哈尔滨工业大学计算机科学与技术学院

软件系统设计与开发实践
（2016年春季）

刘铭

liuming1981@hit.edu.cn

2016年6月2日

4. 版本控制软件

版本控制软件

- 1.版本控制软件介绍
- 2.SVN使用简介和Eclipse中使用SVN
- 3.github的使用简介和Eclipse中使用github

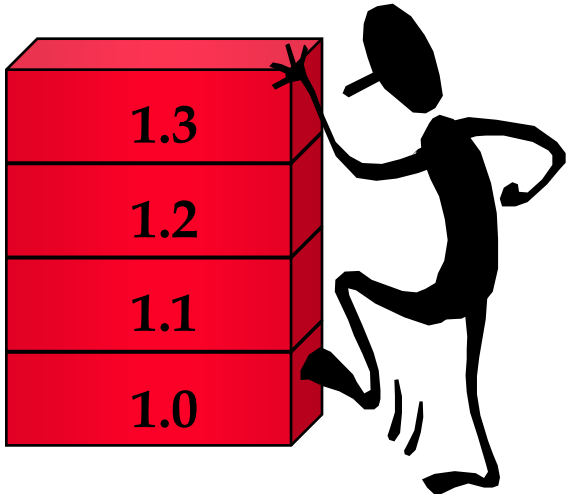
为什么需要版本控制

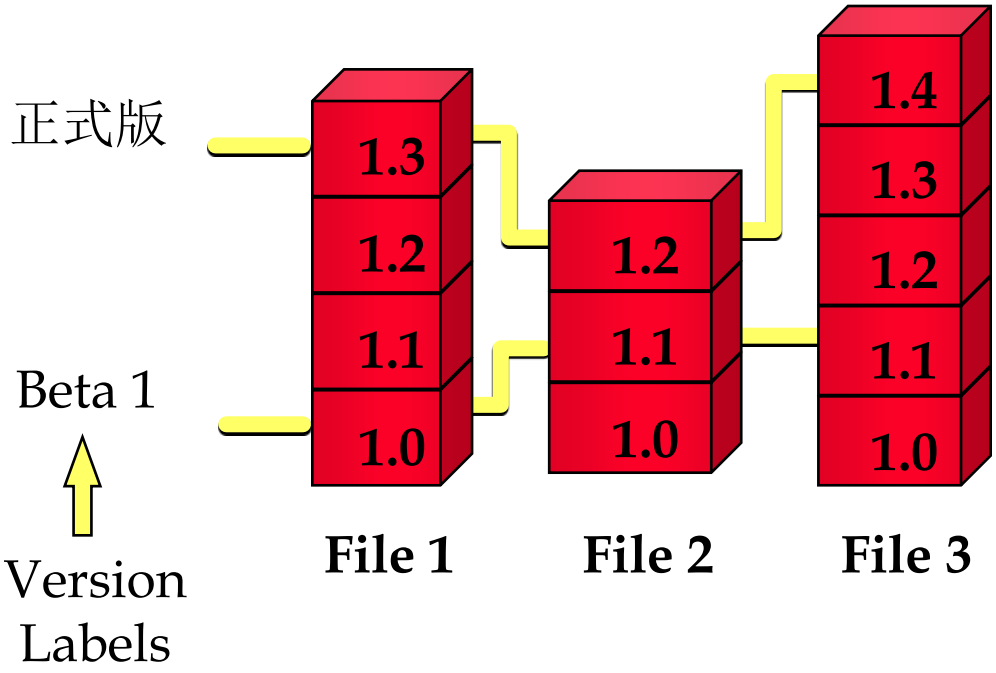
- “即使你的团队只有你一个人，你的项目只需一周的时间；即使是“用过就”的原型；即使你的工作并非源码；确保每样东西都处在源码控制之下--文档、电话号码表、给供应商的备忘录、makefile、构建与发布流程、烧制CD的shell 脚本--每样东西。”

为什么需要版本控制

- 及时了解团队中其他成员的进度；
- 轻松比较不同版本间的细微差别；
- 记录每个文件成长的每步细节，利于成果的复用(reuse)；
- 资料共享，避免以往靠邮件发送文件造成的版本混乱；
- 人人为我，我为人人。所有成员维护的实际是同一个版本库，无需专人维护所有文件的最新版本；
- 协同工作，大大提高团队工作效率，无论团队成员分布在天涯还是海角；

版本控制


1.3
1.2
1.1
1.0


正式版
Beta 1
Version Labels
1.3
1.2
1.1
1.0
1.2
1.1
1.0
1.4
1.3
1.2
1.1
1.0
File 1
File 2
File 3

主流的版本控制软件

- 版本控制软件(VCS，Version Control System)
  - VSS (Microsoft Visual SourceSafe)
  - CVS (Concurrent Version System)
  - IBM Rational ClearCase
  - SVN (Subversion)
  - Git
  - ……

VCS的发展—手工方式

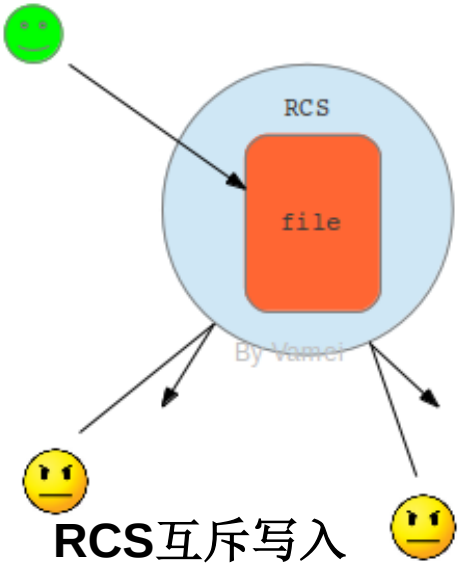
- 早期的软件开发大部分是愉快的个人创作。比如UNIX下的sed是L.E.McMahon写的，Python的第一个编译器是Guido写的，Linux最初的内核是Linus写的。
- 这些程序员可以用手工的方式进行备份，并以注释或者新建文本文件来记录变动。

VCS的发展—互斥写入方式

- 19世纪70年代末80年代初，VCS的概念已经存在，比如UNIX平台的RCS (Revision Control System)。
- RCS对文件进行集中式管理，主要目的是避免多人合作情况下可能出现的冲突。
  - 如果多用户同时写入同一个文件，其写入结果可能相互混合和覆盖，从而造成结果的混乱。
- RCS允许多个用户同时读取文件，但只允许一个用户锁定(locking)并写入文件 。
  - 当一个程序员登出(check-out)某个文件并对文件进行修改的时候。只有在这个程序完成修改，并登入(check-in)文件时，其他程序员才能登出文件。
- 用户可以对原文件进行许多改变 (change, or file delta)。一旦重新登入文件，这些改变将保存到RCS系统中。通过check-in将改变永久化的过程叫做提交(commit)。

VCS的发展--互斥写入方式

- RCS的互斥写入机制避免了多人同时修改同一个文件的可能。
- 但代价是程序员长时间的等待，给团队合作带来不便。如果某个程序员登出了某个文件，而忘记登入，那他就要面对队友的怒火了。（从这个角度上来说，RCS造成的问题甚至大于它所解决的问题……）



RCS互斥写入

VCS的发展--互斥写入方式

- 文件每次commit都会创造一个新的版本(revision)。
- RCS给每个文件创建了一个追踪文档来记录版本的历史。这个文档的名字通常是原文件名加后缀,v (比如main.c的追踪文档为main.c,v)。
- 追踪文档中包括：最新版本的文件内容，每次check-in的发生时间和用户，每次check-in发生的改变。在最新文档内容的基础上，减去历史上发生的改变，就可以恢复到之前的历史版本。这样，RCS就实现了备份历史和记录改变的功能。

RCS历史版本追踪

VCS的发展--互斥写入方式

- 相对与后来的版本管理软件，**RCS**纯粹线性的开发方式非常不利于团队合作。但**RCS**为多用户写入冲突提供了一种有效的解决方案。**RCS**的版本管理功能逐渐被其他软件**(**比如**CVS)**取代。

VCS的发展--CVS

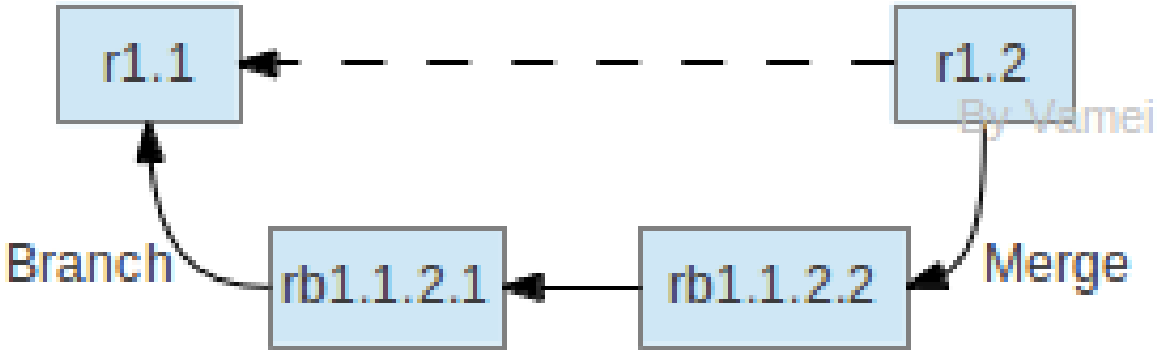
- 1986年，Dick Grune写了一系列的shell脚本用于版本管理，并最终以这些脚本为基础，构成了CVS (Concurrent Versions System)。CVS后来用C语言重写。
- CVS是开源软件。CVS被包含在GNU的软件包中，并因此得到广泛的推广，最终击败诸多商业版本的VCS，呈一统天下之势。
- CVS继承了RCS的集中管理的理念。
- 在CVS管理下的文件构成一个库(repository)。与RCS的锁定文件模式不同，CVS采用复制-修改-合并(copy-modify-merge)的模式，来实现多线开发。
- CVS引进了分支(branch) 的概念。多个用户可以从主干(也就是中心库)创建分支。分支是主干文件在本地复制的副本。用户对本地副本进行修改。用户可以在分支提交(commit)多次修改。用户在分支的工作结束之后，需要将分支合并到主干中，以便让其他人看到自己的改动。

VCS的发展--CVS

- 所谓的合并，就是CVS将分支上发生的变化应用到主干的原文件 上。比如下面的过程中，我们从r1.1分支出rb1.1.2.*，并最终合并回主干，构成r1.2。
- 在合并的过程中，CVS将两个change应用于r1.1，就得到了r1.2:

r1.2 = r1.1 + change(rb1.1.2.2 - rb1.1.2.1) + change(rb1.1.2.1-r1.1)

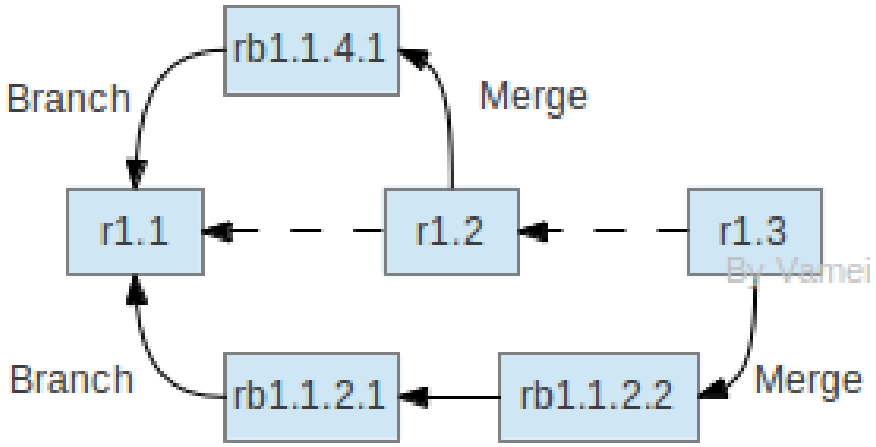
- 注：箭



copy-modify-merge

VCS的发展--CVS

- 在多用户情况下，可以创建多个分支进行开发。
- 在这样的多分支合并的情况下，有可能出现冲突(colliding)。
- 比如图中，第一次合并和第二次合并都对r1.1文件的同一行进行了修改，那么r1.3将不知道如何去修改这一行（第二次合并比图示的要更复杂一些，分支需要先将主干拉到本地，合并过之后传回主干)。
- CVS要求冲突发生时的用户手动解决冲突。用户可以调用编辑器，对文件发生合并冲突的地方进行修改，以决定最终版本(r1.3)的内容。

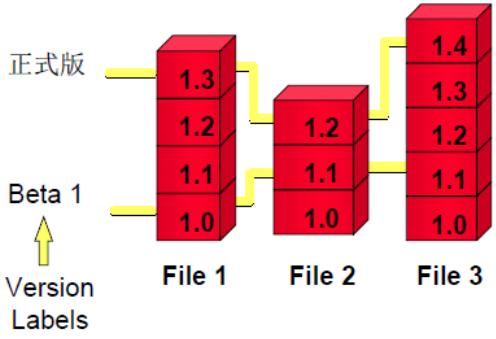


VCS的发展--CVS

- CVS管理下的每个文件都有一系列独立的版本号(比如上面的r1.1,r1.2,r1.3)。但每个项目中往往包含有许多文件。CVS用标签(tag)来记录一个集合，这个集合中的元素是一对(文件名：版本号)。比如我们的项目中有三个文件(file1, file2, file3)，我们创建一个v1.0的标签：

```
tag v1.0 (file1:r1.3) (file2:r1.1) (file3:r1.5)
```

- v1.0的tag中包括了r1.3版本的文件file1，r1.1版本的file2…… 一个项目在发布(release)的时候，往往要发布多个文件。标签可以用来记录该次发布的时候，是哪些版本的文件被发布。



VCS的发展--CVS

- **CVS应用在许多重要的开源项目上。在90年代和00年代初，CVS在开源世界几乎不二选。尽管CVS已经长达数年没有发布新版本，我们依然可以在许多项目中看到CVS的身影。**

- **CVS也有许多常常被人诟病的地方，比如下面几条：**
  - 合并不是原子操作(atomic operation)：如果有两个用户同时合并，那么合并结果将是某种错乱的混合体。如果合并的过程中取消合并，不能撤销已经应用的改变。
  - 文件的附加信息没有被追踪：一旦纳入CVS的管理，文件的附加信息(比如上次读取时间)就被固定了。CVS不追踪它所管理文件的附加信息的变化。
  - 主要用于管理ASCII文件：不能方便的管理Binary文件和Unicode文件。
  - 分支与合并需要耗费大量的时间：CVS的分支和合并非常昂贵。分支需要复制，合并需要计算所有的改变并应用到主干。因此，CVS鼓励尽早合并分支。

VCS的发展--SVN

- 随着时间，人们对CVS的一些问题越来越感到不满（而且程序员喜欢新鲜的东西），Subversion应运而生。Subversion的开发者Karl Fogel和Jim Blandy是长期的CVS用户。赞助开发的CollabNet, Inc.希望他们写一个CVS的替代VCS。这个VCS应该有类似于CVS的工作方式，但对CVS的缺陷进行改进，并提供一些CVS缺失的功能。
- 总体上说，SVN在许多方面沿袭CVS，也是集中管理库，通过记录改变来追踪历史，允许分支和合并，但并不鼓励过多分支。
- SVN在一些方面得到改善。SVN的合并是原子操作。它可以追踪文件的附加信息，并能够同样的管理Binary和Unicode文件。

VCS的发展--SVN

- **CVS和SVN的主要不同：**
  - 与CVS的,v文件存储模式不同，SVN采用关系型数据库来存储改变集。VCS相关数据变得不透明。
  - CVS中的版本是针对某个文件的，CVS中每次commit生成一个文件的新版本。SVN中的版本是针对整个文件系统的(包含多个文件以及文件组织方式)，每次commit生成一个整个项目文件系统树的新版本。
  - Subversion依赖类似于硬连接(hard link)的方式来提高效率，避免过多的复制文件本身。Subversion不会从库下载整个主干到本地，而只是下载主干的最新版本。

VCS的发展--SVN

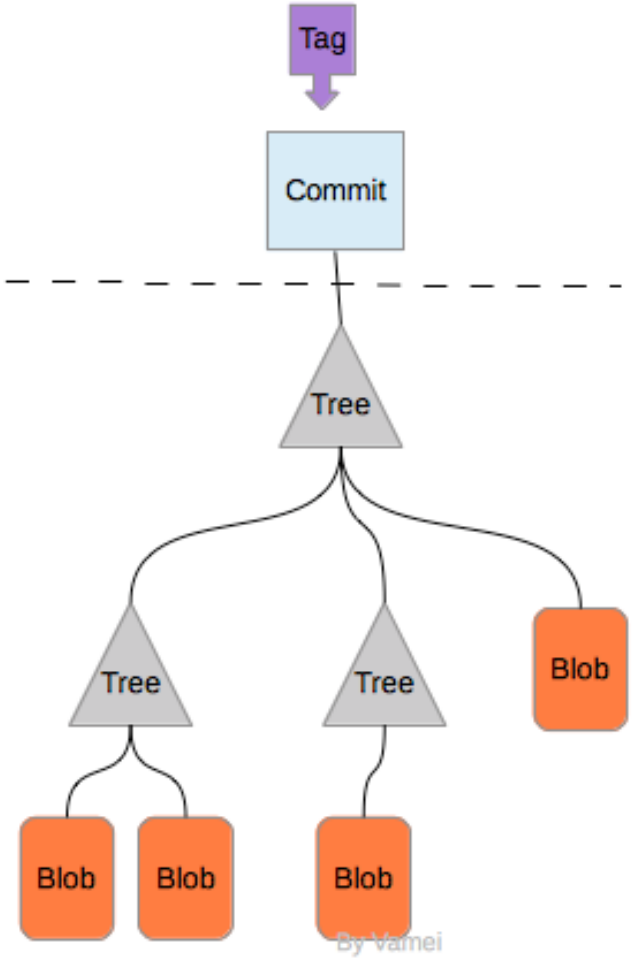
- 在SVN刚刚诞生的时候，来自CVS用户的抱怨不断。但随着时间的推移，SVN逐渐推广（SVN已经是Apache中自带的一个模块了，SVN应用于GCC、SourceForge，新浪APP Engine等项目），并依然有活跃的开发，而CVS则逐渐沉寂。事实上，许多UNIX的参考书的新版本中，都缩减甚至删除了CVS的内容。

VCS的发展--git

- git的作者是Linus Torvald，就是写Linux Kernel的人。
- Linus Torvald本人相当厌恶CVS(以及SVN)。然而，操作系统内核是复杂而庞大的代码“怪兽” （2012年的Linux Kernel有1500万行代码，Windows的代码不公开，估计远远超过这一数目）。
- Linux内核小组最初使用.tar文件来管理内核代码，但这远远无法匹配Linux内核代码的增长速度。
- Linus转而使用BitKeeper作为开发的VCS工具。BitKeeper是一款分布式的VCS工具，它可以快速的进行分支和合并。然而由于使用证书方面的争议(BitKeeper是闭源软件，但给Linux内核开发人员发放免费的使用证书)。
- Linus最终决定写一款开源的分布式VCS软件：git。
- 例子：12306的抢票插件拖垮了GitHub (GitHub基于git)

