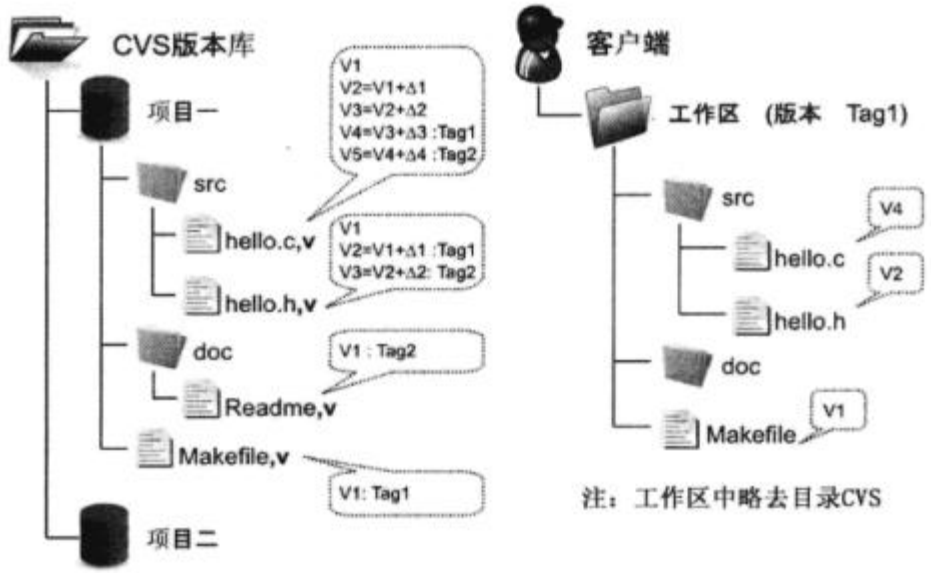
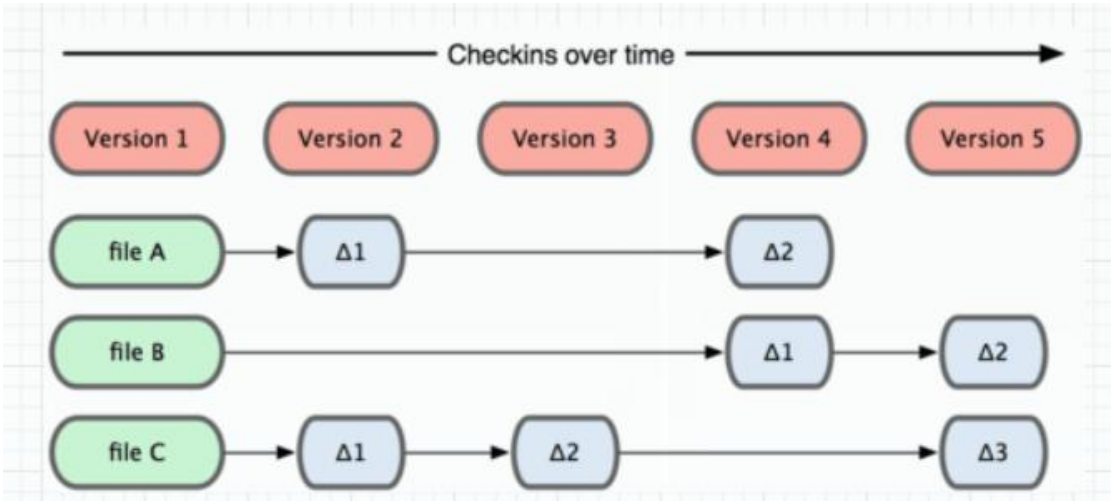
版本号	修改者	明细	时间
V0.1	付勤	创建	2018-05-02

