版本控制系统简明教程

作者：郭宗辉

年级：2010

目录

1.概论

当我们想要查看文件以前的版本，就要查看以前的备份，但是我们的备份一般就是以时间命名的，对于大型的项目，对着一大堆的备份文件去找想要的版本非常困难。对于这种现象，版本控制系统就出现了。大部分的公司都是使用版本控制系统，所以学习它迫在眉睫。

起初，我们的主要也是唯一的目的就是对代码进行监控，使我们能够安全快速的返回到旧的版本，以便于我们能够诊断代码中的问题。我们知道在项目开发中，团队之间的合作是非常重要的，所以代码控制系统关注点又发展到如何使人于人之间的合作更为畅通。这个关注点并不是要取代对代码的监控，而是以代码监控为基础，并建立于其上的。现在，我们有越来越关注使用这些工具来描述代码的变更，因此就出现了对于重写代码历史命令的需求。

版本控制系统的核心任务就是实现协作编辑和数据共享。

鉴于 linux 系统的重要性，全文都是建立在 linux 上的版本控制系统的使用，在此还是要建议大家学习 linux 操作系统。

2.版本控制系统的文件共享

2.1 版本库

为了能实现文件的共享，就需要版本库来来存储文件。图 1“典型的客户/服务器系统”就展示了这种系统。

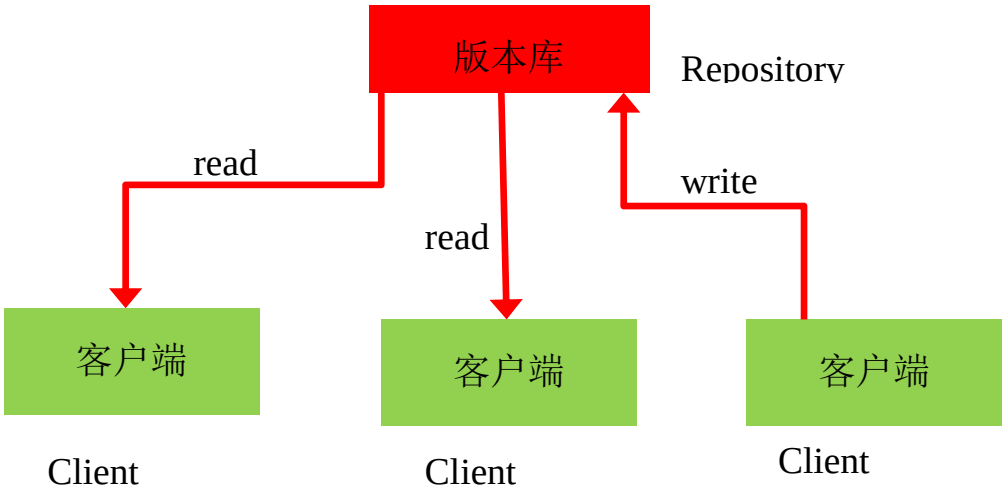


图 1. 典型的客户/服务器系统

以 Subversion 为例，Subversion 的版本库不是一般的文件服务器，特别之处在于，它会记录每一次的改变：每个文件的改变，甚至是目录树本身的改变，例如文件和目录的添加、删除和编辑等。

一般情况下，客户端从版本库中获取的数据是文件系统树中的最新数据。但是客户端也需要具备查看文件系统树以前任何一个状态的能力。例如，客户端有时会对一些历史性的问题比较感兴趣，比如“上周一时目录结构是什么样的？”或者“谁最后修改了某个文件，都修改了什么？”。这些都是版本控制系统所要完成的任务：设计用来记录和跟踪数据变化的系统。