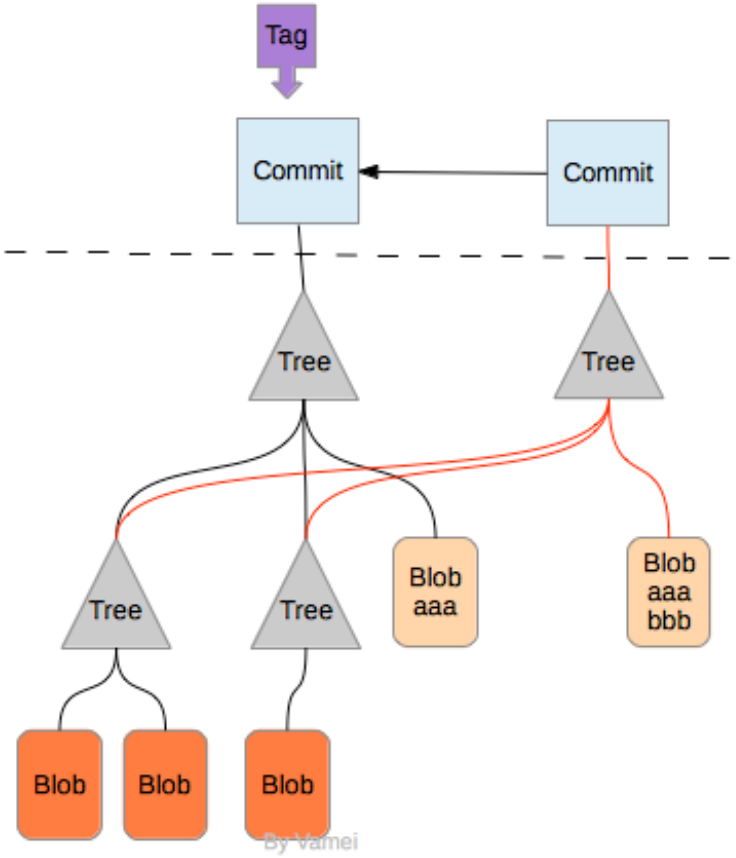
VCS的发展--git

- **对于一个开发项目，git会保存blob, tree, commit和tag四种对象。**
  - 文件被保存为blob对象。
  - 文件夹被保存为tree对象。tree对象保存有指向文件或者其他tree对象指针。
- **上面两个对象类似于一个UNIX的文件系统，构成了一个文件系统树。**
  - 一个commit对象代表了某次提交，它保存有修改人，修改时间和附加信息，并指向一个文件树。这一点与SVN类似，即每次提交为一个文件系统树。
  - 一个tag对象包含有tag的名字，并指向一个commit对象。
- **虚线下面的对象构成了一个文件系统树。在git中，一次commit实际上就是一次对文件系统树的快照(snapshot)。**



VCS的发展--git

- 在整个开发过程中，可能会有许多次提交(commit)。每次commit的时候，git并不总是复制所有的对象。git会检验所有对象的HASH值。如果该对象的HASH值已经存在，说明该对象已经保存过，并且没有发生改变，所以git只需要调整新建tree或者commit中的指针，让它们指向已经保存过的对象就可以了。
- git在commit的时候，只会新建HASH值发生改变的对象。如图所示，我们创建新的commit的时候，只需要新建一个commit对象，一个tree对象和一个blob对象就足够了，而不需要新建整个文件系统树。

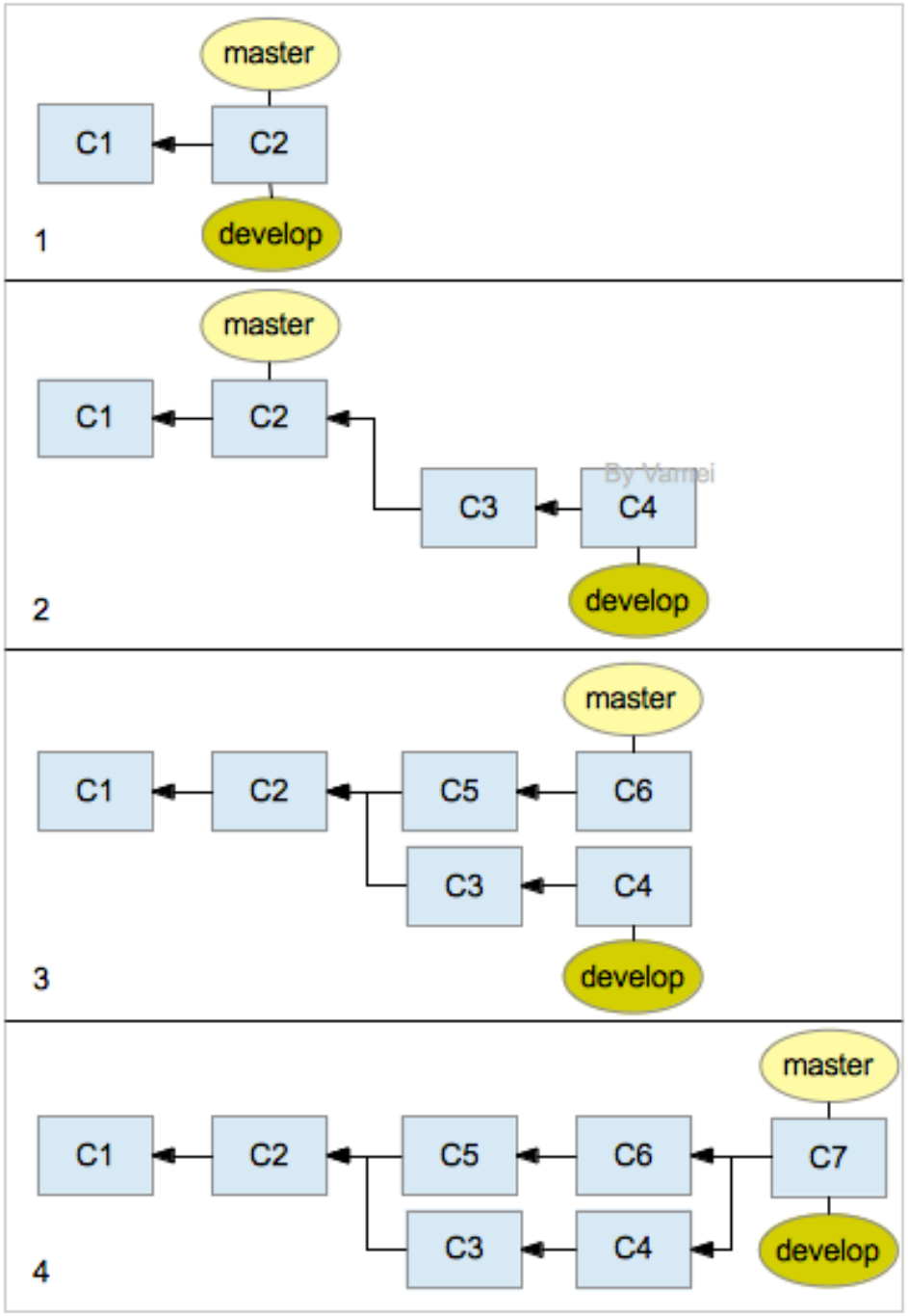


VCS的发展--git

- 可以看到，与CVS，SVN保存改变(file delta)的方式形成对照，git保存的不是改变，而是此时的文件本身。由于不需要遵循改变路径来计算历史版本，所以git可以快速的查阅历史版本。git可以直接提取两个commit所保存的文件系统树，并迅速的算出两个commit之间的改变。
- 同样由于上面的数据结构，git可以很方便的创建分支(branch)。实际上，git的一个分支是一个指向某个commit的指针。合并时，git检查两个分支所指的两个commit，并找到它们共同的祖先commit。git会分别计算每个commit与祖先发生的改变，然后将两个改变合并(同样，针对同一行的两个改变可能发生冲突，需要手工解决冲突)。整个过程中，不需要复制和遵循路径计算总的改变，所以效率提高很多。

VCS的发展--git

- 比如图1中有两个分支，一个master和一个develop。
- 我们先沿着develop分支工作，并进行了两次提交(比如修正bug1)，而master分支保持不变。
- 随后沿着master分支,进行了两次提交(比如增加输入功能)，develop保持不变。
- 在最终进行图4中的合并时，我们只需要将C4-C2和C6-C2的两个改变合并，并作用在C2上，就可以得到合并后的C7。合并之后，两个分支都指向C7。我们此时可以删除不需要的分支develop。



VCS的发展--git

- 由于git创建、合并和删除分支的成本极为低廉，所以git鼓励根据需要创建多个分支。
- 实际上，如果分支位于不同的站点(site)，属于不同的开发者，那么就构成了分布式的多分支开发模式。每个开发者都在本地复制有自己的库，并可以基于本地库创建多个本地分支工作。开发者可以在需要的时候，选取某个本地分支与远程分支合并。git可以方便的建立一个分布式的小型开发团队。比如我和朋友两人各有一个库，各自开发，并相互拉对方的库到本地库合并(如果上面master，develop代表了两个属于不同用户的分支，就代表了这一情况)。当然，git也允许集中式的公共仓库存在，或者多层的公共仓库，每个仓库享有不同的优先级。git的优势不在于引进了某种开发模式，而是给了你设计开发模式的自由。

VCS的发展--git

- 需要注意的是，GitHub尽管以git为核心，但并不是Linus创建的。事实上，Linus不接收来自GitHub的Pull Request。Linus本人将此归罪于GitHub糟糕的Web UI。但正是GitHub的Web页面让许多新手熟悉并开始使用git。

VCS的发展—三者比较总结

- VCS主流的三个工具还没有决出最终胜负
- 或许SVN会继续在一些重要项目上发挥作用
- 或许git会最终一统江山
- 或许CVS可以有新的发展并最终逆袭
- 又或许，一款新的VCS将取代所有的前辈。

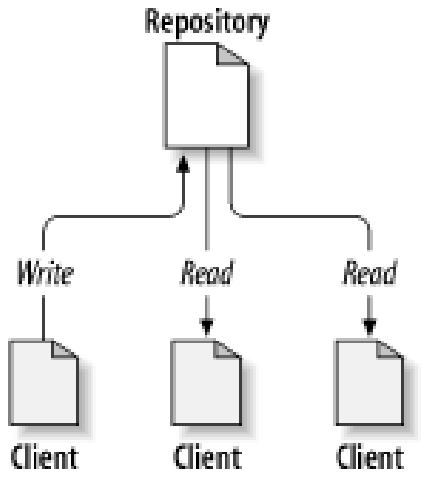
- 一款优秀的VCS可以提高了我们管理项目的能力，降低我们犯错所可能支付的代价。随着开发项目越来越庞大和复杂，这一能力变得越来越不可缺少。花一点时间学习VCS，并习惯在工作中使用VCS，将会有意想不到的回报。

版本控制软件

- 1.版本控制软件介绍
- 2.SVN使用简介和Eclipse中使用SVN
- 3.github的使用简介和Eclipse中使用github

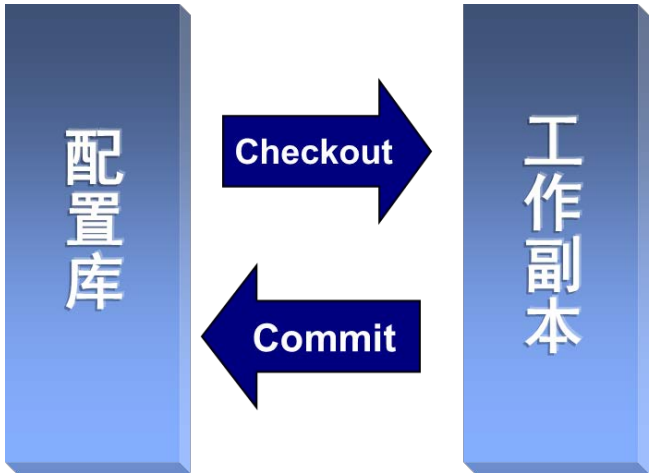
SVN的客户/服务器架构

- 版本库（Repository）：SVN的核心是版本，储存所有的数据，版本库按照文件树形式储存数据－包括文件和目录，任意数量的客户端可以连接到配置库，读写这些文件。通过写数据，别人可以看到这些信息；通过读数据，可以看到别人的修改。
- 经过授权的客户端可以连接到版本库，读写库中的文件
- 版本库和普通文件服务器的不同：版本库会记录每一次的更改，所以，客户端可以任意查询更改的历史。
- 例如：ApplicationContext.java的1451版和1450版相比修改了什么？谁做的修改？什么时候做的修改？等等



SVN的客户/服务器架构

- 工作副本（WorkSpace）
  - 与位于中央配置库相对应的是每个人的工作空间，它是每个程序员工作的地方，程序员从配置库拿到源代码，放在本地作为工作副本，在工作副本上进行查看、修改、编译、运行、测试等操作，并把新版本的代码从这里提交回配置库中。

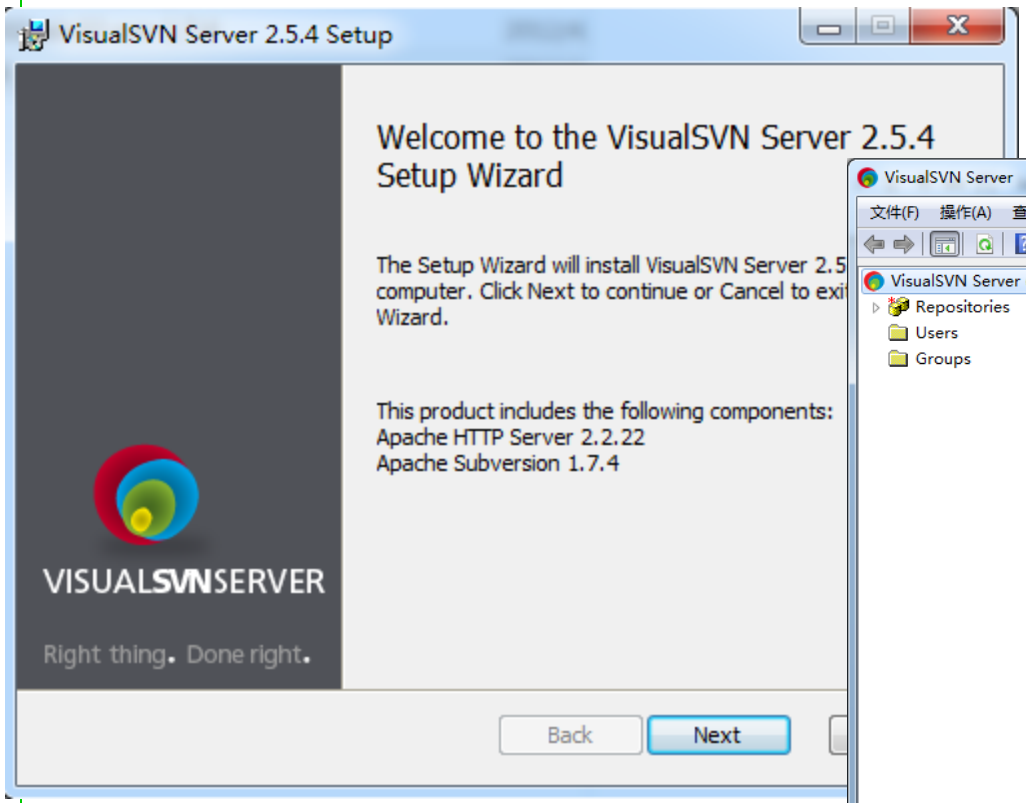


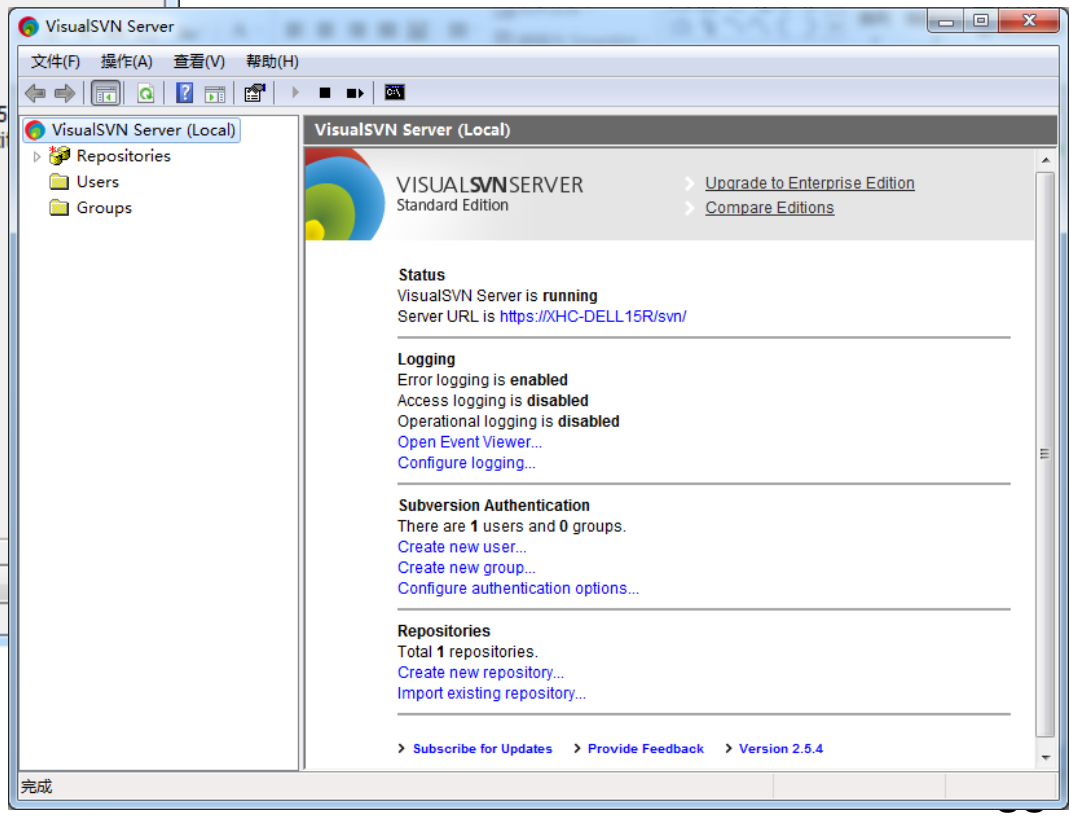
SVN的相关软件

- SVN需要安装server和client软件 ftp://172.16.0.46
- Subversion server
  - 下载地址：http://subversion.apache.org/packages.html 注意操作系统和版本号
  - Server安装后可以通过命令行进行操控
  - Server也可以通过第三方的GUI控制界面进行控制，如VisualSVN
- SVN Client －TortoiseSVN
  - TortoiseSVN：SVN的客户端工具，和资源管理器完美集成，易于使用。
  - http://tortoisesvn.net/downloads.html 安装程序和中文语言包