Git 的前世今生

CVS-开启版本控制大爆发

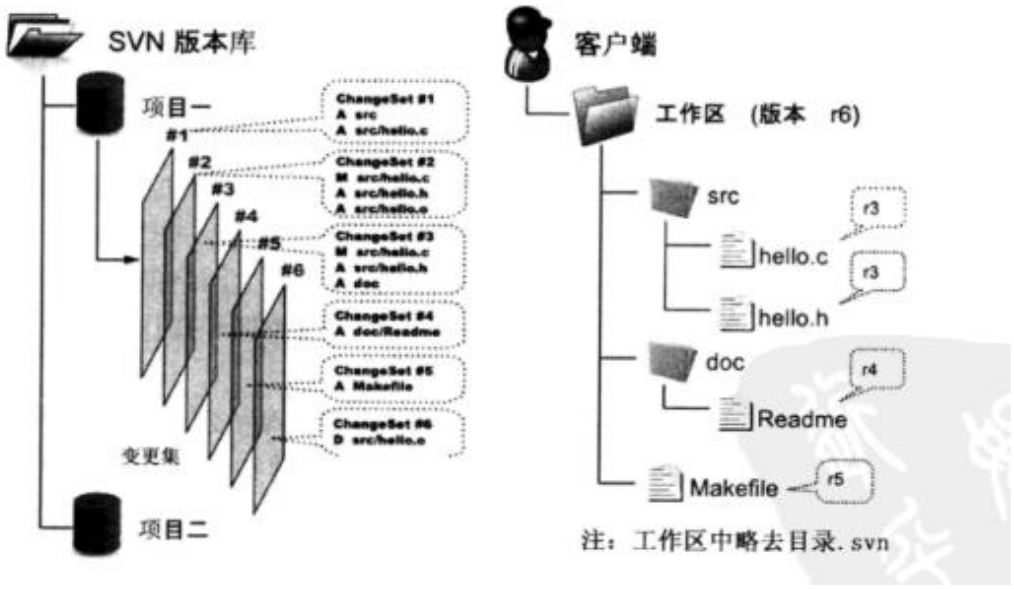
诞生于 1985 年，是由荷兰阿姆斯特丹 VU 大学的 Dick Grune 教授实现的，工作原理如下：





SVN-集中式版本控制集大成者

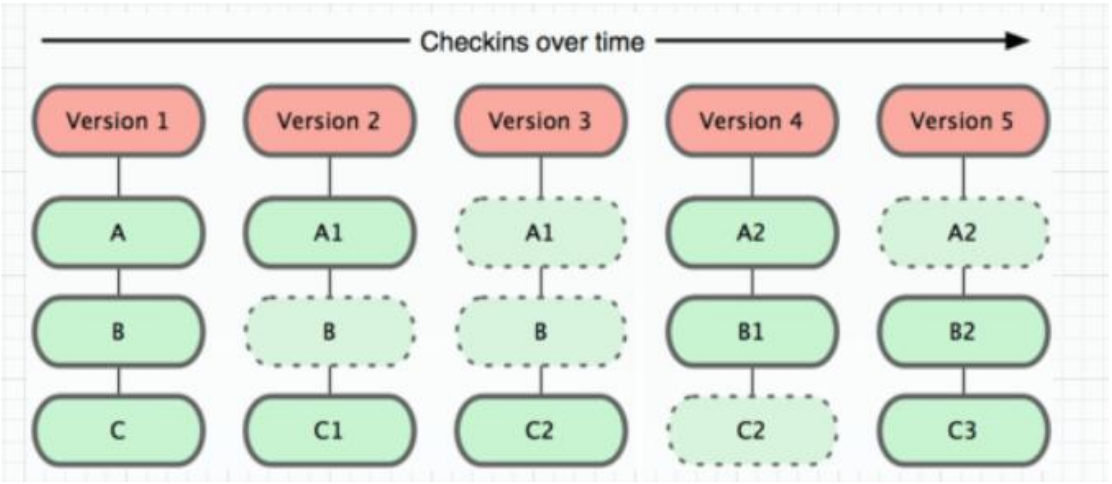
SVN 由 CollabNet 公司于 2000 年资助并开始开发，工作原理如下：



Git-分布式版本控制器

Git 是 Linux 之父 Linux 的第二个伟大作品，Linux 是坚定的 CVS 和 SVN 反对者，在 1991-2002 年间，Linux 宁愿以手工修补文件的方式维护代码，也不使用 CVS 和 SVN。2002 年至 2005 年，Linux 顶着开源社区精英们的口诛笔伐的压力，选择了一个商业版本控制系统 BitKeeper 作为基础，开发出了 Git。

