## 1：git和svn的区别

**git介绍：**

Git 是一个开源的分布式版本控制系统，用于敏捷高效的处理任何项目的版本问题。

Git 是 Linus Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。

Git 与常用的版本控制工具 CVS,SVN 等不同，它采用了分布式版本库的方式，不必服务器端软件支持。

**关于项目版本管理：**

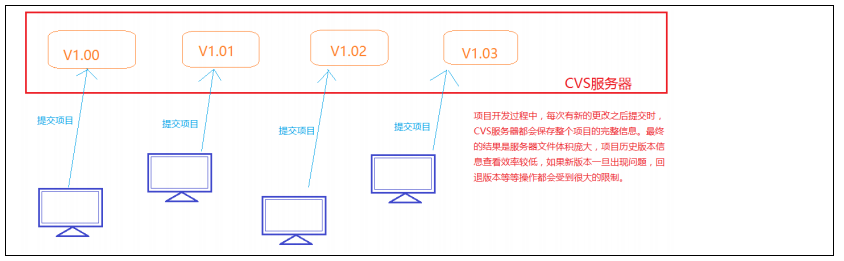
项目在开发的过程中，经常会出现多人分工协作进行项目分发并开发整合的过程，所以项目在刚开始实行的时候经常会出现一些协作开发的同步的问题，同时存在项目整体进度的控制和管理的问题，所以在程序开发行业衍生出来了版本管理工具。

版本管理工具，首先是一个内容管理工具，可以将项目的内容信息存放在版本管理服务器上方便项目组人员进行访问和查询修改。

版本管理具有里程碑意义的主要有三个阶段：CVS 阶段 SVN 阶段 Git 阶段

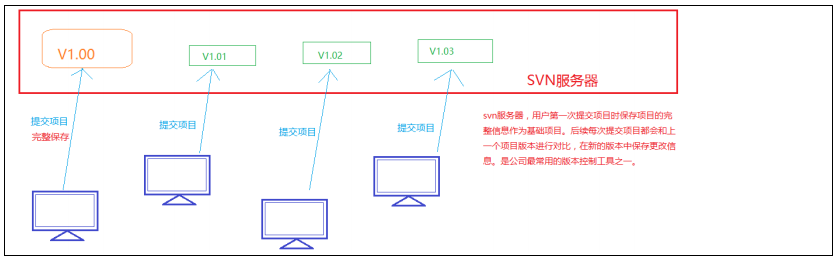
**CVS 阶段：**

项目搭建开发过程中，每次提交项目都会将整个项目提交到服务器进行保存，服务器存储着项目的 N 个备份，开发过程中的协作效率较低，同时也出现了各种传输的问题，所以慢慢淡出了行业。



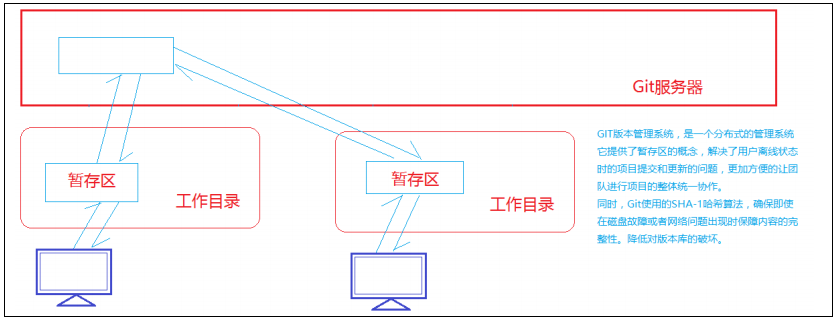
**SVN阶段：**

考虑到 CVS 的缺陷，开发人员研发出专门针对项目版本控制的软件 Subversion （简称 SVN）， SVN 同样也是搭建服务器，让项目组成员将数据存储在服务器上，但是每次改动并提交的时候， SVN 服务器并不重新保存整个项目的完整信息，而是和原来的项目进行对比，只保存改动的信息。这样就在很大的程度上对于项目版本服务器、项目协作效率有了显著的提升。所以至今为止，有很多公司依然选用 SVN 作为公司内部项目协作的版本控制软件。



**Git阶段：**

前面的 CVS 和 SVN 都是基于一个服务器的，如果脱离服务器，项目的版本保存就没有了任何意义，Git 恰恰处理了这样的问题，Git 是一个分布式的版本控制系统，在 Git 中即使用户离线，也能进行项目的提交和更新操作，等到下次连线服务器时进行整体的同步操作。



**Git与SVN比较：**

git 是分布式的，svn 不是。

git 按照元数据的方式存储内容，svn 是按照文件的形式存储

git 和 svn 中的分支不同。

git 没有全局版本号，svn 有。

git 内容的完整性优于 svn。

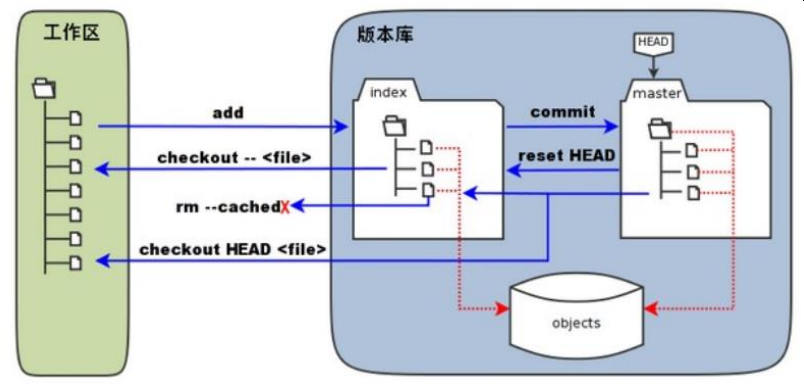
**工作区、暂存区及版本库的概念：**

工作区：个人 PC 中能看到的文件目录结构

暂存区： stage/index，一般存放在.git/index 中，所以 git 中的暂存区也会称为索引。

版本库：工作区中的.git 隐藏文件，不算是工作区，而是 Git 的版本库。

图中左侧为工作区，右侧为版本库。在版本库中标记为 "index" 的区域是暂存区（stage, index），标记为 "master" 的是 master 分支所代表的目录树。