SVN server的安装

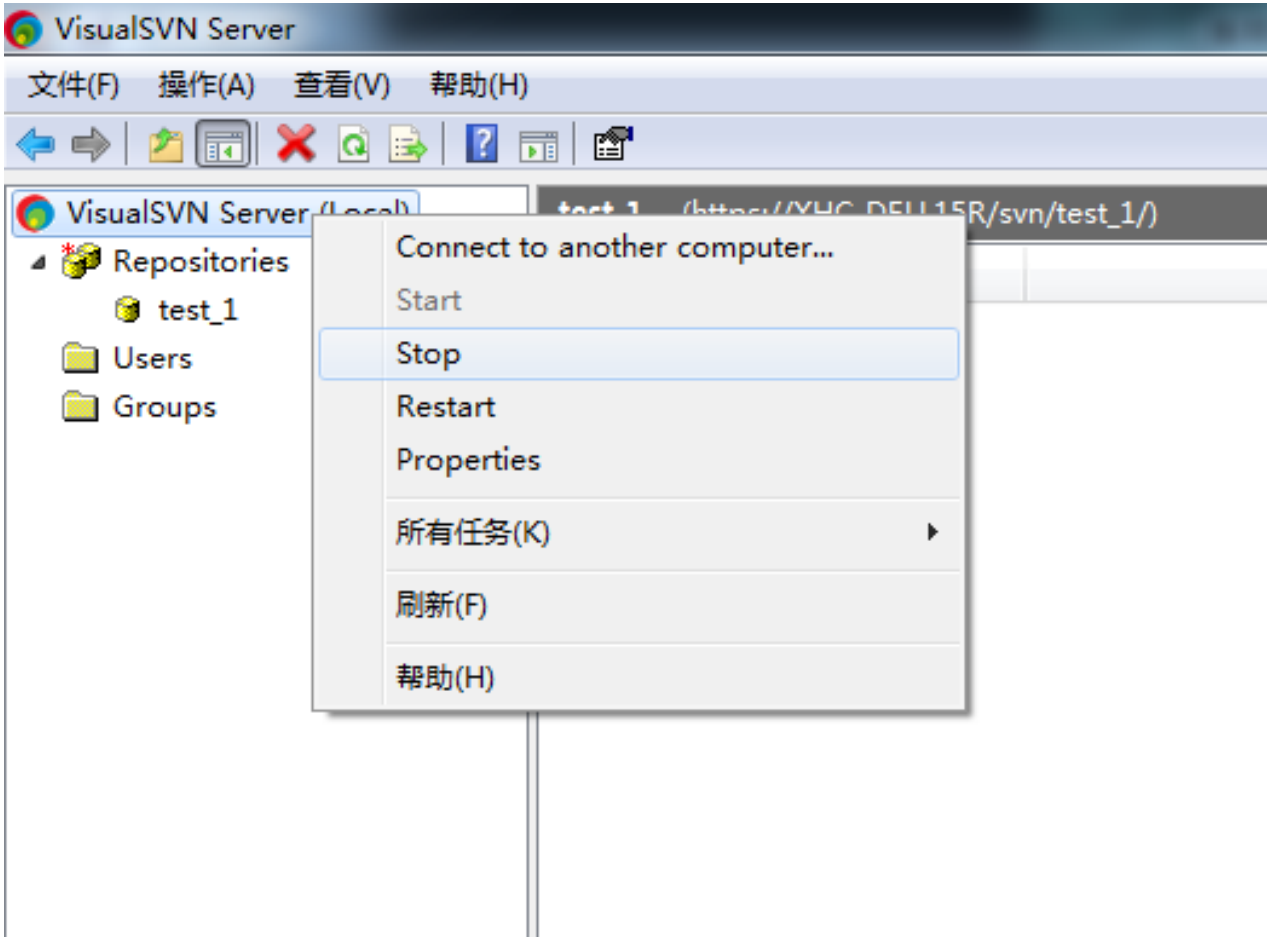
- **SVN server 软件 –VisualSVN**





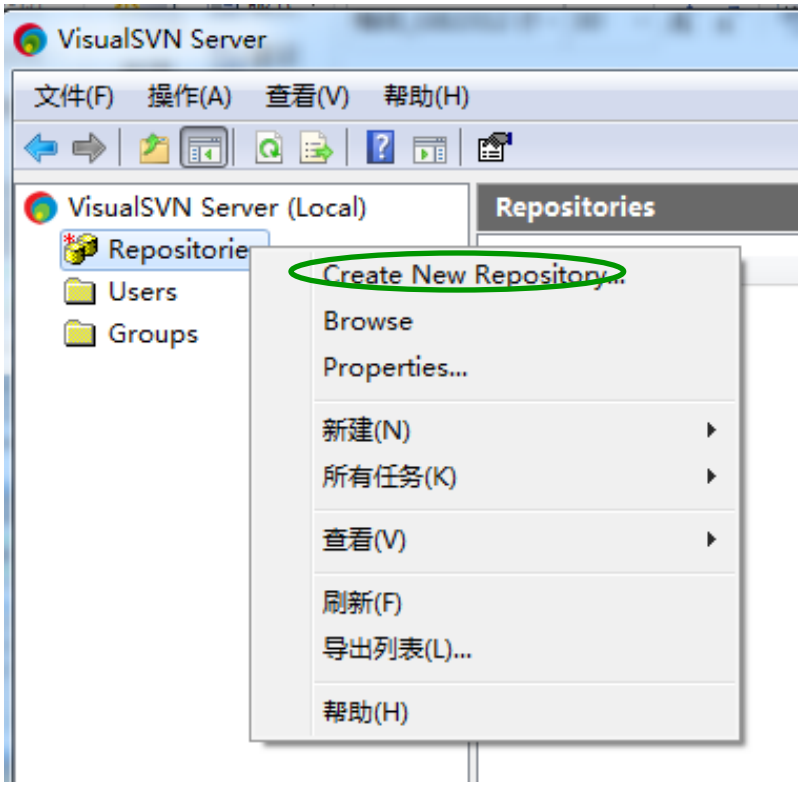
VisualSVN的设置

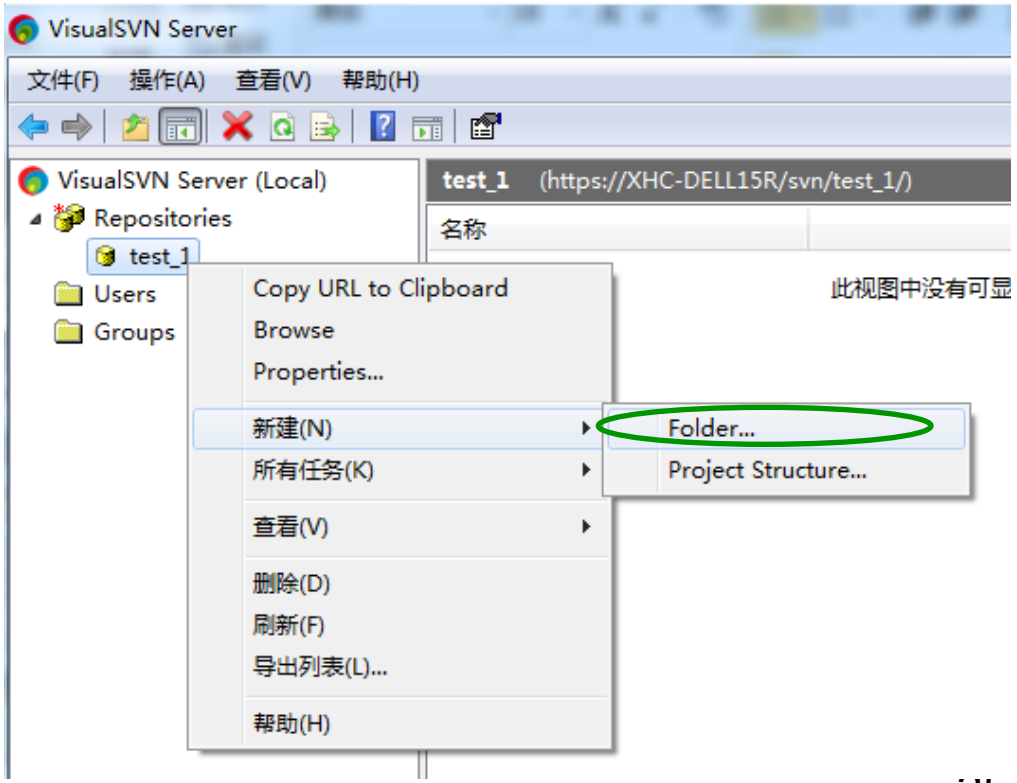
- 启动/关闭Server



VisualSVN的设置

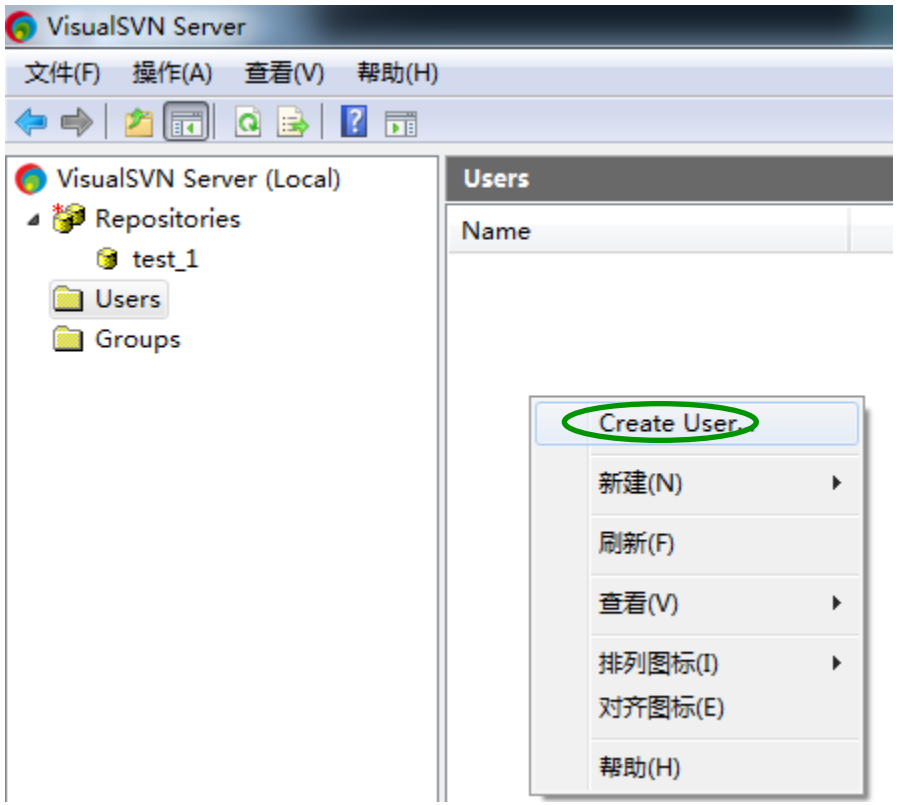
- 创建版本库 （Create New Repository）
- 在所建版本库中可以根据需要创建子目录

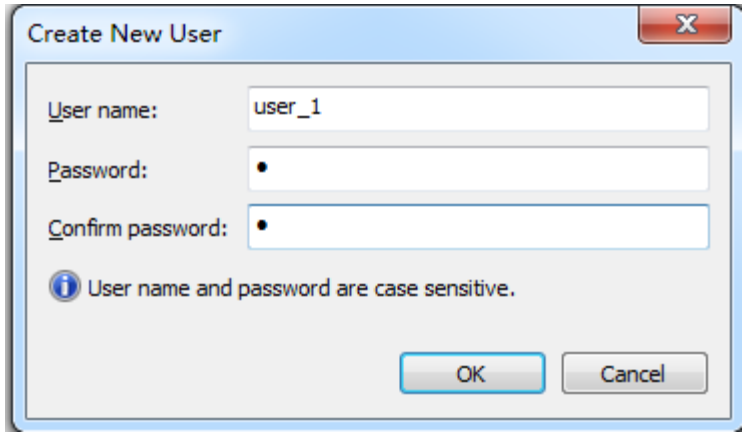




VisualSVN的设置

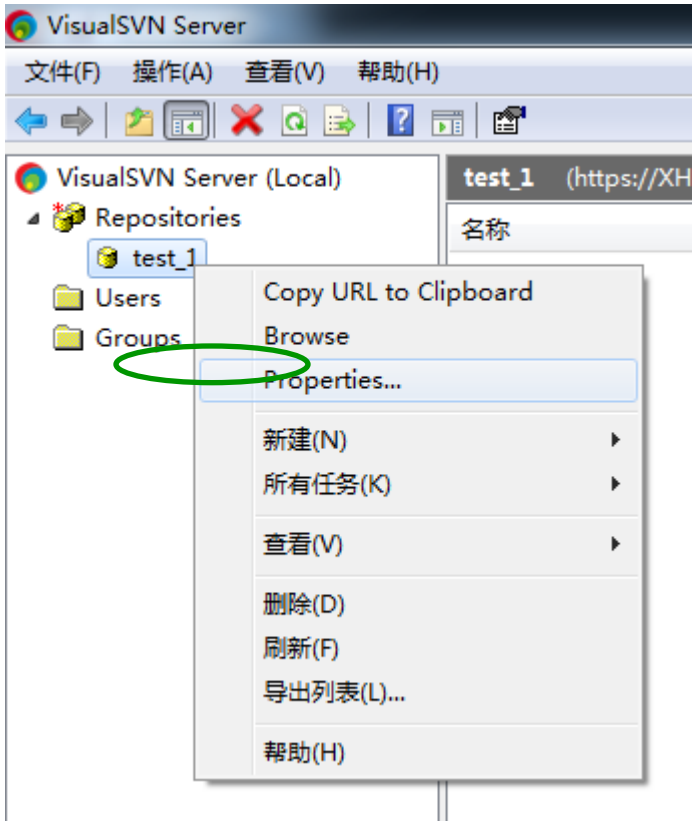
- 创建用户(Create User)

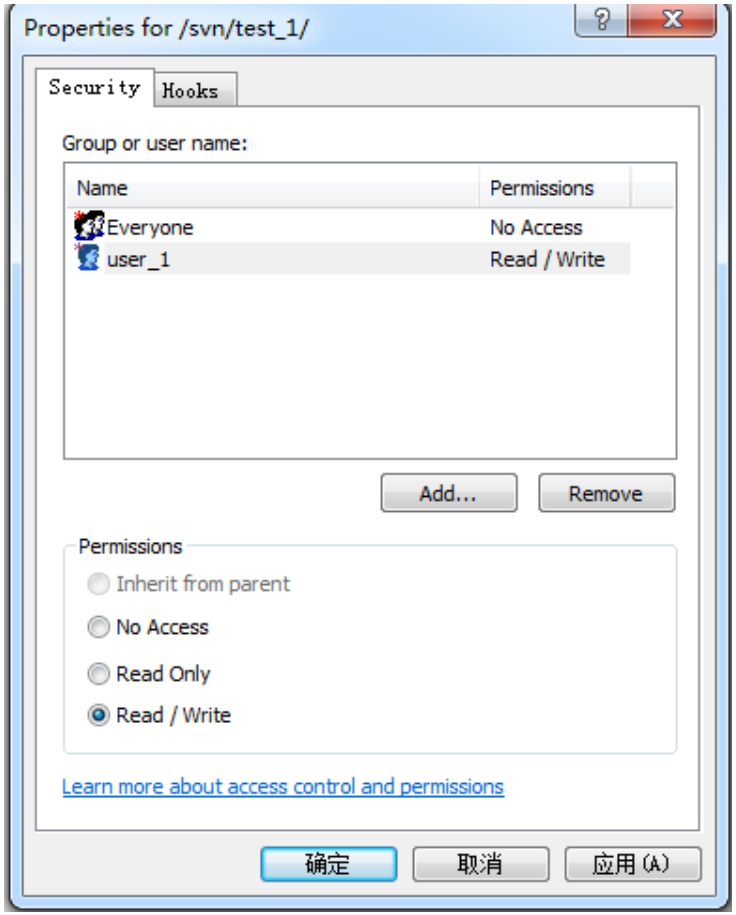




VisualSVN的设置

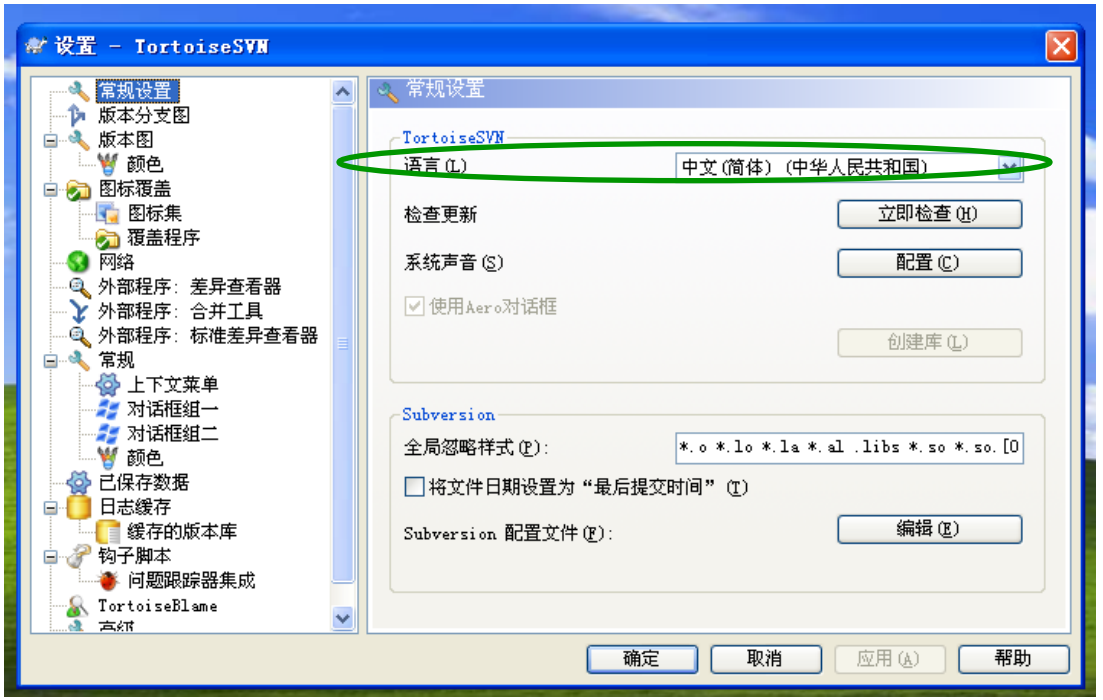
- 指定配置库的访问权限
- 建议：取消“Everyone”用户的权限，添加自建用户的权限为“Read/Write”





SVN Client －TortoiseSVN

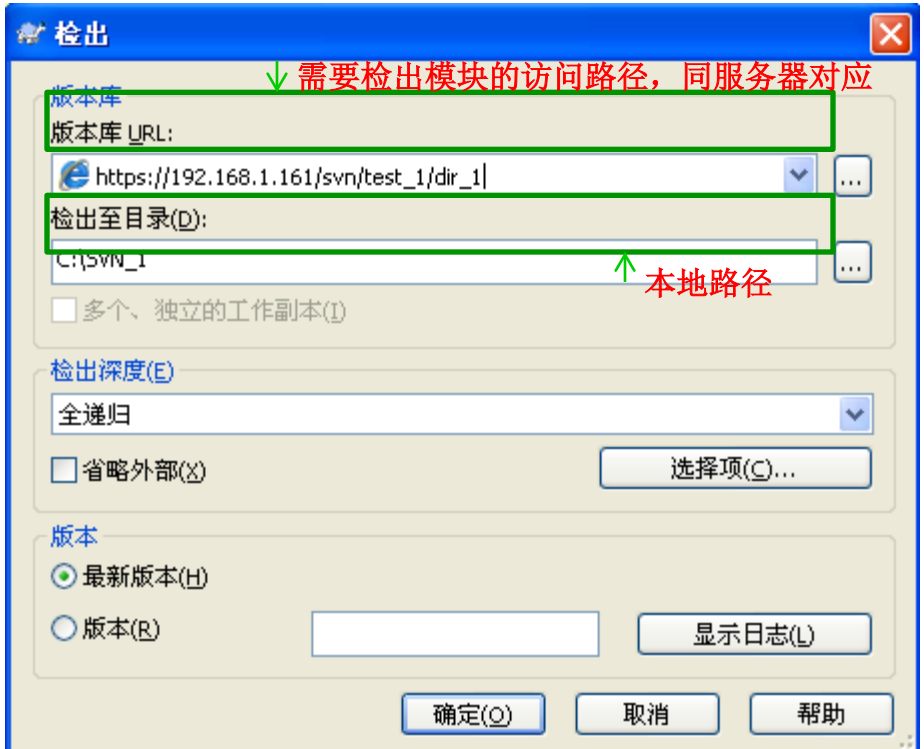
- 安装时按照提示安装即可。
- 如果需要可安装中文语言包。

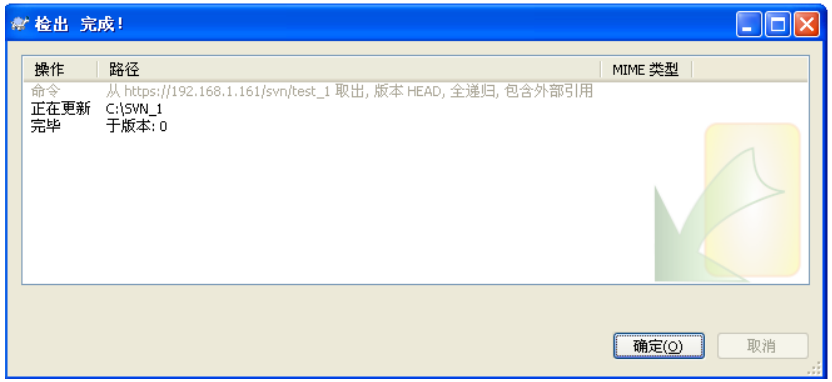


CheckOut（检出）

- 作用：将版本库中的内容检出到本地工作副本
- 步骤：

1. 新建一个空文件夹； 比如：C:\SVN_1
2. 在此目录中点击右键-> SVN Checkout...





CheckOut的注意项

- **检出深度：**
  - 全递归(默认选择)
    - 检出完整的目录树，包含所有的文件或子目录。
  - 直接节点，包含目录
    - 检出目录，包含其中的文件或子目录，但是不递归展开子目录。
  - 仅文件子节点
    - 检出指定目录，包含所有文件，但是不检出任何子目录。
  - 仅此项
    - 只检出目录。不包含其中的文件或子目录。

省略外部引用：如果项目含有外部项目的引用，而这些引用我们不希望同时检出，请选中忽略外部项目复选框。如果选中了这个复选框，更新的时候要使用命令”更新至版本Update to Revision...”

