# Git基本工作原理

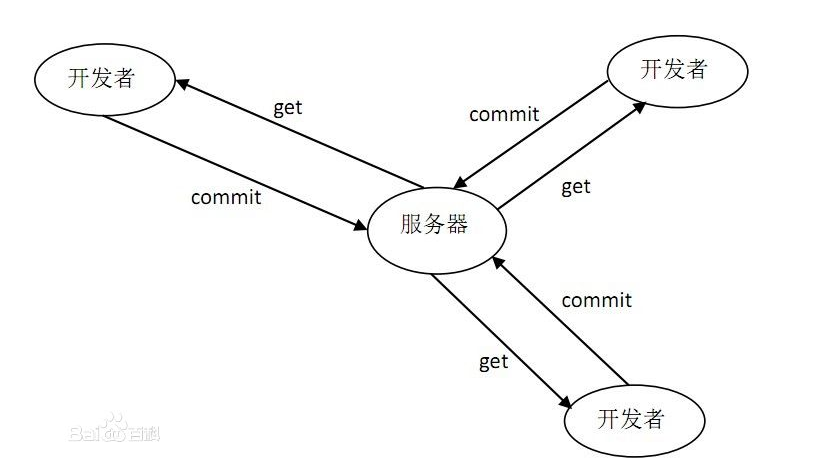
## 与SVN对比

* 1. Snv集中式版本控制系统

Svn版本控制系统是集中式的数据管理,存在一个中央版本库.

SVN的特点概括起来主要由以下几条：  
1）每个版本库有唯一的URL（官方地址），每个用户都从这个地址获取代码和数据；  
2）获取代码的更新，也只能连接到这个唯一的版本库，同步以取得最新数据；  
3）提交必须有网络连接（非本地版本库）；  
4）提交需要授权，如果没有写权限，提交会失败；  
5）提交并非每次都能够成功。如果有其他人先于你提交，会提示“改动基于过时的版本，先更新再提交”… 诸如此类；  
6）冲突解决是一个提交速度的竞赛：手快者，先提交，平安无事；手慢者，后提交，可能遇到麻烦的冲突解决。

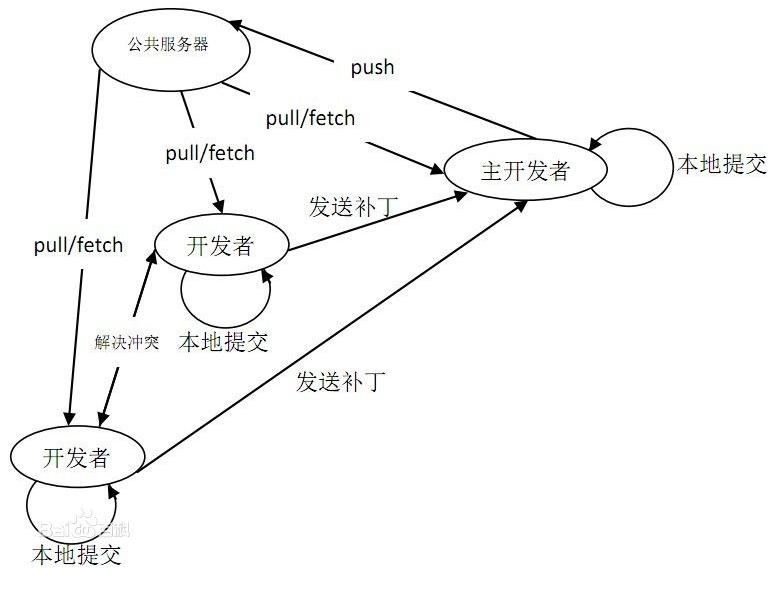
好处：每个人都可以一定程度上看到项目中的其他人正在做些什么。而管理员也可以轻松掌控每个开发者的权限。  
缺点：中央服务器的单点故障。



* 1. Git分布式的版本控制系统

和集中式版本控制系统相比，分布式版本控制系统的安全性要高很多，因为每个人电脑里都有完整的版本库.

