**课程概要：**

1. GIT体系概述
2. GIT 核心命令使用
3. GIT 底层原理

一、GIT体系概述

**提问：**

1. 大家公司是用什么工具来管理代码版本？SVN、CVS、GIT
2. GIT和SVN有什么区别呢？

GIT 与 svn 主要区别：

1. 存储方式不一样
2. 使用方式不一样
3. 管理模式不一样

1、存储方式区别

GIT把内容按元数据方式存储类似k/v数据库，而SVN是按文件(新版svn已改成元数据存储)

* + 演示git 存储过程演示

cd .git/objects/df/

git cat-file -p df70460b4b4aece5915caf5c68d12f560a9fe3e4

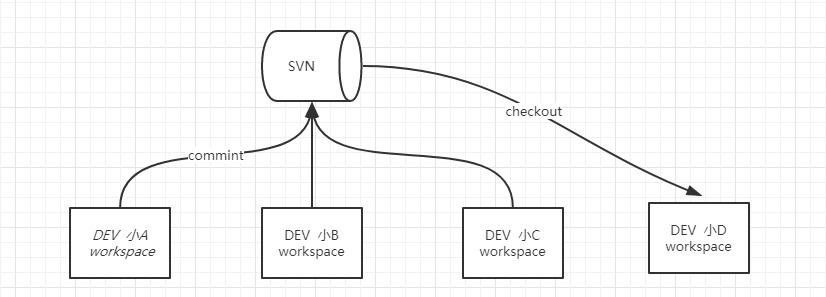
echo 'version1' > text.txt

git hash-object -w text.txt

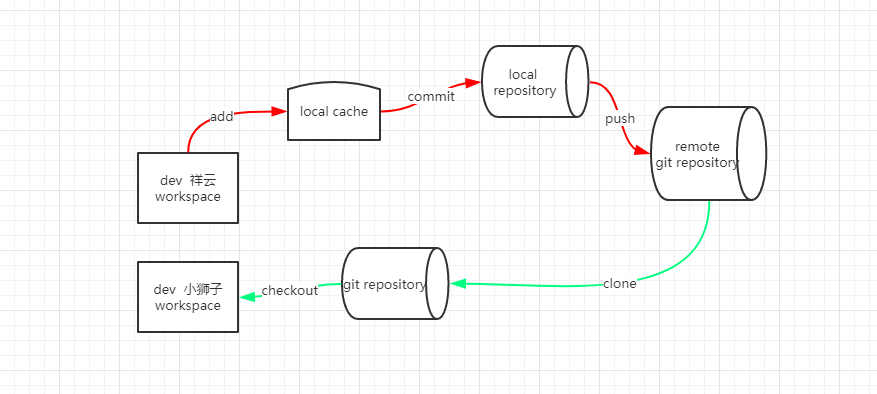
2、使用方式区别

从本地把文件推送远程服务，SVN只需要commint 而GIT需要 add、commint、push 三个步骤

SVN基本使用过程



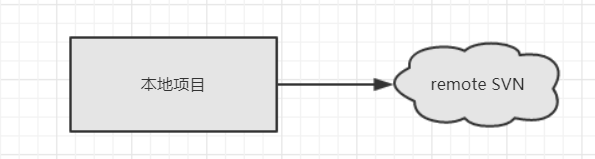
Git基本使用过程



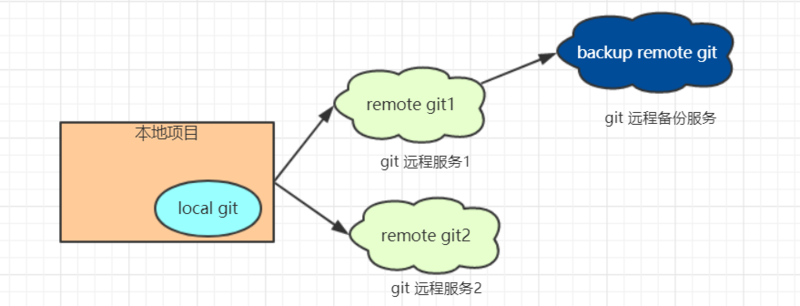
3、版本管理模式区别

git 是一个分布式的版本管理系统，而要SVN是一个远程集中式的管理系统

集中式



分布式



二、GIT 核心命令使用

主要内容:

