Git Introduction

An open source distributed version control system

DropFan@Gmail.com
2015.04

What is Git?

- Git是一个分布式版本控制系统，由Junio Hamano和Linux Torvalds开发
- Git不需要中心服务器
- Git可以运行在Linux, BSD, Solaris, Darwin, Windows, Android等操作系统

Git vs SVN

- Git是分布式的，SVN是集中式
  这是Git和SVN最核心的区别。Git也可以有中心服务器，但每个人的本地都是一个完整的Git仓库。团队成员之间也可以同步代码。分布式最大的优点，没有网络也可以正常工作。查看log diff 提交commit 建立branch、tag等操作都无需联网，在有网络的时候同步就可以了。而集中式的一旦断网则无法正常工作。

- Git把内容按元数据方式存储，而SVN是按文件
  对于同样几个分支，SVN都是完整的一份拷贝，占用空间大。而Git只保存其中的差异。

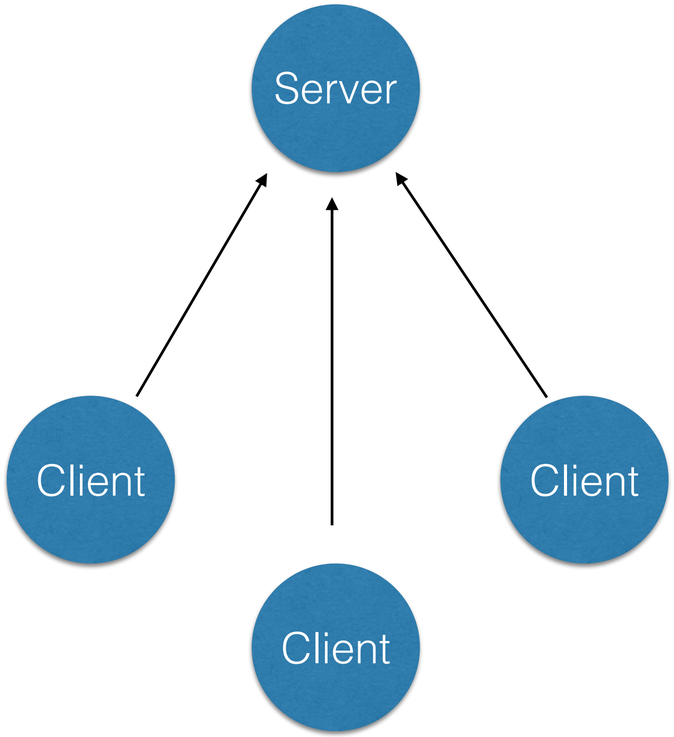
- GIT分支和SVN的分支不同
  SVN的分支实际上只是某一版本代码的拷贝，占用空间大。并且不具备智能化管理的功能，对分支的任何改动都会影响其他人
  Git的分支十分智能，可以任意创建和删除而不影响其他人工作副本

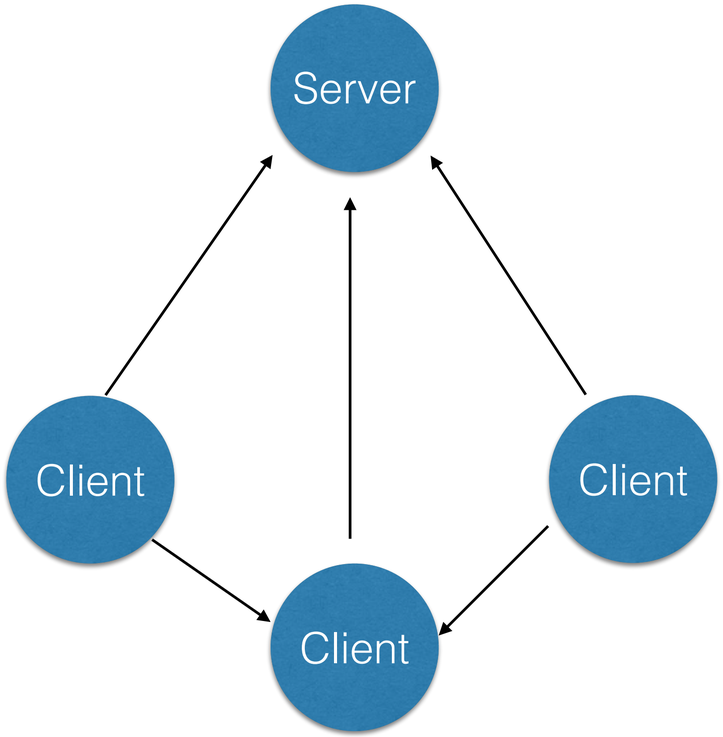
- GIT没有一个全局的版本号，而SVN有
  SVN的版本管理是线性的，而Git每一次提交都有一个唯一的hash作为标记