2.2 文件共享问题

所有的版本控制系统都需要解决这样一个基础问题：怎样让系统允许用户共享信息，而不会让他们因意外而互相干扰？因为版本库里意外覆盖别人的更改是非常容易的。

假使现在又两个工作者，张三和李四，他们工作时对版本库操作

会出现图 2 所示的情况。

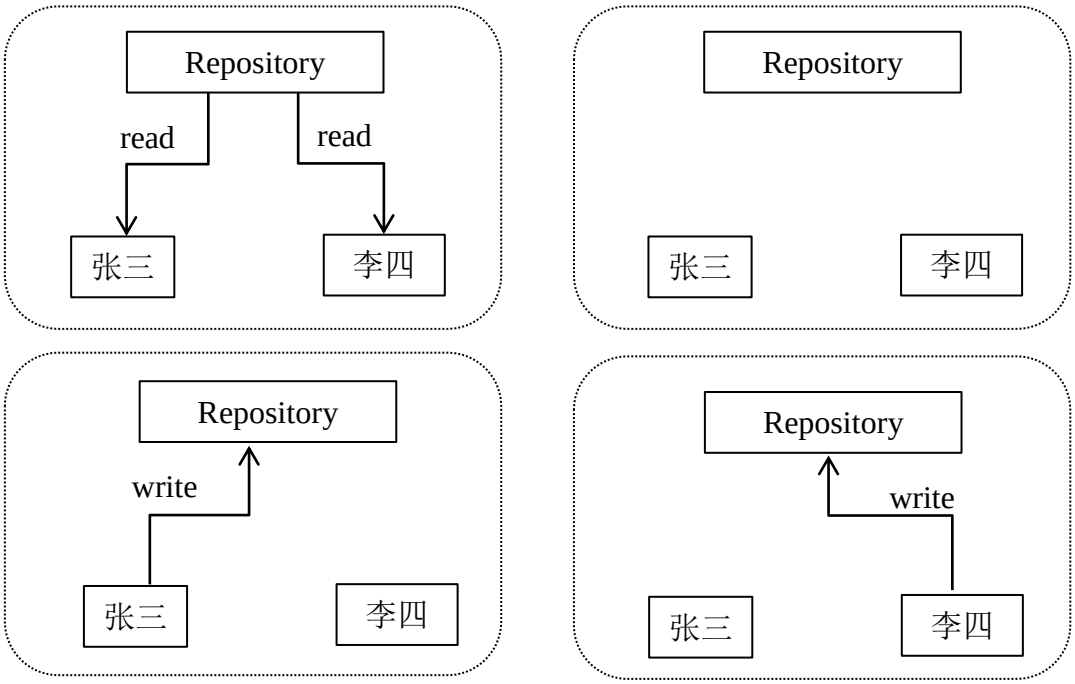


图 2. 需要避免的问题

在上图中大家是否已经想到在对版本库操作中出现的问题？张三和李四想同时编辑版本库中的同一个文件，如果张三先将他的修改保存到版本库中了，过了一会，李四也提交了自己的修改，此时，李四修改的文件就将张三的文件覆盖，虽然张三的修改不会永远消失（因为系统已经保存了版本库的每一次的修改），但是张三所有的修改不会出现在李四新版本的文件中，所以张三的工作还是丢失了。这就是在文件共享中所要避免的重要问题。

下面两节就是针对上述问题所提出的两种解决方案。

2.3 锁定-修改-解锁 方案

看到题目，我们就基本明白了本方案对版本库的操作流程，在这样的模型里，一段时间内只允许被一个人修改。首先在修改之前，张

三先“锁住”这个文件进行修改，在解锁之前，其他人只能阅读这个文件，不能对这个文件做任何修改。图 3 就能明确表示“锁定-修改-解锁”方案的流程。

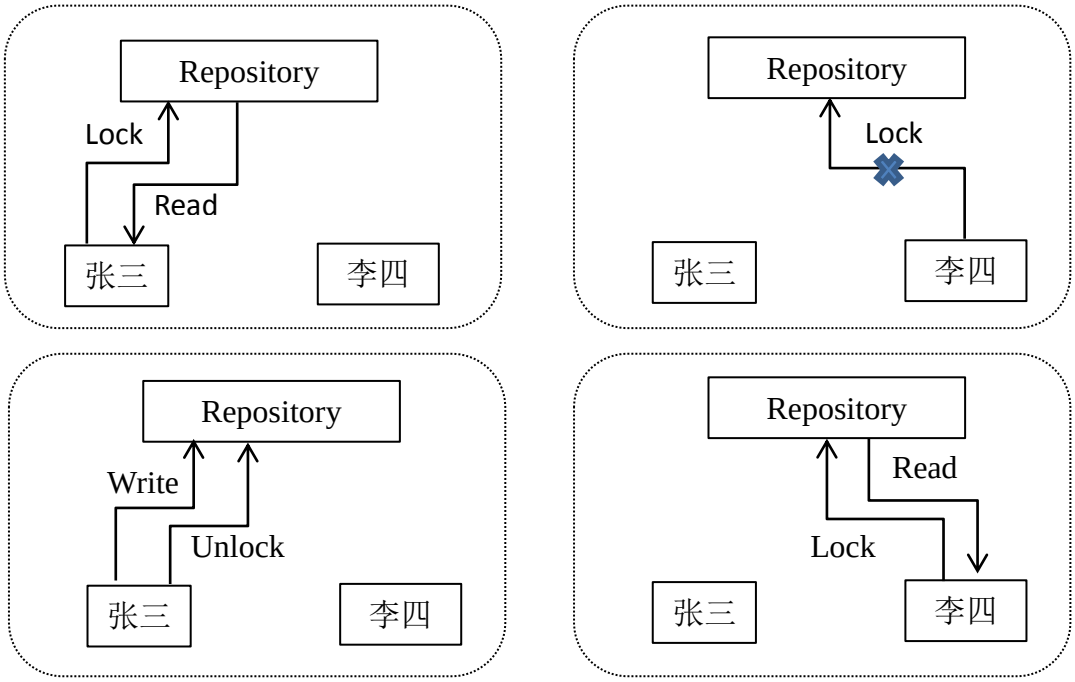


图 3. 锁定-修改-解锁 方案

锁定-修改-解锁模型的问题就是限制太多，经常成为用户的障碍：

1. 锁定可能导致管理问题。如果张三忘记了锁住文件的事情，而李四一直等待解锁，或是张三的修改时间过长，李四就无法工作，这样就会导致不必要的耽误和时间的浪费。
2. 锁定可能导致不必要的线性开发。如果张三编辑一个文件的开始，李四想编辑同一个文件的结尾，这样的修改不会冲突，设想修改可以正确的合并到一起，他们可以并行工作而没有太多的坏处，没有必要再轮流工作了。
3. 锁定可能导致错误的安全状态。假设张三锁定和编辑文件 A，同时李四锁定并编辑文件 B，如果 A 和 B 相互依赖，这种变化必须

