git基础概念

* 客户端并不只是提取最新版本的文件快照，而是把代码仓库完整的镜像下来
* git核心本质上是一个键值对数据库。可以向该数据库插入任意类型的内容，
  他会返回一个键值，通过该值可以在任意时刻再次检索该内容。

区域

* 工作区 （沙箱环境 git不会管理 随便更改操作）

* 暂存区 （记录文件的操作）

* 版本库 （最终的代码实现提交到这里 .git目录就是版本库）

.git目录下文件的介绍

* hooks (钩子函数的一个库 类似于回调函数)
* info (包含一个全局性的排除文件)
* objects (目录存储所有数据内容)
* refs (目录存储指向数据（分支）的提交对象的指针)
* config (文件包含项目特有的配置选项)
* description (显示对仓库的描述信息)
* HEAD (文件目前被检出的分支)
* logs (日志信息)
* index (文件保存暂存区的信息)

基础Linux命令

* clear 清除屏幕
* echo 'hello word '>test.tet 命令台书写内容
* ll 将当前目录下的子目录展现出来
* find ./ 将当前目录下的子目录以及文件也展现出来
* find ./ -type f 只讲文件展现出来
* rm text.txt 删除文件
* MV a.txt b.txt 更名字
* cat a.txt 查看文件内容
* vim a.txt 编辑内容 i 插入 ese : wq 保存退出 :set nu 设置行号 q! 强制退出不保存

对象

* Git对象
    - echo "hello" | git hash-object --stdin
      - 这句命令返回一个hash值用来标识这句话 但是并没有写到数据库中
      - 内容不一样对应的hash值不一样

    - echo "hello" | git hash-object -w --stdin
      - 这句命令返回一个hash值用来标识这句话 并写到数据库中
      - 查看有没有存在 可以通过 find ./ -type f 找对应hash的文件
      - 里面的内容是压缩的 通过 git cat-file -p hash注意前面的那两个字母也加上
      - git cat-file -t hash 查看git对象的类型 blob

    - 将新创建的文件添加到git数据库中即生成一个git对象
      - git hash-object -w ./a.txt

    - 如果文件更改git数据库里面不会自动的添加要手动添加过去 这时会在添加一个git对象
      - git hash-object -w ./a.txt

    - 实质上Git对象是一个KYE：VALUE hash/value
    - git对象不能当作项目的一次快照 只是组成项目的一部分

    - 存在的问题？
      - 记住文件的每一个（版本）对应的hash值并不现实
      - 在git中，文件名并没有被保存，只能通过hash

    - 注意：此时的操作只是针对本地数据库进行操作，不涉及暂存区。

* 树对象
    * 树对象能够解决文件名保存的问题，也允许我们将多个文件组织到一起。

    * 构建树对象
      - git update-index --add --cacheinfo 100644 915c628f360b2d8c3edbe1ac65cf575b69029b61 test.txt
      - 文件模式为100644 表明这是一个普通文件
      - 文件模式为100755 表明这是一个可执行文件
      - 文件模式为120000 表明这是一个符号连接
      - --add 因为此前该文件并没有在暂存区中 首次要加add
      - --cacheinfo 因为要添加的文件在git数据库中,没有位于当前目录下

    * 暂存区做一个快照生成一个对象放到git数据库中
      - git write-tree
        对象类型是一个树对象
        树对象里面的内容是暂存区的快照（项目的快照）

```
    * 暂存区中文件名字不变 如果改变文件的内容，就会重新生成一个hash

    * 存在的问题?
      - 不知道hash值对应的是哪一个版本
      - 不知道这个版本的一些基础信息

* 提交对象
    - 提交对象完美的解决了上面的问题
    - 本质就是给树对象做一层包裹包含项目的基础信息
    - commit-tree创建一个提交对象，为此需要指定一个树对象的hash值,以及该提交的父提交对象
    - echo "second commit" | git commit-tree 019fb2c522b604cd94929085bbac93d60e2f2063 -p  d248eb19a125c

    - 真正代表一个项目的是一个提交对象（数据和基本信息）这是一个链式的!!
```

