   Git是一个分布式的版本控制工具，本篇文章从介绍Git开始，重点在于介绍Git的基本命令和使用技巧，让你尝试使用Git的同时，体验到原来一个版 本控制工具可以对开发产生如此之多的影响，文章分为两部分，第一部分介绍Git的一些常用命令，其中穿插介绍Git的基本概念和原理，第二篇重点介绍 Git的使用技巧，最后会在Git Hub上创建一个开源项目开启你的Git实战之旅

**1、Git是什么**

        Git 在Wikipedia上的定义：它是一个免费的、分布式的版本控制工具，或是一个强调了速度快的源代码管理工具。Git最初被Linus Torvalds开发出来用于管理Linux内核的开发。每一个Git的工作目录都是一个完全独立的代码库，并拥有完整的历史记录和版本追踪能力，不依赖 于网络和中心服务器。

        Git 的出现减轻了许多开发者和开源项目对于管理分支代码的压力，由于对分支的良好控制，更鼓励开发者对自己感兴趣的项目做出贡献。其实许多开源项目 包括Linux kernel, Samba, X.org Server, Ruby on Rails，都已经过渡到使用Git作为自己的版本控制工具。对于我们这些喜欢写代码的开发者嘛，有两点最大的好处，我们可以在任何地点(在上班的地铁 上)提交自己的代码和查看代码版本;我们可以开许许多多个分支来实践我们的想法，而合并这些分支的开销几乎可以忽略不计。

**2、Git 1+1**

        现在进入本篇文章真正的主题，介绍一下Git的基本命令和操作，会从Git的版本库的初始化，基本操作和独有的常用命令三部分着手，让大家能够开始使用Git。

        Git通常有两种方式来进行初始化:

        git clone: 这是较为简单的一种初始化方式，当你已经有一个远程的Git版本库，只需要在本地克隆一份，例如'git clone git://github.com/someone/some\_project.git some\_project'命令就是将'git://github.com/someone/some\_project.git'这个URL地址的远程版 本库完全克隆到本地some\_project目录下面

        git init和git remote：这种方式稍微复杂一些，当你本地创建了一个工作目录，你可以进入这个目录，使用'git init'命令进行初始化，Git以后就会对该目录下的文件进行版本控制，这时候如果你需要将它放到远程服务器上，可以在远程服务器上创建一个目录，并把 可访问的URL记录下来，此时你就可以利用'git remote add'命令来增加一个远程服务器端，例如'git remote add origin git://github.com/someone/another\_project.git'这条命令就会增加URL地址为'git: //github.com/someone/another\_project.git'，名称为origin的远程服务器，以后提交代码的时候只需要使用 origin别名即可

**3、Git的基本命令**

     现在我们有了本地和远程的版本库，让我们来试着用用Git的基本命令吧：

**git pull：**从其他的版本库(既可以是远程的也可以是本地的)将代码更新到本地，例如：'git pull origin master'就是将origin这个版本库的代码更新到本地的master主枝，该功能类似于SVN的update

**git add：**是将当前更改或者新增的文件加入到Git的索引中，加入到Git的索引中就表示记入了版本历史中，这也是提交之前所需要执行的一步，例如'git add app/model/user.rb'就会增加app/model/user.rb文件到Git的索引中

**git rm：**从当前的工作空间中和索引中删除文件，例如'git rm app/model/user.rb'

**git commit：**提交当前工作空间的修改内容，类似于SVN的commit命令，例如'git commit -m "story #3, add user model"'，提交的时候必须用-m来输入一条提交信息

**git push：**将本地commit的代码更新到远程版本库中，例如'git push origin'就会将本地的代码更新到名为orgin的远程版本库中

**git log：**查看历史日志

**git revert：**还原一个版本的修改，必须提供一个具体的Git版本号，例如'git revert bbaf6fb5060b4875b18ff9ff637ce118256d6f20'，Git的版本号都是生成的一个哈希值、

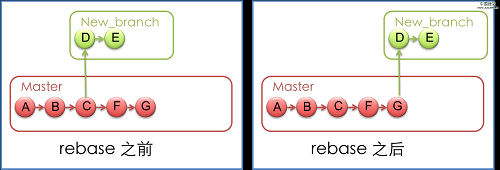
        上面的命令几乎都是每个版本控制工具所公有的，下面就开始尝试一下Git独有的一些命令：

**4、Git独有的一些命令**

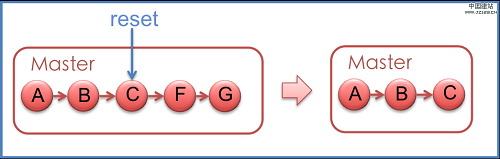
**git branch**： 对分支的增、删、查等操作，例如'git branch new\_branch'会从当前的工作版本创建一个叫做new\_branch的新分支，'git branch -D new\_branch'就会强制删除叫做new\_branch的分支，'git branch'就会列出本地所有的分支

**git checkout**：Git 的checkout有两个作用，其一是在不同的branch之间进行切换，例如 'git checkout new\_branch'就会切换到new\_branch的分支上去;另一个功能是还原代码的作用，例如'git checkout app/model/user.rb'就会将user.rb文件从上一个已提交的版本中更新回来，未提交的内容全部会回滚