是同时的，这样 A 和 B 就不能正确工作了，锁定机制对防止此类问题将无能为力，从而产生了一种处于安全状态的假象。

4. 出现不匹配的错误。张三和李四都在独立的工作，就不能够尽早的发现他们不匹配的修改。

2.4 拷贝-修改-合并 方案

Subversion，mercurial 和一些版本控制系统都是采用拷贝-修改-合并模型，在这种模型里，每一个用户在项目版本库建立一个个人工作拷贝—版本库中文件和目录的本地映射。用户并行工作，修改各自的工作拷贝，最终，各个私有的拷贝合并在一起，成为最终的版本，这种系统通常可以辅助合并操作，但是最终要靠人工去确定正误。

图4和图5能够将拷贝-修改-合并模型的设计流程基本表示出来。

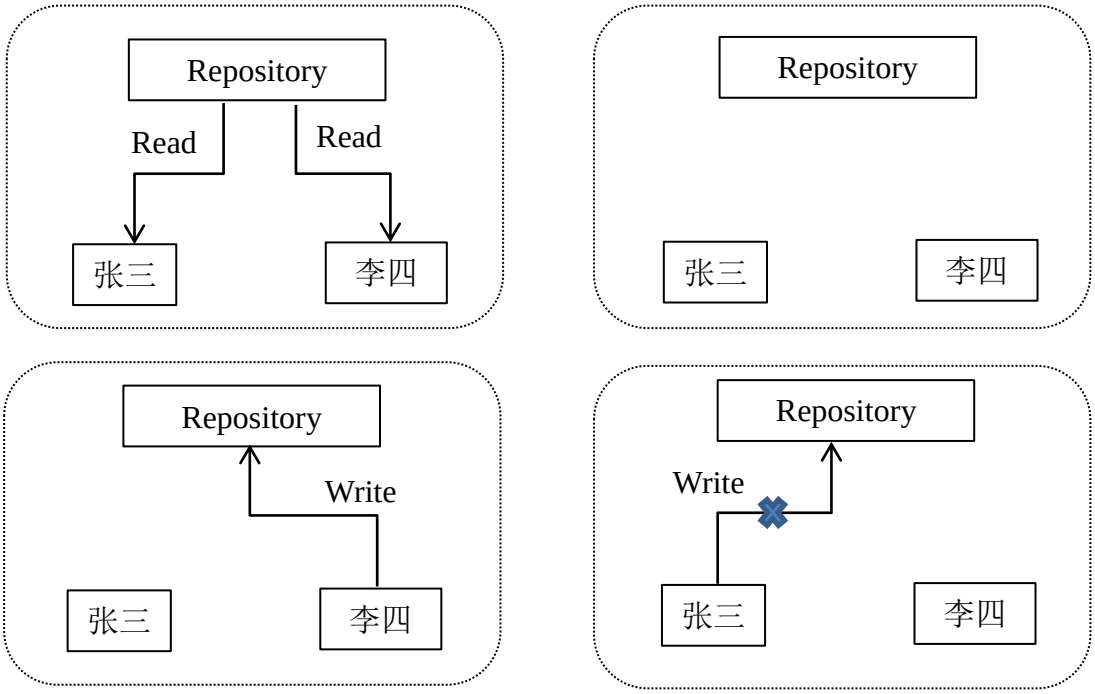


图 4 拷贝-修改-合并 （1）

图 4 中张三不能提交的原因是因为李四在之前已经提交了更改，

张三现有版本不是最新的了，所以他只能先更新自己的工作拷贝，再合并。

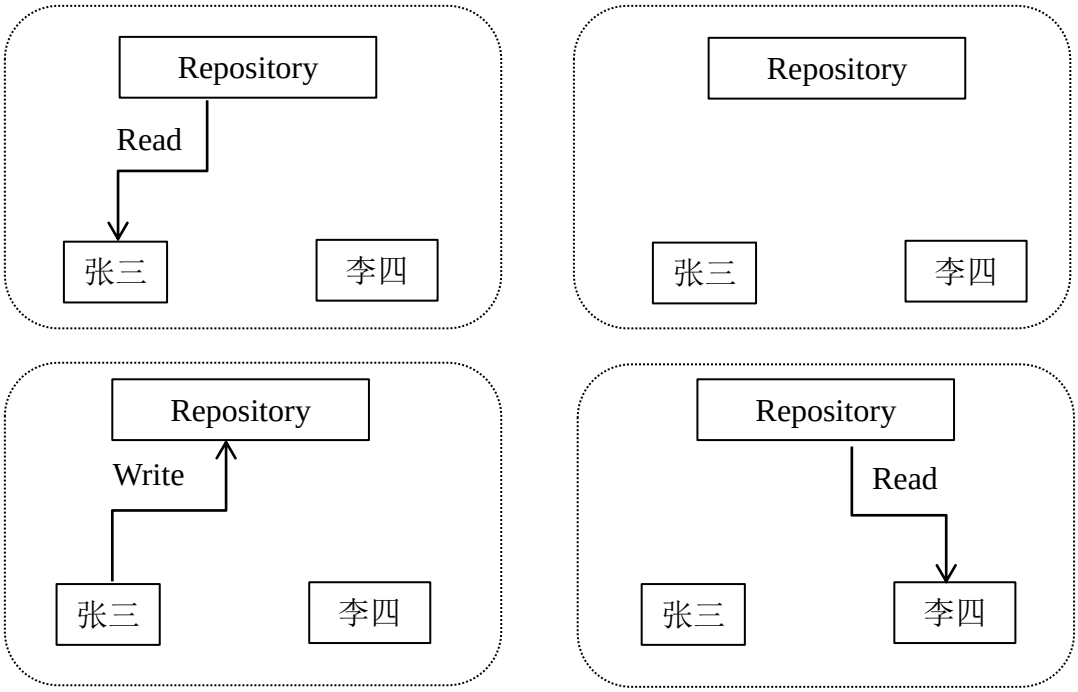


图 5 拷贝-修改-合并 （2）

但是如果张三和李四的修改交迭了该怎么办？这种情况叫做冲突，这通常不是个大问题，当张三告诉他的客户端去合并版本库的最新修改到自己的工作拷贝时，他的文件 A 就会处于冲突状态：他可以看到一对冲突的修改集，并手工的选择保留一组修改。需要注意的是软件不能自动的解决冲突，只有人可以理解并作出智能的选择，一旦张三手工的解决了冲突（也许需要与李四讨论）它可以安全的把合并的文件保存到版本库。