1. git 客户端安装配置
2. 整体认识GIT的基本使用
3. 分支管理
4. 标签管理
5. 远程仓库配置

1、安装git 客户端安装

官方客户端： [httpsd://git-scm.com/downloads](https://git-scm.com/downloads)

其它客户端：<https://tortoisegit.org/download/>

2、认识GIT的基本使用

1. git 项目创建与克隆
2. 文件提交与推送

完整模拟从项目添加到push 过程

* 创建项目
* 初始化git仓库
* 提交文件
* 远程关联
* push 至远程仓库

**本地初始化GIT 仓库:**

#基于远程仓库克隆至本地

git clone <remote\_url>

#当前目录初始化为git 本地仓库

git init  <directory>

基于mvn 模板创建项目

mvn archetype:generate

本地添加

#添加指定文件至暂存区

git add <fileName>

#添加指定目录至暂存区

git add <directory>

#添加所有

git add -A

#将指定目录及子目录移除出暂存区

git rm --cached target -r

#添加勿略配置文件 .gitignore

本地提交

#提交至本地仓库

git commit file -m '提交评论'

#快捷提交至本地仓库

git commit -am '快添加与提交'

3、分支管理

#查看当前分支

git branch [-avv]

#基于当前分支新建分支

git branch <branch name>

#基于提交新建分支

git branch <branch name> <commit id>

$ git branch -d {dev}

#切换分支

git checkout <branch name>

#合并分支

git merge <merge target>

#解决冲突，如果因冲突导致自动合并失败，此时 status 为mergeing 状态.

#需要手动修改后重新提交（commit）

4、远程仓库管理

#查看远程配置

git remote [-v]

#添加远程地址

git remote add origin http:xxx.xxx

#删除远程地址

git remote remove origin

#上传新分支至远程

git push --set-upstream origin master

#将本地分支与远程建立关联

git branch --track --set-upstream-to=origin/test test

5、tag 管理

#查看当前

git tag

#创建分支

git tag <tag name> <branch name>

#删除分支

git tag -d <tag name>

6、日志管理

#查看当前分支下所有提交日志

git log

#查看当前分支下所有提交日志

git log {branch}

# 单行显示日志

git log --oneline

# 比较两个版本的区别

git log master..experiment

#以图表的方式显示提交合并网络

git log --pretty=format:'%h %s' --graph

三、git 底层原理

* GIT存储对像
* GIT树对像
* GIT提交对像
* GIT引用

1、GIT存储对像(hashMap)

Git 是一个内容寻址文件系统，其核心部分是一个简单的键值对数据库（key-value data store），你可以向数据库中插入任意内容，它会返回一个用于取回该值的hash 键。

# git 键值库中插入数据

echo 'luban is good man' | git hash-object -w --stdin

79362d07cf264f8078b489a47132afbc73f87b9a

#基于键获取指定内容

git cat-file -p 79362d07cf264f8078b489a47132afbc73f87b9a

Git基于该功能 把每个文件的版本中内容都保存在数据库中，当要进行版本回滚的时候就通过其中一个键将期取回并替换。

* 模拟演示git 版写入与回滚过程

# 查找所有的git 对像

  find .git/objects/ -type f

# 写入版本1

echo 'version1' > README.MF; git hash-object -w README.MF;

# 写入版本2

echo 'version2' > README.MF; git hash-object -w README.MF;

# 写入版本3

echo 'version3' > README.MF; git hash-object -w README.MF;

