git 使用简介

指导老师：邵志远
作者：　　卓达城
邮箱：　　zhuodc@qq.com
单位：　　华中科技大学服务计算技术与系统/集群与网格计算实验室
简介：　　作者是华中科技大学 2010 级计算机学院计算机系统结构专业研究生

一、简介：

网上找到的 git 的中文资料，大部分是讲 git 的命令的使用，对于 git 的工作流程和如何实现团队合作的介绍少之又少，特别是对于团队代码库管理者的文档，几乎没有，这份文档将从开发者和管理者两方面介绍如何使用 git 进行团队合作开发。

二、git 和 svn 的差异

git 和 svn 最大的差异在于 git 是分布式的管理方式而 svn 是集中式的管理方式。如果不习惯用代码管理工具，可能比较难理解分布式管理和集中式管理的概念。下面介绍两种工具的工作流程（团队开发），通过阅读下面的工作流程，你将会很好的理解以上两个概念。

集中式管理的工作流程如下图（图 2.1）：

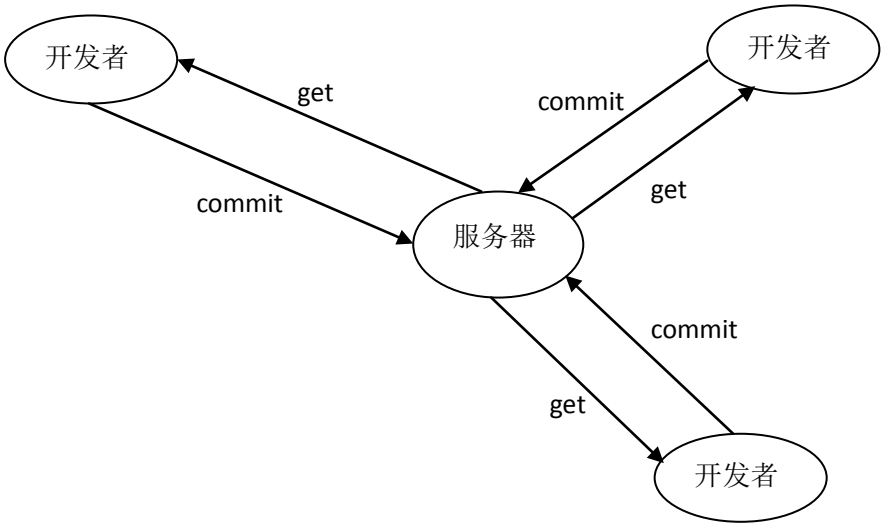

图 2.1

集中式代码管理的核心是服务器，所有开发者在开始新一天的工作之前必须从服务器获取代码，然后开发，最后解决冲突，提交。所有的版本信息都放在服务器上。如果脱离了服务器，开发者基本上是不可以工作。下面举例说明：

开始新一天的工作：

1：从服务器下载项目组最新代码。

2：进入自己的分支，进行工作，每隔一个小时向服务器自己的分支提交一次代码（很多人

都有这个习惯。因为有时候自己对代码改来改去，最后又想还原到前一个小时的版本，或者看看前一个小时自己修改了那些代码，就需要这样做了）。
3：下班时间快到了，把自己的分支合并到服务器主分支上，一天的工作完成，并反映给服务器。

这就是经典的 svn 工作流程，从流程上看，有不少缺点，但也有优点。
缺点：
1、 服务器压力太大，数据库容量暴增。
2、 如果不能连接到服务器上，基本上不可以工作，看上面第二步，如果服务器不能连接上，就不能提交，还原，对比等等。
3、不适合开源开发（开发人数非常非常多，但是 Google app engine 就是用 svn 的）。但是一般集中式管理的有非常明确的权限管理机制（例如分支访问限制），可以实现分层管理，从而很好的解决开发人数众多的问题。

优点：
1、 管理方便，逻辑明确，符合一般人思维习惯。
2、 易于管理，集中式服务器更能保证安全性。
3、 代码一致性非常高。
4、 适合开发人数不多的项目开发。
5、大部分软件配置管理的大学教材都是使用 svn 和 vss。

