# GIT内部培训

主讲人：郑泽楷

# 目录

CONTENTS


01. Git简介

02. Git基本操作

03. Git版本管理

04. 多人协作场景

05. 代码合并指令使用讨论及版本回退

# 一、Git简介

● What is Git?

- Git是目前世界上最先进的分布式版本控制系统（没有之一）

● Why is Git?

- 优点：1、适合分布式开发，强调个体，离线工作。

2、公共服务器压力和数据量都不会太大。

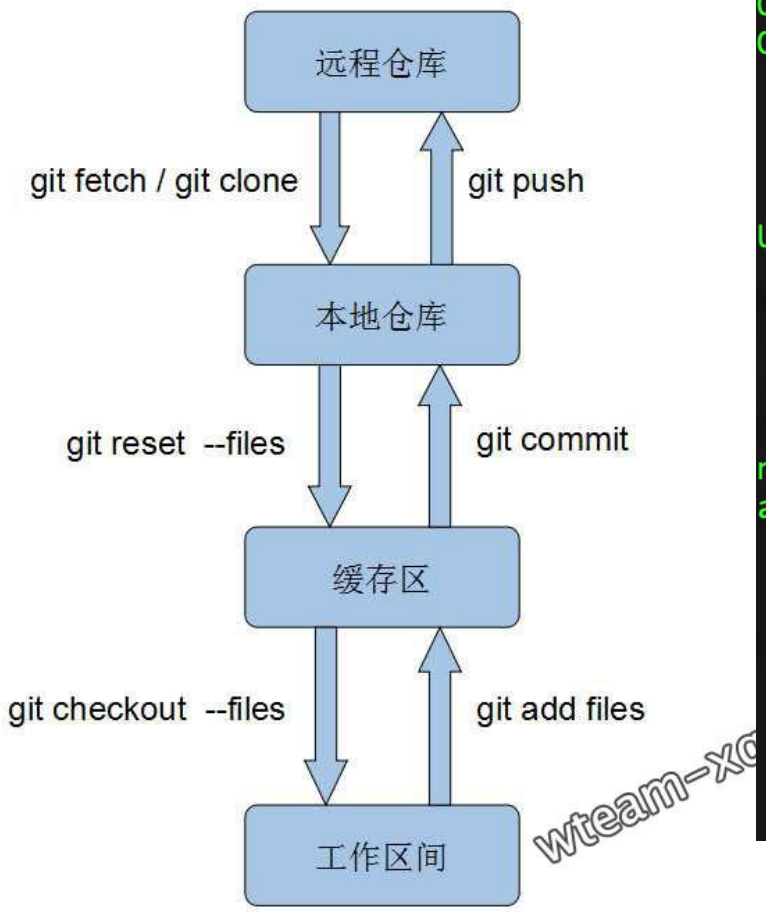
3、速度快、灵活。

4、任意两个开发者之间可以很容易的解决冲突。

- 缺点：代码保密性差，一旦开发者把整个库克隆下来就可以完全公开所有代码和版本信息。

如果需要像SVN一样变态地控制权限，用Gitolite。

如果需要进行代码审核，可以用gerrit。




```
hellogit — -zsh — 80×24
app@appdeMacBook-Pro hellogit % vim CMakeLists.txt
app@appdeMacBook-Pro hellogit % git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   CMakeLists.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .DS_Store
        .vscode/
        main.cpp

no changes added to commit (use "git add" and/or "git commit -a")
app@appdeMacBook-Pro hellogit %
```

使用.gitignore让git工程忽略无关的文件或文件夹

# 二、Git基本操作

廖雪峰的Git教程：地址：https://www.liaoxuefeng.com/wiki/896043488029600

git init

git add

git commit

git status

git branch

git push

git fetch

git log https://git-scm.com/book/zh/v2/Git-基础-查看提交历史

git checkout

git merge --no-ff/ git merge --abort https://blog.csdn.net/chaiyu2002/article/details/81020370