- GIT的内容完整性要优于SVN
  GIT的内容存储使用的是SHA-1哈希算法。这能确保代码内容的完整性，确保在遇到磁盘故障和网络问题时降低对版本库的破坏。

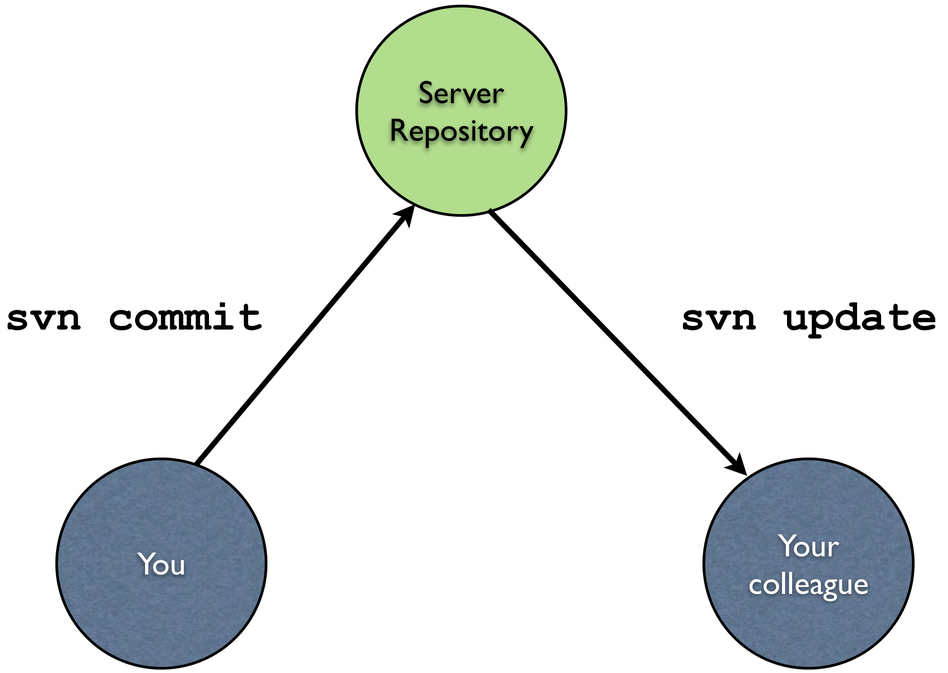
- Git速度要比SVN快

集中式 VS 分布式



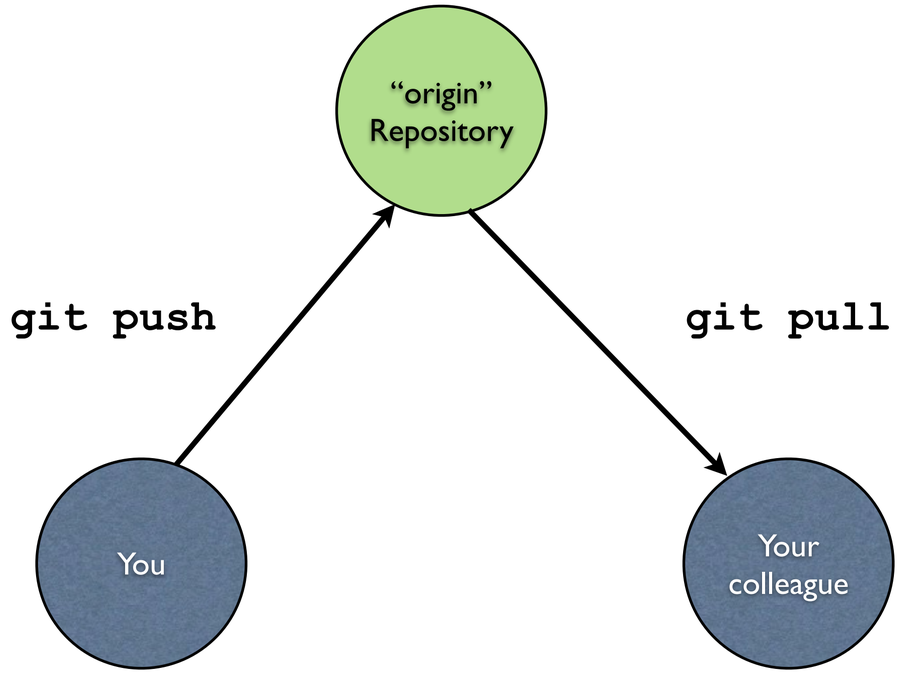


The SVN model

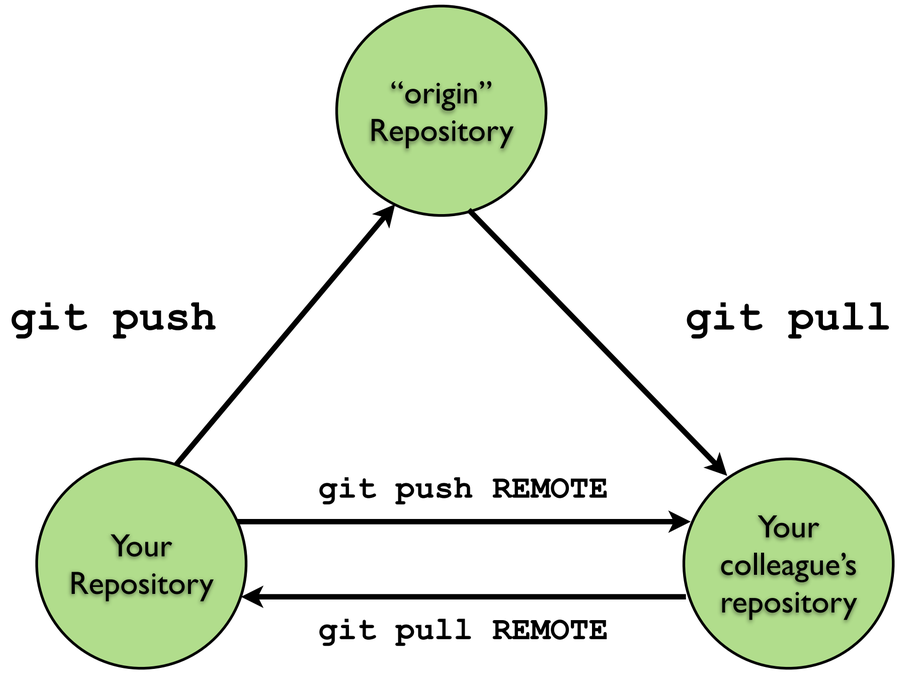


Commits are only shared through the server.

The Git model



The Git model



You can push/pull commits to any remote repository,
there is no difference between server and client.

Git优点

- 速度快
- 十分灵活，尤其对于分支和回退操作
- 无需联网，离线工作
- 团队协作开发功能强大，团队成员之间操作互不影响
- 强大的分支和合并操作

Git仓库(repository)

git init 初始化一个仓库(repo)