ADD(添加)

- 选中文件/文件夹（在新文件/文件夹所在父文件夹点击右键），
- 在菜单中选择“添加Add”命令。不需要受SVN控制的文件请取消打钩。

Add to Ignore List （忽略文件）

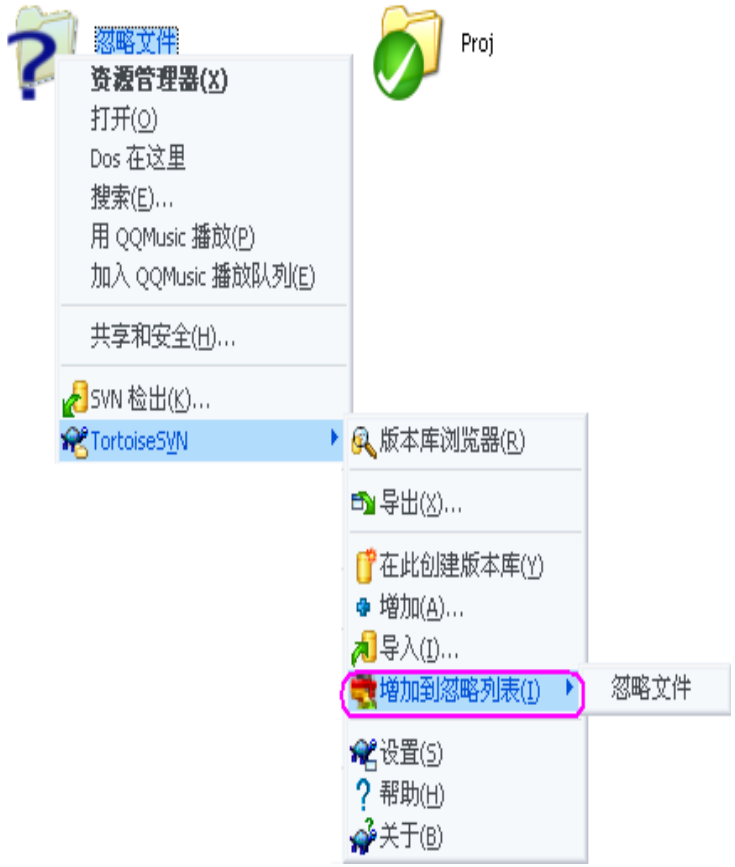
- **添加忽略文件**

  右键一个单独的未进入版本控制文件→TortoiseSVN →(加入忽略列表)Add to Ignore List，会出现一个子菜单允许你仅选择该文件，或者所有具有相同后缀的文件。

- **删除忽略文件：**

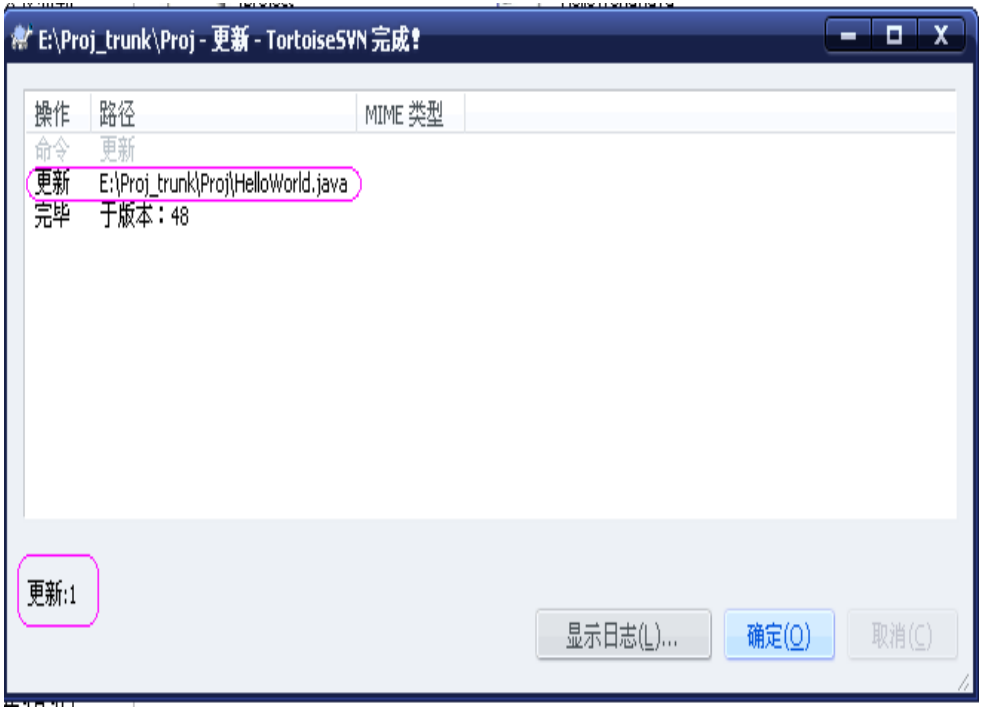
  如果你想从忽略列表中移除一个或多个条目，右击这些条目，选择TortoiseSVN → 从忽略列表删除。

- **已进入版本控制的文件或目录不能够忽略**



Update(更新)

- 作用：更新工作副本使其成为版本库中的最新版本（当别人也在处理此文件时）
- SVN将显示出更新的文件和更新的次数



Commit(提交)

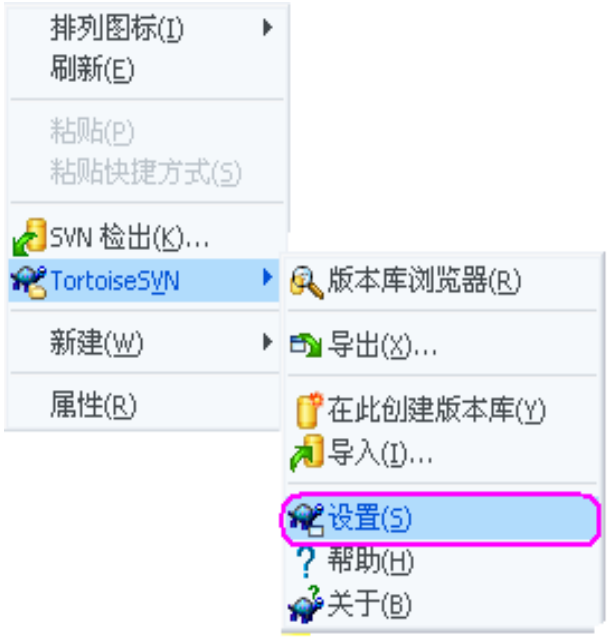
- 对工作副本进行编辑后提交到SVN
- 在右键菜单中点击SVN Commit
- 提交前写好信息，点击确定

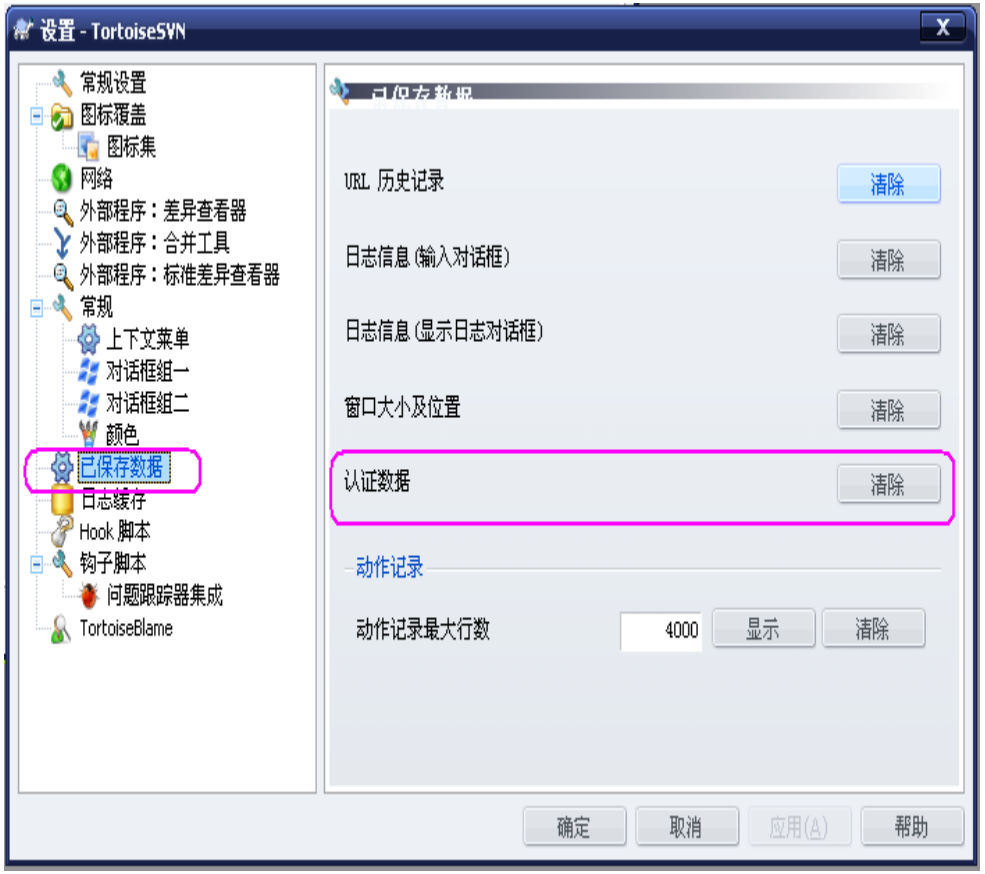
权限控制

- 当进行提交文件操作的时候您将看到权限提示信息
- 输入您的用户名和密码
- 保存权限设置（见红圈），可以避免将来重复输入用户名和密码

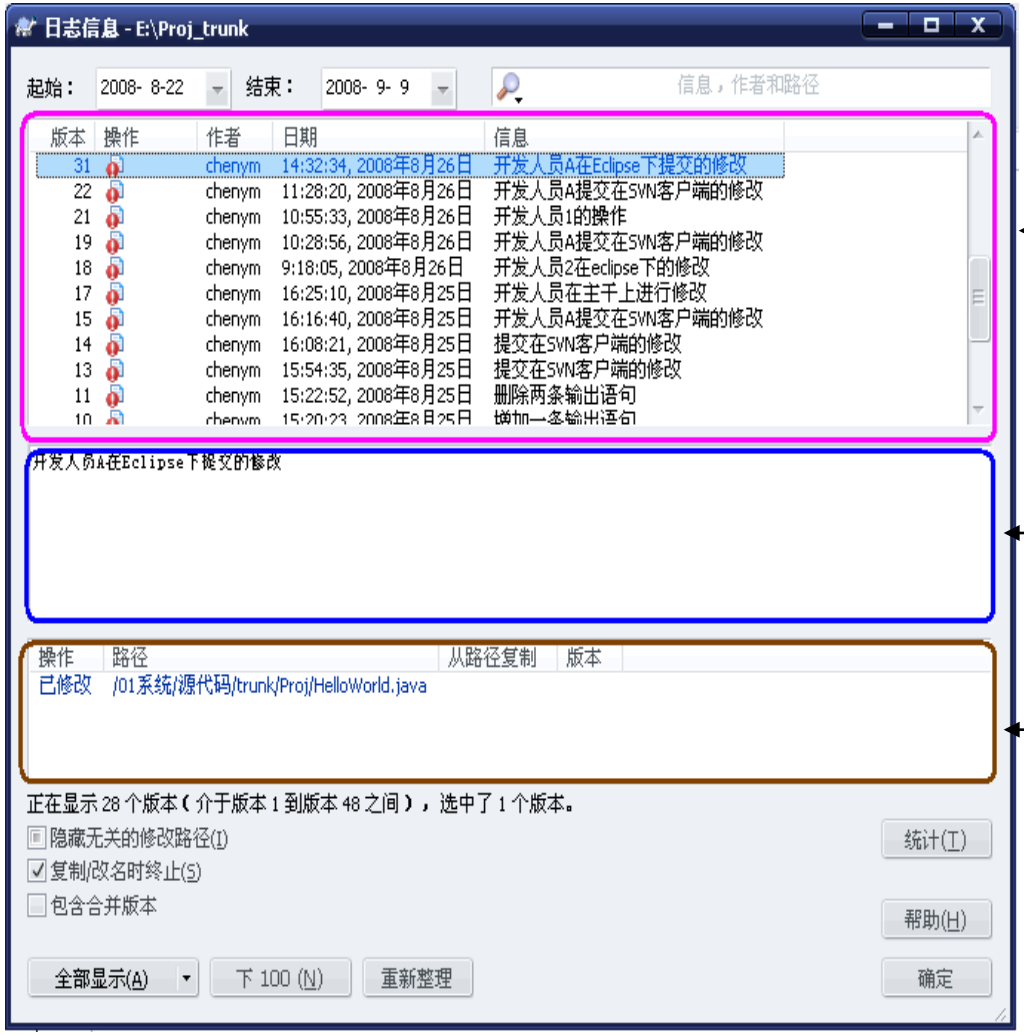
如何删除认证数据

步骤：点击右键→选择设置→以保存数据→清除认证数据





Show log（显示日志）



显示版本列表，包含了日期和时间，以及提交的用户和日志信息开头的部分内容。

显示了被选中的版本的完整的日志信息

显示了被选中版本中都对哪里文件和文件夹进行了修改。

日志信息填写规则

不规范的日志	规范的日志
去除无用文件	删除分支中的无用文件attachment.js_BAK_和moderation.js”
线索报警	修改线索的样式
更新配置文件	更新公司主题包和栏目配置文件

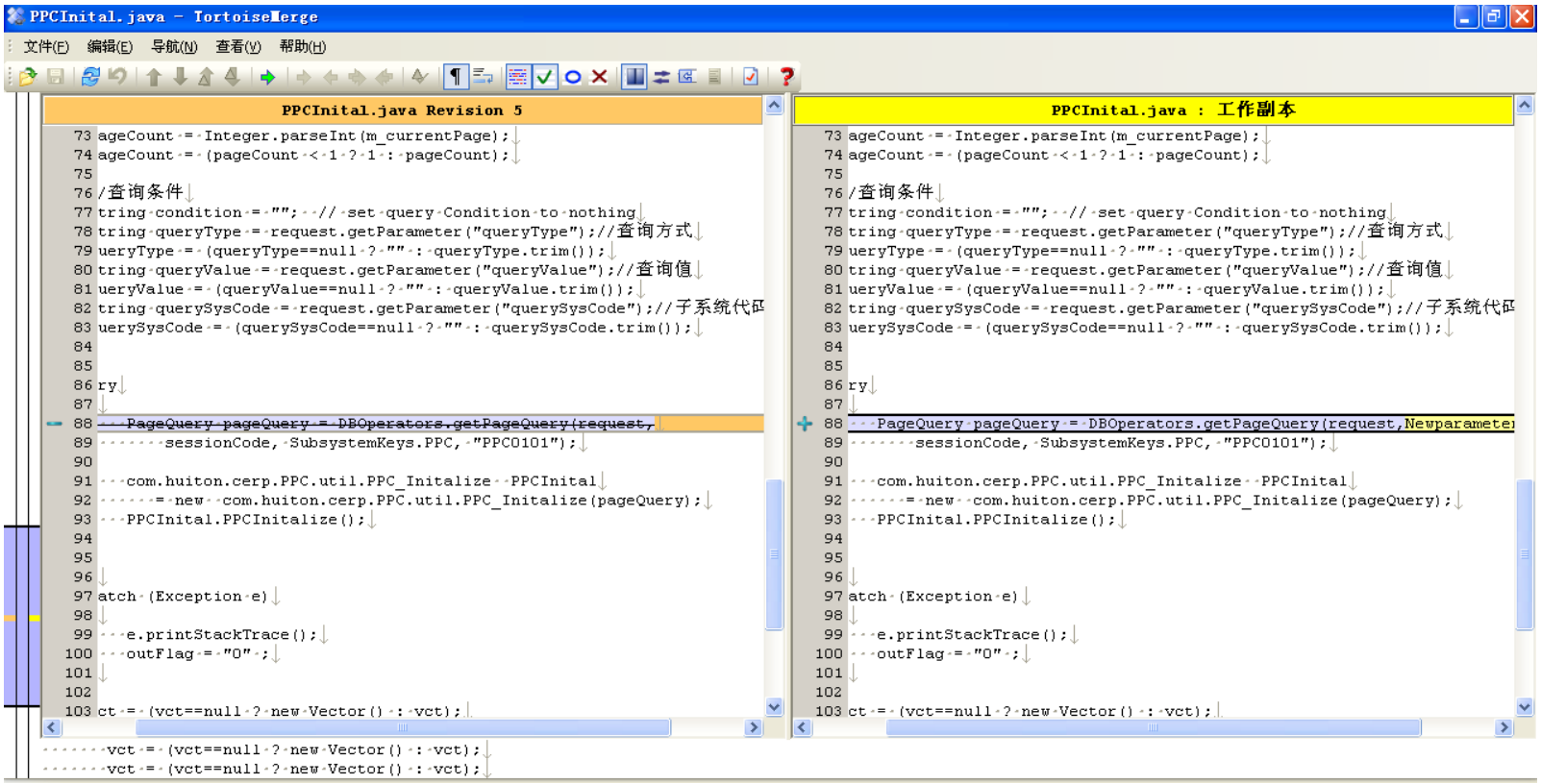
- 好的日志信息和糟糕的日志信息

  日志信息主要记录的是每次的修改内容。建议把一些重要数据、关键操作写到日志信息中。

- 注：修改人和提交时间由软件自动记录，无需人工写入日志信息

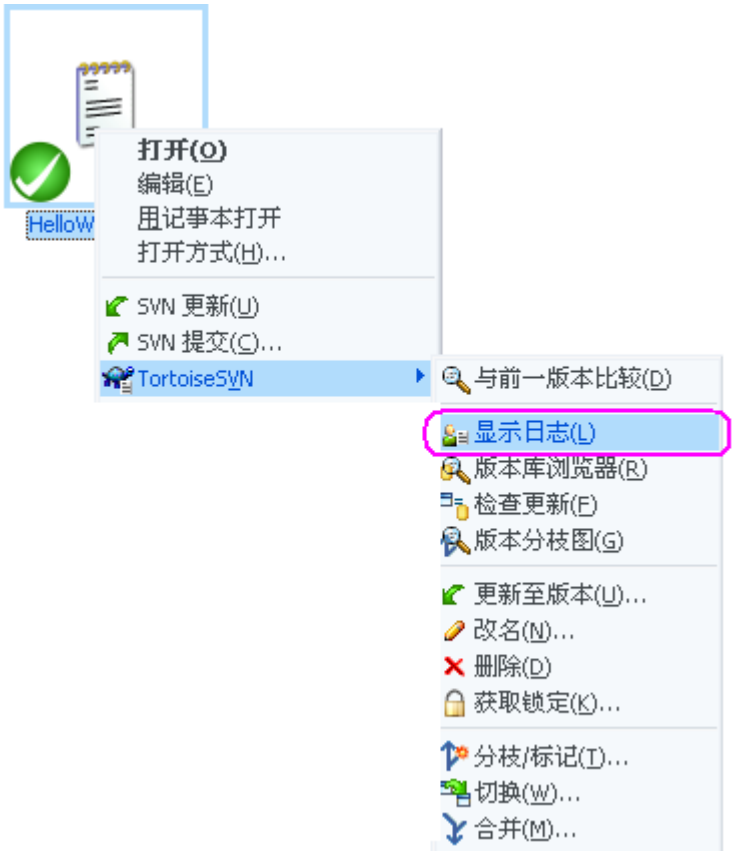
同历史版本进行比较

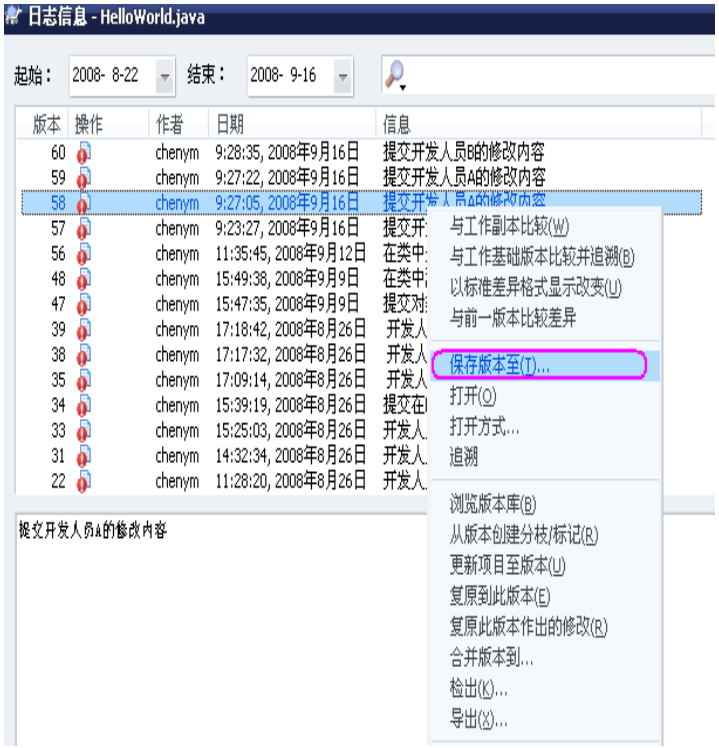
- 工作副本右键→显示日志→选择所需的版本号→与工作副本比较
- 支持文本文件的比较，也支持Word、Powerpoint、Excel文件的比较（需要安装相应的软件）



