Git 相关学习：

1. Git 是一个版本管理系统，由linux之父使用C语言创建的区别于SVN集中式管理的另一种分布式版本管理工具。
2. 目前Git已经广泛应用在Mac OS 、Windows 、 Unix 、Linux 等平台上。
3. 在Mac 上创建一个空的文件夹

```
$ mkdir fileName    #创建一个文件（文件名：fileName）
$ cd fileName       #进入到fileName文件中
$ pwd               #显示fileName文件的具体路径  eg:/Users/huwenyang/fileName
```

4. 创建一个git 库，添加文件，提交文件

```
$ git init                  #生成一个repository(仓库)

$ git add fileNma.tex    #添加一个文件，若添加多个文件使用空格隔开
$ git add fileName/      #如果fileName是一个文件夹，使用/表示把文件夹所有的文件全加进去
$ git add *              #使用*表示全部文件

$ git commit -m “message” #提交这些add的文件（变化的文件），-m后面的message是描述这些变化
```

5. 链接远程仓库

```
如果当前没有连接上远程库：$ git remote add origin <server>
推送本地库 到 远程库： $ git push -u <branchName>
下拉远程 到 本地库：$ git pull
删除远程库的一个分支：$ git push -u origin --delete <branchName>

$ git clone <server>       #克隆远程库到本地
#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。
```

6. 查看当前git库中的变化

```
$ git status            #当前git库的状态，展示当前分支信息，以及都有哪些修改和未提交，以及哪里产生冲突信息
$ git diff              #当修改了某一个文件后，想展示都哪里做了修改，可以使用该命令
```

7. 查看当前git库的提交记录，以及根据commit id 回退版本，前进版本

```
$ git log          #查看当前git库的提交记录，当记录比较多时，可以通过添
```

加--pretty=oneline

$ git log —pretty=oneline　　#简单显示多行提交记录

$ git log --author=bob　　#打印作者bob的所有提交记录

$ git log --graph --oneline --decorate --all　#你想通过 ASCII 艺术的树形结构来展示所有的分支, 每个分支都标示了他的名字和标签