git --bare init 建立一个空仓库（没有工作目录，可直接作为server，接受push）

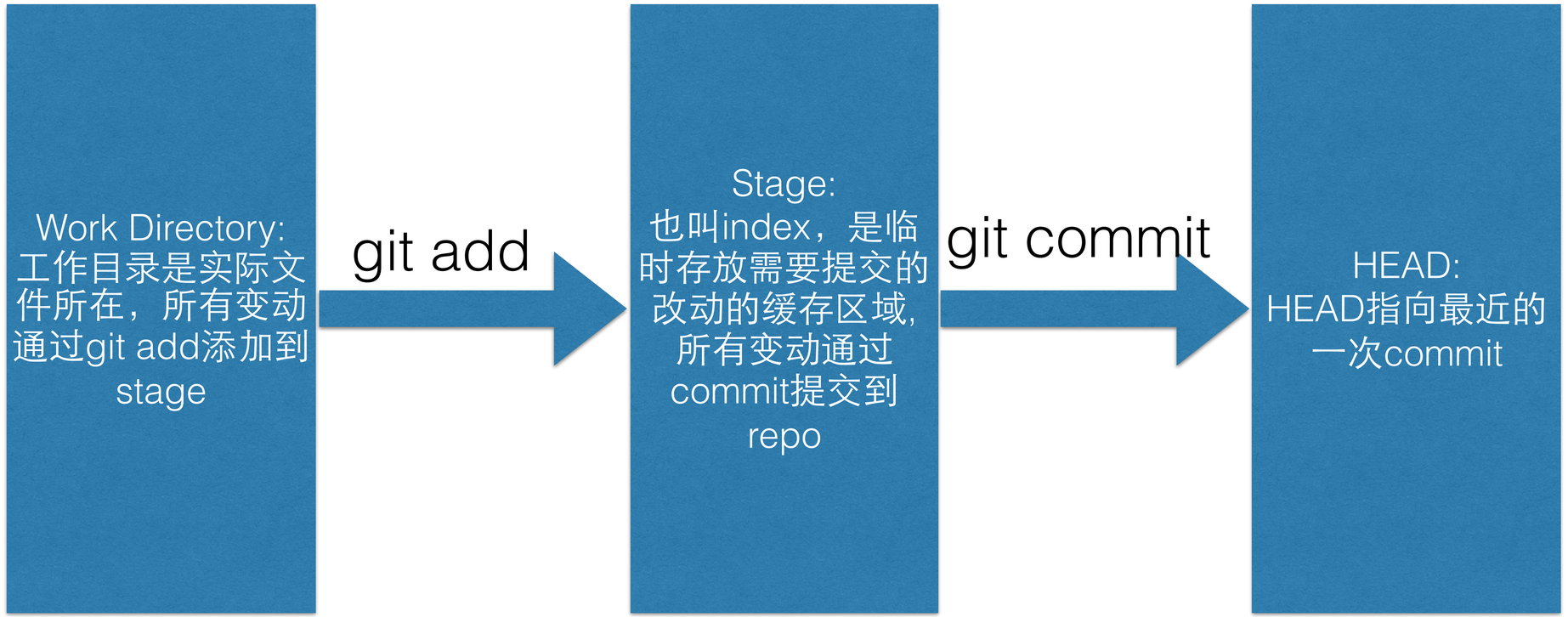
库结构(.git)
hooks：存储钩子的文件夹
logs：存储日志的文件夹
refs：存储指向各个分支的指针（SHA-1标识）文件
objects：存放git对象
config：存放各种设置文档
HEAD：指向当前所在分支的指针文件路径，一般指向refs下的某文件

more about .git:
http://gitready.com/advanced/2009/03/23/whats-inside-your-git-directory.html

Git仓库

- 克隆本地仓库：
  git clone /path/to/repository
- 克隆远程仓库：
  git clone username@host:/path/to/repository
- 远程仓库支持git:// ssh:// https:// 多种协议
  git:// 通过ssh协议速度最快；https最慢，并且每次都需要用户验证
- 添加名为name的远端仓库
  git remote add name url

本地仓库工作流


Work Directory:
工作目录是实际文件所在，所有变动通过git add添加到stage
git add
Stage:
也叫index，是临时存放需要提交的改动的缓存区域,所有变动通过commit提交到repo
git commit
HEAD:
HEAD指向最近的一次commit

How to use Git?

使用Git开发的一般流程

1. git clone REMOTE 在本地克隆一个远程版本库
2. 在本地做出改动
3. git add 相关文件
4. git commit -m ‘message’ 提交改动
5. git push 推送到远程仓库

git commit

- 每次commit都会生成唯一的哈希值用作不同版本的区分
- 每次commit都会保存提交者的name和email，还有commit的说明信息
- 设置name和email可以通过git config命令，或者编辑config文件
- git config -- local 则仅作用于当前repo
- git config --global user.name 'Your Name'
- git config --global user.email you@somedomain.com
- git commit -a -m 自动提交所有文件的改动，但不包括新文件
- git commit -- amend 修改最后一次提交

git reset

- git reset 使 stage 区域恢复到上次提交时的状态，不改变现在的工作目录
- git reset HEAD 最后一次提交 git reset HEAD^上上一次 git reset HEAD^^倒数第三次
- git reset HEAD~0 最后一次提交 git reset HEAD~1上上一次 git reset HEAD~2 倒数第三次
- git reset file 仅恢复某个文件的提交
- git reset --hard commit使 stage 区域恢复到上次提交时的状态，覆盖现在的工作目录
- git reset commit 将当前分支恢复到某次提交，不改变现在的工作目录，在工作目录中所有的改变仍然存在
- $ git reset --hard commit 将当前分支恢复到某次提交，覆盖现在的工作目录，并且删除所有未提交的改变和指定提交之后的所有提交

git revert

- git revert 语法和git reset类似
- git revert 是撤销某一次提交所做的改变，并不删除commit。revert之后提交一个新的commit保存到repo
- git reset 是还原到某次commit，之后的commit会被删除

stash

- git stash 可以保存当前的工作状态，以便做完其他事情之后回到当前状态
  比如想pull 最新代码， 又不想加新commit， 或者另外一种情况，为了fix 一个紧急的bug，先stash, 使返回到自己上一个commit, 改完bug之后再stash pop, 继续原来的工作

