Git相关概念

三个区域 四个对象 快照 SHA1 文件系统

1. 快照

传统版本控制工具记录的是每个文件与初始版本的差异。

Git记录的是项目随时间改变的快照，即每个版本都保存所有文件。

2. SHA1

Git中所有用来表示项目历史信息的文件，是通过一个40字符的"对象名"索引，即通过SHA1算法索引。

优点：1.比较对象名快速判断对象　2.同样的内容存储在不同仓库中拥有相同的对象名　3.可通过检查对象内容的SHA1的哈希值与对象名是否相同判断内容正确

3. 四个对象

- blob　存储文件数据，通常是一个文件
- tree 类似目录，管理一些tree/blob
- commit　指向一个tree对象且带有相关描述信息
- tag　标记某一个commit的方法

4. 三个区域

- 工作区　对项目的某个版本独立提取出来的内容
- 暂存区　保存下次将提交的文件列表信息
- 版本库　保存项目元数据和对象数据库

5. 两种状态

- 已跟踪　已纳入版本控制的文件，上一快照内有其记录，工作一段时间后可能处于未修改、已修改或已放入暂存区
- 未跟踪　既不存在于上次快照中，也未放入暂存区

6. 基本工作流程

工作区修改文件 -》 暂存文件，将快照放入暂存区 -》 提交更新，找到暂存区文件，快照永久存储到版本库

Git命令

```
1. git配置命令
git config --global user.name ""    //设置git全局用户名
git config --global user.email xxx  //设置git全局用户邮箱
git help config //配置命令提示
git config --list   //当前配置以列表显示

2. 基本操作
git clone <url> //支持https、git、ssh三种传输协议进行克隆操作
git status  //查看当前状态：  已修改、已暂存、未跟踪
git add .   //跟踪所有文件，包括新文件和原有已修改文件
git add -u  //不跟踪新文件，只把已跟踪文件放入暂存区，需要显式add新文件
```

```
git diff    //查看详细修改，通过不同参数，查看工作区、暂存区、版本库间两两内容的差异
    .1 --cached 暂存区与HEAD比较 .2 HEAD 工作区与HEAD比较 .3 无参数 工作区与暂存区比较
git commit  //修改并提交到仓库
git commit --amend  //当提交后发现漏掉n个文件未提交或提交信息错误，使用该命令回溯
    exp:
        git commit -m "initial commit"
        git add forgotten_file
        git commit --amend
        //最终只会有一次提交记录，第二次将覆盖第一次提交
git log //查看提交历史
git reset HEAD <file>   //取消暂存，将committed的文件移除出来
git reset --hard //不能找回丢失数据，硬重置使工作区文件被覆盖
git checkout -- <file>  //撤销对某个文件的修改，将其还原为上次提交时,撤销后无法记录之前的修改

3. 分支
//git分支本质上是指向提交对象的可变指针，它会在每次提交操作中自动向前移动
//git使用HEAD指向当前所在的本地分支，HEAD随提交操作自动向前移动
git branch testing //从当前分支拉取新分支testing
git checkout testing    //从当前分支切换到testing分支
git checkout -b iss53   //从当前分支拉取新分支iss53并切换过去
git branch -d hotfix    //删除分支hotfix

4. 远程仓库
git remote  //查看当前远程仓库
git remote -v   //origin表示远程库服务器的默认名
git remote add google xxxx  //添加远程仓库
//远程分支的命名 remote/branch 如： origin/master

git checkout -b [branch] [remotename]/[branch]
git checkout [remotename]/[branch]  //从远程拉取一个跟踪分支
//从一个远程分支拉取一个本地分支将自动创建一个跟踪分支 git branch -vv 查看所有跟踪分支

//git pull = git fetch + git merge

git push origin master
git push origin refs/heads/master:refs/heads/master //本地分支:远程分支

5. rebase
git checkout experiment
git rebase master   //把分叉experiment变基到master

git rebase [--onto <newbase>] [<upstream>] [<branch>]
git rebase --onto master servcer client //筛选出在client分支上存在但server分支上不存在的提交，将其链接到master分支后
//交互式用法 git rebase -i topicA~5 //-i交互用法
//Cherry-pick方式 移除某些提交,每次“复制”一个提交并在当前分支做一次完全一样的新提交

6. 命令区别
git merge有2种模式： 快进模式、三方合并模式
    fast-forward 快进式合并，被合并分支和主分支在一条线上，不存在分叉
    diverged 分叉式合并，被合并分支和主分支存在分叉，执行三方合并后向前移动新出现一个节点
git reset有2种用法： 撤销暂存、重置引用
git checkout有2种用法： 撤销工作区文件的修改、切换分支

7. 使用原则
使用Merge和Rebase的基本原则
- 下游分支更新上游分支内容时使用Rebase
```