图中我们可以看出此时 "HEAD" 实际是指向 master 分支的一个"游标"。所以图示的命令中出现 HEAD 的地方可以用 master 来替换。

图中的 objects 标识的区域为 Git 的对象库，实际位于 ".git/objects" 目录下，里面包含了创建的各种对象及内容。

当对工作区修改（或新增）的文件执行"git add"命令时，暂存区的目录树被更新，同时工作区修改（或新增）的文件内容被写入到对象库中的一个新的对象中，而该对象的ID被记录在暂存区的文件索引中。

当执行提交操作（git commit）时，暂存区的目录树写到版本库（对象库）中，master 分支会做相应的更新。即master指向的目录树就是提交时暂存区的目录树。

当执行 "git reset HEAD" 命令时，暂存区的目录树会被重写，被 master 分支指向的目录树所替换，但是工作区不受影响。

当执行 "git rm --cached <file>" 命令时，会直接从暂存区删除文件，工作区则不做出改变。

当执行 "git checkout ." 或者 "git checkout -- <file>" 命令时，会用暂存区全部或指定的文件替换工作区的文件。这个操作很危险，会清除工作区中未添加到暂存区的改动。

当执行 "git checkout HEAD ." 或者 "git checkout HEAD <file>" 命令时，会用 HEAD 指向的 master 分支中的全部或者部分文件替换暂存区和以及工作区中的文件。这个命令也是极具危险性的，因为不但会清除工作区中未提交的改动，也会清除暂存区中未提交的改动。

## ：具体使用

<https://git.oschina.net/> 如果想要自己搭建git服务器 gitlab

1-ws 在版本控制设置 路径 C:\Program Files\Git\cmd\git.exe

hbuild中需要装一个插件egit

2-第一次进公司需要下载项目

3-克隆clone <https://git.oschina.net/xiang/elm.git>

4-修改一下代码，提交本地 commit

5- 每提交给服务器之前都需要pull（拉）一下，把最新的代码拉下来

1. 提交给服务器 push

Linux 命令

ls 查看文件夹下面的内容

Pwd 查看自己所在的路径

Cd 进入某个文件夹

Clear 清空命令

Touch 创建文件

mkdir test11111 创建文件夹

Rm -rf

用命令行clone 一份代码 git clone <https://git.oschina.net/xiang/elm.git>

如果commit 不了的话，就去跟踪 你修改的文件 git add 文件名 git add . 跟踪所有文件

提交到本地 git commit -m '222222'

Pull 拉取 git pull

提交到服务器 git push -u origin master

分支 ： 主分支 和 开发分支

新建分支 switch to new branch

合并分支 merge

## 3：requireJS模块化开发

什么是模块化开发？







requireJS基本用法

require会定义三个变量：define,require,requirejs，其中require === requirejs，一般使用require更简短

define 从名字就可以看出这个api是用来定义一个模块

require 加载依赖模块，并执行加载完后的回调函数

参数1：依赖的模块，数组形式，

参数2：回调函数，参数：跟每个 依赖模块exports的对象顺序是一致的

require(['module1','module2'],function(m1,m2){

//TODO:

})

1-config.js 配置文件 配置所有的模块

//paths 配置所有模块的路径

// 'myDoduleName'(模块名称):'myDodulePaths'（模块路径）

requirejs.config({

paths:{

// 'myDoduleName':'myDodulePaths'

'myLunBo':'lunbo',

'myTest':'test',

'jquery':'jquery-1.11.3.min'

}

})

2-定义模块 define（‘【module1,module2】’，function(m1,m2){

}）

2-//定义一个模块,模块对外提供了 对一个对象  
*define*(function () {  
 var obj = {  
 LunBo:function () {  
   
 }  
 }  
   
   
 return obj;  
})

4-使用模块

*require*(['config'],function () {  
  
 console.log(1);  
 //引入config文件， 知道了所有的模块  
  
 *require*(['myLunBo'],function (m1) {  
 m1.LunBo();  
 console.log(m1)  
 })  
  
})

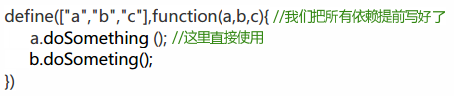
## 4：模块化开发规范 AMD和CMD规范

AMD是require所倡导的模块化开发的方式。遵循AMD规范的还有一个叫CommonJS的，不过还是和requireJS有些区别，主要针对NODE后端开发。

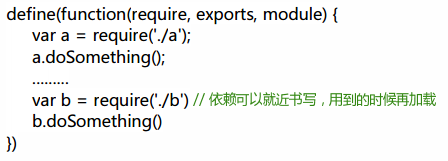
另外还有一个大名鼎鼎的SeaJS，作者是淘宝的玉伯，于是诞生了CMD规范，根据作者自己的介绍，SeaJS各方面都比requireJS强大。这个就不做评价了。

**区别：**

AMD推崇的是依赖前置。



CMD推崇的是依赖就近。



当然requireJS也支持CMD的写法，不过作者本人是不推荐这么写的。

## 综合应用：

1：掌握模块化开发。

2：掌握git。