* git reflog 程序员的后悔药

git diff

git difftool

git stash

# 三、Git版本管理

git tag -a v1.0-beta -m "v1.0 beta版本发布上线"

git tag # 查看tag列表

git tag -l # 查看tag列表

# 此处对历史提交做tag处理

git log --pretty=oneline --abbrev-commit

git tag -a v0.9 -m "v0.9版本发布上线" <commit-id> # 对历史提交做tag处理

git push origin v1.0-beta # 推送到远程仓库

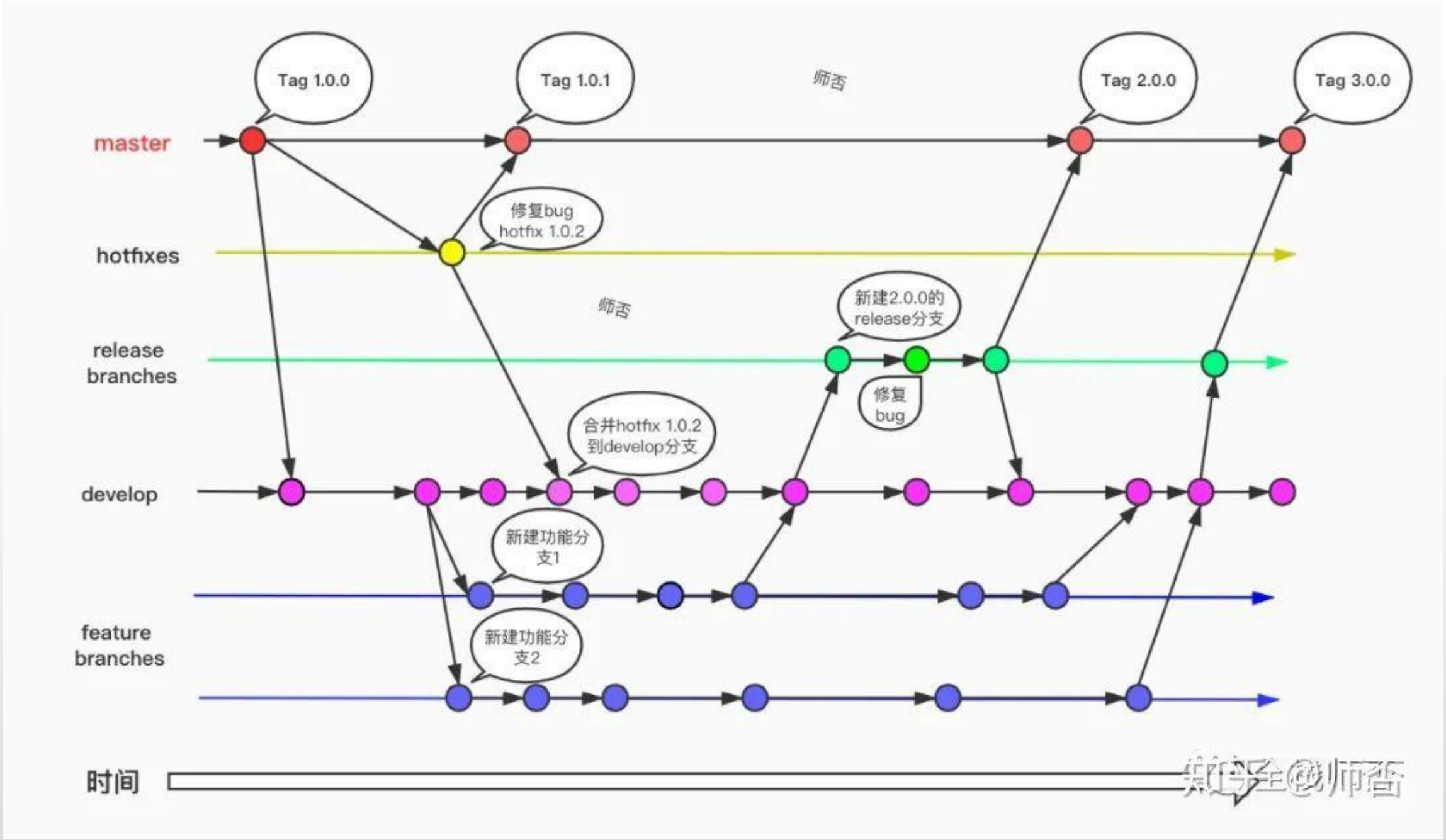
git push origin --tags # 一次性推送全部尚未推送到远程的本地tags