下面开分布式管理的工作流程，如下图（图 2.2）：

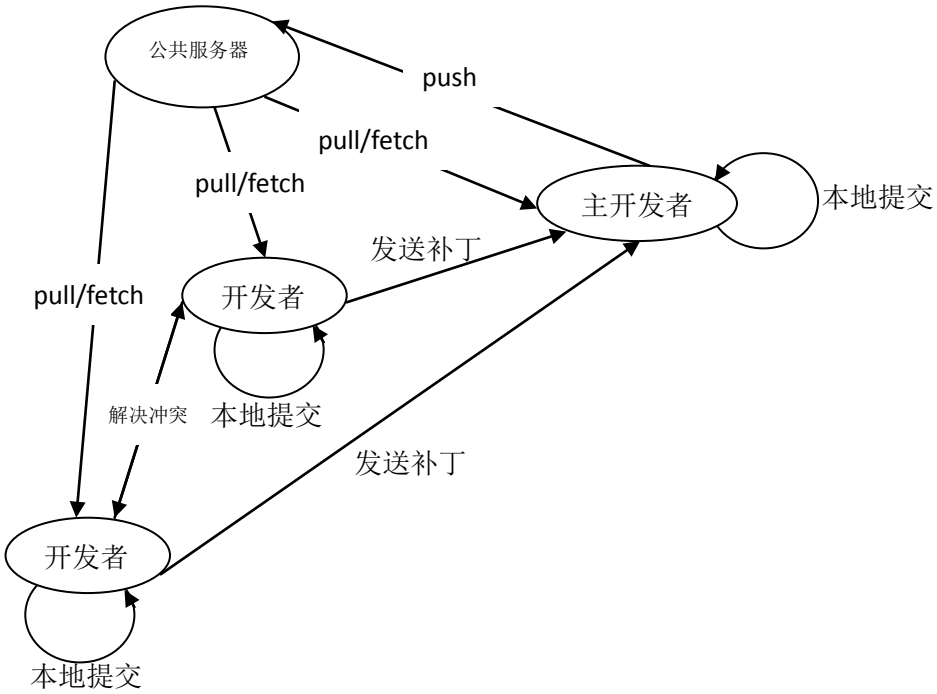

图 2.2

分布式和集中式的最大区别在于开发者可以本地提交。每个开发者机器上都有一个服务器的数据库。

图 2.2 就是经典的 git 开发过程。步骤如下：

一般开发者的角度：
1：从服务器上克隆数据库（包括代码和版本信息）到单机上。
2：在自己的机器上创建分支，修改代码。
3：在单机上自己创建的分支上提交代码。
4：在单机上合并分支。
5：新建一个分支，把服务器上最新版的代码 fetch 下来，然后跟自己的主分支合并。
6：生成补丁（patch），把补丁发送给主开发者。
7：看主开发者的反馈，如果主开发者发现两个一般开发者之间有冲突（他们之间可以合作解决的冲突），就会要求他们先解决冲突，然后再由其中一个人提交。如果主开发者可以自己解决，或者没有冲突，就通过。
8：一般开发者之间解决冲突的方法，开发者之间可以使用 pull 命令解决冲突，解决完冲突之后再向主开发者提交补丁。

主开发者的角度（假设主开发者不用开发代码）：
1：查看邮件或者通过其它方式查看一般开发者的提交状态。
2：打上补丁，解决冲突（可以自己解决，也可以要求开发者之间解决以后再重新提交，如果是开源项目，还要决定哪些补丁有用，哪些不用）。
3：向公共服务器提交结果，然后通知所有开发人员。