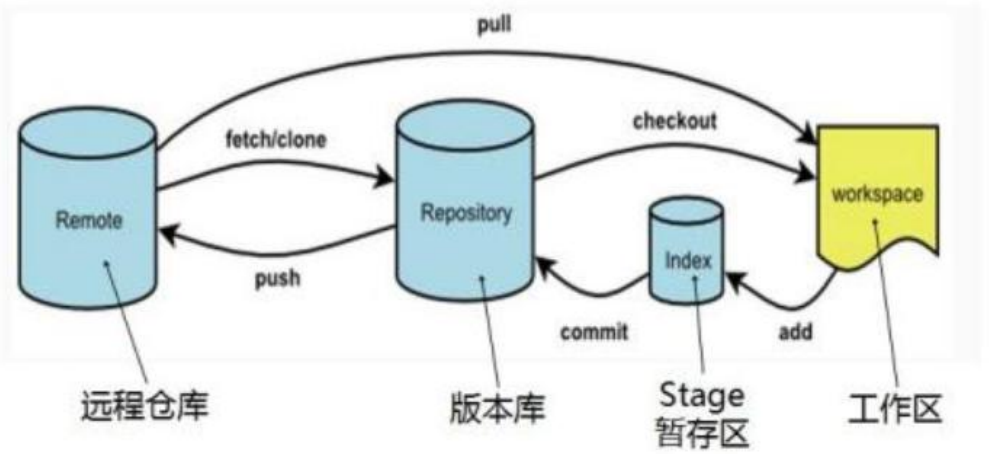
分布式版本控制系统最大的反传统之处在于可以不需要集中式的版本库，每个人都工作在通过克隆建立的本地版本库中，也就是说每个人都拥有一个完整的版本库，查看提交日志、提交、创建里程碑和分支、合并分支、回退等所有操作都直接在本地完成而不需要网络连接，每个人都是本地版本库的主人，不再有谁能提交谁不能提交的限制，加上多样的协同工作模式（版本库间推送，拉回，以及不定文件传送等）让开源项目的参与度有爆发式增长。



Git 构成

构成：本地仓库 和 远程仓库

本地仓库：工作区，暂存区，版本库



git add .　　工作区提交至暂存区

git commit　　暂存区提交至本地仓库

git push　　本地仓库提交至远程仓库

Git 的常用命令

学习网站：*https://git-scm.com/docs*

设置用户名和邮箱

全局设置：

git config --global user.name zhangsan

git config --global user.email zhangsan@china.cn

局部设置：

git config user.name zhangsan

git config user.email zhangsan@china.cn

区别：全局设置作用域是全部 git 项目，C:\Users\Administrator\ .gitconfig

局部设置作用域是某一个 git 项目，../.git/ config

优先级：局部 > 全局

查看配置信息：git config --global –list

git config --list

给命令取别名

好处：能将较长的命令简短化

git config --global alias.<别名> <command>

仓库的初始化

将一个普通的文件夹初始化为一个 git 仓库

git init

This command creates an empty Git repository - basically a `.git` directory with subdirectories for `objects`, `refs/heads`, `refs/tags`, and template files. An initial `HEAD` file that references the HEAD of the master branch is also created.

.git 文件夹解析：*https://blog.csdn.net/mayfla/article/details/78653396*

clone

从远程仓库克隆一个项目到本地
git clone <remote_url>

git 远程通讯支持 Https 和 SSH 两种协议，使用 Https 时每次远程操作（push,pull,fetch 等）都需要输入密码，很不方便，所以一般情况采用 SSH 方式免密远程操作。

（配置 ssh 免密操作）

add/commit

diff

用于比较两次修改的差异

① 比较工作区与暂存区

git diff　　默认比较当前文件目录下的所有差异

git diff <file_path>　　比较某个文件

② 比较暂存区与最新本地版本库（本地库中最近一次 commit 的内容）

git diff --cached [<file_path>...]

③ 比较工作区与最新本地版本库

Git diff HEAD [<file_path>...]

④ 比较工作区与指定版本的差异

git diff <commit_id> [<file_path>...]

⑤ 比较暂存区与指定版本的差异

git diff –cached <ccommit_id> [<file_path>...]