Git特点：  
1）Git中每个克隆(clone)的版本库都是平等的。你可以从任何一个版本库的克隆来创建属于你自己的版本库，同时你的版本库也可以作为源提供给他人，只要你愿意。  
2）Git的每一次提取操作，实际上都是一次对代码仓库的完整备份。  
3）提交完全在本地完成，无须别人给你授权，你的版本库你作主，并且提交总是会成功。  
4）甚至基于旧版本的改动也可以成功提交，提交会基于旧的版本创建一个新的分支。  
5）Git的提交不会被打断，直到你的工作完全满意了，PUSH给他人或者他人PULL你的版本库，合并会发生在PULL和PUSH过程中，不能自动解决的冲突会提示您手工完成。  
6）冲突解决不再像是SVN一样的提交竞赛，而是在需要的时候才进行合并和冲突解决。



* 1. 应用场景:

1)适用对象不同。Git适用于参与开源项目的开发者。他们由于水平高，更在乎的是效率而不是易用性。SVN则不同，它适合普通的公司开发团队。使用起来更加容易。

2)使用的场合不同。Git适用于通过Internet，有多个开发角色的单个项目开发，SVN适合企业内部由项目经理统一协调的多个并行项目的开发。

3)权限管理策略不同。Git没有严格的权限管理控制，只要有帐号，就可以导出、导入代码，甚至执行回退操作。SVN则有严格的权限管理，可以按组、按个人进行针对某个子目录的权限控制。区分读、写权限。更严格的，不支持回退操作。保证代码永远可以追踪。

4)分支(branch)的使用范围不一样。Git中，你只能针对整个仓库作branch,而且一旦删除，便无法恢复。而SVN中，branch可以针对任何子目录，它本质上是一个拷贝操作。所以，可以建立非常多、层次性的branch,并且，在不需要时将其删除，而以后需要时只要checkout老的SVN版本就可以了。

5)基于第三点，Git适用于单纯的软件项目，典型的就是一些开源项目，比如Linux内核、busybox等。相反，SVN擅长多项目管理。比如，你可以在一个SVN仓库中存放一个手机项目的bsp/设计文档/文件系统/应用程序/自动化编译脚本，或者在一个SVN中存放5款手机项目的文件系统。git中必须建立n(项目数)\*m（组件数）个仓库。SVN中只需要最多n或者m个就可以了。

6)Git使用128位ID作为版本号，而且checkout时要注明是哪个branch,而SVN使用一个递增的序列号作为全局唯一的版本号，更加简明易懂。虽然可以使用gittag来建立一些文字化的别名，但是毕竟那只是针对特殊版本。

7)可跟踪性，git的典型开发过程为：建立分支，进行开发，提交到本地master，删除分支。这样做的后果是以前的修改细节会丢失。而在SVN下做同样的事情，不会丢失任何细节。这里是一个有趣的链接，表明了git下典型的工作方式：（以master为核心，不断创建新branch,删除旧branch):

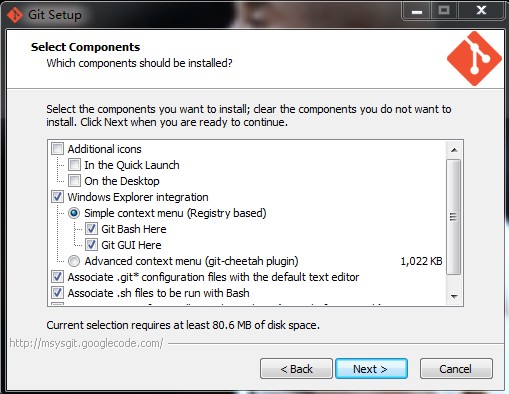
8)局部更新，局部还原。SVN由于是在每个文件夹建立一个.svn文件夹来实现管理，所以可以很简单实现局部更新或者还原。假如你只希望更新某些部分，则svn可以很好实现。同时代码写错了，同时可以很好实现局部还原，当然git也可以通过历史版本还原，但是无法简单地实现局部还原。

