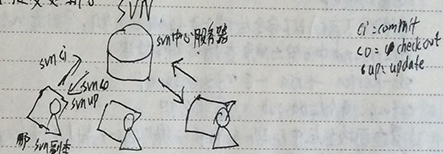
版本控制

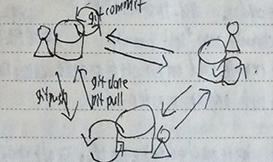
1. 简介：

版本控制的目的是实现开发团队并行开发、提高开发效率。其目的在于对软件开发进程中文件或目录的发展过程提供有效的追踪手段，保证在需要时可回到旧的版本，避免文件的丢失、修改的丢失和相互覆盖，通过对版本库的访问控制避免未经授权的访问和修改，达到有效保护企业软件资产和知识产权的目的。下面将介绍2种主流版本控制工具：SVN和Gitlab

1. svn和git的区别
2. svn是集中化的版本控制系统，只有单一的集中管理的服务器，保存所有文件的修订版本，而协同工作的人员都通过客户端连到这台服务器，取出最新的文件或提交更新。



1. git是分布式的版本控制系统，每一个终端都是一个仓库，客户端不仅提取新版本的文件快照，而是把原始的代码仓库完整地镜像下来。每一次的提取操作，实际上都是对一次代码仓库的完整备份。



1. 使用GitLab进行版本控制

简介：说到git工具，首先想到的肯定是github。github主要适用于做开源项目的版本控制，而gitlab是本地服务器，使用和github差不多，而且可以创建私有项目，适用于企业产品开发用

1. gitlab创建工程
2. 关于namespace：可选择个人或组，这会影响项目最终访问路径，如suning创建工程suningTest，则对应工作路径:https://domain.com/suning/suningTest
3. Visibility Level：权限分三种，private(私有，只有自己或组内成员能访问、internal(所有登陆的用户)、public(公开的，所有人都可以访问)
4. 添加ssh key

git仓库之间的代码传输协议主要是用ssh协议。但是本地服务器gitlab在搭建时的git用户一般是没密码的，因此直接ssh是不能登录的，就需要使用ssh－keygen上传公钥，使用非对称加密传输。

1. 生成ssh key

在终端敲入以下命令，生成一对私钥和公钥，分别在~/.ssh/id\_rsa和~/.ssh/id\_rsa.pub中。第二步查看公钥字符串

ssh-keygen -t rsa –C “$your\_email”

cat ~/.ssh/id\_rsa.pub//显示公钥

1. 在面板上点击profile Settings->SSH Keys ->Add SSH Keys，然后把公钥粘贴保存在gitlab账户中
2. 初始化上传代码
3. 进入工程目录：cd $project\_root
4. 初始化git仓库：git init
5. 添加文件到仓库：git add.
6. 提交代码到本地仓库：git commit –m “init commit”
7. 链接到git server：git remote add origin [git@example.com:namespace/projectname.git](mailto:git@example.com:namespace/projectname.git)
8. push代码至服务器：git push origin master
9. 克隆代码到本地

在svn中，我们都叫checkout，把代码checkout到本地。而git中我们 叫克隆，克隆会把整个仓库都拉到本地。如：把刚才的工程再克隆到本地

git clone [git@example.com:namespace/projectname.git](mailto:git@example.com:namespace/projectname.git)

1. 设置gitignore
2. 有一些文件或文件夹是我们不希望被版本控制的，比如.DS\_Store build\xcuserdata thumbs.db等，git提供了一种忽略方案
3. 在项目根目录下创建.gitignore文件，然后把需要忽略文件或者文件夹写进去。这样就可以忽略这些文件受版本控制了。

svn也提供了类似方案，svn也可以设置全局忽略。svn此配置位于~/.subversion/config中global-ignores的值

通过设置ignore，可以实现git与svn双管理，即svn忽略.git文件夹，自在gitignore忽略 .svn文件夹

1. git配置用户信息

git config --global user.name “”

git config --global user.email “”