⑥ 比较两个指定版本间的差异

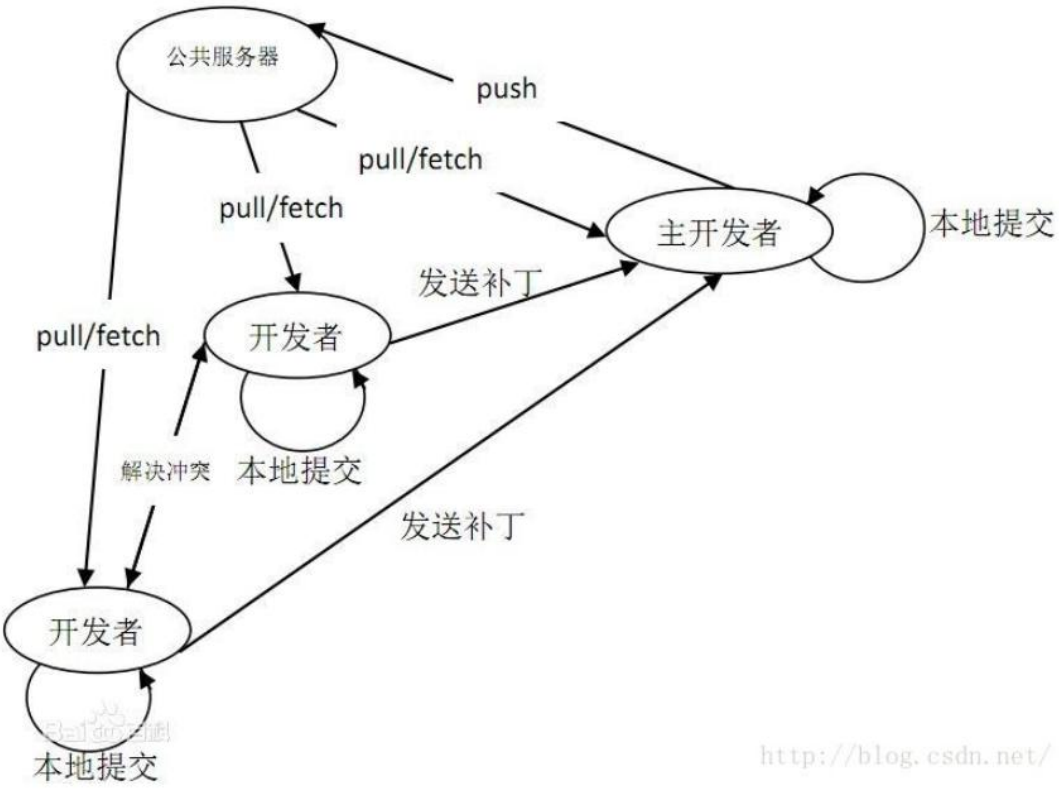
git diff <commit_id1> <commit_id2> [<file_path>...]

⑦ 比较本地分支和本地远程分支的文件的差异

git diff origin/<branch_name> [<file_path>...]

⑧　打补丁

工作模式：



生成的补丁保存在当前目录下

git diff <file_path> > <patch_name>　　将工作区和暂存区的差异打成补丁

git diff --cached <file_path> > <patch_name>　将暂存区和本地仓库的差异打成补丁

git diff --HEAD <file_path> > <patch_name>　将工作区与本地仓库的差异打成补丁

git pull 和 git fetch

两个命令的作用都是将远程的变更信息同步到本地，但是他们有质上的区别。

（模拟演示流程）

总结：git pull = git fetch + git merge

日志

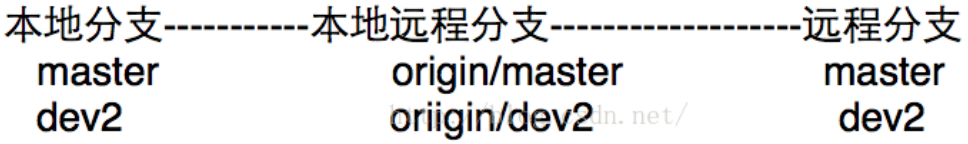
每一次的提交操作都会产生 log 日志，一条 log 包含作者信息，版本号，备注和提交时间等。

git log　　查看日志记录

git log --oneline　　日志以精简的方式展示

分支

独立于其他分支的一个隔离空间，在该空间内完成的所有增删改均不影响其他分支。好处就是实现多个开发人员并行开发。



git branch　　列出本地所有分支，且当前分支高亮

git branch -r　查看所有远程分支

git branch -a　查看本地分支和远程分支

git checkout <branch_name>　　切换到<branch_name>分支

git checkout -b <branch_name>　切换分支，如果<branch_name>不存在，则新建

git checkout -B <branch_name>　切换分支，如果<branch_name>不存在，则新建；如果已存在，则覆盖。

添加远程地址

完成 git init 后，项目只限于在本地进行各项操作，当需要把本地的改动上传至远程仓库时，就需要先添加远程仓库地址。

git remote　　查看当前

git remote add <remote_name> <remote_url>　　添加远程地址
remote_name 一般设为 origin

git remote rm <remote_name>　　删除远程地址

里程碑 - tag

在 CVS 中已经存在 tag 的概念，亦即“里程碑”