$ git log --name-status #看看哪些文件改变了

$ git log --help　　#还有很多其他指令

「 $ q　　#当git log之后，使用 q退出 」

根据上图当前最近的提交版本是——「测试一下diff」

回退版本的话有两种方法：

一个是直接根据commit id （提交ID）来切换，一般适用于提交记录特别庞大，且回退距离较大的；

第二种是使用 HEAD^^，来表示自从当前版本开始向前回退版本，根据^的个数来表示回退几个版本。如果回退距离较

大，可根据数字 HEAD~100 （自当前版本开始向前回推100个版本）。

$ git reset —hard HEAD^　　#回退到上个版本

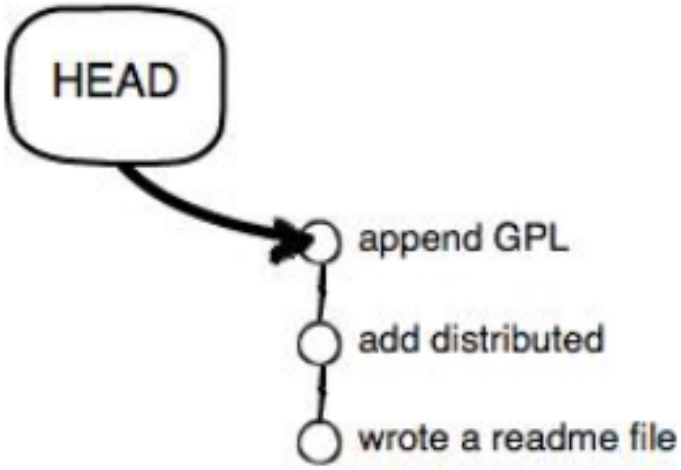
$ git reset —hard HEAD^^　　#回退到上上版本

$ git reset —hard HEAD~100　　#会退到前100版本

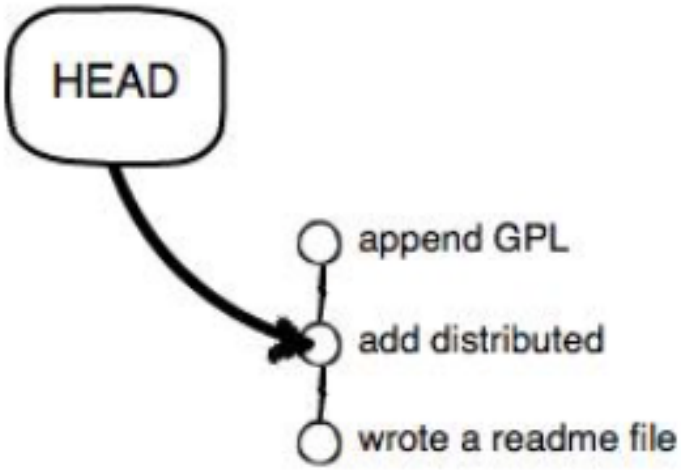
$ git reset —hard <commit id>　　#根据commit id 跳转到指定的版本，commit id是一个哈希值是以十六进制的格式展示「e2f7426bcf3c5aadafc3ebada6e5409470d2310c」，我们指定版本的时候，不用将这一长串的哈希数值写全，可只写前五六位即可，git会根据这个值去自动搜索。

```
[huwenyangdeMacBook-Air:TestFile huwenyang$ git log --pretty=oneline
e2f7426bcf3c5aadafc3ebada6e5409470d2310c (HEAD -> master) 测试一下diff
89a8f1141e32c6dd58d5fca1ccad9957416bb775 (origin/master) 提交一下主线
20643066103d91db238d13a1f5c29404c46e49d5 修改了工程
0a14018e8888290f1ee3a38cdf3c1d4011281b1d 修改README文件
7f8f7fbd1231f313903c1f97bfd907aaf8d2699e 添加了test工程
d0d6862296d05f9947f37868861d7921f9a6c7a2 更改README文件
2bb3737de47e407c36c061e6862cc214ecdd8d80 Initial commit
[huwenyangdeMacBook-Air:TestFile huwenyang$ git reset -hard HEAD^
error: 你的意思是 `--hard`（有两个短线？）
[huwenyangdeMacBook-Air:TestFile huwenyang$ git reset --hard HEAD^
HEAD 现在位于 89a8f11 提交一下主线
[huwenyangdeMacBook-Air:TestFile huwenyang$ git log --pretty=oneline
89a8f1141e32c6dd58d5fca1ccad9957416bb775 (HEAD -> master, origin/master) 提交一
下主线
20643066103d91db238d13a1f5c29404c46e49d5 修改了工程
0a14018e8888290f1ee3a38cdf3c1d4011281b1d 修改README文件
7f8f7fbd1231f313903c1f97bfd907aaf8d2699e 添加了test工程
d0d6862296d05f9947f37868861d7921f9a6c7a2 更改README文件
2bb3737de47e407c36c061e6862cc214ecdd8d80 Initial commit
[huwenyangdeMacBook-Air:TestFile huwenyang$ git reset --hard e2f74
HEAD 现在位于 e2f7426 测试一下diff
huwenyangdeMacBook-Air:TestFile huwenyang$
```

#####git版本切换特别的快，其根本原理是：改变HEAD 指针的指向，然后更新展示的信息，并没有将未来的版本删除，只是不将其显示而已，当我们后悔回退的话，我们可以再次更改HEAD指针，指向之前的未来的版本号（一般的记录一下commit id）。#####



改为指向`add distributed`：



######### $ git reflog　　#当我们忘记之前都针对于git库做了哪些命令操作，我们可以通过这个命令将之前使用的git指令记录全部调出来。

```
huwenyangdeMacBook-Air:TestFile huwenyang$ git reflog
e2f7426 (HEAD -> master) HEAD@{0}: reset: moving to e2f74
89a8f11 (origin/master) HEAD@{1}: reset: moving to HEAD^
e2f7426 (HEAD -> master) HEAD@{2}: commit: 测试一下diff
89a8f11 (origin/master) HEAD@{3}: checkout: moving from featrue_x to master
b265dad (origin/featrue_x, featrue_x) HEAD@{4}: commit: 修改了README文件
20eed5d HEAD@{5}: checkout: moving from master to featrue_x
89a8f11 (origin/master) HEAD@{6}: reset: moving to HEAD
89a8f11 (origin/master) HEAD@{7}: commit: 提交一下主线
2064306 HEAD@{8}: checkout: moving from featrue_x to master
20eed5d HEAD@{9}: commit: 提交分支改动
076e0f1 HEAD@{10}: checkout: moving from featrue_x to featrue_x
076e0f1 HEAD@{11}: commit: 添加一个模型文件
2064306 HEAD@{12}: checkout: moving from master to featrue_x
2064306 HEAD@{13}: checkout: moving from featrue_x to master
2064306 HEAD@{14}: checkout: moving from master to featrue_x
```

8. 放弃更改指令，删除一个文件

$ git checkout — <file>　　#放弃当前文件的最近所有修改

$ rm <fileName>　　#使用文件管理器删除一个文件，当前文件夹中已经将该文件删除，那么git会提示我们当前文件已经被删除了，如果我们真是想删除该文件，我们需要在git库中删除它，具体指令如下：

