* 1. 软件项目管理的重要性

软件的项目管理，是保证软件项目按照预定的成本、进度、质量顺利完成的基础。它所涉及的范围覆盖了整个软件工程过程，关键问题是必须对软件项目的工作范围、可能风险、需要资源、要实现的任务、经历的里程碑、花费工作量、进度安排等做好合理的管理。而软件项目管理的根本目的，就是为了让软件项目尤其是大型项目的整个软件从分析、设计、编码到测试、维护等全部生命周期，都能在管理者的控制之下，以预定成本按期，按质的完成软件交付用户使用。软件项目管理在项目计划、项目控制和人员管理等方面的内容是，软件开发中具有决定性意义的过程，这些工作做的好坏，直接决定着整个软件开发项目的成败。

什么是软件项目管理？大家都知道，软件开发中有太多的不可预知性，这些不可预知的事物就是潜在的风险源。如果缺乏好的管理，这些不可预知的事物就会带领你一步一步的走向失败；相反，通过良好的管理，合理的规避风险，有效的控制这些不可预知的事物，软件项目就会一步一步随着你的设计思路起向成功，这就需要我们了解什么是软件的项目管理。软件的项目管理，类似于传统意义上的项目管理，最早出现在美国， 20世纪70年代中期，当时美国国防部专门研究了软件开发不能按时提交，预算超支和质量达不到用户要求的原因，结果发现70%的项目是因为管理不善引起的，而非技术原因。于是软件开发者开始逐渐重视起软件开发中的各项管理。

软件的项目管理，类似于传统意义上的项目管理，最早出现在美国， 20世纪70年代中期，当时美国国防部专门研究了软件开发不能按时提交，预算超支和质量达不到用户要求的原因，结果发现70%的项目是因为管理不善引起的，而非技术原因。于是软件开发者开始逐渐重视起软件开发中的各项管理。

软件的项目管理目的就是保证软件项目按照预定的成本、进度、质量顺利完成。它所涉及的范围覆盖了整个软件工程过程，关键问题是必须对软件项目的工作范围、可能风险、需要资源、要实现的任务、经历的里程碑、花费工作量、进度安排等做好合理的管理。 这种管理在技术工作开始之前就应开始，在软件从概念到实现的过程中继续进行，当软件工程过程最后结束时才终止。

同时，由于软件企业与传统工业企业不同，与现代企业的其他行业也不同，所以软件的项目管理和其他的项目管理相比有其特殊性。软件是纯知识产品，其开发进度和质量很难估计和度量，生产效率也难以预测和保，软件系统的复杂性也导致了开发过程中各种风险的难以预见和控制。因此，软件企业最主要的“资产”是一批掌握技术、熟悉业务、懂得管理的“人”。软件企业主要的成本是人的成本，软件企业主要的财富积累是知识和经验的积累。因此，软件项目组的管理过程，几乎全部是围绕“人”来进行的管理。而作为被管理对象的“人”本身管理的讨论，则越来越成为软件领域所要讨论的核心问题。

那软件项目管理的意义是什么？同时，由于软件企业与传统工业企业不同，与现代企业的其他行业也不同，所以软件的项目管理和其他的项目管理相比有其特殊性。软件是纯知识产品，其开发进度和质量很难估计和度量，生产效率也难以预测和保，软件系统的复杂性也导致了开发过程中各种风险的难以预见和控制。因此，软件企业最主要的“资产”是一批掌握技术、熟悉业务、懂得管理的“人”。软件企业主要的成本是人的成本，软件企业主要的财富积累是知识和经验的积累。因此，软件项目组的管理过程，几乎全部是围绕“人”来进行的管理。而作为被管理对象的“人”本身管理的讨论，则越来越成为软件领域所要讨论的核心问题。

而软件项目管理的根本目的，就是为了让软件项目尤其是大型项目的整个软件从分析、设计、编码到测试、维护等全部生命周期，都能在管理者的控制之下，以预定成本按期，按质的完成软件交付用户使用。

