

## 版本控制系统

1.记录历史版本信息（记录每次的修改）

2方便团队之间协作开发

### 常用的版本控制系统

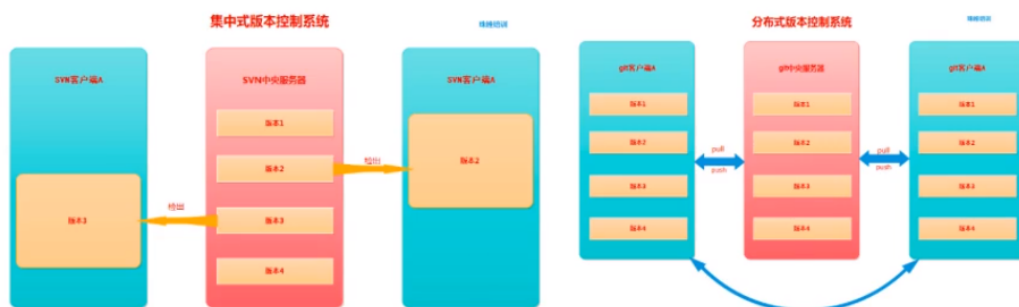
cvs/svn:集中式的版本控制系统

git: 分布式版本控制系统

### SVN和GIT的管理形式



#### ※ GIT基础管理



## SVN的运行模式

01-GIT的基础概述（集中式和分布式） 正在分享屏幕 01:05:06 王亚...等56人正在观看

北京珠峰培训  
微信: zhufengpeixunjiuye

### SVN为代表性的集中式

=>必须有一个中央总控的服务器（用来存储历史版本和代码信息）

**中央服务器**

**代码仓库：存储最新代码**

**A 把代码提交到中央服务器上**

**B 从中央服务器上拉取最新的代码（代码中包含每个同事写的内容）**

历史版本1  
历史版本2  
历史版本3

从中央服务器上获取某个版本信息，回滚一下，让对应的版本中的代码覆盖本地的代码

### 每天早上来了

1. 拉取中央服务器上的最新代码到本地
2. 开始一天的工作

### 每天走之前或者上传之前

1. 上传前一定要先拉取，如果有冲突，自己本地把冲突的代码进行合并
2. 把自己写完没有问题的代码上传到中央服务器上

### SVN弊端：

1. 需要连网才能回退或查看历史版本信息
2. 中央服务器毁坏了，一切OVER
3. 所有的上传和下载都是基于文件传输方式完成的，速度会慢

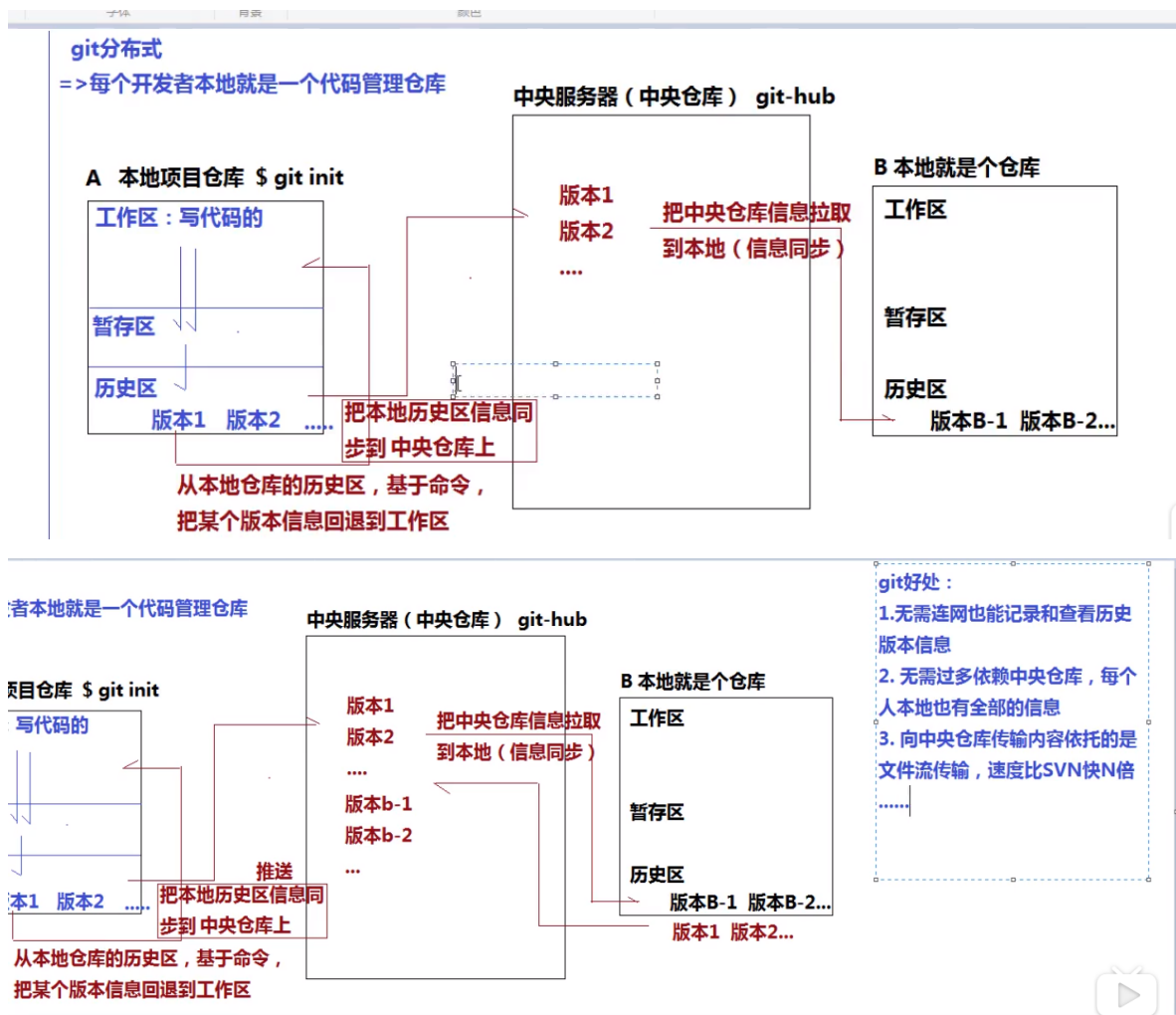
