项目Github主页，点“Fork”在自己账号下克隆仓库。
- 2 $ git clone git@github.com:username/repositoryname.git

移除远程库

```
$ git remote rm origin
```

远程库默认名为origin，这是因为关联远程库时我们取名为origin，这个名称是可以更改的。

- 1 **未添加远程库 `origin_name`自定**

  ```
  $ git remote add origin_name git@github.com:username/repositoryname.git
  ```

- 2 **已关联远程库**

  ```
  先取消关联，再次关联的时候修改远程库名称
  ```