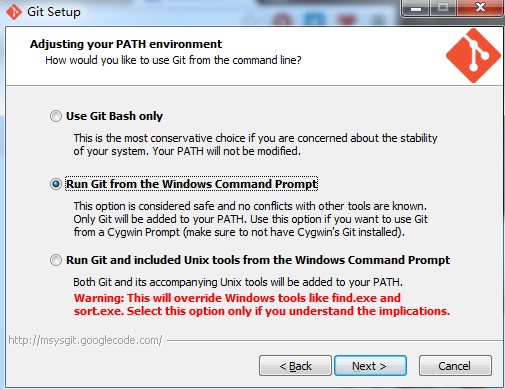
## Git for window安装和基本设置

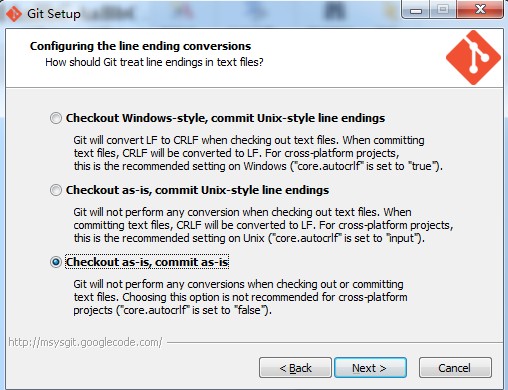
1、下载地址：

<http://msysgit.github.io/>

2、下载完成后安装，安装路径自己选择，其他的选项参照下图：







其他的一步一步往下即可，最后Finish完成安装；

3、配置github的ssh密钥:

(1)打开Git Bash查看电脑上是否已经存在SSH密钥：

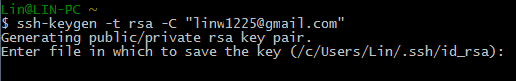
输入 cd ~/.ssh

https://images0.cnblogs.com/i/335529/201405/122329362033539.png

若如上图显示无法找到该文件则要创建新的ssh key;

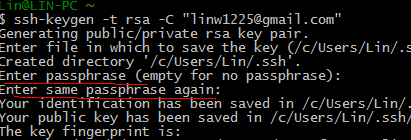
(2)创建新的ssh key:

输入 ssh-keygen -t rsa -C "your\_email@youremail.com"



执行这条命令会如上图提示文件保存路径，可以直接按Enter，

然后提示输入 passphrase（密码），输入两次（可以不输直接两次Enter），



然后会在 .ssh 目录生产两个文件：id\_rsa和id\_rsa.pub

用记事本打开.ssh目录下的id\_rsa.pub文件，复制里面的内容；

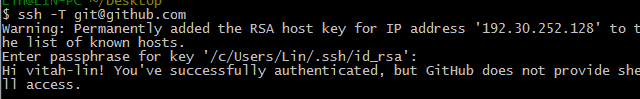
4、复制ssh key到github：

On the GitHub site Click “Account Settings” > Click “SSH Keys” > Click “Add SSH key”

打开github网站，点击右上角扳手图标，然后点击左边菜单的 ssh key， 然后右边页面的 add ssh key，将复制的内容粘贴到github的key中，title可以不填，直接保存即可。

5、测试 ssh 链接 github：

输入 ssh -T git@github.com



出现Successfully就OK；

6、设置自己的git信息：

输入

git config --global user.name "Firstname Lastname" （此处name可修改也不是用于登录github的登录名）

git config --global user.email "your\_email@youremail.com"

设置自己的git信息即完成安装和设置，可以输入git config --list查看自己的git信息。

# Git常用操作及问题处理

## 工程初始化及克隆

## 文件提交

## 分支及常用标签应用

## 远程仓库管理

## 合并与冲突解决

## 规范Git操作

1）Git也可以模拟集中式的工作模式  
Git版本库统一放在服务器中  
可以为 Git 版本库进行授权：谁能创建版本库，谁能向版本库PUSH，谁能够读取（克隆）版本库  
团队的成员先将服务器的版本库克隆到本地；并经常的从服务器的版本库拉（PULL）最新的更新；  
团队的成员将自己的改动推（PUSH）到服务器的版本库中，当其他人和版本库同步（PULL）时，会自动获取改变  
2）Git 的集中式工作模式非常灵活  
你完全可以在脱离Git服务器所在网络的情况下，如移动办公／出差时，照常使用代码库  
你只需要在能够接入Git服务器所在网络时，PULL和PUSH即可完成和服务器同步以及提交  
Git提供rebase 命令，可以让你的改动看起来是基于最新的代码实现的改动  
3）Git有更多的工作模式可以选择，远非 Subversion能比的。