拷贝-修改-合并模型感觉有一点混乱，但在实践中，通常运行的很平稳，用户可以并行的工作，不必等待别人，当工作在同一个文件上时，也很少会有交迭发生，冲突并不频繁，处理冲突的时间远比等待解锁花费的时间少。

最后，一切都要归结到一条重要的因素：用户交流。当用户交流贫乏，语法和语义的冲突就会增加，没有系统可以强制用户完美的交流，没有系统可以检测语义上的冲突，所以没有任何证据能够承诺锁定系统可以防止冲突。

3.集中式版本控制系统（CVCS）

集中式版本控制系统采用单一的中央代码库模型，所有工作者都要对中央代码库操作，例如 subversion 是一种集中式版本控制系统，采用“拷贝-修改-合并”模型，工作者在中央代码库中拷贝，然后在本地修改，最后将修改合并到中央版本库中。如图 6 所示。

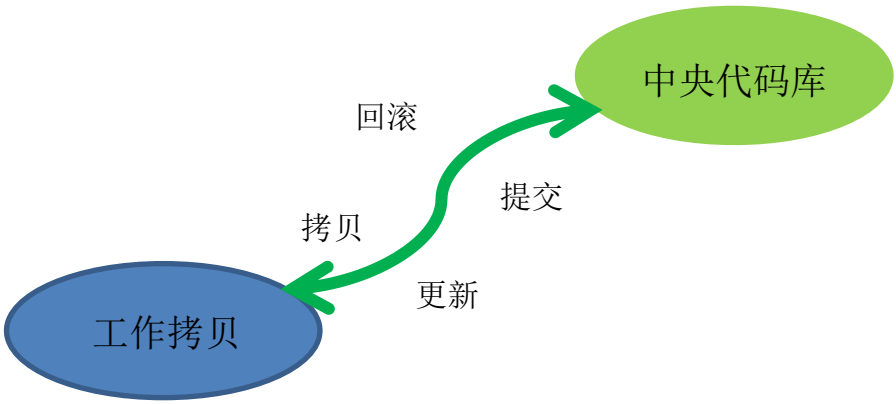


图 6. 集成式版本控制系统工作模型

在上图中我们就能够想到集中式版本控制系统有两个明显的限制：

1、对中央版本库操作比较频繁，受网络速度的影响会特别的严重。

2、单一的中央代码库，如果服务器出现问题导致数据丢失，后果是非常严重的，所以对服务器数据备份也是比较高的要求。

4.分布式版本控制系统（DVCS）

针对 CVCS 的限制，DVCS 逐渐进入了我们的视线，在 DVCS 产品中，比较成熟的产品有 Git、Mercurial 和 Bzr，相比之下 Mercurial 对 Linux，Mac 和 Windows 平台有良好的支持，支持通过 Web 方式访问代码库，并存在成熟的 IntelliJ、Eclipse 插件，所以我们实验室选择了 Mercurial，下文都是以 Mercurial 使用方法展开的。