如何得到历史版本

- 工作副本右键→显示日志→选择所需的版本号→保存版本至





Revert(还原)

- 作用：撤销本地所有未提交的修改，还原到上次Commit之前的状态
- 注意：还没有执行Commit操作之前执行此命令才可以，否则无效

Eclipse下使用SVN

- 需要安装SVN客户端插件，安装方法参考Eclipse下安装Windowbuilder插件的过程。
- SVN插件下载：实验室ftp://172.16.0.46

Eclipse下使用Svn的主要功能

- 在Eclipse下使用Svn我们主要使用功能:

  1)将版本库导入到SVN资源库

  2)将新建项目导入到版本库

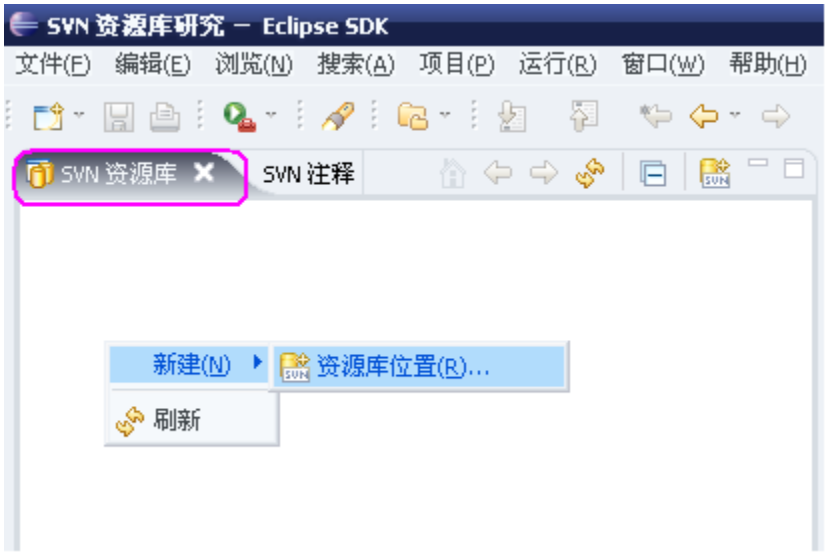
  3)小组

  4)比较对象

  5)替换为

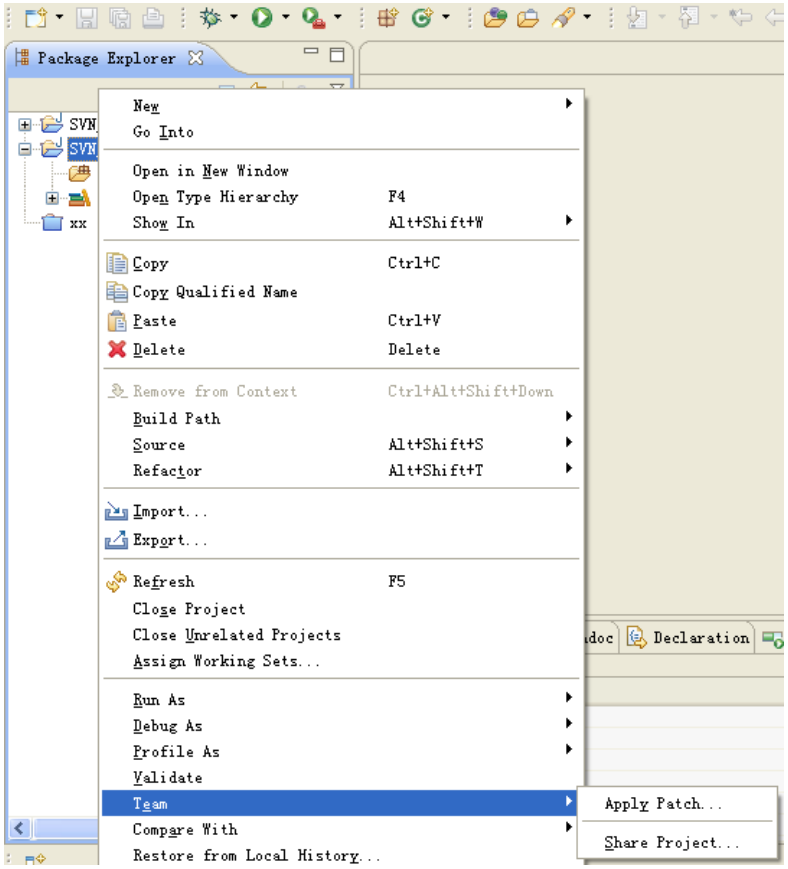
将版本库导入到SVN资源库

- Window→Open Perspective→SVN资源库研究(或者Other中选择)



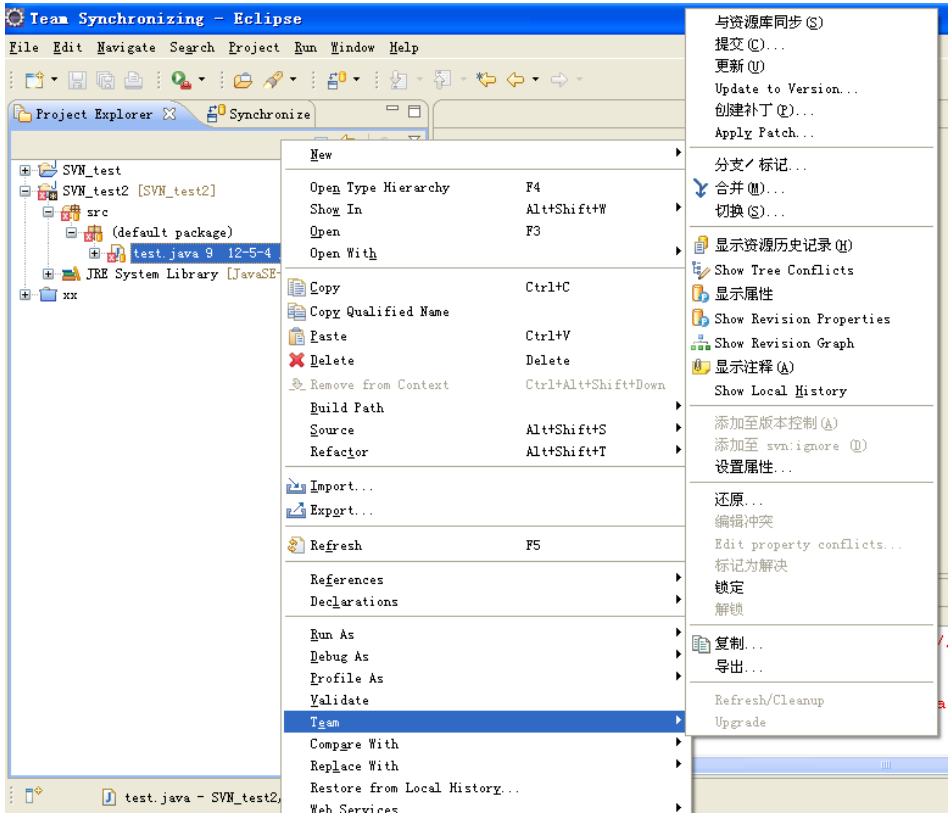
将新建项目导入到版本库

- 在项目上鼠标右键单击→Team→share project→SVN→使用现有资源库（也可根据实际情况创建新的）



Eclipse中Team的使用

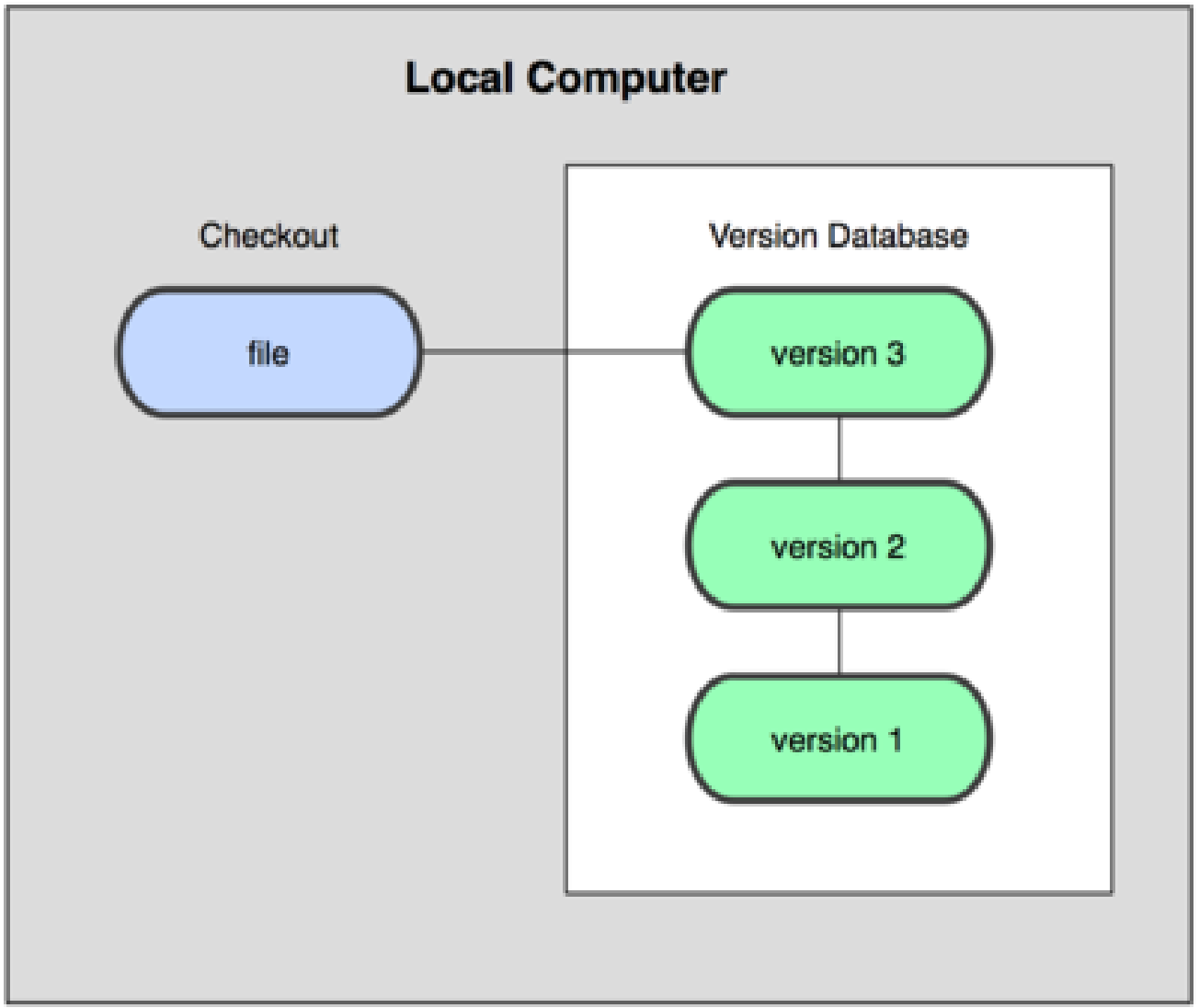
- 右键点击需要文件或文件夹，利用“Team”菜单中的功能进行版本控制；
- 在Eclipse中使用SVN的步骤同在操作系统中使用SVN客户端相同。



版本控制软件

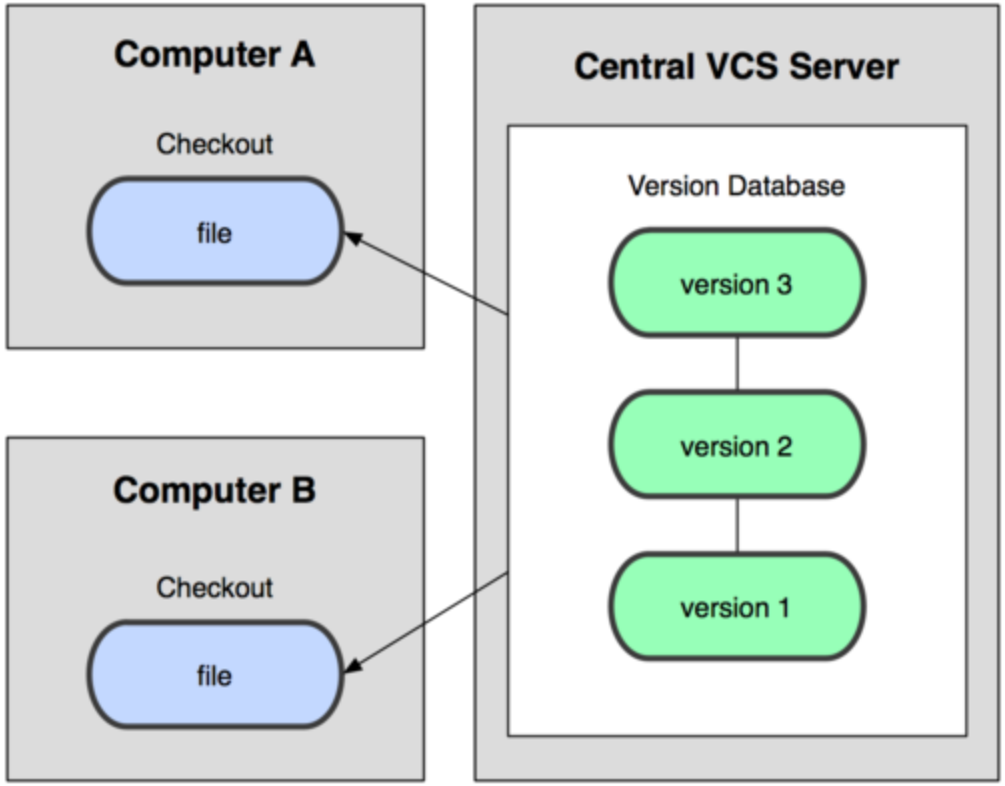
- 1.版本控制软件介绍
- 2.SVN使用简介和Eclipse中使用SVN
- 3.github的使用简介和Eclipse中使用github

本地版本控制系统


Local Computer
Checkout
file
Version Database
version 3
version 2
version 1

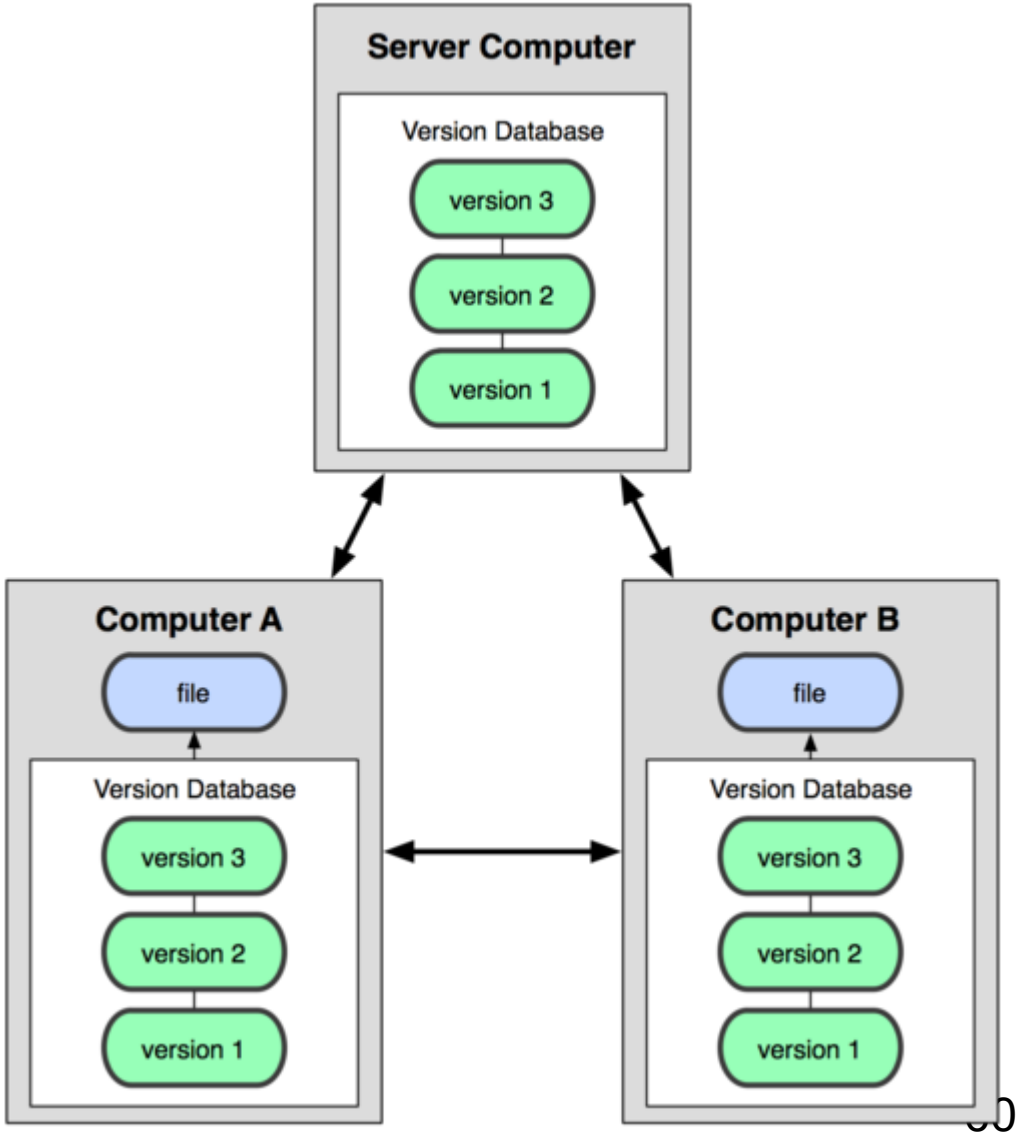
集中化的版本控制系统

- CVS，Subversion 以及 Perforce 等，有一个单一的集中管理的服务器，保存所有文件的修订版本，而协同工作的开发者通过客户端连到这台服务器，取出最新的文件或者提交更新。



分布式版本控制系统

- 客户端并不只提取最新版本的文件快照，而是把原始的代码仓库完整地镜像下来。
- 任何一处协同工作用的服务器发生故障，事后都可以用任何一个镜像出来的本地仓库恢复。
- 每一次的提取操作，实际上都是一次对代码仓库的完整备份。



Git如何出现的？

- Git是一个分布式的版本控制工具。
- Linux 内核开源项目有着为数众广的参与者，绝大多数的 Linux 内核维护工作都花在了提交补丁和保存归档的繁琐事务上；
- 最初由Linus Torvalds开发，用于管理Linux内核的开发；