- 上游分支更新下游分支内容时使用Merge

一般使用原则
- 对还未推送到服务端的提交，推荐使用Rebase
- 切勿对服务端已有的提交做Rebase(git pull --rebase) & 强推

对Git flow流程的理解

> git flow流程是我在实习时所接触到的一种开发过程中的版本控制思想，意在使项目流程清晰、规划完善，避免每位开发者对应一堆提交而导致项目混乱、主线不清晰、难以协调维护的问题。
>
> 总的而言，git flow是为了**维护多人参与的大项目的开发流程清晰，代码提交、合并流程规范。**

1. Git flow的分支类别

- **Production** 即master分支，只能从其他分支合并，不能在该分支直接修改，该分支为最近发布到生产环境的代码。
- **Develop** 主开发分支，包含所有要发布到下一版本的代码，主要由Feature分支合并。
- **Feature** 主要用来开发一个新功能，一旦开发完成，将其合并回Develop分支。
- **Release** 版本发布分支，包含所有要发布到下一版本的代码，由Develop分支拉取，完成发布后合并回Develop/Production分支。
- **Hotfix** 当生产环境上版本出现Bug时，创建该分支进行Bug修复，完成后合并回Develop/Production分支。

2. Git flow的理论使用

- Production/Develop分支

Develop分支基于Production分支拉取，Production分支为线上版本稳定代码，Develop分支为开发中版本的代码。

- Feature分支

新功能开发完成后合并回Develop分支，随后一般会删除该分支。

- Release分支

区别于Develop分支，该分支主要用于下一版本的代码整合、测试、发布阶段，基于Develop分支拉取，该分支应记录相应的迭代版本号，在发布完成合并回Production/Develop分支后应Tag记录版本号，随后可删除该Release分支。

- Hotfix分支

基于Production分支拉取，修复完成后合并回Production/Develop分支时建议Tag记录。

3. Git flow的实际使用

以我的实习经历为例，master分支仅限部门主管具有合并拉取权限；release分支的合并拉取权限进一步扩大到项目资深工程师，根据版本迭代时间可看到一连串release分支，如当前版本预计在2021年9月11日上线，则项目负责人将从master分支或已完成交付的最近release分支拉取一个新分支，并将之命名为release/20210911，测试、发布初期将主要关注于该分支，经过线上运行稳定后再由项目负责人将其合并到master分支中；所有参与者皆有权限合并拉取feature分支，版本迭代按照业务需求分配到开发人员，随后根据个人任务从release分支拉取feature分支，如当前任务为“新功能”，则分支名大致为feature/20210911_new_func，当开发完成则向对应版本的release分支发起一个merge request请求，项目负责人/资深工程师对代码修改进行评估后将其合并到release分支，完成某个需求的整合；hotfix分支仅用于线上热修复的分支，仅项目负责人有操作权限，当线上版本出现未知问题需要即刻修复时将拉

取该分支，解决问题后合并到release、master分支。四类分支在完成相应作用后均不会被删除，以此可对开发隐患有较好的溯源能力，也可记录每个版本所做出的迭代工作。

4. 对git flow流程现存的疑惑

假设同一时间前后AB两个版本同步开发，存在两个release分支且B分支最终要将A分支合并。此时开发者各自完成的feature分支merge到对应的release分支，此时若A版本顺利交付，合并到B版本时出现冲突，若提前完成的B版本业务功能与A版本出现冲突，则将存在“轮子白造了”的情况。

Git在AS中的使用

AS自带git可视化工具，相比较纯命令行式操作更直观简便，本节将于学习AS开发工具时再总结。