初始化git

```
* git init （初始化仓库 生成.git文件）
* git config --global user.name "Is zyd"
* git config --global user.email 1426593075@qq.com
* git config --list
```

添加到暂存区

```
* git add ./   首先将工作区（文件）做成git对象放到版本库 然后再放到暂存区 但是这里没     有生成树对象
* git ls-files -s  查看暂存区的当前状态
```

添加到版本库

```
* git commit -m '提交的信息'
```

结论

```
* 一次完整的项目提交 包括至少一个提交对象 一个树对象 0或多个git对象
* 工作目录中文件只有两种状态 已跟踪（只要第一次add就跟踪上了） 未跟踪
* 已经跟踪的文件还有三种状态 已提交 已修改 已暂存
* 如果一个已经提交的文件再次修改要重新添加到暂存区否则显示已修改状态
* 如果一个文件暂存完了没有提交前还要在修改 这时会出现一个暂存一个已修改的情况需要重新add
*
```

* 分支就是为了保护代码方便更改存在的 假如master里面的提交对象完美了就可以在创建一个分支 添加功能如果可以就可以master合并 不行的话就可以删除这个分支 这样对于master没有影响

* 新建一个分支到一个提交对象上面 这样做的好处是实现版本回推但是不改边主仓库的东西 用完 删除这个分支就可以了特别方便

* 合并分支一定要注意顺序 后面的可能会过期还会存在bug 会产生冲突
    - 快速合并 一条分支
    - 典型合并 多条分支 会有冲突（打开冲突文件看哪里要留 然后暂存提交）
    - 同事之间的冲突才是最麻烦的

高级命令（crud）

```
* git init                  初始化
* git add ./                添加暂存区
* git commit -m "注释"       提交项目
* git status                查看当前状态
* git diff                  查看当先做的哪些更新没有暂存
* git diff -cached          查看哪些已经更新好了准备下次提交
* git commit -a -m          git自动将已经跟踪过的文件暂存起来一并提交
* rm yd.txt                 删除文件 暂存区里没有文集 版本库多了一个提交对象不过没有内容
* 删除文件属于修改操作 跟上面的提交步骤一样
* mv z.txt zx.txt           重新起名字跟已修改一样的操作
* git add ./
* git commit -m "rename zx"
* git log --oneline        查看提交历史记录
* git commit --amend       查看最近的提交信息
```

分支

```
* 每一个功能都可以开一个分支，不影响主线的分支
* 分支就是一个活动的指针就在提交对象的前面指向最新提交
* master默认是主分支
* git branch test      会在当前的提交对象上创建一个分支
* git checkout test   将分支转到test上面来
* git branch -D test   删除分支 不能自己删自己
* git log --oneline --decorate --graph --all   查看完整的分支图（没删除前）
```

* git config --global alias.lol ‘log --oneline --decorate --graph --all’ 陪别名
* git branch -v 查看分支的最后一个提交
* git branch test hash 新建一个分支到hash所对应的提交对象上去 很重要
* git checkout -b test 创建分支并且切换过去
* git checkout name 切换分支的时候一定要提交完的时候再切否则会出现问题
  每次切换分支前当前分支一定要是已提交状态 否则会污染主分支 如果第一次提交了再修改的时候没有提交他就不让切换分支了
* git merge hotbug(分支名)
* git branch -a 查看所有分支
* git branch -r 查看远程分支

存储

* 解决的问题 不想过多的创建提交比如iss53那个分支
* git stash list 查看存储
* git stash apply 拿出栈顶的元素 但是不会消除
* git stash drop 名字

后悔药

* 工作区撤回在工作目录中的修改
  * git checkout -- filename 本质是相当于重置
* 暂存区撤回自己的暂存
  * git reset HEAD filename
* 提交区注释写错了修改注释
  * git commit --amend