# 回滚指定版本

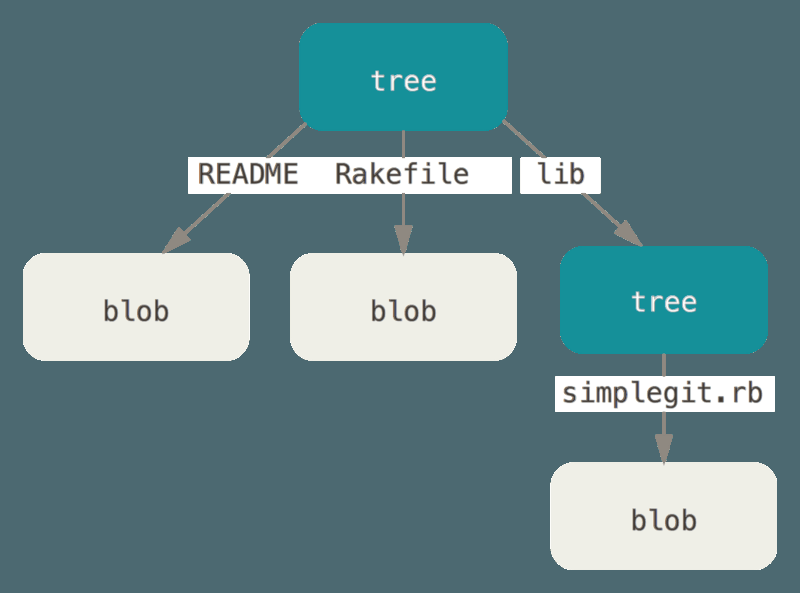
git cat-file -p c11e96db44f7f3bc4c608aa7d7cd9ba4ab25066e > README.MF

所以我们平常用的 git add 其实就是把修改之后的内容 插入到键值库中。当我们执行 git add README.MF 等同于执行了 git hash-object -w README.MF 把文件写到数据库中。

我们解决了存储的问题，但其只能存储内容同并没有存储文件名，如果要进行回滚 怎么知道哪个内容对应哪个文件呢？接下要讲的就是树对象，它解决了文件名存储的问题 。

2、GIT树对像

树对像解决了文件名的问题，它的目的将多个文件名组织在一起，其内包含多个文件名称与其对应的Key和其它树对像的用引用，可以理解成操作系统当中的文件夹，一个文件夹包含多个文件和多个其它文件夹。



每一个分支当中都关联了一个树对像，他存储了当前分支下所有的文件名及对应的 key.

通过以下命令即可查看

#查看分支树

git cat-file -p master^{tree}

3、git提交对象

一次提交即为当前版本的一个快照，该快照就是通过提交对像保存，其存储的内容为：一个顶级树对象、上一次提交的对像啥希、提交者用户名及邮箱、提交时间戳、提交评论。

$ git cat-file -p b2395925b5f1c12bf8cb9602f05fc8d580311836

tree 002adb8152f7cd49f400a0480ef2d4c09b060c07

parent 8be903f5e1046b851117a21cdc3c80bdcaf97570

author tommy <tommy@tuling.com> 1532959457 +0800

committer tommy <tommy@tuling.com> 1532959457 +0800

通过上面的知识，我们可以推测出从修改一个文件到提交的过程总共生成了三个对像：

一个内容对象 ==> 存储了文件内容

一个树对像 ==> 存储了文件名及内容对像的key

一个提交对像 ==> 存储了树对像的key 及提交评论。

* 演示文件提交过程

4、GIT引用

当我们执行 git branch {branchName} 时创建了一个分支，其本质就是在git 基于指定提交创建了一个引用文件，保存在 .git\refs\heads\ 下。

* 演示分支的创建

 git branch dev

cat.git\refs\heads\dev

git 总共 有三种类型的引用：

1. 分支引用
2. 远程分支引用
3. 标签引用

#查询比较两个版本

git log master..experiment

#版本提交历史网络

git log --pretty=format:'%h %s' --graph

#查看分支树

git cat-file -p master^{tree}