优点：
适合分布式开发，强调个体。
公共服务器压力和数据量都不会太大。
速度快、灵活。
任意两个开发者之间可以很容易的解决冲突。
离线工作。

缺点：
资料少（起码中文资料很少）。
学习周期相对而言比较长。
不符合常规思维。
代码保密性差，一旦开发者把整个库克隆下来就可以完全公开所有代码和版本信息。

三、git 常用命令介绍
git init
创建一个数据库。

git clone
复制一个数据到指定文件夹

git add 和 git commit
把想提交的文件 add 上，然后 commit 这些文件到本地数据库。

git pull
从服务器下载数据库，并跟自己的数据库合并。

git fetch
从服务器下载数据库，并放到新分支，不跟自己的数据库合并。

git whatchanged
查看两个分支的变化。

git branch
创建分支，查看分支，删除分支

git checkout
切换分支

git merge
合并分支，把目标分支合并到当前分支

git config
配置相关信息，例如 email 和 name

git log
查看版本历史

git show
查看版本号对应版本的历史。如果参数是 HEAD 查看最新版本。

git tag
标定版本号。

git reset
恢复到之前的版本
----mixed 是 git-reset 的默认选项，它的作用是重置索引内容，将其定位到指定的项目版本，而不改变你的工作树中的所有内容，只是提示你有哪些文件还未更新。
--soft 选项既不触动索引的位置，也不改变工作树中的任何内容。该选项会保留你在工作树中的所有更新并使之处于待提交状态。相当于再--mixed 基础上加上 git add .
--hard 把整个目录还原到一个版本，包括所有文件。