.....

359, 91像素 1948 x 576像素 大小: 65.7KB 2019/7/29 44:00 / 61:55

# Git的运行模式和分布式开发

中央服务器的作用：中央服务器的作用主要是将各个本地仓库的项目进行流转

- 1: 通过将代码缓存到本地的形式来解决SVN模式的联网问题
- 2: 将全部的项目信息放到本地仓库中，解决团队开发时对于中央服务器的依赖
- 3: 向中央仓库传输内容依托的是文件流传输,速度比SVN快N倍



## 1.GIT工作原理

工作区:我们能看到的,并且用来写代码的区域

暂存区:临时存储用的

历史区:生成历史版本

工作区→暂存区→历史区

不能从历史区撤回到暂存区

GIT的全局配置

第一次安装完git之后，我们在全局环境下配置基本信息：我是谁？

```
$ git config -l 察看全局配置
$ git config --global -l 察看全局配置信息
配置全局信息
$ git config --global user.name 'xxx' 用户名
$ git config --global user.email 'xxx@xx.xx' 邮箱

$ clear 清屏
```

## 创建仓库，完成版本控制

创建本地仓库

**从工作区提交暂存区,从暂存区提交到历史区:是把内容复制一份传过去的,本区域中依然存在这些信息(只有这样才能对比出哪些文件在某个区)**

```
$ git init
//=>会产生一个隐藏文件夹".git"(这个文件夹不能删除,因为暂存区和历史区还有一些其他重要信息都在这个文件夹里)
```