1.2配置管理工具的功能分析

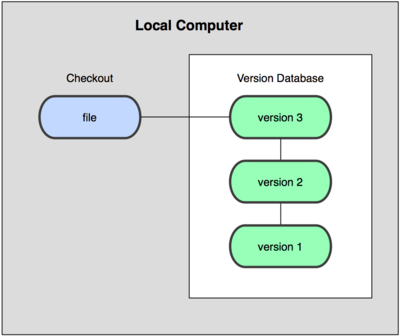
首先要了解什么是版本控制？

如果你是位图形或网页设计师，可能会需要保存某一幅图片或页面布局文件的所有修订版本（这或许是你非常渴望拥有的功能）。采用版本控制系统 （VCS）是个明智的选择。有了它你就可以将某个文件回溯到之前的状态，甚至将整个项目都回退到过去某个时间点的状态。你可以比较文件的变化细节，查出最 后是谁修改了哪个地方，从而导致出现怪异问题，又是谁在何时报告了某个功能缺陷等等。使用版本控制系统通常还意味着，就算你乱来一气把整个项目中的文件改 的改删的删，你也照样可以轻松恢复到原先的样子。但额外增加的工作量却微乎其微。

### 本地版本控制系统

许多人习惯用复制整个项目目录的方式来保存不同的版本，或许还会改名加上备份时间以示区别。这么做唯一的好处就是简单。不过坏处也不少：有时候会混淆所在的工作目录，一旦弄错文件丢了数据就没法撤销恢复。

为了解决这个问题，人们很久以前就开发了许多种本地版本控制系统，大多都是采用某种简单的数据库来记录文件的历次更新差异（见图 1-1）。

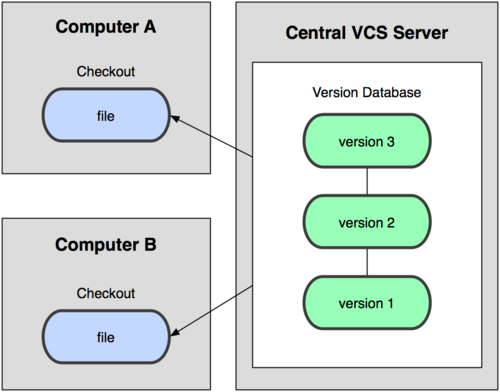


1-2-1本地版本控制系统

其中最流行的一种叫做 rcs，现今许多计算机系统上都还看得到它的踪影。甚至在流行的 Mac OS X 系统上安装了开发者工具包之后，也可以使用 rcs 命令。它的工作原理基本上就是保存并管理文件补丁（patch）。文件补丁是一种特定格式的文本文件，记录着对应文件修订前后的内容变化。所以，根据每次 修订后的补丁，rcs 可以通过不断打补丁，计算出各个版本的文件内容。

### 集中化的版本控制系统

接下来人们又遇到一个问题，如何让在不同系统上的开发者协同工作？于是，集中化的版本控制系统（ Centralized Version Control Systems，简称 CVCS ）应运而生。这类系统，诸如 CVS，Subversion 以及 Perforce 等，都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，而协同工作的人们都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。多年以来，这 已成为版本控制系统的标准做法（见图 1-2）。



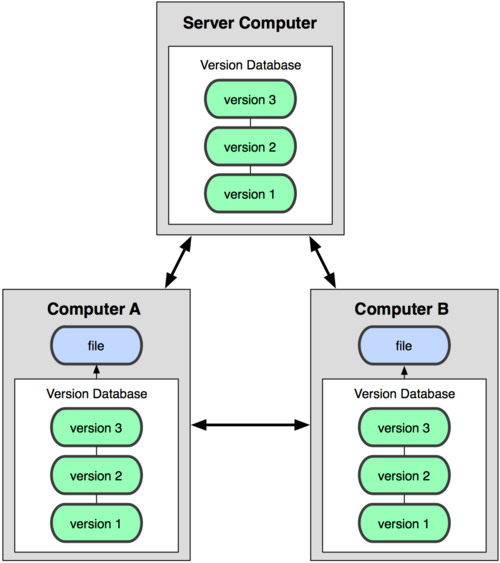
1-2-2 集中化的版本控制系统

这种做法带来了许多好处，特别是相较于老式的本地 VCS 来说。现在，每个人都可以在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，并且管理一个 CVCS 要远比在各个客户端上维护本地数据库来得轻松容易。

