2018-10-24

1、创建初始账户

在git bash 中执行

git config –global user.name “XXXX”

git config –global user.emai [lj.bao@huahengweld.com](mailto:lj.bao@huahengweld.com)

2、创建仓库，目的 供git管理的目录

（1）直接将你的项目所在的目录变为git仓库

进入你的项目所在目录，在git bash中cd 到你的项目所在目录执行git init （执行完毕，生成.git 文件，表示成功），此后可以执行 git add file 及 git commit –m “注释，说明本次提交的内容或者其他等”

（2）创建空目录，设置为git仓库，然后在增加项目内容。

在git bash 中cd到某个目录，执行 mkdir testgit，生成一个testgit 的文件夹，再执行git init,

其他操作同上。

这里的仓库都是指在我本地的一个工作区。

3、区域概念

当你将某个目录git init 意味着这个目录变为你的工作区，此时如果工作有文件为add,执行git status 会发现该文件或者文件夹的目录是Untrack 状态。

当你git add file ,则意味着这个文件保存在了git 内在的暂存区，此时这个文件属于 unstage,等待commit。

当你git commit 后，此时git status 发现 会提示你没有可以去提交的文件，此时文件均属于stage状态，已经被提交到某个分支（后续讲分支的概念，现在默认提交到master的分支下）

4、修改程序，查看修改内容

git status 随时用来跟踪工作区的文件状态

git diff git status告诉你有文件被修改过，用git diff可以查看修改内容。

-------早上早起学习的感觉棒棒哒。------后续要坚持，早睡早起早上学习。git

5、git log 查看修改记录

git log 查看修改记录，时间由近及远。主要会提示你commit 时的备注



如果觉得显示的内容太多，眼花缭乱，可以 --pretty=oneline

2种写法 git log pretty=oneline git log –oneline





其中这里一大串16进制的数字表示‘commit id ’ 即所谓的版本号，与SVN不同没有使用1,2,3，主要是考虑到git的分布式属性及后续多人同在一个版本工作会出现冲突，因此就有了这么个sha1 计算出来的数字。

如果想反向排序，从一个文件一开始到最新的commit 状态

git log –revise –oneline



6、搜索日志

1> git log --author [yourname]

筛选出yourname 提交的所有日志，yourname可以保护通配符（\*） （疑问：每个电脑只能有一个name么，同一个电脑可以使用不用的name去提交等操作么？）

$ git log --author ljbao

$ git log --author lj\*

2> git log –grep [keywords]

从提交的关键字抓取匹配的commit项



3> git log –p –-[file]

根据文件名获取仅和这个文件有关的提交

***待验证该功能***

4> git log --oneline --before={} --after={}

查看某个时间阶段的commit记录， 前 before，since，后 until，after

git log --oneline --before={3.weeks.ago} --after={2018-10-24}

 

7、版本回退

git reset –hard commitid

这里的commitid 是指提交时生成的16进制的数，可以通过git log –oneline 去查看



此时版本已有最新的哈哈 回退到 验证回退功能

如果 出现上述 unlink of file 的提示，这说明你的文件现在处于打开状态，回退需要在关闭当前文件或者程序的状态。

git log --oneline 发现只能查到这个版本之前的commit记录，那要是想回退刚刚我们回退之前的版本怎么办呢？ 别担心，再git这里你可以吃到后悔药。git reflog



你可以查看到整个版本的全部信息，这个时候你在使用git reset --hard commitID,即可回到回退之前的版本



版本回退总结：

1、Head 就是当前的版本。git 允许我们在版本的历史之间来穿梭，git reset –hard commitid

2、git log –oneline 查看提交历史记录，以便于回退到哪个历史版本中。

3、git reflog 查看命了历史，以便于回到未来的某个版本。

8、git 工作区和暂存区

Git 和 SVN的一个不同之处在于，他有属于自己的暂存区

* **工作区：**就是你在电脑里能看到的目录。
* **暂存区：**英文叫stage, 或index。一般存放在 ".git目录下" 下的index文件（.git/index）中，所以我们把暂存区有时也叫作索引（index）。
* **版本库：**工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库。



版本库中存放很多东西，最重要的就是stage（或者Linux 中叫index）暂存区，所有未add之前的文件全部在工作区，add 之后未commit之前在存放于暂存区，commit 之后存放于分支中。当我们建立一个版本库后，git 默认给我们创建了一个master 分支以及指向master 的第一个指针HEAD.

**后续区分master 和HEAD**

图中的 objects 标识的区域为 Git 的对象库，实际位于 ".git/objects" 目录下，里面包含了创建的各种对象及内容。

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，同时工作区修改（或新增）的文件内容被写入到对象库中的一个新的对象中，而该对象的ID被记录在暂存区的文件索引中。

当执行 "git rm --cached <file>" 命令时，会直接从暂存区删除文件，工作区则不做出改变。

9、如何管理修改

git commit 只针对已经add的修改，若在工作区修改，未add,那么commit后，库里的仍是第一次add后的结果。

下面我们以实际运行查看，绿色字体代表为add 但未提交的，存在于工作区的，白色字体说明已经add 且commit成功





2015-10-25

10、撤销修改

2种情况下的修改：

1、只是在工作区作了修改，未add 到stage（暂存区）

Git checkout [file] 将file恢复到最后一次commit状态

2、add 到stage 后，作了修改，那么git checkout [file]，则是恢复到add 之前的状态

11、删除修改

(1) 删除工作区和版本库中的该文件

（出现一个疑惑，如果每个人都可以删除版本库里的文件，那么到底以哪个版本库的为基准呢）

Git rm –r file(文件夹)

Git rm file(文件)

Git commit –m ‘beizhuxinxi ’

(2) 删除已修改且add后未commit 的工作区，暂存区和版本库

Git rm –f file

Git rm –f -r file(文件夹)

(3) 工作区保留，仅仅删除版本库

Git rm –cached file

（二）远程仓篇

远程仓有两种

1：运用现有的GitHub 远程托管（国外），当然还有其他形式的码云（[gitee.com](https://gitee.com/)）（国内）

2：在公司搭建自己的git服务器，后续将，目前先掌握如何将远程仓和本地仓的关联。

远程仓控制：

1：将本地仓内容push 到远程仓

（1）远程仓新建 repository 库名

Git remote add origin <https://github.com/HHblj/testgit.git>

Git push –u origin master

由于远程库是空的所以第一次推送是需要参数-u, git 不但把本地的master 内容推送至远程的master 分支，还会把本地的master分支和远程master 分支关联起来。

2：将远程仓clone 到本地仓

远程仓已有，在本地选择一个目录去clone 远程仓的文件，无需任何其他操作。

git clone <https://github.com/HHblj/testgit2.git>

完成clone ，查看本地确实存在此仓库。

3、查看当前的远程库

Git remote

Git remote –v

**总结：**

要关联一个远程库，使用命令git remote add origin git@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；

分布式版本系统的最大好处之一是在本地工作完全不需要考虑远程库的存在，也就是有没有联网都可以正常工作，而SVN在没有联网的时候是拒绝干活的！当有网络的时候，再把本地提交推送一下就完成了同步，真是太方便了！

(三) 分支

Master 与分支

1、新建分支 git brach jojo

2、切换分支 git checkout jojo

3、研究master分支和其他分支如何并行工作。

4、验证分支合并功能，再次验证如何工作

5、hello branch