git tag -d v1.0-beta # -d参数删除掉tag

git push origin :refs/tags/v1.0-beta # 删除掉远程仓库的tag,名称为v1.0的tag

# 四、多人协作场景

强烈推荐使用Vincent Driessen 提出的GIT Flow


Tag 1.0.0
Tag 1.0.1
师否
Tag 2.0.0
Tag 3.0.0
master
修复bug
hotfix 1.0.2
hotfixes
师否
新建2.0.0的
release分支
release
branches
修复
bug
合并hotfix 1.0.2
到develop分支
develop
新建功能分
支1
feature
branches
新建功能分
支2
时间
知乎@师否

# 使用方法

1.一个普通的git工程，基于当前master分支最新的提交切出开发分支develop，所有开发任务基于该分支。

2.需要开发新功能时，基于develop分支的最新提交切换出feature分支，在feature上单独进行功能开发。开发完成，自测通过后，合并分支代码到develop分支。

3.需要将develop分支的代码提测一个版本给到生产环境时，首先基于最新的develop分支的提交，切换出一个release分支，该分支用于给到测试部同事进行测试。如果测试过程中发现一些bug，则在release分支上进行改动，最后测试通过，所有bug解决，将release分支和master分支、develop分支合并，并且使用git tag给合并后的master的最新提交打上标签，发布该版本到生产环境。

注意：如果有人正在release分支上进行相关工作，也就是由他负责在release分支的问题都解决完了之后发布一个版本，在这个人发布版本之前，其他人不应该自行准备发布一个版本，也创建一个release分支进行工作。一个工程同时刻应该只有一个release分支，由单人负责跟进和解决问题。等到他发布版本之后，release分支的代码会合并到develop分支，这时如果还有谁需要发布版本到生产环境，再从develop分支切到release，开始工作。

4.生产环境运行的代码出现了bug，则基于git tag版本号找到对应的代码，切换出hotfix分支，在hotfix分支上解决问题后，将代码合并到develop分支和master分支，master分支的版本号可以带后缀-hotfix-bugId来做记录。

有经验的人总是推荐git fetch + git merge而不是git pull

Why?

还有用git fetch + git rebase 的? git rebase又是啥玩意?