git push
向其他数据库推送自己的数据库。

git status
显示当前的状态。

git mv
重命名文件或者文件夹。

git rm
删除文件或者文件夹。

git help
查看帮助，还有几个无关紧要的命令，请自己查看帮助。

四、git 开发模式
1：大项目开发模式（如图 4.1）（不适合我们实验室使用）

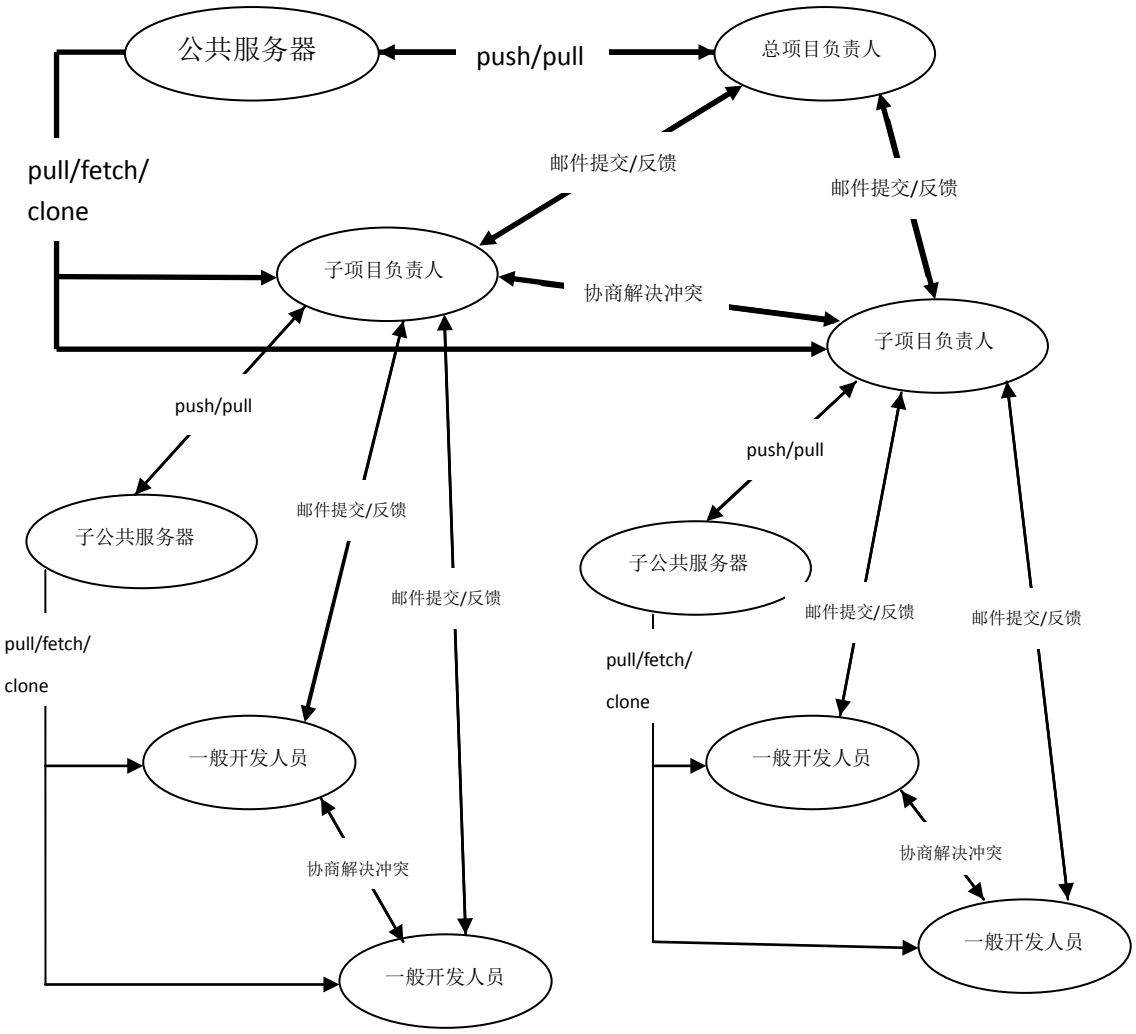

图 4.1

对于项目负责人
1：初始化
对于最终项目负责人：
使用 git init --bare 在公共服务器上建立一个空数据库，在自己的机器上通过

```
git pull  公共服务器 ip 地址:数据库所在文件夹
```

获得数据库（这里需要设置一下访问权限，由于 git 没有提供权限管理功能，所以要通过 ssh 设置，具体是对下一级子项目经理可读不可写，对自己可读可写）。
新建一些必要的文件夹和文件在放到自己的数据库上

```
新建文件
git add .
git commit
```

然后使用

```
git push  公共服务器 ip 地址:数据库所在文件夹  master
```

提交到公共服务器上，作为原始版本。
告诉下级公共服务器的地址。

对于子项目负责人：
在子公共服务器上克隆一个数据库

```
git clone 上级公共服务器 ip 地址:数据库所在文件夹
```

设置访问权限，对下级可读不可写，对自己可读可写。
在自己的计算机中克隆一个数据库

```
git clone  同级公共服务器 ip 地址：数据库所在文件夹
```

告诉下级子公共服务器地址。

对于最底层的开发人员：
在上级公共服务器中克隆一个数据库

```
git clone 上级公共服务器 ip 地址:数据库所在文件夹
```

2：开展工作
对于最终项目负责人
收来自下级的邮件
在自己的数据库上建分支，并转到分支上

```
git branch temp
git checkout temp
打补丁（patch 命令）
git whatchanged master temp
git add .
```

```
git commit
git checkout master
git merge master temp
git push 公共服务器 ip 地址:数据库所在文件夹
通知下级人员已经更新
```

重复上述步骤，直到所有补丁打完。
如果发现在合并分支的时候发现有些冲突需要下级项目负责人协助解决的话，可以通知下级项目负责人。

对于子项目负责人：
如果上级项目负责人需要他们之间合作解决某些冲突，他们可以通过

```
git fetch  子项目负责人服务器 ip 地址:数据库所在文件夹  master:temp
git merge temp
解决冲突
git add .
git commit
git branch -d temp              删除分支
git fetch 上级项目公共服务器 ip 地址:数据库所在文件夹  master:temp
git merge temp                                  保证代码是最新的
git diff master temp > patch
发补丁给上级
```

解决冲突。
如果上级项目负责人没有要求合作解决冲突，那项目负责人应该做以下事情：

```
git pull  上级项目公共服务器 ip 地址:数据库所在文件夹
解决冲突
git push  对应的公共服务器 ip 地址:数据库所在文件夹
通知下级人员服务器已经更新
```

然后项目负责人就开始接收邮件，然后打补丁

```
git branch temp
git checkout temp
打补丁（patch 命令）
git whatchanged master temp
git add .
git commit
git checkout master
git merge temp
git branch -d temp
```

重复上述工作，直到补丁全部打完。

下面是向上级提交更新的过程

```
git fetch  上级项目公共服务器 ip 地址:数据库所在文件夹  master:temp
git merge temp                  保证代码是最新的。
git diff temp master > patch
发邮件提交补丁
```

对于最底层的开发人员：
如果上级要求解决冲突，同样是要解决冲突，然后再提交补丁

```
git fetch 对应开发人员的服务器 ip 地址:数据库所在文件夹  master:temp
git merge temp
解决冲突
git add .
git commit
git branch -d temp              删除分支
git fetch 上级项目公共服务器 ip 地址:数据库所在文件夹  master:temp
git merge temp                                  保证代码是最新的
git diff master temp > patch
发补丁给上级
```