查看配置信息：git config --list

1. git提交命令：git commit –m “initial project version”
2. git查看当前文件状态：git status
3. vim编辑文件：vim文件名，I(输入状态)，esc(命令状态)，wq(写入文件并退出)
4. git status 在changes not staged for commit下的文件，需git add进缓存区
5. git提交更新commit是一次对项目做的快照，以后可以回到这个状态，或进行比较
6. 跳过暂存区直接提交 git commit -a –m “” //把文件暂存并提交
7. 移除文件: git rm文件，文件会跑到未跟踪清单，只要把文件从目录删掉，就不会出现在未跟踪清单中。如果已修改后并放到暂存区域，删除加-f，以免误删文件后丢失修改内容
8. 移动文件(重命名)：git mv 文件名 新文件名 相当于mv文件名 新文件名 git rm旧文件名 git add新文件名
9. 查看git分支：git branch、git branch –a
10. 提取远程分支：git checkout –b ver1.1 origin/ver1.1
11. 查看提交历史：git log 常用-p展开显示每次提交内容差异；-2则仅显示最近两次更新；git log –p -2，此外--prety 指定使用完全不同于默认格式的提交历史如：git log --prety=oneline让信息只在一行显示
12. 撤销提交：当commit之后发现想修改提交说明，或想修改点东西再提交，则可以先修改完成后，git commit - -amend，会覆盖上次提交
13. 取消已经暂存的文件：git reset HEAD 文件名
14. 提交到远程仓库：
15. git commit -a -m ””
16. git fetch origin ver1.1
17. git merge origin/ver1.1
18. git push origin ver1.1
19. git log origin/ver1.1
20. 当merge失败，冲突解决不了，想回到merge前重新merge，使用

git reset - -hard 会滚。这样可能会丢失之前的修改内容

git reset - -hard HEAD ~2//回滚至前三次提交

git reset - -soft 回滚，所有修改退回暂存区

1. 已经push到远程分支，想撤销：使用git revert + commit号，这一撤销会作为新的修改存储起来，当与服务器同步时，不会产生副作用：如删掉的文件又回来了
2. 删除本地分支：git branch -d 分支名
3. 取消文件版本控制：git checkout - - 文件名
4. 使用svn进行版本控制
5. 在code.taobao.org 服务器，新建私人项目，获得项目地址
6. xcode新建项目时不要勾选git responsitory
7. xcode的preference的accounts创建respository，type改为subversion，address为项目地址，并输入svn账号和密码

打开工程，checkout 一份项目到本地，把工程目录移至svn目录下后执行(4)第2)步

1. 1)命令行checkout:checkout path - -username “” - -password “”
2. 然后把代码拷至svn目录，cd目录，svn add代码目录，svn update，svn commit －m“”
3. 删除文件：svn delete 文件或目录后svn commit -m “”提交修改
4. 更新文件：svn update:将当前目录及子目录文件都更新到最新

svn update –r 200 test.cpp 将test.cpp文件还原到200版本

svn update 文件名:当前文件更新到最新

1. 查看文件或目录状态：svn status –v 目录路径/名
2. 查看日志：svn log 文件名
3. 查看文件详细信息：svn info 文件名
4. svn帮助：svn help 显示所有命令选项

svn help ci：指定命令说明

1. 创建纳入版本控制的新目录：svn mkdir 目录名，记得回根目录update一下
2. 恢复本地文件修改：svn revert 文件名，回解决冲突情况。不会恢复呗删除的目录

svn revert - -recurive .：恢复整个目录文件，.为当前目录

1. 解决冲突：当svn update时可能出现冲突，产生三个新文件。可以解决了冲突后删除其它文件后提交，也可以解决后执行svn resolved 本地目录全路径，这个命令删除冲突文件，还修正了一些纪录在工作拷贝管理区域的纪录数据
2. 新建一个分支：svn copy branchA branchB –m “make branchB”
3. 合并内容到分支merge：svn merge branchA branchB ／/对branchA的修改合并到branchB分支