git pull & git fetch + git m

1. 相同点

首先在作用上

2. 不同点

先补充一些g

首先我们要说 （生成
本地仓的cor 悉。

我们本地的g mit ID
（可以有多个

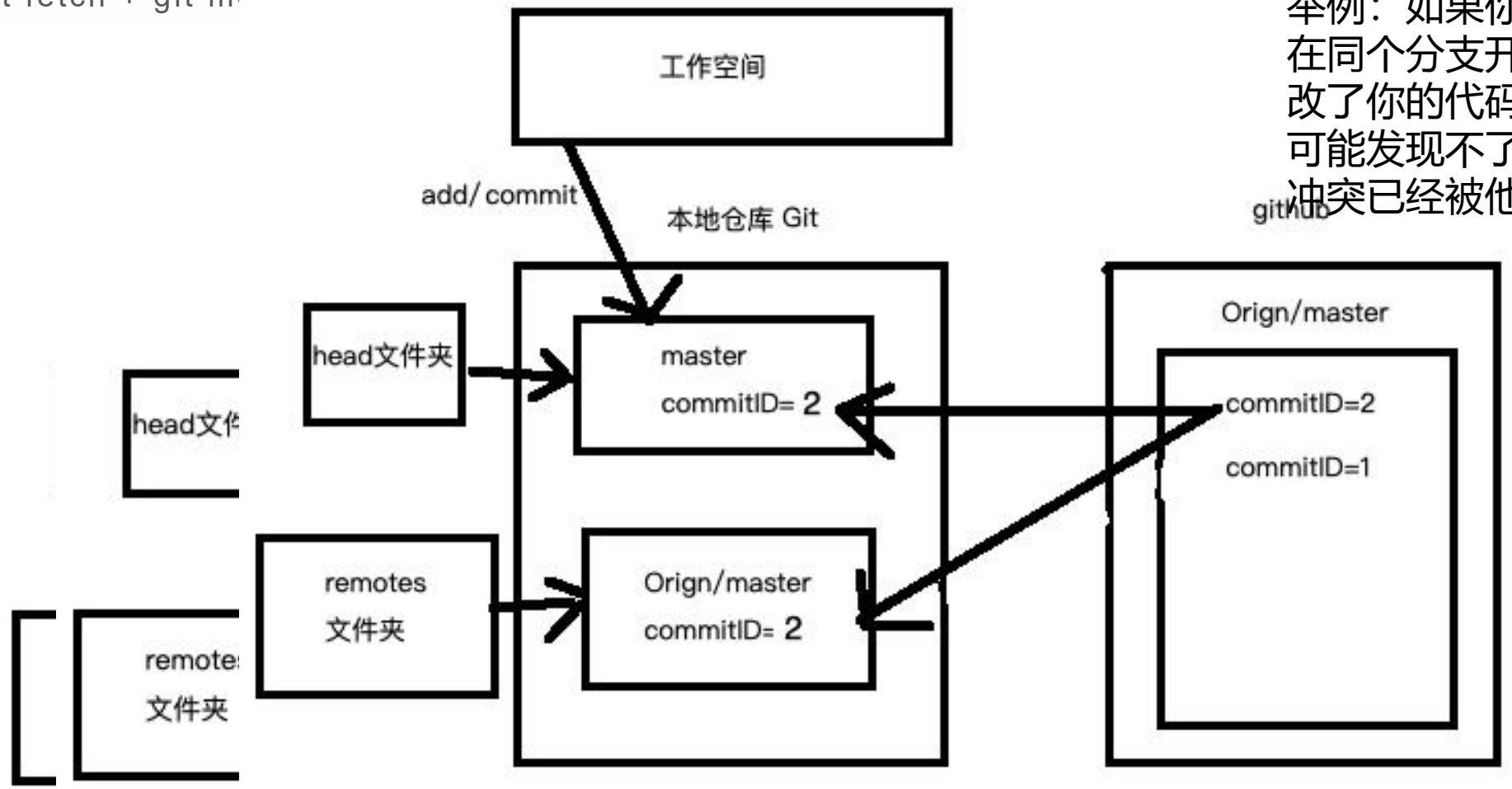
.git/refs/hea

.git/refs/rem

其中head就 交的
SHA1校验和

但是，不管他

更改远端跟踪 分支操
作，我们必须




举例：如果你和别人一起在同个分支开发，他偷偷改了你的代码，你直接pull可能发现不了错误，因为冲突已经被他解决过了。


https://blog.csdn.net/weixin_41975655

不要用git pull，用git fetch和git merge代替它。

git pull的问题是它把过程的细节都隐藏了起来，以至于你不用去了解git中各种类型分支的区别和使用方法。当然，多数时候这是没问题的，但一旦代码有问题，你很难找到出错的地方。看起来git pull的用法会使你吃惊，简单看一下git的使用文档应该就能说服你。

将下载（fetch）和合并（merge）放到一个命令里的另外一个弊端是，你的本地工作目录在未经确认的情况下就会被远程分支更新。当然，除非你关闭所有的安全选项，否则git pull在你本地工作目录还不至于造成不可挽回的损失，但很多时候我们宁愿做的慢一些，也不愿意返工重来。

git rebase

https://www.cnblogs.com/cangqinglang/p/12419568.html

git revert / git reset

回滚xxx　/ 回滚到xxx

git cherry-pick

单独取出某次提交或某几次提交的代码合并过来

git rm --cached 去掉已追踪的文件，但不在磁盘上删除

git clean

# Thanks!