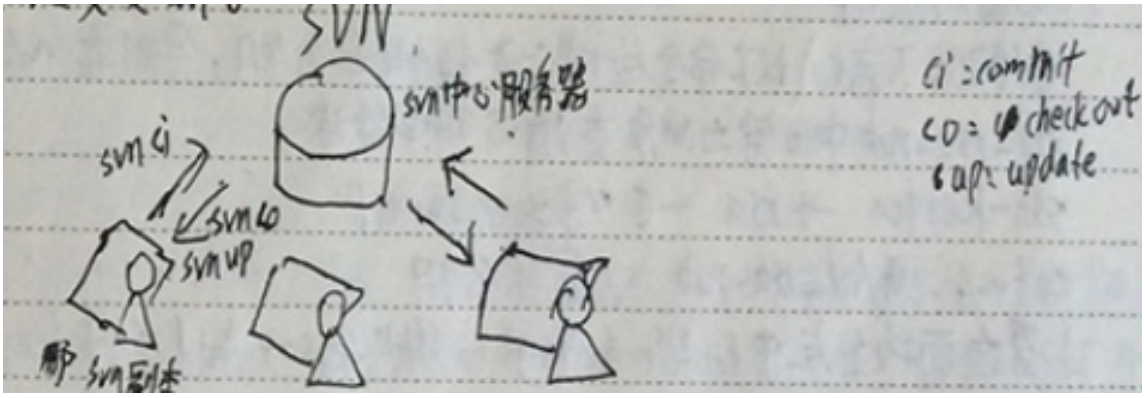
版本控制

1. 简述：

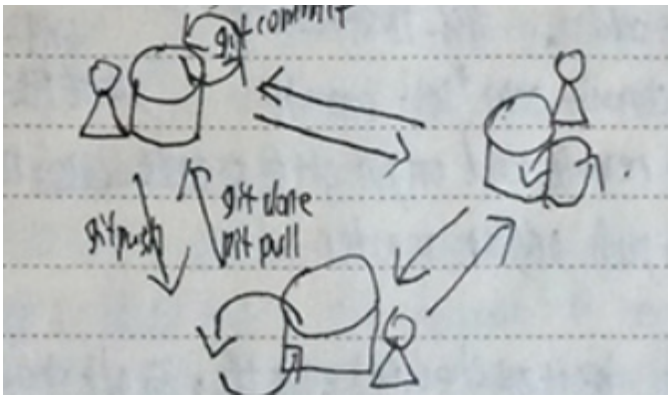
   版本控制的目的是实现开发团队并行开发、提高开发效率。其目的在于对软件 开发进程中文件或目录的发展过程提供有效的追踪手段，保证在需要时可回到旧的 版本，避免文件的丢失、修改的丢失和相互覆盖，通过对版本库的访问控制避免未 经授权的访问和修改，达到有效保护企业软件资产和知识产权的目的。下面将介绍 2 种主流版本控制工具:SVN 和 Gitlab

2. git 和 svn 的区别

   (1) 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。



   (2) 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。



3. 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

1) 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

2) 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)

(2) 添加 ssh key

1) 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。

2) 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

3) 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

(3) 初始化上传代码

1) 进入工程目录:cd $project_root

2) 初始化 git 仓库:git init

3) 添加文件到仓库:git add.

4) 提交代码到本地仓库:git commit -m “init commit”

5) 链接到 git server:git remote add origin

git@example.com:namespace/projectname.git

6) push代码至服务器:git push origin master

(4) 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 [/目标目录/本地显示的项目目录名]

(5) 设置 gitignore

1) 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

2) 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

(6) 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

(7) git管理相关命令

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 (git add .表示全部add进暂存区)

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unstage区直接提交 git commit -a -m “” //把unstage区文件放到暂存区并提交，相当于add+commit