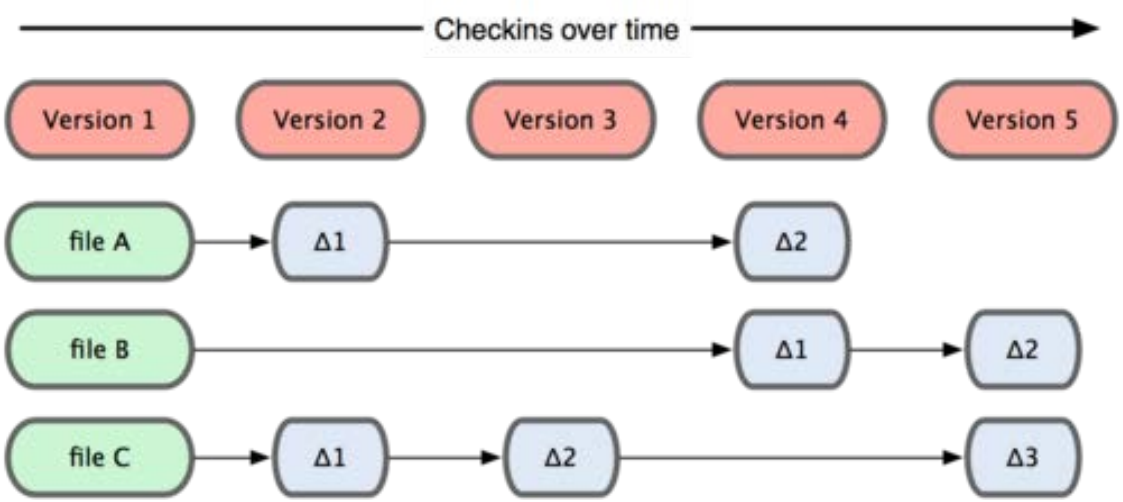
Git的基本思想

- 其他大多数SCM系统主要关心文件内容的具体差异(每次记录有哪些文件作了更新，以及都更新了哪些行的什么内容)；

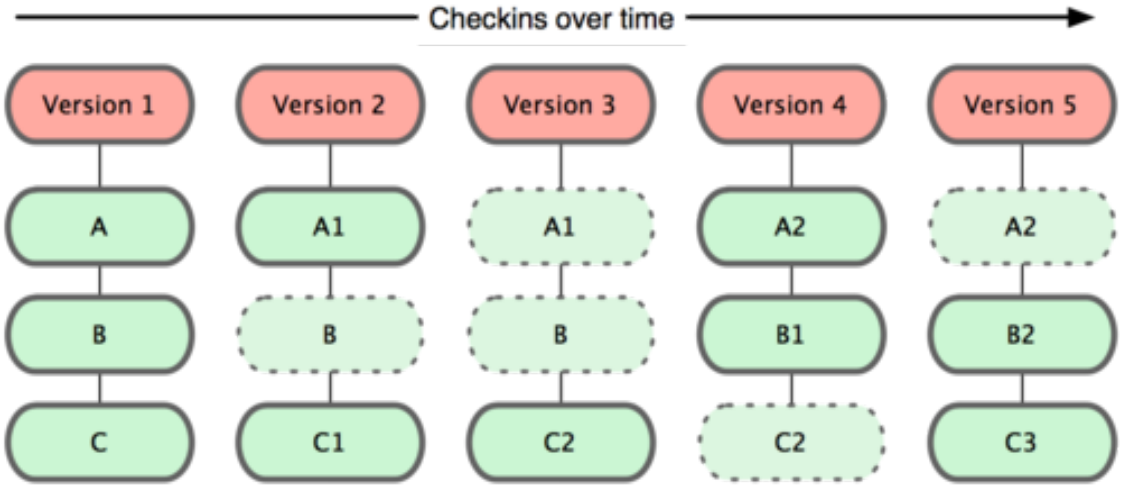
- Git关心文件数据的整体是否发生变化，并不保存这些前后变化的差异数据
  - 把变化的文件作快照后，记录在一个微型的文件系统中。
  - 每次提交更新时，它会纵览一遍所有文件的指纹信息并对文件作一快照，然后保存一个指向这次快照的索引。
  - 为提高性能，若文件没有变化，Git不会再次保存，而只对上次保存的快照作一链接。

Git的基本思想

其他SCM软件

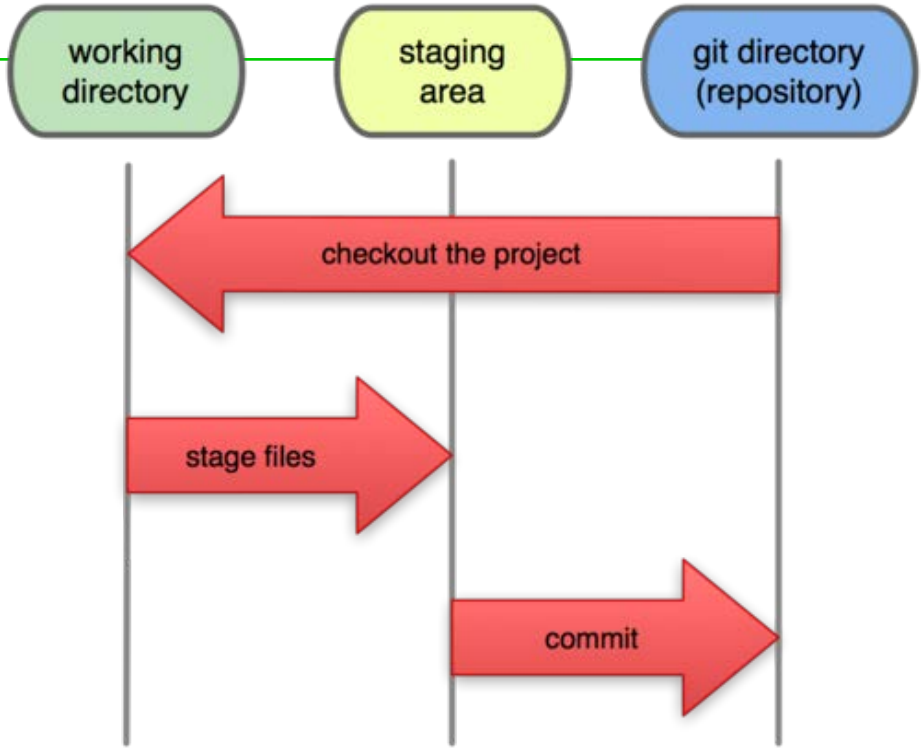


git



Git中文件的三种状态

- **Git管理项目的三个工作区域：**
  - 工作目录
  - 暂存区域
  - 本地仓库



- **对于任何一个文件，在 Git 内都只有三种状态：**
  - 已提交(committed)：表示该文件已经被安全地保存在本地数据库中；
  - 已修改(modified)：表示修改了某个文件，但还没有提交保存；
  - 已暂存(staged)：表示把已修改的文件放在下次提交时要保存的清单中。

Git的工作区

- 每个项目都有一个Git目录(Git directory)，用来保存元数据和对象数据库。每次clone镜像仓库的时候，实际拷贝的就是这个目录里面的数据。
- 从Git目录中取出某个版本的所有文件和目录，用以开始后续工作，形成工作目录(working directory)，接下来就可以在工作目录中对这些文件进行编辑。
- 所谓的暂存区域(staging area)只不过是个简单的索引文件，是一个包含文件索引的目录树，像是一个虚拟的工作区。在这个虚拟工作区的目录树中，记录了文件名、文件的状态信息（时间戳、文件长度等），文件的内容并不存储其中，而是保存在 Git 对象库中。暂存区域的设计可以更有效、灵活的控制要提交的文件对象。

Git的基本工作流程

- 1. 在工作目录中修改某些文件。
- 2. 对修改后的文件进行快照，然后保存到暂存区域。
- 3. 提交更新，将保存在暂存区域的文件快照永久转储到Git目录中。

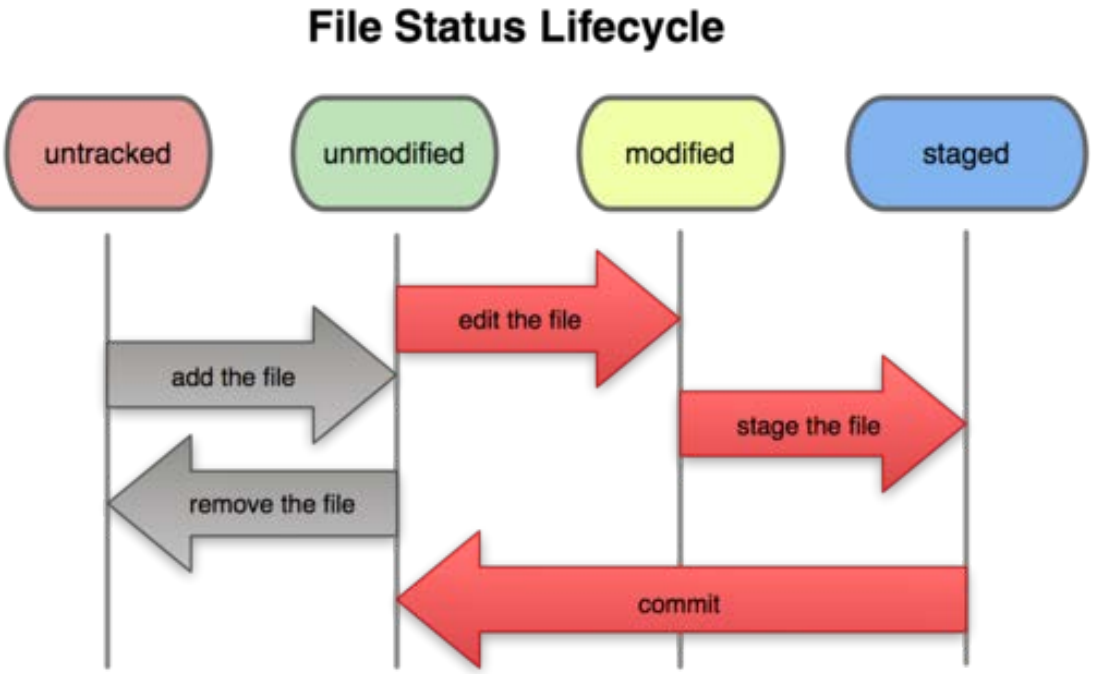
- 可以从文件所处的位置来判断状态：
  - 如果是 Git 目录中保存着的特定版本文件，就属于已提交状态；
  - 如果作了修改并已放入暂存区域，就属于已暂存状态；
  - 如果自上次取出后，作了修改但还没有放到暂存区域，就是已修改状态。

基本的Git命令：取得项目的 Git 仓库

- **本机安装设置Git环境**
- **在工作目录中初始化新仓库**
  - 要对现有的某个项目开始用Git管理，只需到此项目所在的目录，执行**git init**命令，用 **git add** 命令告诉Git开始对这些文件进行跟踪，然后提交：
    - git add *.c
    - git add README
    - git commit -m 'initial project version'
- **从现有仓库克隆：复制服务器上项目的所有历史信息到本地**
  - git clone [url]

基本的Git命令：记录每次更新到仓库

- 在工作目录对某些文件作了修改之后，Git将这些文件标为“已修改”，可提交本次更新到仓库。
- 逐步把这些修改过的文件放到暂存区域，直到最后一次性提交所有这些暂存起来的文件，如此重复。



基本的Git命令：检查当前文件状态

- **要确定哪些文件当前处于什么状态，用`git status`命令：**
  - # On branch master nothing to commit (working directory clean)
    当前没有任何跟踪着的文件，也没有任何文件在上次提交后更改过
  - # On branch master # Untracked files: …
    有未跟踪的文件，使用git add开始跟踪一个新文件
  - # On branch master # Changes to be committed:
    有处于已暂存状态的文件

基本的Git命令：跟踪新文件、暂存已修改文件

- 使用git add开始跟踪一个新文件（使某个文件纳入到git中管理）。
- 一个修改过的且被跟踪的文件，处于暂存状态。

- git add后面可以指明要跟踪的文件或目录路径。如果是目录的话，就说明要递归跟踪该目录下的所有文件。
- git add的潜台词：把目标文件快照放入暂存区域，也就是 add file into staged area，同时未曾跟踪过的文件标记为需要跟踪。

- 若对已跟踪的文件进行了修改，使用git add命令将其放入暂存区；
- 运行了git add之后又对相应文件做了修改，要重新git add。

基本的Git命令：查看已暂存和未暂存的更新

- git status回答：当前做的哪些更新还没有暂存？有哪些更新已经暂存起来准备好了下次提交？
- 如果要查看具体修改了什么地方，可以用git diff命令，使用文件补丁的格式显示具体添加和删除的行。
- 要查看尚未暂存的文件更新了哪些部分，不加参数直接输入git diff：
  - 比较的是工作目录中当前文件和暂存区域快照之间的差异，也就是修改之后还没有暂存起来的变化内容。
- 若要查看已暂存起来的文件和上次提交时的快照之间的差异，可以用 git diff --cached 命令。

基本的Git命令：提交更新

- **在使用git commit命令进行提交之前，要确认是否还有修改过的或新建的文件没有git add过，否则提交的时候不会记录这些还没暂存起来的变化。**
  - 每次准备提交前，先用git status进行检查，然后再运行提交命令git commit。
- **提交后返回结果：**
  - 当前是在哪个分支(master)提交的
  - 本次提交的完整 SHA-1 校验和是什么
  - 在本次提交中，有多少文件修订过、多少行添改和删改过。
- **提交时记录的是放在暂存区域的快照，任何还未暂存的仍然保持已修改状态，可以在下次提交时纳入版本管理。每一次运行提交操作，都是对项目做一次快照，以后可以回到这个状态，或者进行比较。**

基本的Git命令：跳过使用暂存区域、移除文件

- Git提供了一个跳过使用暂存区域的方式，只要在提交的时候，给 git commit 加上-a 选项，Git 就会自动把所有已经跟踪过的文件暂存起来一并提交，从而跳过git add步骤；

- 使用git rm命令从Git中移除某个文件，把它从已跟踪文件清单(暂存区域)中移除，并连带从工作目录中删除指定的文件。

基本的Git命令：对远程仓库的操作

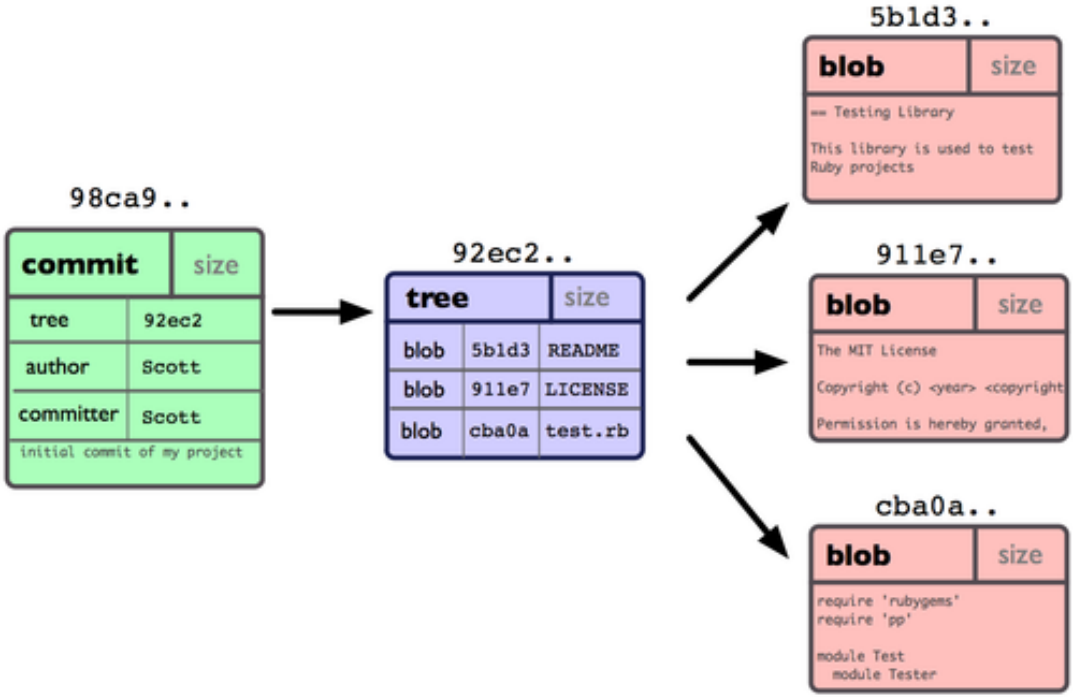
- **远程仓库：托管在网络上的项目仓库；**
- **多人协作开发某个项目时，需要管理这些远程仓库，以便推送或拉取数据，分享各自的工作进展。**
- **管理远程仓库：添加远程库、移除废弃的远程库、管理各式远程库分支、定义是否跟踪这些分支。**
  - git remote：获取当前配置的所有远程仓库；
  - git remote add [shortname] [url]：添加一个远程仓库；
  - git fetch：从远程仓库抓取数据到本地；
  - git push [remote-name] [branch-name]：将本地仓库中的数据推送到远程仓库；
  - git remote show [remote-name]：查看某个远程仓库的详细信息；
  - git remote rm：从本地移除远程仓库；

Commit

- **在 Git 中提交时，会保存一个提交(commit)对象，该对象包含一个指向暂存内容快照的指针，包含本次提交的作者等相关附属信息，包含零个或多个指向该提交对象的父对象指针：**
  - 首次提交是没有直接祖先的；
  - 普通提交有一个祖先；
  - 由两个或多个分支合并产生的提交则有多个祖先。

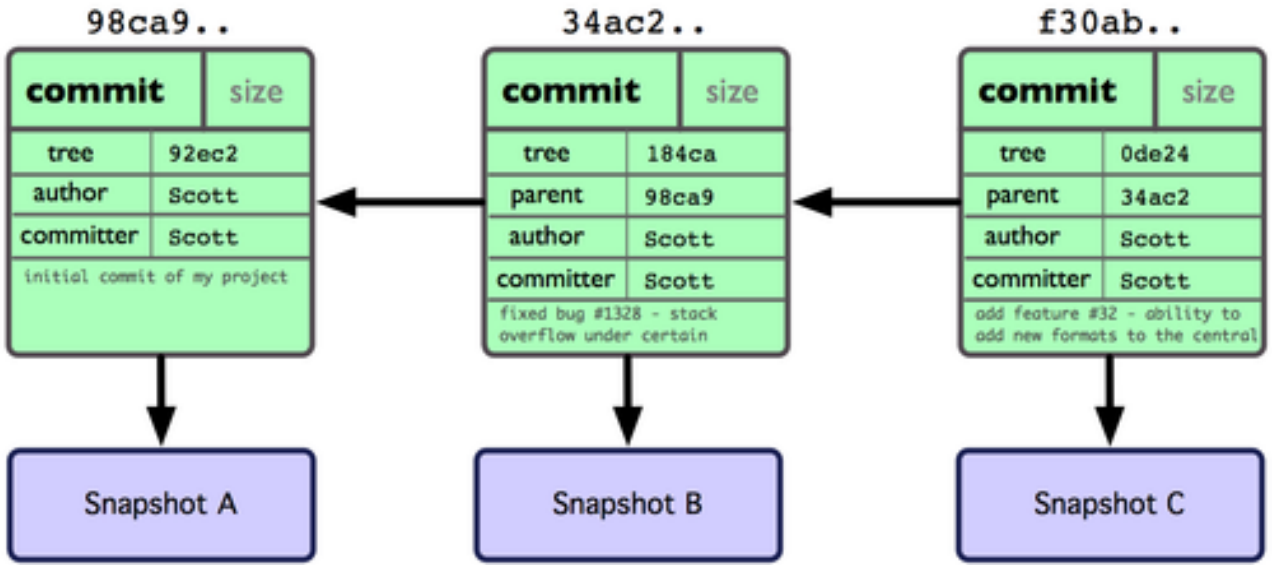
单个提交(Commit)对象在仓库中的数据结构

- 多个表示待提交的文件快照内容的blob对象；
- 一个记录着目录树内容及其中各个文件对应 blob 对象索引的 tree 对象；
- 一个包含指向 tree对象(根目录)的索引和其他提交信息元数据的commit对象。



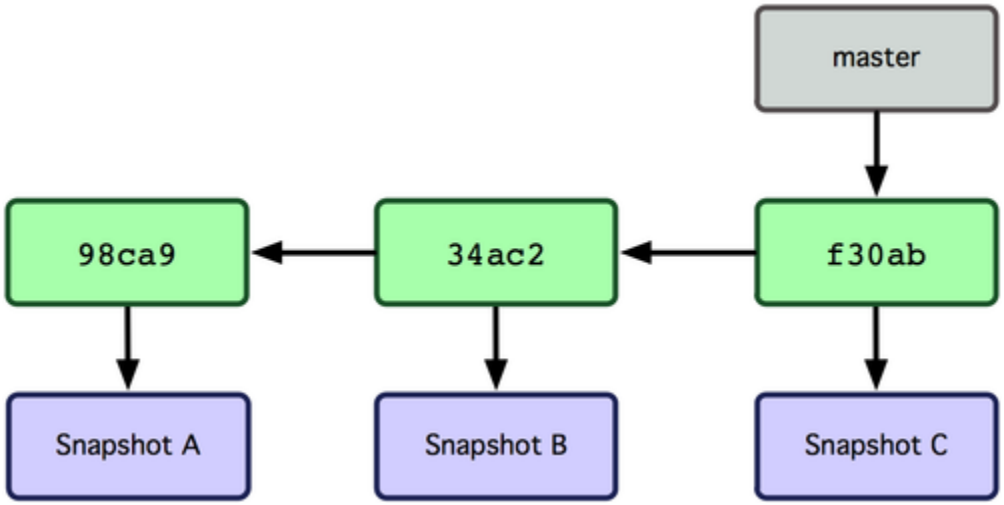
多次提交之后

- 某些文件修改后再次提交，这次的commit对象会包含一个指向上次提交对象的指针。



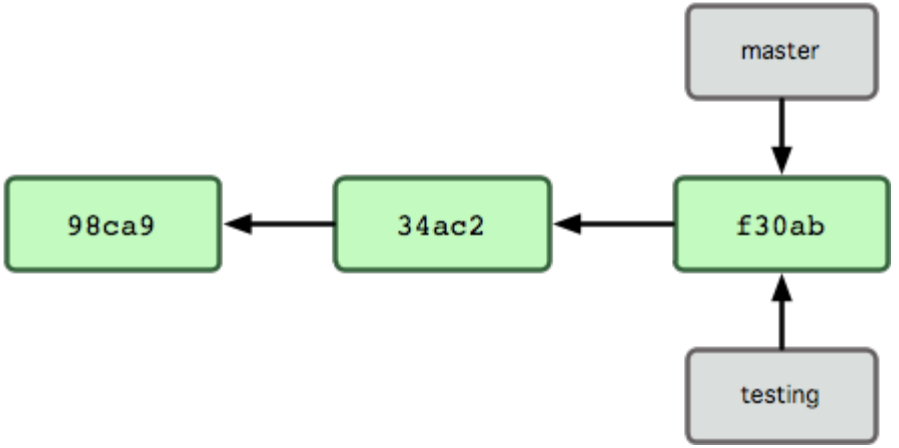
Git的分支

- Git中的分支本质上仅仅是个指向 commit 对象的可变指针。
- Git使用master作为分支的默认名字。
- 若干次提交后，master分支指向最后一次提交对象，它在每次提交的时候都会自动向前移动。