在本地编写完代码后(在工作区), 把一些文件提交打暂存区

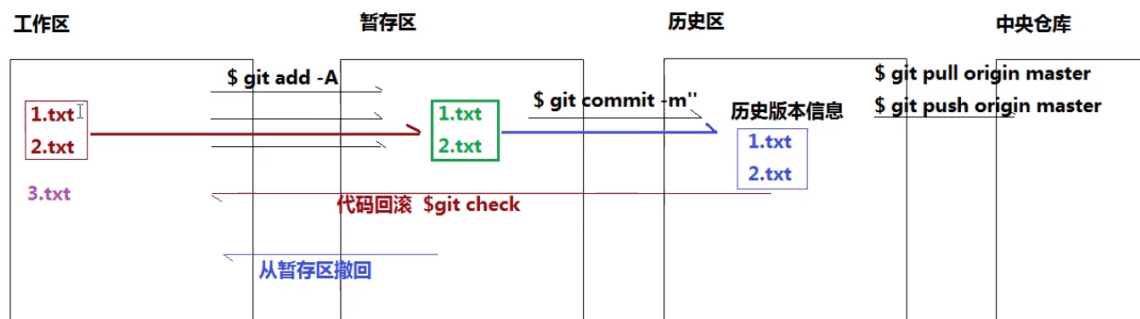
```
$ git add xxx 把某一个文件提交到暂存区
$ git add . 或$ git add -A 把所有的新修改文件提交到暂存区
```

```
$ git status
```

察看先在文件的状态, 红色在工作区, 绿色就已经提交到了暂存区, 没有东西表示所有的文件已经提交到了历史区

把暂存区内容提交到历史区

```
$ git commit -m'描述信息: 本次提交内容的一个描述'
$ git commit 提交到历史区
$ git log 查看提交记录
$ git relog 查看所有的历史记录(包括历史区回滚后)版本号前七位
```



**从工作区提交暂存区,从暂存区提交到历史区:是把内容复制一份传过去的,本区域中依然存在这些信息(只有这样才能对比出哪些文件在某个区)**

## GIT和GIT-HUB

[www.github.com](http://www.github.com)

一个网站(一个开源的源代码管理平台),用户注册后,可以在自己账户下创建仓库,用来管理项目的源代码(源代码是基于就传到仓库中)

### 本地仓库和远程仓库之间的操作只与本地仓库的历史区打交道

在cmd中输入密码不会显示出来

new repository->填写信息> Create repository

public公共仓库作为开源的项目

private私有仓库作为内部团队协作管理的项目

Settings→>删除仓库 Delete this repository

Collaborators设置协作开发的人员

### 将本地仓库提交到远程仓库

```
//建立本地仓库和远程仓库的链接
//察看本地仓库和哪些远程仓库链接
$ git remote -v
//让本地仓库和远程仓库建立一个新的链接，origin是随便起的一个链接名(可以改成自己想要的,只不过一般都用这个名字)
$ git remote add [name] [GIT仓库地址]
//删除关联信息
$ git remote rm [name]

//提交之前最好先拉取
$ git pull [name] master
//把本地代码提交到远程仓库(需要输入 git thub的用户名密码)
$ git push [name] master
//拉取远程仓库:
$ git pull [name] [别名, 可以不设置, 默认为仓库名]
//克隆项目
$ git clone
```

真实项目开发流程:

1. 组长或者负责人先创建中央仓库(增加协作者)
2. 小组成员基于 `$ git clone`把远程仓库及默认的内容克隆到本地一份(解决了三个事情:初始化一个本地仓库“`git init`”、和对应的远程仓库也保持了关联“`git remote add`”、把远程仓库默认内容拉取到本地“`git pull`”)
3. 每个组员写完自己的程序后,基于“`git add/ git commit`”把自己修改的内容存放到历史区,然后通过“`git pull/ git push`”把本地信息和远程仓库信息保持同步即可(可能涉及冲突的处理)