4.1 Mercurial 的优势

4.1.1 快速可靠

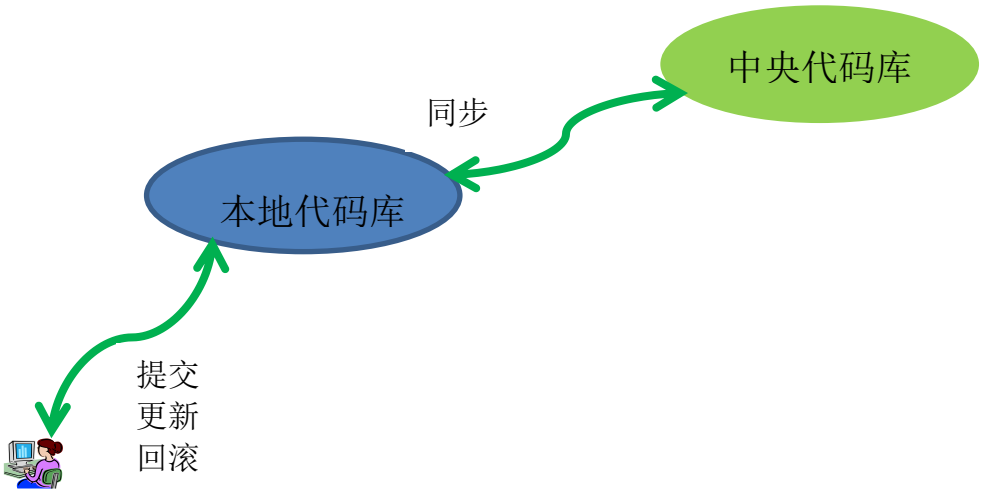


图 7. Mercurial 工作快速性

在图 7 中 Mercurial 带给我们的第一个体验就是快，原因很简单，由于 DVCS 的工作目录与中央仓库（Central Repository）别无二致，同样保存了全部的元数据，那么 Subversion 需要通过网络完成的操作(诸如提交、追溯历史、更新等), Mercurial 可以在离线条件下通过操作本地仓库完成。

通过减少与中央仓库的通信, Mercurial 加快了操作速度，减小了

网络环境对团队的影响，非常符合我们的需求。这种速度和可靠性的提高，对于时刻与版本管理工具打交道的开发者是一种非常愉悦的工作体验。

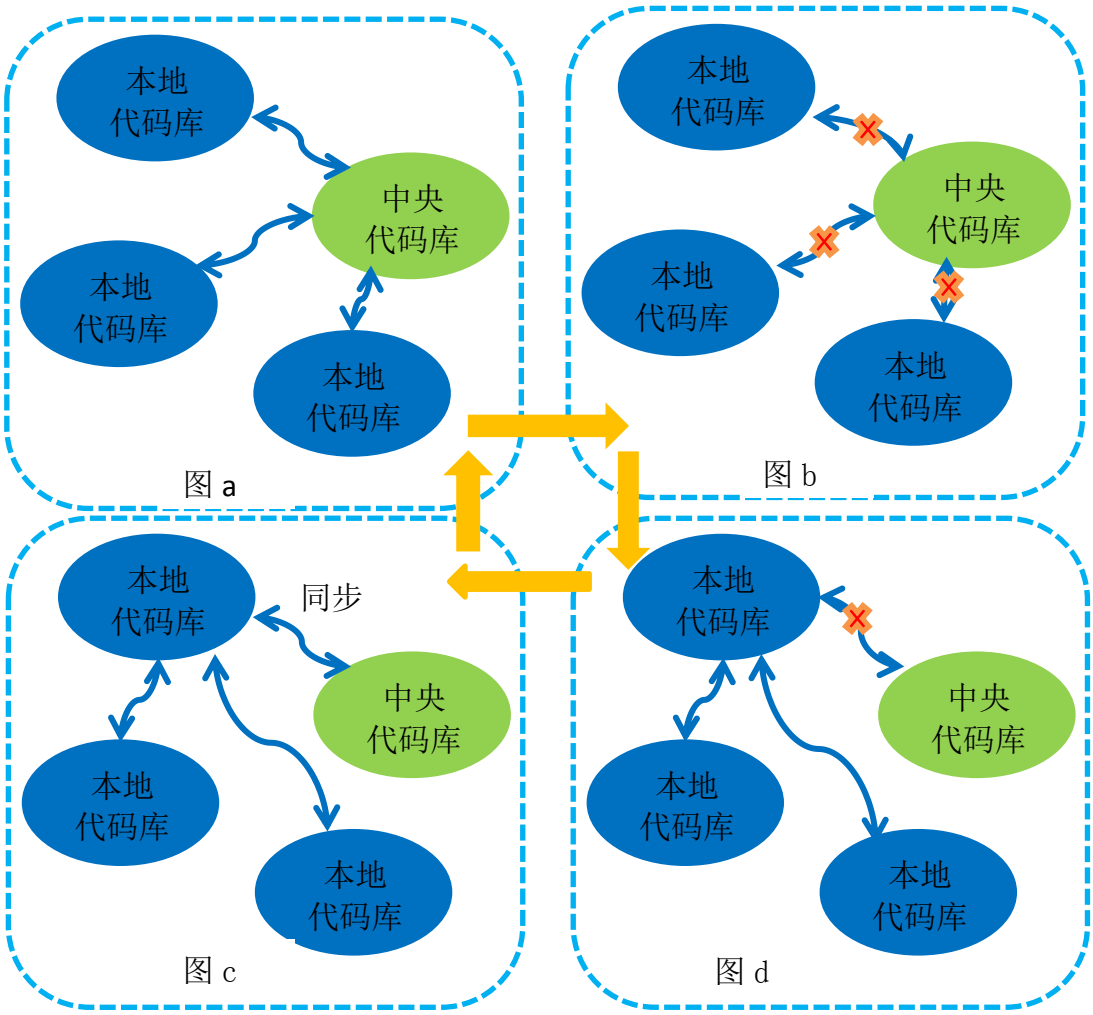


图 7. Mercurial 工作可靠性

通过图 7 可以看出，包含了全部元数据的工作目录可以在中央仓库出现问题时（图 2-b）成为备用仓库（图 2-c），而整个过程只需运行一条命令即可。

这样，在修复中央仓库的过程中，开发团队依然可以通过备用仓库交换修订，日常工作在没有中央仓库的情况下依然可以正常开展，中央仓库恢复后，再将宕机期间所有的修订通过备用仓库同步到中央