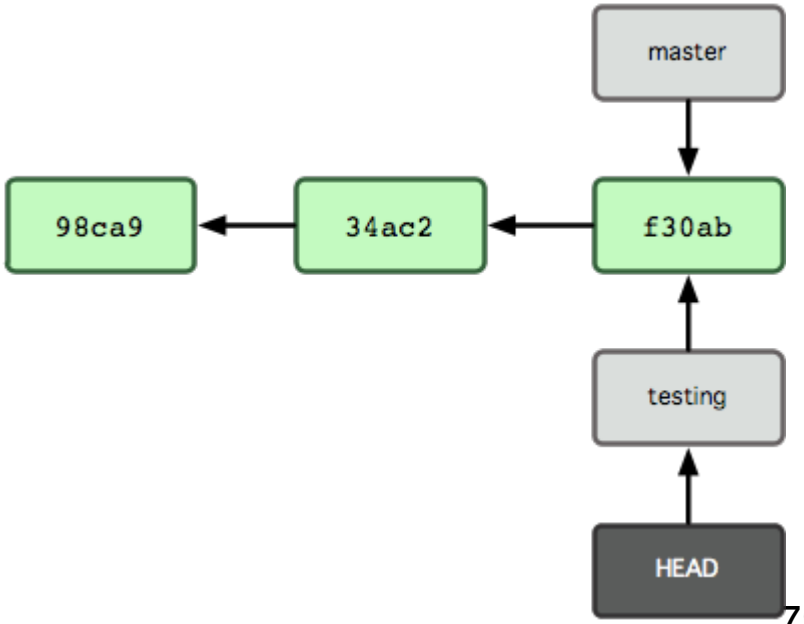
创建一个新的分支

- git branch在当前commit对象上新建一个分支指针；



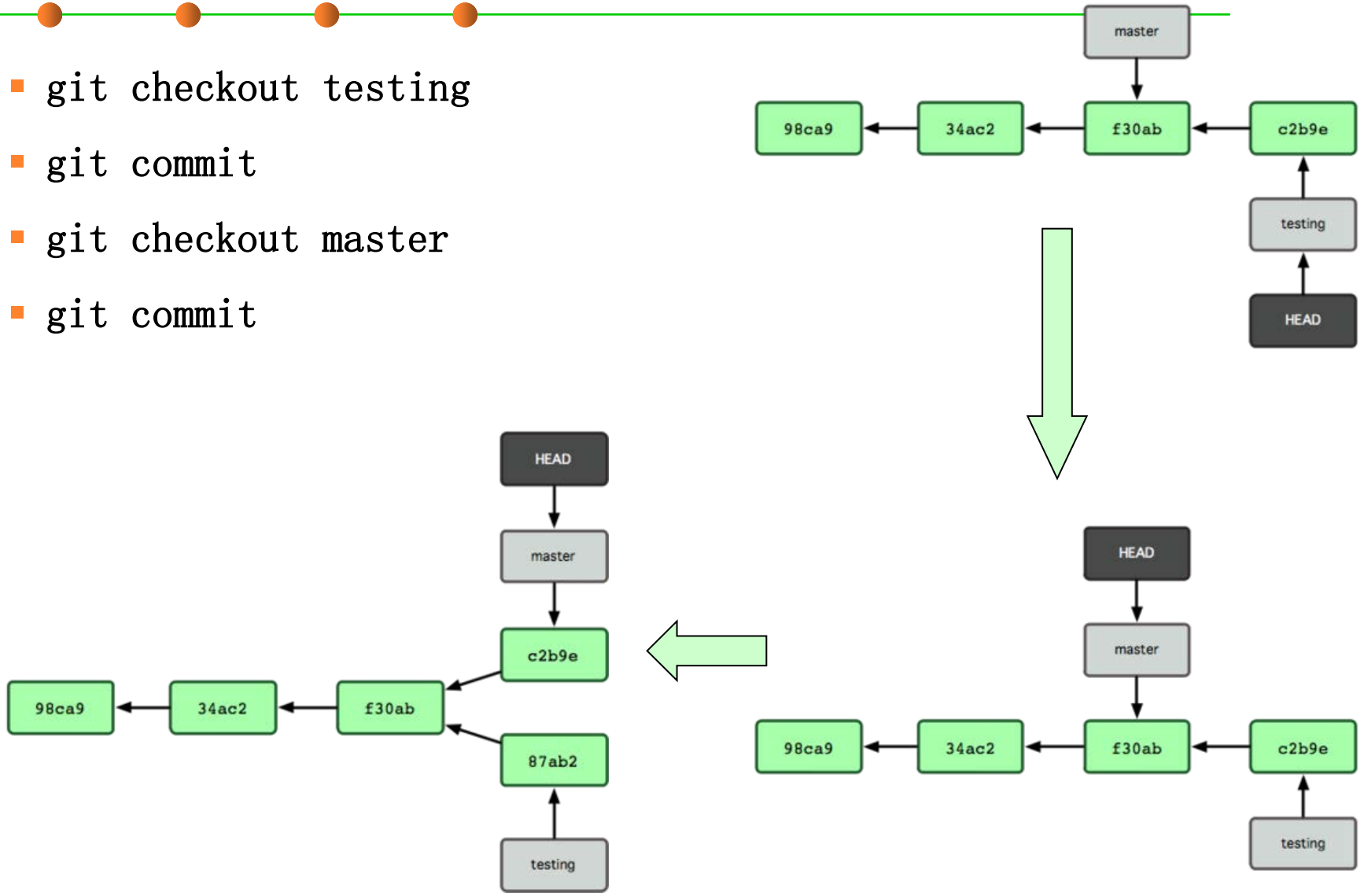
- Git使用一个叫做HEAD的特别指针来获知你当前在哪个分支上工作。

- 要切换到其他分支，可以执行git checkout命令。



在新的分支上提交

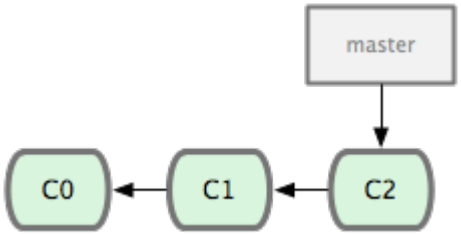
- git checkout testing
- git commit
- git checkout master
- git commit

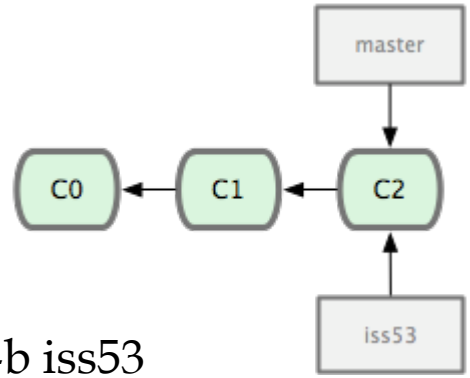


Git的分支机制的优越性

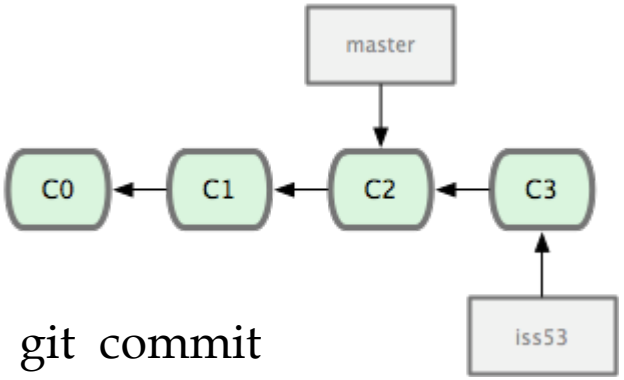
- Git 中的分支实际上仅是一个包含所指对象校验和(40 个字符长度SHA-1字串)的文件，非常方便；
- 传统的版本控制系统则采用将文件备份到目录的方式。
- Git的分支实现与项目复杂度无关，可以在几毫秒的时间内完成分支的创建和切换。
- 因为每次提交时都记录了祖先信息(即parent 对象)，将来要合并分支时，寻找恰当的合并基础(即共同祖先)，实现起来非常容易。

分支的新建与合并：例子

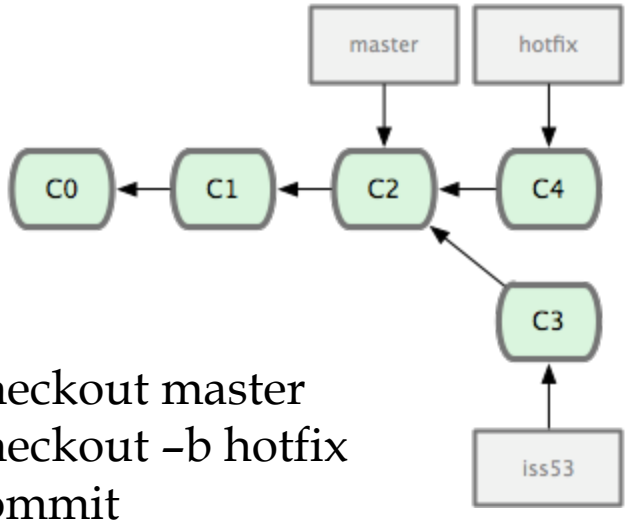




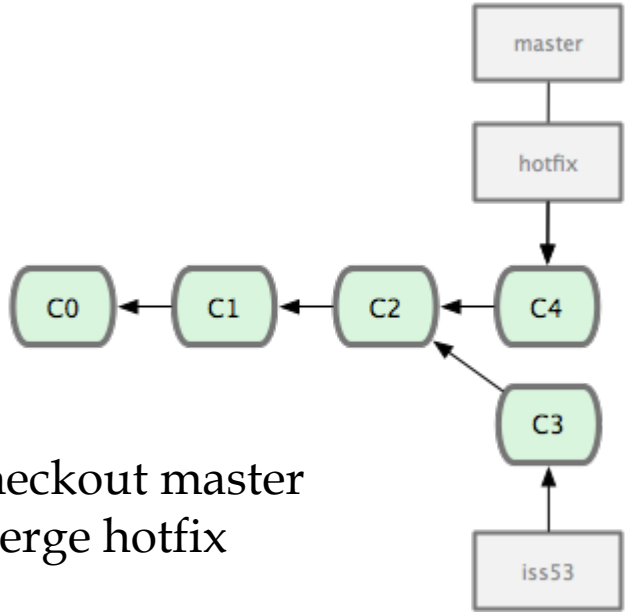
git checkout –b iss53



git commit

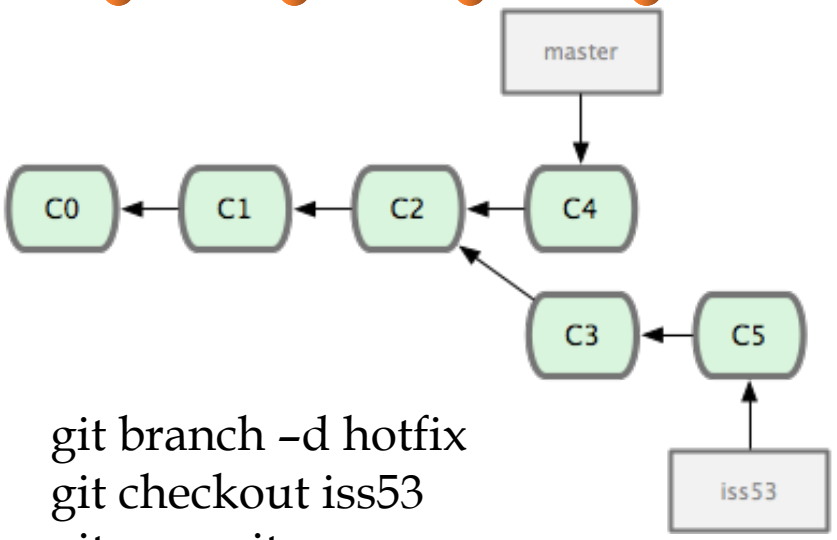


git checkout master
git checkout –b hotfix
git commit

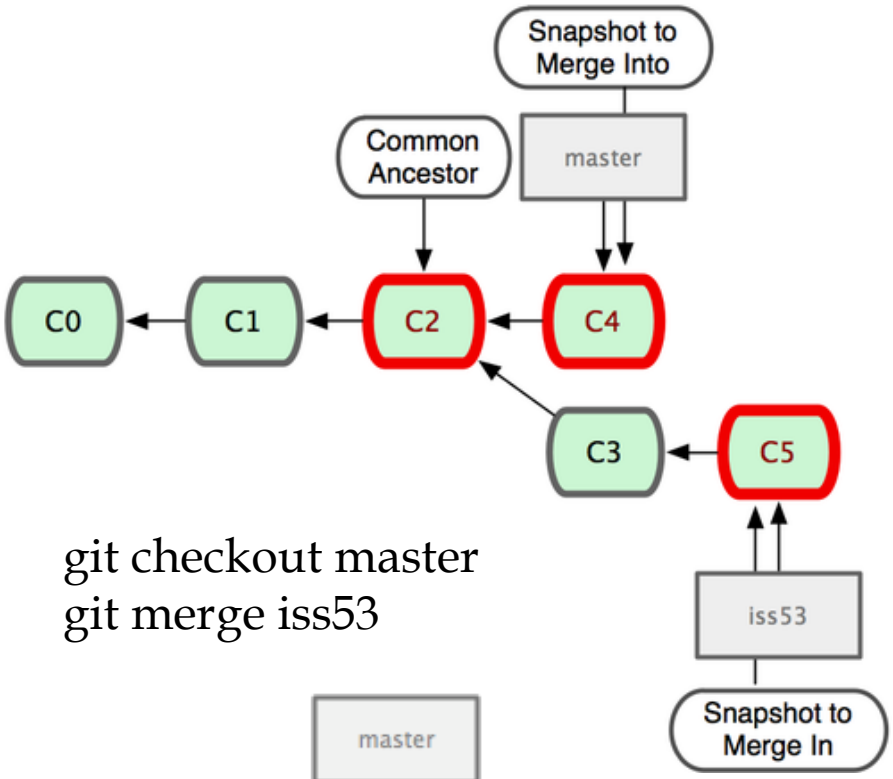


git checkout master
git merge hotfix

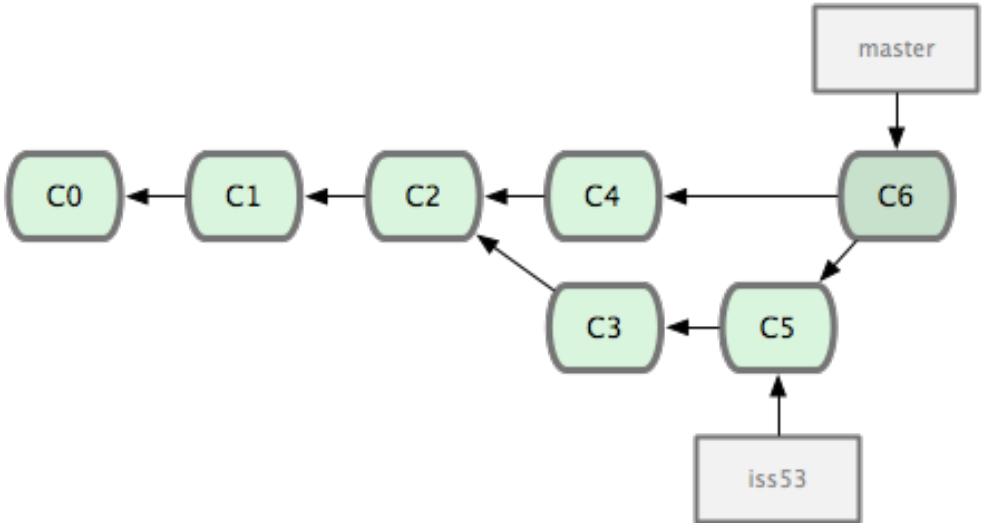
分支的新建与合并：例子



git branch –d hotfix
git checkout iss53
git commit



git checkout master
git merge iss53



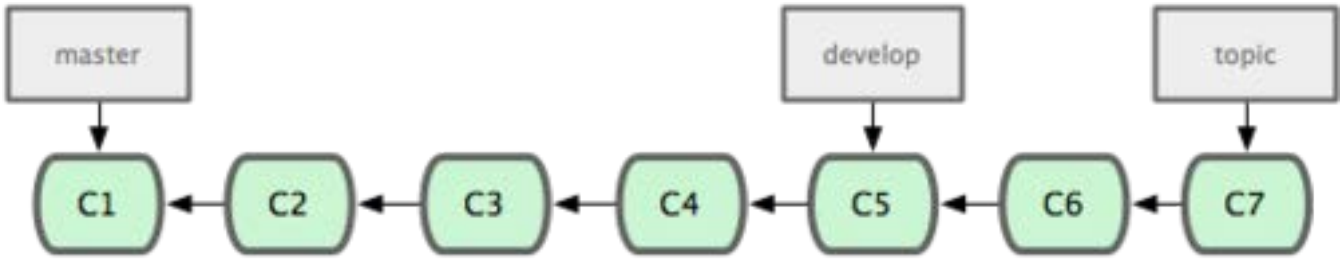
分支的新建与合并：有冲突时如何？

- 如果在不同的分支中都修改了同一个文件的同一部分，Git无法干净地把两者合到一起，需要依赖于人的裁决。
- 使用git status查看任何包含未解决冲突的文件，在有冲突的文件里加入标准的冲突解决标记，可以通过它们来手工定位并解决这些冲突。

```
<<<<<<< HEAD
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>>> iss53
```

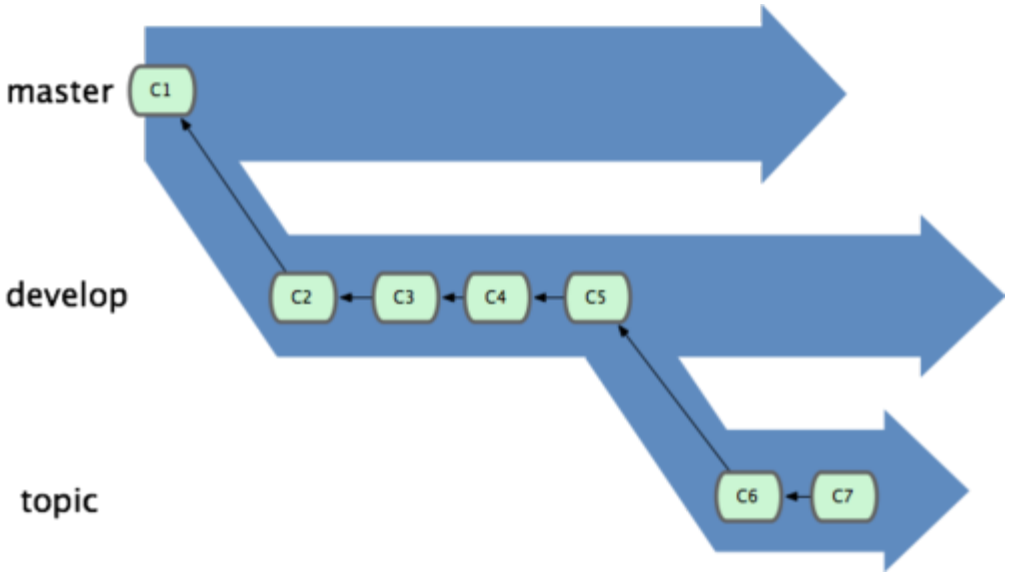
利用分支进行开发的工作流程

- **长期分支：可以同时拥有多个开放的分支，每个分支用于完成特定的任务，随着开发的推进，可以随时把某个特性分支并到其他分支中。**
  - 仅在 master 分支中保留完全稳定的代码，即已经发布或即将发布的代码。
  - 同时还有一个名为develop或next的平行分支，专门用于后续的开发，或仅用于稳定性测试，一旦进入某种稳定状态，便可以把它合并到master 里。
  - 这样，在确保这些已完成的特性分支能够通过所有测试并且不会引入更多错误之后，就可以合并到主干分支中，等待下一次的发布。
  - 随着提交对象不断右移的指针。稳定分支的指针总是在提交历史中落后一大截，而前沿分支总是比较靠前。