项目经理远程操作github仓库步骤

1. 先在github上创建一个空的仓库new repository 注意不要有readme.md文件
2. 创建本地仓库然后基础设置　git init
3. 然后给github上面的地址起别名和用户别名
  * git remote add use https://github.com/zhaoyuanmeng/git_to_use.git
  * git config --list
4. 注意如果是复制别人的github 要把.git删掉只复制项目部分代码到自己的本地仓库
5. 注意凭据 本人是可以直接上传的
6. 检查完毕后推送到远地仓库　git push use(别名) master（分支）
7. 给员工开放权限通过github里面的manage access contributor
8. 获取员工上传的代码 git fetch use(别名)
9. 切换成远程跟踪分支 git checkout use/master
10. 合并远程跟踪分支 git merge use/master
11. git pull　获取数据并合并

员工拉取项目经理的仓库代码

1. 本地不用创建仓库 直接克隆下来
  * git clone https://github.com/zhaoyuanmeng/git_to_use.git
2. 它自动创建一个别名; 查看别名git remote -v
3. 创建新的文件 echo "hello world">test.txt
4. git add ./
5. git commit -m "tijiao"
6. git push origin master

本地分支 远程分支 远程跟踪分支

* 本地分支是本机电脑的
* 远程分支是github上对应的分支
* 远程跟踪分支是本地与远程分支的一个映射
* 成员克隆远程仓库以后默认本地分支和对应的远程跟踪分支有同步关系
* 在push的时候会生成对应的远程跟踪分支
* 在fetch的时候把数据下载到远程跟踪分支里面
* 注意成员开辟新分支提交的时候 经理在fetch的时候要创建对应的分支不用加别名
* 主分支和远程跟踪分支自动绑定的功能(默认情况下push的时候)
* 建立同步关系 git branch -u (远程跟踪分支) 注意要在那个分支里面输入这个命令
* git checkout --track remote别名/分支名　最自动创建本地分支并且与远程跟踪分支绑定
* git checkout -b 分支名 remote别名/分支名 效果与上面一样

删除远程分支

* git push use(别名) --delete (分支名) 删除远程分支
* git remote prune use --dry-run 列出仍在远程跟踪但是远程分支已经被删除的无用分支
* git remote prune use 清除上面的命令列出来的远程跟宗

冲突

* git本地操作的冲突
    * 典型合并的时候
    *
* git远程协作的时候
    * push
        * 两个人同时推（更改同一个文件）解决办法只能先把远程仓库拉下来 然后再更改那个 文件然后再add commit push
    * pull
        * 更改完以后不push 直接pull会报错 远程仓库会覆盖更改的内容建议push不过push还会出错就是上面那个错误

参加开源项目的步骤 （pull request）

* 如果参加某个项目时，但是没有推送权限，这时候可以通过对这个项目进行fork。这会在
  你的空间中创建一个完全属于你的项目副本，且你对其具有推送权限。通过这个方式项目的
  管理者不用忙着添加贡献者，人们可以fork这个项目将修改推送到项目副本上，并通过pull
  request来将他们的改动进入源版本库

1. 先将源仓库fork到自己的仓库
2. 然后clone到本地仓库
3. 更改后提交到自己的远程仓库
4. pull request
5. 管理人审核 然后merge

* 不重新fork怎么解决
  * git 支持同时跟踪多个仓库
    git remote add 别名2 地址
    <!-- git remote rm 别名1 -->
    git fetch 别名2
    git branch -u 远程跟踪分支
    git merge 对应的远程跟踪分支
    git push (这里还是会提交到自己上面)

然后pull request

SSH

* github特有的一种协议 走的不是验证密码的道路而是密匙
* 配置步骤
  * ssh -keygen -t rsa -C 邮箱名
  * 在c:\users\Adminstor\.ssh下生成公私密匙
  * ssh -T git@github.com 测试一下
  * 把公共的密匙复制到github账户里面setting下
  * 好处是不用每次输密码 而且项目经理不用设置贡献者 直接把他们的公共密匙复制到自己账户 就可以了

.gitnore文件

* 这个文件可以直接从github上下载
* 目的是不在git仓库上传不必要的文件