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:

git commit -a -m ”“

git fetch origin ver1.1

git merge origin/ver1.1

git push origin ver1.1

git log origin/ver1.1

15) 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

16) 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

17) 删除本地分支:git branch -d 分支名

18) 取消文件版本控制:git checkout - - 文件名

19) 远程分支重命名，原理：1.删除远程分支 2.重命名本地分支 3.push本地分支到git服务器

(1) git push :原分支名

(2) git branch -m 原分支名 新分支名

(3) git push origin 新分支名

20) 从远程仓库里拉取一条本地不存在的分支

git checkout -b 本地分支名 origin/远程分支名

21) 合并分支某个commit到目的分支

git checkout master

git cherry-pick 62ecb3

22) 本地fetch的远程分支和github上的远程分支保持同步(自动删除github上不存在的分支)

git pull -p

23) github误删远程分支

git push origin 目标分支名

24) git切换本地分支

git checkout ver1.1

25) git新建本地分支

git checkout -b ver1.1

26) 分支间合并法

比如：当前分支是ver1.1,已经在开发ver1.2了(review1.1中开发)，发现之前ver1.1存在bug。。。然后，你想把review1.1的bug修复全部同步到ver1.1分支的话，可以使用

git fetch origin review1.1

git merge origin/review1.1

27) 单行查看日志

git log origin/ver1.1 -oneline

28) 查看版本操作记录

git reflog

4. 使用 svn 进行版本控制

(1)在 code.taobao.org 服务器，新建私人项目，获得项目地址

(2)xcode 新建项目时不要勾选 git responsitory

(3)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

(4)1)命令行checkout:checkout path - -username “” - -password “”

3) 然后把代码拷至 svn 目录,cd 目录,svn add 代码目录,svn update，svn commit -m“”

(5)删除文件:svn delete 文件或目录后 svn commit -m “”提交修改

(6)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

(7)查看文件或目录状态:svn status -v 目录路径/名

(8)查看日志:svn log 文件名

(9)查看文件详细信息:svn info 文件名

(10) svn帮助:svnhelp显示所有命令选项

svn help ci:指定命令说明

(11) 创建纳入版本控制的新目录:svnmkdir目录名，记得回根目录update一下

(12) 恢复本地文件修改:svnrevert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

(13) 解决冲突:当svnupdate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

(14) 新建一个分支:svn copy branchA branchB -m “make branchB”

(15) 合并内容到分支merge:svnmergebranchAbranchB//对branchA的修改合并到 branchB 分支

(16) 解决冲突文件:将<<<===>>>相关代码整理好后resolve文件后更新并提交

5. git版本回滚

(1) git checkout commandid --文件路径文件名

将某个文件的修改撤销到某个版本，默认commandid为head

当commandid=head的时候可以省略，作用，取消这个文件当前版本的所有修改

(2) git reset --hard

```
cwndeiMac:Local_RollBackComandsTest ios31$ git log dev1.0 --oneline
1a3e599 测试提交2
5a4c9a4 添加到暂存区并提交
69956d5 test项目初始化


cwndeiMac:Local_RollBackComandsTest ios31$ git reset 5a4c9a4 --hard
HEAD is now at 5a4c9a4 添加到暂存区并提交


cwndeiMac:Local_RollBackComandsTest ios31$ git log dev1.0 --oneline
5a4c9a4 添加到暂存区并提交
69956d5 test项目初始化
```

这种方式直接丢失修改，如果回滚错了，要撤销。先找到commandid：1a3e599,这时候git log是看不到了，不过可以通过如下命令找到

```
cwndeiMac:Local_RollBackComandsTest ios31$ git reflog
5a4c9a4 HEAD@{0}: reset: moving to 5a4c9a4
1a3e599 HEAD@{1}: commit: 测试提交2
5a4c9a4 HEAD@{2}: commit: 添加到暂存区并提交
69956d5 HEAD@{3}: checkout: moving from master to dev1.0
69956d5 HEAD@{4}: checkout: moving from dev1.0 to master
69956d5 HEAD@{5}: checkout: moving from dev1.0 to dev1.0
69956d5 HEAD@{6}: checkout: moving from master to dev1.0
69956d5 HEAD@{7}: commit (initial): test项目初始化
```

然后，再次reset id --hard，强行回到原来版本

```
git reset head —hard
```

可以看到当前所在版本

(3) git reset --soft

```
cwndeiMac:Local_RollBackComandsTest ios31$ git reset 5a4c9a4 --soft
cwndeiMac:Local_RollBackComandsTest ios31$ git status
On branch dev1.0
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   JKDB_Test.xcworkspace/xcuserdata/ios31.xcuserdatad/UserInterfaceState.xcuserstate
        modified:   JKDB_Test/ViewController.m
```