如果不用，就从上级服务器更新

```
git pull  上级项目公共服务器 ip 地址:数据库所在文件夹
解决冲突
```

然后建分支，并开发代码

```
git branch xxx
git checkout xxx
开发
git checkout master
git merge xxx
```

然后是向上级提交代码

```
git fetch  上级项目公共服务器 ip 地址:数据库所在文件夹  master:temp
git merge temp                  保证代码是最新的。
git diff temp master > patch
发邮件提交补丁
```

以上就是 git 在大项目开发中的应用。
但是明显是不适合我们实验室的。
原因有三：
1、我们学生中没有专门的维护人员。
2、我们学生中没有对全局都很了解架构师。
3、我们的老师可以担当此重任也只有我们的老师有这样的实力，但是老师太忙，没时间每天做这些琐事。

所以我们需要一种新的合作模式（一种没有项目负责人的模式）。
这种模式对开发人员的素质要求很高。
合作模式如下图（图 4.2）：（适合我们实验室使用）

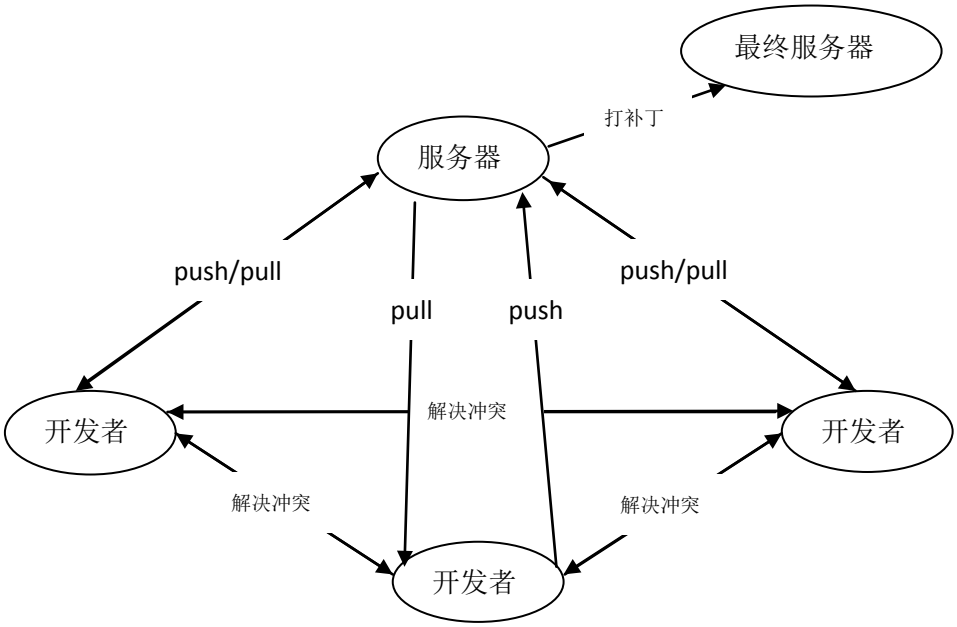

图 4.2

这种模式的开发流程如下：
1、由其中一个开发者这服务器上建立一个数据库。
所有开发者都可以向数据库提交和下载东西，这里必须规定一定的时间间隔（一周或者一天）必须提交一次，不然以后解决冲突时是个大问题。

如果每个人的开发耦合度很高，我们可在服务器上建立分支，然后每人每次提交到自己的分支上，过一段时间之后（不能太久）有一个人去合并分支。然后所有人更新自己的数据库的 master 分支，使之跟服务器上的 master 分支同步。

2、这样服务器会有非常多的版本信息，集合了每个人的版本信息。
过了一段时间之后，例如有里程碑的出现。再由一个人把所有改动打补丁到最终服务器上。这样最终服务器的版本信息就会很精练。

3、当我们的服务器无限膨胀到一定程度的时候我们可以把它删除，然后用最终服务器上的一个版本作为起始版本。

五、通过 ssh 进行权限管理

这个问题在第一版暂时不讲。目的是规定哪些人可以对服务器做什么事情（读？写？）。