应用场景：根据需求和上线目的不同，项目一般会分成几个关键阶段，在每个阶段开发完成后就可打上一个 tag。如果新上线的版本中，由于外部原因导致早期的功能模块不可用，此时可以基于某个稳定的里程碑新建一个分支，在新分支上完成打包和重新部署，不影响 master 分支上的新功能代码，发布完成后为了便于分支管理，一般会将新建分支删除。

git branch <分支名称> <tag>　　基于某个 tag 新建一个分支

常用命令：

git tag -a <tag_alias> -m '<tag_comment>'　　在当前版本基础上，创建一个新的 tag，“-a”给 tag 取别名，类似“v0.1”；“-m”给 tag 添加备注。

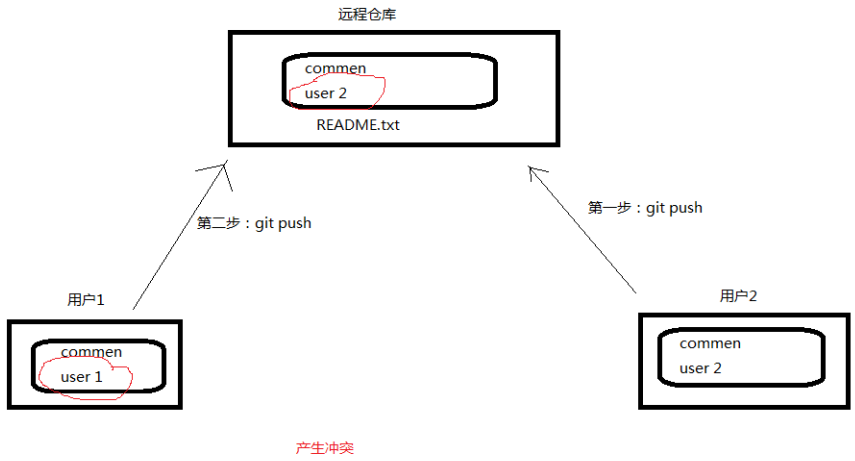
（tag 存放路径展示）

git tag　　查看存在的所有 tag

git show <tag_alias>　　查看某个 tag 的具体信息

git push origin <tag_alias>　　将某个 tag 上传远程仓库，与其他用户共享

解决冲突



当我们执行完毕 git add, git commit，正在 git push 时，gitbash 报错

```
$ git push -u origin master
To github.com:pc-fuqq/mongo_example.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'git@github.com:pc-fuqq/mongo_example.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

完成以下几步完成解决冲突：

第一步：git pull origin <branch_name> 从远程将最新的变动拉下来

第二步：git status 查看有哪些文件产生了冲突

第三步：针对冲突的文件进行 Resolve。vim 进入冲突文件，会看到
<<<<<<<, =======, 和 >>>>>>>的标记，其中
<<<<<<<, =======之间的内容是本地的改动，

=======, >>>>>>>之间的内容为远程的改动，

根据具体情况取舍相应变动，并且将上面三种符号删除。

如果存在多处冲突，则上面的分隔符也会存在多组。

第四步：重新执行 git add, git commit 和 git push 操作。

（实际显示，和结合 idea 的 Git 插件显示）

版本回退

即放弃当前版本，撤回到某个历史版本。

应用场景：当在开发一个不确定模块时，如果后面决定放弃该模块转向另一个方向开发，此时的代码已经包含了很多无用代码，即垃圾代码，可以采用回退到某个历史版本，然后基于这个纯净的版本再开发。

第一步：确定要回退到某个历史版本，或者 tag

git log –oneline　　查看所有的历史提交记录

git tag　　查看所有 tag 信息

第二步：git reset--hard <版本号/tag>　　实现版本回退

第三步：git push -f -u origin master　　将回退的历史版本推送至远程仓库，并且强制覆盖（-f 强制覆盖，如不加此参数，需要解决冲突）