$ git rm <fileName>　　#git库中删除当前文件，然后commit一下就可以了。

9. 创建分支，切换分支（创建git库的时候，会默认创建一个主分支master分支）

$ git branch dev　　#创建一个dev分支

$ git checkout dev　　#切换到dev分支

$ git checkout -b dev　　#创建一个dev分支，并且转换到dev分支，相当于上面两句代码

$ git branch　　#查看所有的分支，当前分支前有 *

```
huwenyangdeMacBook-Air:TestFile huwenyang$ git branch
  featrue_x
* master
huwenyangdeMacBook-Air:TestFile huwenyang$
```

10. 合并分支，删除分支

$ git branch -d <branchName>　　#删除某一个分支

$ git merge <branchName>　　#将branchName分支合并到当前分支上

$ git merge --abort　　#终止合并

#合并分支的时候，一般默认Fast Forword模式快速合并，同时可以使用 —no-ff 关闭快速合并模式。

$ git merge —no-ff -m “添加相关的合并信息标注 ” <branchName>

11. 解决合并冲
    1. 一般都是通过 git status命令，查看是哪一个文件有冲突，然后手动打开文件，将里面的======== >>>>>> <<<<<<< 的特殊符号找到，确定冲突问题，保留应该保留的。
    2. git add *
    3. git commit -m “解决了我们的冲突”
    4. 合并成功，然后就可以愉快的删除分支了

12. 保存工作现场，恢复工作现场

使用场景是：当在dev分支开发的过程中突然接到一个任务要修改一个bug，但是当前做的修改又不能直接提交，因为
提交之后会影响到其他小伙伴的开发，那么我们可以使用当前的做的所有修改保存起来【$ git stash　#保存当前所有
的修改】

```
huwenyangdeMacBook-Air:TestFile huwenyang$ git diff
diff --git a/devReadme.md b/devReadme.md
index ba3a0cb..37ba6f2 100644
--- a/devReadme.md
+++ b/devReadme.md
@@ -1,2 +1,6 @@
 #1.0
 development 开发开始
+#1.1
+我卡卡卡卡做了好多修改，现在突然发现有一个bug要修改，但是呢当前代码又不能提交，
因为j提交了就影响到其他人的工作，
+这个时候就可以暂时存放的功能，等bug修改完成之后再恢复，继续之前的修改
+
huwenyangdeMacBook-Air:TestFile huwenyang$ git stash
Saved working directory and index state WIP on dev: a84c1e0 创建一个devReadme。m
d
huwenyangdeMacBook-Air:TestFile huwenyang$ git status
位于分支 dev
您的分支与上游分支 'origin/dev' 一致。

无文件要提交，干净的工作区
```

一般保存完成，再次使用【$ git status　#查看当前有什么修改】发现，当前分支没有任何修改，此时可以切换
分支，或者建立新分支去修改bug。
当修改bug完成之后，再次切换到当前dev分支，查看stash list ，找到相应的保存，再恢复。

工作现场恢复：Git把stash内容存在某个地方了，但是需要恢复一下，有两个办法：

1. 用git stash apply恢复，但是恢复后，stash内容并不删除，你需要用git stash drop来删除；
2. 用git stash pop，恢复的同时把stash内容也删了。

13. tag 创建
1.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
2.命令git tag -a <tagname> -m "blablabla..."可以指定标签信息
3.命令git tag可以查看所有标签

14. tag操作
1.命令git push origin <tagname>可以推送一个本地标签；
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；

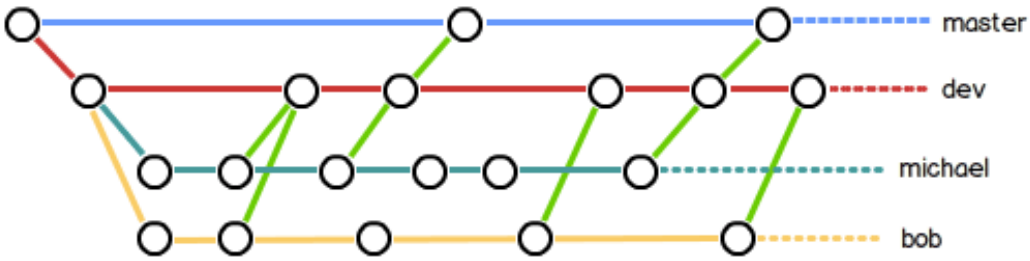
4.命令git push origin :refs/tags/<tagname>可以删除一个远程标签。

#开发策略#

#1　正式的发版项目应该是在master分支上面，开发的过程应该是在dev（开发分支上），然后开发成员每人一个分支，然后不时的向dev分支合并，等到要发布版本的时候，我们就可以将dev合并到master主分支上

#2　开发一个新feature，最好新建一个分支

#3　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


master
dev
michael
bob