- git stash save “message" 保存stash时添加注释

- git stash list 查看当前保存的list

- git stash apply stash@{0} 取出最后一次保存的stash

- git stash pop 从stash的栈里取出最后一次的stash并且从list里去除

- git stash clear 清除stash内的所有内容

多人协作的一般流程

1. git push origin branch-name 推送自己的修改
2. 如果推送失败，则因为远程分支比本地更新，需要先用git pull 合并远端代码
3. 如果合并有冲突，则解决冲突，并在本地提交
4. 没有冲突或者解决掉冲突后，再推送自己的修改 git push origin branch-name
5. 如果git pull提示“no tracking information”，则说明本地分支和远程分支的链接关系没有创建，git branch --set-upstream branch-name origin/branch-name

Branch

- git branch 列出所有分支 当前分支用*标记
- git branch name 建立名为name的分支
- git checkout branch切换到branch分支
- git checkout -b name 创建并切换到name分支
- git branch -d name 删除名为name的分支
- git branch -m oldName newName 重命名分支
- git branch --set-upstream branch origin/branch 链接本地和远端的dev分支(没有链接无法pull)
- git checkout -b branch-name origin/branch-name 在本地创建和远端对应的分支
- git pull name branch 从远端仓库name下的branch分支同步代码
- git push [远端仓库] [本地分支名]:[远端分支名] 把branch分支推送到远端仓库
- git push name:branch 删除远程分支 注意冒号

merge

- git merge 将所有分支的改动合并到当前分支
- git merge branchname 将指定分支合并到当前分支
- 如果有冲突的话则需要用git status和git diff查看并解决掉
  再用git add和git commit提交改动

Tag

- tag多用来标记发布点/版本号，相当于一次快照 snapshot
- git tag 列出当前分支下所有标签
- git tag name 建立名为name的标签 默认为HEAD 也可以指定一个commit
- git tag -a name -m ‘message’ 建立带有说明的标签-a 标签名 -m 说明(推荐)
- git show tag 查看标签信息
- git checkout tag 切换到tag标签
- git tag -d name 删除name标签
- git push remote --tags 把所有标签推送到远端仓库
- git push remote tag 把tag标签推送到远端仓库

- 删除远程标签
  先删除本地标签 然后git push origin :refs/tags/tagname

常用Git命令

- git init 建立新仓库
- git clone /repo/path克隆一个仓库
- git add filename 添加filename到stage git add . 添加所有改动到stage
- git rm 删除文件 git rm -- cached 在git中删除文件(但保留本地文件)
- git mv 移动/重命名文件 类似git rm --cached old; mv old new; git add new
- git commit -m ‘message’ 提交stage的改动
- git commit -am ‘message’ 添加所有改动并提交(=git add . & git commit)
- git status 查看当前状态(包括尚未commit和没有add到stage的改动)
- git diff 查看尚未缓存的改动(git add之前) git diff —cached 查看已缓存的改动(git add的改动)
- git diff HEAD 查看所有没有commit的改动 git diff commit commit 比较两个commit的改动
- git log 查看日志 git log -n 查看最近的n次commit
- git push 将改动推送到远端仓库 git push REMOTE BRANCH 将某分支推送到远端
- git pull 将远端仓库pull到本地
- git checkout 检出版本库 git checkout NAME 切换分支/标签
- git reset 还原改动，删除commit git revert 撤销改动，不删除commit 提交一个新的commit
- git tag 查看标签 git tag name 建立name标签
- git branch 查看分支 git branch name 建立name分支
- git config 设置相关命令 git config -- global 操作全局配置 git config -- local 仅对当前仓库
- git remote 远端仓库相关命令 git remote -v 查看本地链接的远端仓库 git remote show name 查看远端仓库信息
- git help 查看帮助
- git help [command] 查看某个git命令的帮助

Thanks.