仓库上（图 2-d），这套机制可以作为经费和硬件设施有限团队的备份方案。即便中央仓库完全损毁，所造成的损失也非常有限，避免了使用 CVCS 时将“所有鸡蛋放在一个篮子里”的风险。

4.1.2 便于协同工作

我们实验室大部分还在用 win 系统，DVCS 对 linux 和 win 兼容性比较好。工作人员使用不同的操作系统一样可以方便的协同工作。

4.1.3 对小步前进友好

本地仓库的存在，使 Mercurial 对小步前进更加友好 。小步前进意味着开发者在不破坏任何现有功能的前提下，每次修改少量代码并提交。这两个需求让使用集中式版本管理工具的开发者常常处于两难的境地，“不破坏现有功能”与“每次修改少量代码并提交”意味着存在便于分析的细粒度需求以及开发人员必须掌握增量式的对象建模、重构，数据库设计、迁移等技术。难于小步前进体现的是团队成员经验和技术的欠缺，然而解决这些问题不是一朝一夕之功，本地仓库的存在给了开发者更大的自由，允许开发者频繁提交而无需顾忌是否每一次提交都不会破坏现有功能，在代码经过若干次提交到达稳定状态时再与中央数据库同步。通过使用 Mercurial，使得小步前进这个实践得以在团队开展，在大家体会到实践带来的好处后，再追求高质量的小步前进。

5.Mercurial 基础

5.1 Mercurial 安装

Mercurial 提供的二进制安装包适用于现今各种主流的操作系统，一般不会出现兼容性问题。

5.1.1 Windows

在 http://tortoisehg.bitbucket.org/下载 tortoisehg 的最新版本，双击安装即可，安装完成后，在桌面上右键能够出现 TortoiseHg 选项，即表示安装完成。

5.1.2 Linux

linux 版本的工具包中一般都会有 mercurial，命令安装非常方便。

Ubuntu 和 Debian： apt-get install mercurial；

Fedora： yum install mercurial；

OpenSUSE： zypper install mercurial；

Gentoo： emerge mercurial。

5.2 Mercurial 的内置 help

Mercurial 提供了强大的帮助系统，能够帮助我们找到某个命令的用法，我们可以运行一下 hg help；

```
gzh@ouc:~$ hg help
分布式软件配置管理工具 - 水银

命令列表:

 add           add the specified files on the next commit
 addremove     add all new files, delete all missing files
 annotate      show changeset information by line for each file
 archive       create an unversioned archive of a repository revision
 backout       reverse effect of earlier changeset
 bisect        subdivision search of changesets
 branch        set or show the current branch name
 branches      list repository named branches
 bundle        create a changegroup file
```

在 hg help 后我们能能清楚的看到 hg 的命令，当需要查看某个命令的使用方法时，输入“hg help 命令”即可，例如：hg help clone

```
gzh@ouc:~$ hg help clone
hg clone [OPTION]... SOURCE [DEST]

make a copy of an existing repository

    Create a copy of an existing repository in a new directory.

    If no destination directory name is specified, it defaults to the basename
    of the source.

    The location of the source is added to the new repository's .hg/hgrc file,
    as the default to be used for future pulls.

    See "hg help urls" for valid source format details.

    It is possible to specify an "ssh://" URL as the destination, but no
    .hg/hgrc and working directory will be created on the remote side. Please
    see "hg help urls" for important details about "ssh://" URLs.
```

5.3 Mercurial 操作

要学会 mercurial 首先就应该理解 mercurial 基本的使用流程，图 8 就能清晰表示出来。

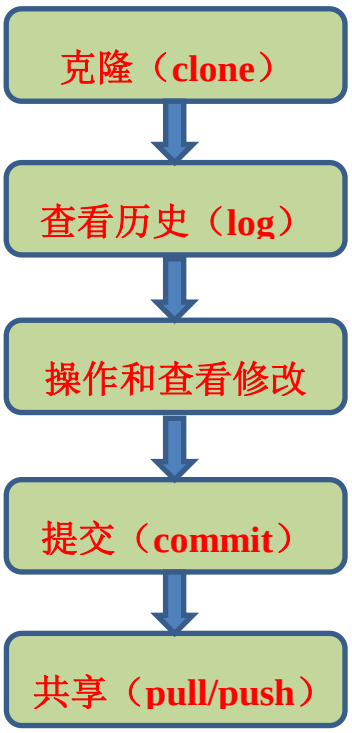


图 8 Mercurial 基本使用流程

5.3.1 克隆

前文讲到 mercurial 是分布式版本控制系统，首先就要从中央代码库中拷贝源代码，然后对本地的版本库进行操作。用到的命令是“hg clone”。服务器中有项目名为 project 的版本，我们以此项目为例展开。

我们用 hg help clone 来查看 clone 命令的用法。

```
gzh@ouc:~$ hg help clone
hg clone [OPTION]... SOURCE [DEST]
```