长期分支

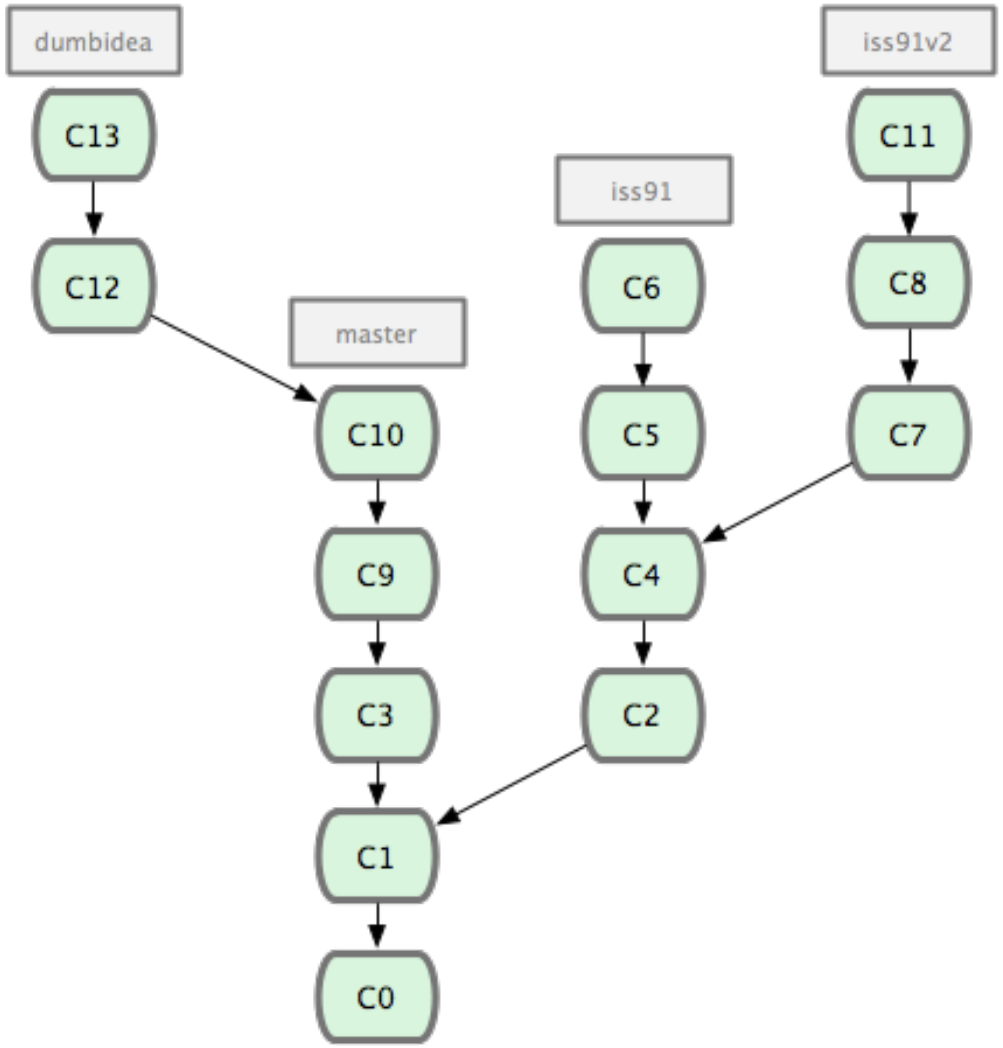
- 这么做的目的是拥有不同层次的稳定性：当这些分支进入到更稳定的水平时，再把它们合并到更高层分支中去。

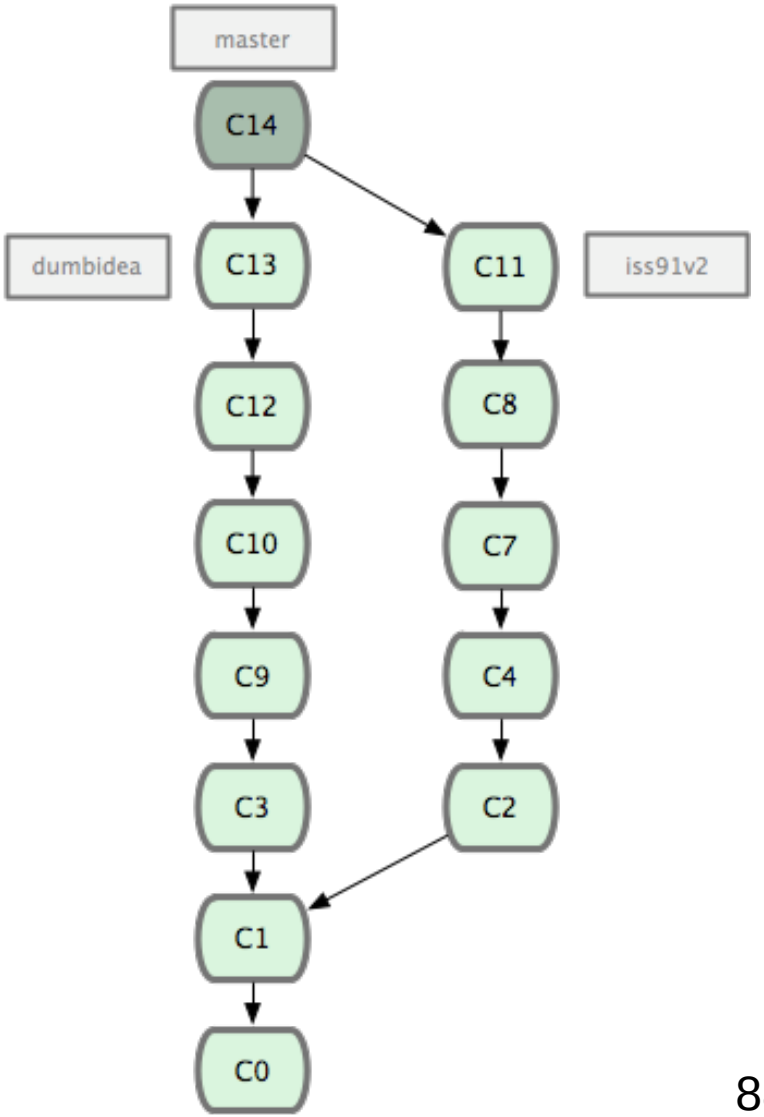


特性分支

- 特性分支(Topic)是指一个短期的，用来实现单一特性或与其相关工作的分支。
- 创建特性分支，在提交了若干更新后，把它们合并到主干分支，然后删除，从而支持迅速且完全的进行语境切换。
- 因为开发工作分散在不同的流水线里，每个分支里的改变都和它的目标特性相关，浏览代码之类的事情因而变得更简单了。
- 可以把作出的改变保持在特性分支中几分钟，几天甚至几个月，等它们成熟以后再合并，而不用在乎它们建立的顺序或者进度。

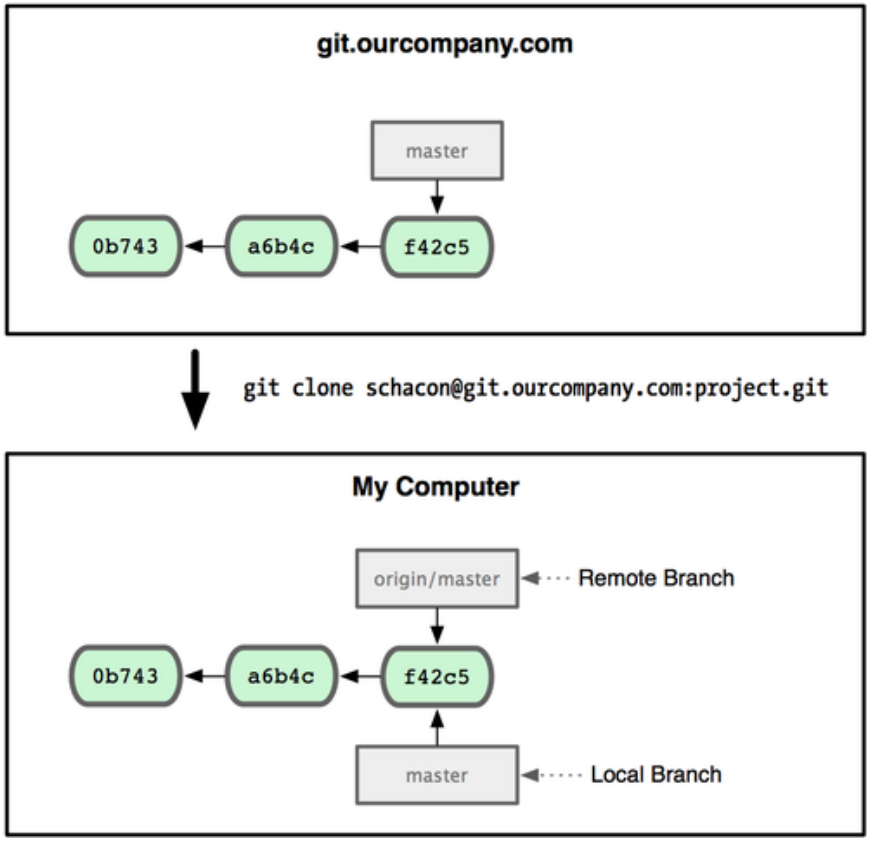
特性分支的例子

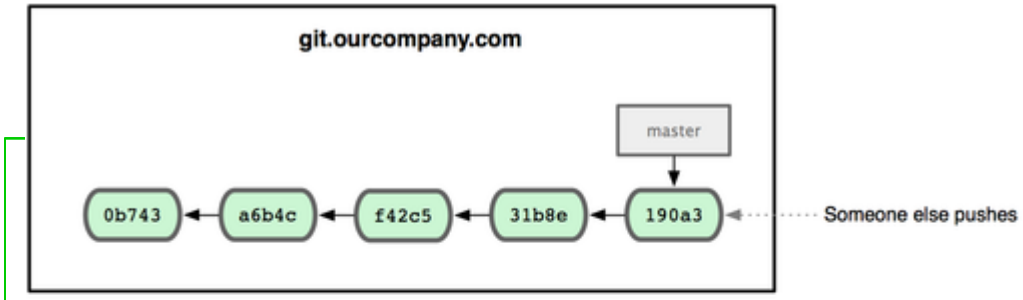

dumbidea
C13
C12
master
C10
C9
C3
C1
C0
iss91
C6
C5
C4
C2
iss91v2
C11
C8
C7

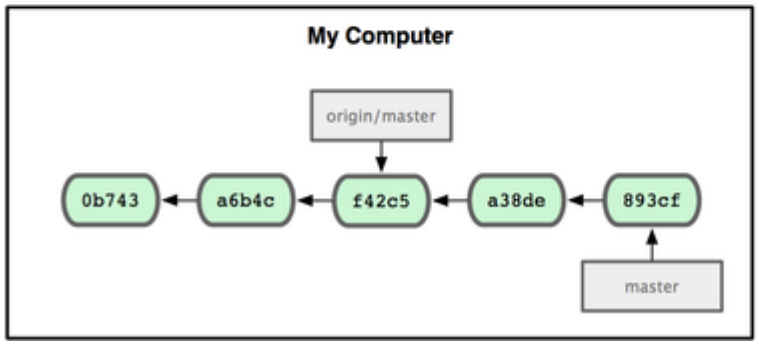

master
C14
dumbidea
C13
C12
C10
C9
C3
C1
C0
C11
C8
C7
C4
C2
iss91v2

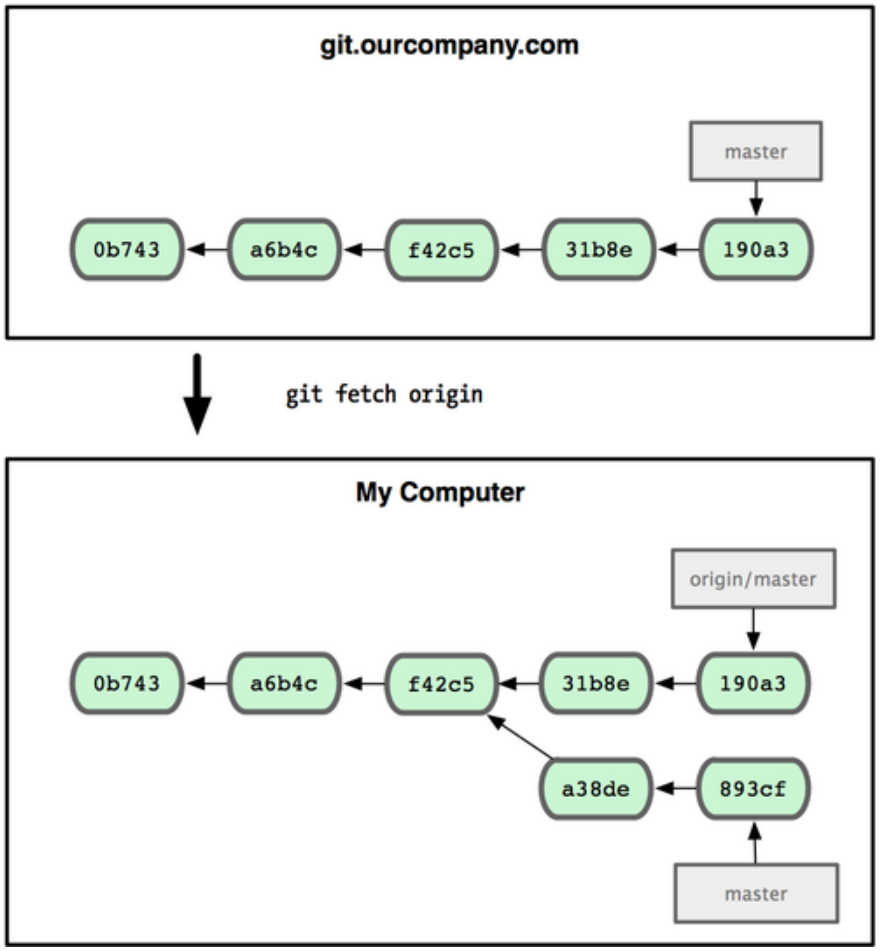
远程分支

- 远程分支(remote branch)：对远程仓库中的分支的索引，类似于书签，提醒上次连接远程仓库时上面各分支的位置。
- 使用“(远程仓库名)/(分支名)”表示。



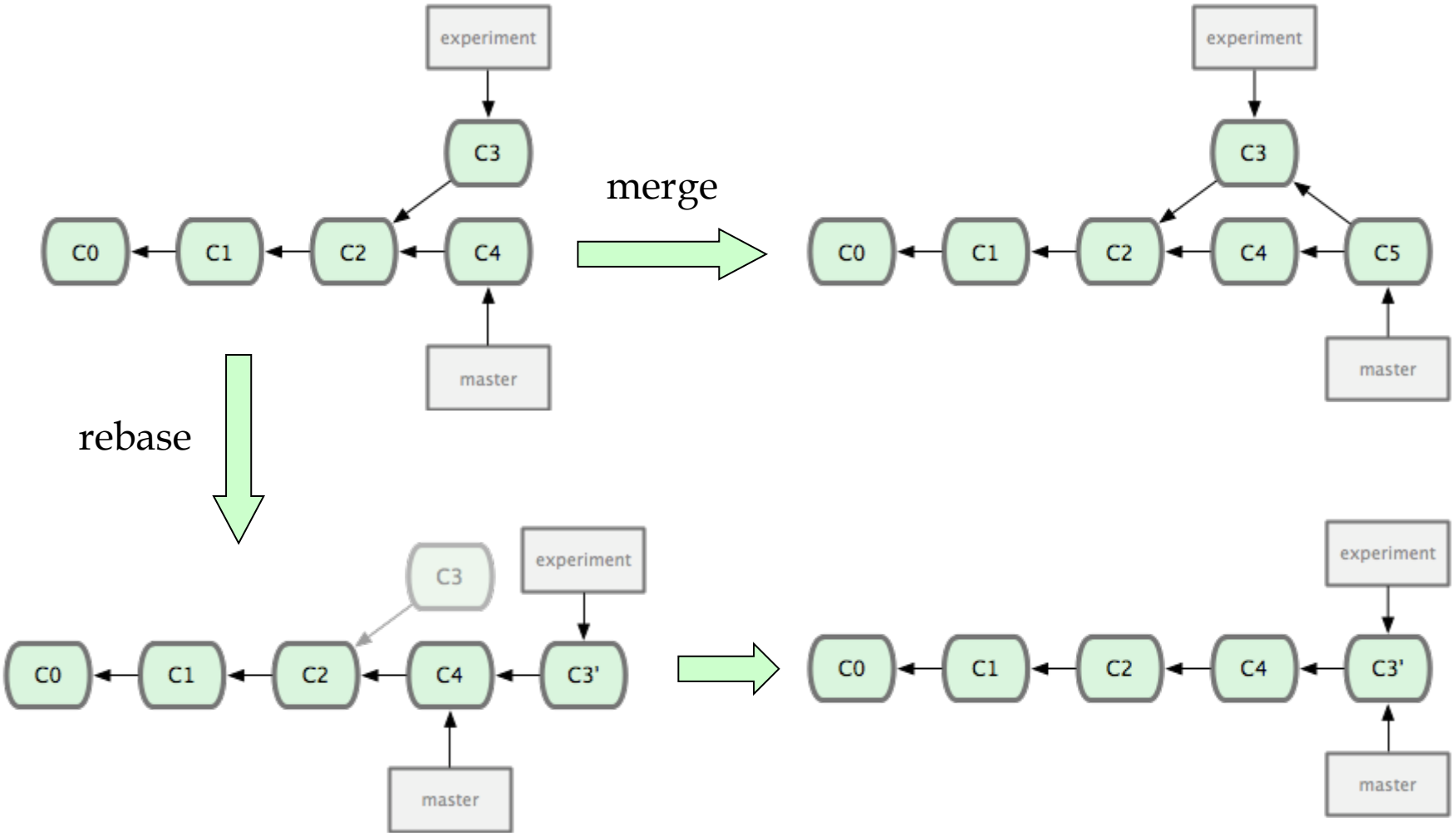

git.ourcompany.com
master
0b743
a6b4c
f42c5
31b8e
190a3
Someone else pushes


My Computer
origin/master
0b743
a6b4c
f42c5
a38de
893cf
master


git.ourcompany.com
master
0b743
a6b4c
f42c5
31b8e
190a3
git fetch origin
My Computer
origin/master
0b743
a6b4c
f42c5
31b8e
190a3
a38de
893cf
master

分支的衍合(rebase)

- 除了git merge，另一种进行分支合并的方法：git rebase。



Github

Build software better, together.

- GitHub：共享虚拟主机服务，用于存放使用Git版本控制的软件代码和内容项目，是目前最流行的Git存取站点，同时提供付费账户和为开源项目提供的免费账户。
- 支持社会化软件开发，允许用户跟踪其他用户、组织、软件库的动态，对软件代码的改动和 bug 提出评论等。
- 提供了图表功能，用于显示开发者们怎样在代码库上工作以及软件的开发活跃程度。

Github

Build software better, together.

- GitHub快速入门可参考

http://www.worldhello.net/gotgithub/index.html

github

- 在github网站上注册 https://github.com/ 帐户
- 在Windows下可以下载客户端进行管理
- 在Linux和Unix下主要通过命令行进行操作
- 在Eclipse中可以通过Egit插件实现github的使用，详见文档 http://wiki.eclipse.org/EGit/User_Guide
- 主要命令：具体的操作命令见相关文档
  - Commit 在本地库中多次提交;
  - Push 发布到github中
  - Clone check-out项目到本地
  - Merge 合并分支

在本地机器上安装Git

- Linux：
  - $ yum install git-core #在redhat等系统下用yum
  - $ apt-get install git-core #在debian, ubuntu等系统下用apt-get
- Windows：http://msysgit.github.io/ 下载安装包
- 远程Git服务器：Github(http://www.github.com)

- 说明：
  - Linux和Windows下的Git安装操作二选一即可
  - 在Github上申请账号
  - 使用命令行方式完成实验，避免图形界面下的操作。

在Eclipse或MyEclipse中安装和使用Git Plugin

- http://blog.csdn.net/chinaonlyqiu/article/details/8830050

结束

2016年6月2日