这种方式修改会退回暂存（回滚前文件的修改不会丢失），如果重新commit的话，会生成新的版本id

```
cwndeiMac:Local_RollBackComandsTest ios31$ git commit -a -m "提交2"
[dev1.0 d29dc30] 提交2
 2 files changed, 1 insertion(+)
 rewrite JKDB_Test.xcworkspace/xcuserdata/ios31.xcuserdatad/UserInterfaceState.xcuserstate (70%)
cwndeiMac:Local_RollBackComandsTest ios31$ git status
On branch dev1.0
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .DS_Store

nothing added to commit but untracked files present (use "git add" to track)
cwndeiMac:Local_RollBackComandsTest ios31$ git log dev1.0 --oneline
d29dc30 提交2
5a4c9a4 添加到暂存区并提交
69956d5 test项目初始化
```

如果还想回退到原来版本，同样可以再次执行reset 1a3e599 --soft

```
cwndeiMac:Local_RollBackComandsTest ios31$ git reset 1a3e599 --soft
cwndeiMac:Local_RollBackComandsTest ios31$ git status
On branch dev1.0
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .DS_Store

nothing added to commit but untracked files present (use "git add" to track)
cwndeiMac:Local_RollBackComandsTest ios31$ git log dev1.0 --oneline
1a3e599 测试提交2
5a4c9a4 添加到暂存区并提交
69956d5 test项目初始化
```

(4) git commit --amend

当你提交了之后想再修改点东西，重新提交可以使用此命令。

和git reset --soft的区别是，不能回退到其他版本，而是回退当前版本。再次提交时--reset生成新的id提交，而--amend会生成新的提交id并覆盖之前的提交

```
cwndeiMac:Local_RollBackComandsTest ios31$ git log dev1.0 --oneline
9b761b1 测试 amend
1a3e599 测试提交2
5a4c9a4 添加到暂存区并提交
69956d5 test项目初始化
cwndeiMac:Local_RollBackComandsTest ios31$ git status
On branch dev1.0
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   JKDB_Test.xcworkspace/xcuserdata/ios31.xcuserdatad/UserInterfaceState.xcuserstate
        modified:   JKDB_Test/ViewController.m

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

        .DS_Store

no changes added to commit (use "git add" and/or "git commit -a")
cwndeiMac:Local_RollBackComandsTest ios31$ git commit --amend -am "编辑后覆盖上次提交"
[dev1.0 ed38306] 编辑后覆盖上次提交
 Date: Thu Mar 8 16:20:01 2018 +0800
 2 files changed, 2 insertions(+)
cwndeiMac:Local_RollBackComandsTest ios31$ git log dev1.0 --oneline
ed38306 编辑后覆盖上次提交
1a3e599 测试提交2
5a4c9a4 添加到暂存区并提交
69956d5 test项目初始化
```

(5) git revert

和git reset - -soft相比，会先pull远程分支，确保没问题了，再回退版本，把远程分支的修改拉下来。然后作为进一步的编辑，重新提交，作为新的提交，不容易冲突。

```
cwndeiMac:Local_RollBackComandsTest ios31$ git log dev1.0 --oneline
050023e 修复冲突
674a3ed 修复冲突后提交
ed38306 编辑后覆盖上次提交
1a3e599 测试提交2
5a4c9a4 添加到暂存区并提交
69956d5 test项目初始化
cwndeiMac:Local_RollBackComandsTest ios31$ git log origin/dev1.0 --oneline
ed38306 编辑后覆盖上次提交
1a3e599 测试提交2
5a4c9a4 添加到暂存区并提交
69956d5 test项目初始化
```

两次尝试revert到599，最终都是新的提交。Reset - -soft的，中间的所有提交如：306都会被抹掉！！！

6. 代码暂存

需求：git旧版本，切换分支必须保证项目干净，没有修改未提交的。但是经常我们都只修改了一半，还不想直接做提交。可以通过如下命令暂存代码。下一次切回来的时候，查看暂存的代码并恢复即可

```
cwndeiMac:Local_RollBackComandsTest ios31$ git status
On branch dev1.0
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   JKDB_Test.xcworkspace/xcuserdata/ios31.xcuserdatad/UserInterfaceState.xcuserstate
        modified:   JKDB_Test/ViewController.m

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

        .DS_Store

no changes added to commit (use "git add" and/or "git commit -a")
cwndeiMac:Local_RollBackComandsTest ios31$ git checkout master
error: Your local changes to the following files would be overwritten by checkout:
        JKDB_Test.xcworkspace/xcuserdata/ios31.xcuserdatad/UserInterfaceState.xcuserstate
        JKDB_Test/ViewController.m
Please commit your changes or stash them before you switch branches.
Aborting
```

git stash //暂存

git stash list//显示暂存状态

git stash pop//恢复