Mercurial 提供常用的三种访问中央版本库的方式，本地访问、SSH、和 http 访问方式。

SSH:ssh://用户名@IP/PATH

例如：hg clone ssh://gzh@222.195.149.33/tmp/project myproject

现在我们将 project 项目克隆出来。

```
gzh@ouc:~/tmp$ hg clone project myproject
updating to branch default
2 files updated, 0 files merged, 0 files removed, 0 files unresolved
gzh@ouc:~/tmp$ cd myproject/
gzh@ouc:~/tmp/myproject$ ls -al
总用量 20
drwxr-xr-x 4 gzh gzh 4096  2月 20 20:01 .
drwxr-xr-x 9 gzh gzh 4096  2月 20 20:01 ..
drwxr-xr-x 3 gzh gzh 4096  2月 20 20:01 .hg
drwxr-xr-x 2 gzh gzh 4096  2月 20 20:01 Lib
-rw-r--r-- 1 gzh gzh   16  2月 20 20:01 readme.txt
```

能够在克隆的目录中看到.hg 文件，就表示克隆成功了。

5.3.2 查看历史

当我们克隆完项目后，查看版本库的历史应该是我们首先想到的。hg log 就能让我们清晰的看到版本库的修改历史。

```
gzh@ouc:~/tmp/myproject$ hg log
修改集:      1:d2d3cde028fa
标签:        tip
用户:        gzh <gzh@example.com>
日期:        Mon Feb 20 19:07:54 2012 +0800
摘要:        change

修改集:      0:4f2f906d4c75
用户:        gzh <gzh@example.com>
日期:        Mon Feb 20 16:55:51 2012 +0800
摘要:        add function
```

默认情况下，hg log 将会打印出所有版本的变化。

打印结果中的选项：

修改集：“版本号（version）：十六进制字符串（hex）”，这样能够明确此版本中的变化，hex 是独一无二的标识：在一个版本库的各个克隆中，相同的 hex 能够表明相同的变化。Version 是递增的整数，不是独一无二的，相同的 version 在不同的克隆中一般表示不同变化。

用户：标识哪个用户创建的修改，一般包括用户名和邮箱。

日期：标识创建修改的时间。

摘要：标识对修改的描述。

标签：一些版本变化，比如图中上半部分有“标签”选项，标签是标识变化的另一种方式，便于用户记住此次修改。（标签中的 tip 一般标识最新的版本）

到此你可能还没有清晰的理解版本的修改，图 9 能帮助你清晰的理解版本的修改历史。

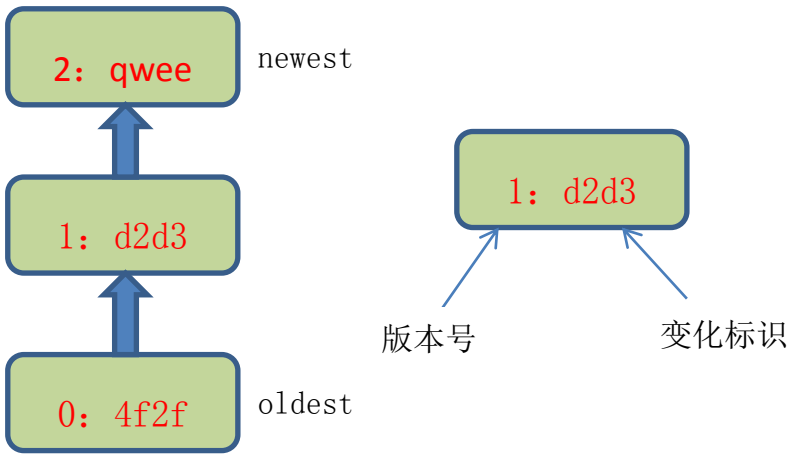


图 9. 版本历史图形表示

5.3.3 操作和查看修改

经过克隆你的电脑应该有了版本库的克隆，现在我们就可以对文件进行修改了。我们来 readme.txt 文件。

```
hello,mercurial
first change
```

添加了“first change”，保存，然后用 hg status 查看修改。

```
gzh@ouc:~/tmp/myproject$ hg status
M readme.txt
```

如果没有对文件修改就用 hg status 的话，默认是没有输出的，现在我们发现输出中的“M readme.txt”，说明 readme.txt 已经修改了，

M 表示还没有提交。在 hg help status 中有详细的讲解。

现在我们也可以用 hg diff 查看详细的修改。

```
gzh@ouc:~/tmp/myproject$ hg diff
diff -r d2d3cde028fa readme.txt
--- a/readme.txt        Mon Feb 20 19:07:54 2012 +0800
+++ b/readme.txt        Mon Feb 20 20:16:16 2012 +0800
@@ -1,1 +1,2 @@
 hello,mercurial
+first change
```

上图打印出了修改的文件，修改的内容。

5.3.4 提交

我们现在就可以把修改提交到本地克隆的版本库了，用 hg commit 提交，hg commit –m “摘要”。你第一次提交时会出现下图的现象。

```
gzh@ouc:~/tmp/myproject$ hg commit -m "first commit"
中止: no username supplied (see "hg help config")
```

会让你设置用户名。在.hg/hgrc 中添加用户名。将 gzh 换成自己的名字。

```
[ui]
username = gzh <gzh@example.com>
```