**git rebase**：用下面两幅图解释会比较清楚一些，rebase命令执行后，实际上是将分支点从C移到了G，这样分支也就具有了从C到G的功能



**git reset**： 将当前的工作目录完全回滚到指定的版本号，假设如下图，我们有A-G五次提交的版本，其中C 的版本号是 bbaf6fb5060b4875b18ff9ff637ce118256d6f20，我们执行了'git reset bbaf6fb5060b4875b18ff9ff637ce118256d6f20'那么结果就只剩下了A-C三个提交的版本

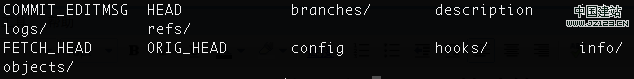


**git stash**：将当前未提交的工作存入Git工作栈中，时机成熟的时候再应用回来，这里暂时提一下这个命令的用法，后面在技巧篇会重点讲解

**git config**： 利用这个命令可以新增、更改Git的各种设置，例如'git config branch.master.remote origin'就将master的远程版本库设置为别名叫做origin版本库，后面在技巧篇会利用这个命令个性化设置你的Git，为你打造独一无二的 Git

**git tag**： 可以将某个具体的版本打上一个标签，这样你就不需要记忆复杂的版本号哈希值了，例如你可以使用 'git tag revert\_version bbaf6fb5060b4875b18ff9ff637ce118256d6f20'来标记这个被你还原的版本，那么以后你想查看该版本时，就可以使用 revert\_version标签名，而不是哈希值了

        Git之所以能够提供方便的本地分支等特性，是与它的文件存储机制有关的。Git存储版本控制信息时使用它自己定义的一套文件系统存储机制，在代码根目录下有一个.git文件夹，会有如下这样的目录结构：

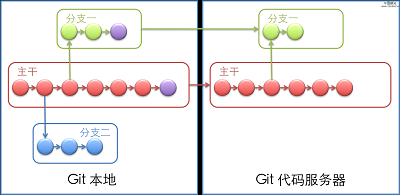


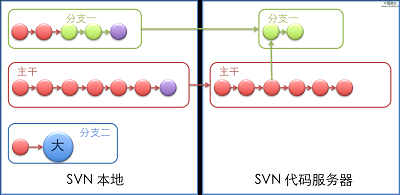
        有 几个比较重要的文件和目录需要解释一下：HEAD文件存放根节点的信息，其实目录结构就表示一个树型结构，Git采用这种树形结构来存储版本信息， 那么HEAD就表示根;refs目录存储了你在当前版本控制目录下的各种不同引用(引用指的是你本地和远程所用到的各个树分支的信息)，它有heads、 remotes、stash、tags四个子目录，分别存储对不同的根、远程版本库、Git栈和标签的四种引用，你可以通过命令'git show-ref'更清晰地查看引用信息;logs目录根据不同的引用存储了日志信息。因此，Git只需要代码根目录下的这一个.git目录就可以记录完 整的版本控制信息，而不是像SVN那样根目录和子目录下都有.svn目录。那么下面就来看一下Git与SVN的区别吧

**5、Git与SVN的不同**

        SVN(Subversion)是当前使用最多的版本控制工具。与它相比较，Git最大的优势在于两点：易于本地增加分支和分布式的特性。

        下面两幅图可以形象的展示Git与SVN的不同之处





        对 于易于本地增加分支，图中Git本地和服务器端结构都很灵活，所有版本都存储在一个目录中，你只需要进行分支的切换即可达到在某个分支工作的效果。 而SVN则完全不同，如果你需要在本地试验一些自己的代码，只能本地维护多个不同的拷贝，每个拷贝对应一个SVN服务器地址。举一个实际的例子，以前我所 在的小组使用SVN作为版本控制工具，当我正在试图增强一个模块，工作做到一半，由于会改变原模块的行为导致代码服务器上许多测试的失败，所以并没有提交 代码。这时候上级对我说，现在有一个很紧急的Bug需要处理， 必须在两个小时内完成。我只好将本地的所有修改diff，并输出成为一个patch文件，然后回滚有关当前任务的所有代码，再开始修改Bug的任务，等到 修改好后，在将patch应用回来。前前后后要完成多个繁琐的步骤，这还不计中间代码发生冲突所要进行的工作量。可是如果使用Git， 我们只需要开一个分支或者转回到主分支上，就可以随时开始Bug修改的任务，完成之后，只要切换到原来的分支就可以优雅的继续以前的任务。只要你愿意，每 一个新的任务都可以开一个分支，完成后，再将它合并到主分支上，轻松而优雅。

        分 布式对于Git而言，你可以本地提交代码，所以在上面的图中，Git有利于将一个大任务分解，进行本地的多次提交，而SVN只能在本地进行大量的一 次性更改，导致将来合并到主干上造成巨大的风险。Git的代码日志是在本地的，可以随时查看。SVN的日志在服务器上的，每次查看日志需要先从服务器上下 载下来。我工作的小组，代码服务器在美国，每次查看小组几年前所做的工作时，日志下载就需要十分钟，这不能不说是一个痛苦。后来我们迁移到Git上，利用 Git日志在本地的特性，我用Ruby编写了一个Rake脚本，可以查看某个具体任务的所有代码历史，每次只需要几秒钟，大大方便我的工作。当然分布式并 不是说用了Git就不需要一个代码中心服务器，如果你工作在一个团队里，还是需要一个服务器来保存所有的代码的。

**总结**

        本篇介绍了Git的基本概念、一些常用命令和原理，大家可以尝试动手体会一下，下一篇会重点介绍Git命令的使用技巧，Git附带的工具，最后会在Git Hub上创建一个开源项目。