再进行提交即可，没有打印输出。用 hg log 便可查看是否提交成功。

```
gzh@ouc:~/tmp/myproject$ hg commit -m "first commit"
gzh@ouc:~/tmp/myproject$ hg log
修改集:      2:c9aa1e6c9c87
标签:        tip
用户:        gzh <gzh@example.com>
日期:        Mon Feb 20 20:25:41 2012 +0800
摘要:        first commit

修改集:      1:d2d3cde028fa
用户:        gzh <gzh@example.com>
日期:        Mon Feb 20 19:07:54 2012 +0800
摘要:        change

修改集:      0:4f2f906d4c75
用户:        gzh <gzh@example.com>
日期:        Mon Feb 20 16:55:51 2012 +0800
摘要:        add function
```

产生了新的版本 2，表示提交成功。

当我们要在本地克隆中添加文件或目录时，首先建立文件或目录，然后用“hg add 文件/目录“增加，否则是无法被 mercurial 记录的。下面我们就版本库中增加一个文件 add.txt。

```
gzh@ouc:~/tmp/myproject$ hg add add.txt
gzh@ouc:~/tmp/myproject$ hg status
A add.txt
```

A 表示 add.txt 等待增加。再用 hg commit –m “”提交。

5.3.5 共享

假如我们已经可以把自己的工作可以共享给别的用户使用了，我们就将自己版本库共享到中央代码库中去。用“pull/push”来共享。

首先我们在本地克隆一个代码库 my_test。

```
gzh@ouc:~/tmp$ hg clone myproject my_test
updating to branch default
3 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

我们照上面的步骤再对 myproject 代码作一些变化，然后共享。

下面我们以 push 为例讲解。

在 push 之前我们可以用 hg outgoing 查看我们 push 后版本库的变化。

```
gzh@ouc:~/tmp/myproject$ hg outgoing ../my_test
comparing with ../my_test
正在搜索修改
修改集:      6:2735c5a3c80d
标签:        tip
用户:        gzh <gzh@example.com>
日期:        Tue Feb 21 09:56:16 2012 +0800
摘要:        push
```

上图表明，当我们 push 版本后，版本库的信息就变为上图打印出的信息了。

```
gzh@ouc:~/tmp/myproject$ hg push ../my_test
正在推到 ../my_test
正在搜索修改
正在增加修改集
正在增加清单
正在增加文件改变
已增加 1 个修改集，包含 1 个改变，修改了 1 个文件
```

这样就表示已经将本地推到中央代码库了。但是还需要“hg update”才能将中央代码库更新。

```
gzh@ouc:~/tmp/project$ hg update
2 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

我们或许认为 update 这一步是没有不必要的，但是正是因为有了 update，就能帮助我们方便的回到以前的版本。例如下图：

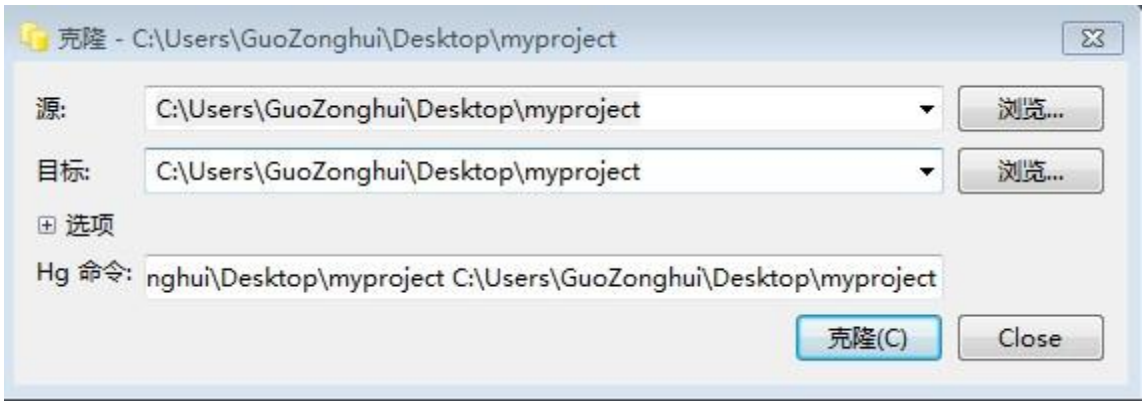
```
gzh@ouc:~/tmp/myproject$ hg update 2
1 files updated, 0 files merged, 1 files removed, 0 files unresolved
```

现在版本库中就是以前版本 2 时的内容了。我们可以用 hg update 回来最新的版本。

5.3.5 Windows 下 tortoise 的使用

如果你能够理解了上面的内容，tortoise 在 windows 下的使用就很容易了。

新建一个存放代码的文件夹 myproject，在 myproject 下右键选择克隆，



我们现在把服务器上的版本库克隆出来，源的地址就是：ssh://gzh@222.195.149.33/home/gzh/tmp/hg　密码是 tank，点击克隆，文件夹 myproject 就会变为 

，内部就是 

。

然后按照 mercurial 操作流程，就学会使用了。

6.总结

版本控制系统不仅可以用于代码的管理，也可以管理 word 等文件，因为文件是以二进制的形式存储在版本控制系统中的。

大家有时间可以在 http://hgbook.red-bean.com/read/index.html 深入学习 mercurial。