事分两面，有好有坏。这么做最显而易见的缺点是中央服务器的单点故障。如果宕机一小时，那么在这一小时内，谁都无法提交更新，也就无法协同工作。要 是中央服务器的磁盘发生故障，碰巧没做备份，或者备份不够及时，就还是会有丢失数据的风险。最坏的情况是彻底丢失整个项目的所有历史更改记录，而被客户端 提取出来的某些快照数据除外，但这样的话依然是个问题，你不能保证所有的数据都已经有人事先完整提取出来过。本地版本控制系统也存在类似问题，只要整个项 目的历史记录被保存在单一位置，就有丢失所有历史更新记录的风险。

### 分布式版本控制系统

于是分布式版本控制系统（ Distributed Version Control System，简称 DVCS ）面世了。在这类系统中，像 Git，Mercurial，Bazaar 以及 Darcs 等，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。这么一来，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。因为每一次的提取操作，实际上都是一次对代码仓库的完整备份（见图 1-3）。



1-2-3 分布式版本控制系统

更进一步，许多这类系统都可以指定和若干不同的远端代码仓库进行交互。籍此，你就可以在同一个项目中，分别和不同工作小组的人相互协作。你可以根据需要设定不同的协作流程，比如层次模型式的工作流，而这在以前的集中式系统中是无法实现的。

对于任何软件项目管理工具，都有几个基本的要点，包括项目计划定制，项目的控制，和人员的管理是核心内容，是软件开发中具有决定性的意义。

1. 软件项目的计划

制定计划的过程就是一个对项目逐渐了解掌握的过程，通过认真地制定计划，项目经理可以知道哪些要素是明确的，哪些要素是要逐

渐明确的，通过渐近明细不断完善项目计划。阶段计划中包含的工作汇报和下一阶段工作安排是掌握项目进度的依据，从阶段计划对照总体计划，才能一目了然地看出工作的进展情况。制定计划的过程，也是在进度、资源、范围之间寻求一种平衡的过程。制定计划的精髓不在于写出一份好看的文档，而在于运用您的智慧去应对各种问题和面临风险并尽可能做出前瞻性的思考。一旦计划被负责任地完成，他就可以给自己一个和管理层或客户交流与协商的基础，帮助你在项目过程中防范各种问题的出现，帮助你保证项目按时完成。软件项目的计划是一个软件项目正式进入系统实施的启动阶段,主要进行的工作包括:确定详细的项目实施范围、定义递交的工作成果、评估实施过程中主要的风险、制定项目实施的时间计划、成本和预算计划、人力资源计划等内容。

同时，计划变更也是计划的重点，由于项目计划无法保证一开始就预测得非常准确，在项目进行过程中也不能保证准确有力的控制，导致项目计划与项目实际情况不符的情况经常发生，所以必须有效处理项目的变更。变更控制计划主要是规定变更的步骤、程序，配置管理计划就是确定项目的配置项和基线，控制配置项的变更，维护基线的完整性，向项目干系人提供配置项的准确状态和当前配置数据。

1. 软件项目的控制

对于软件开发项目而言，控制是十分重要的管理活动，主要包括软件工程控制活动中的质量保证和配置管理。

软件质量保证是在软件过程中的每一步都进行的“保护性活动”。主要有基于非执行的测试（也称为评审）、基于执行的测试（即通常所说的测试）和程序正确性证明。而他的最主要的活动软件评审是在发现及改正错误的成本相对较小时就及时发现并排除错误。审查和走查是进行正式技术评审的两类具体方法。审查过程不仅步数比走审多，而且每个步骤都是正规的。由于在开发大型软件过程中所犯的错误绝大数是规格说明错误或设计错误，而正式的技术评审发现这两类错误的有效性高达75%，因此是非常有效的软件质量保证方法。

软件配置管理是应用于整个软件过程中的保护性活动，它是在软件整个生命周期内管理变化的一组活动。由一组相互关联的对象组成，这些对象也称为软件配置项，它们是作为某些软件工程活动的结果而产生的。除了文档、程序和数据这些软件配置项之外，用于开发软件的开发环境也可置于配置控制之下。一旦一个配置对象已被开发出来并且通过了评审，它就变成了基线。对基线对象的修改导致建立该对象的版本。版本控制是用于管理这些对象而使用的一组规程和工具。变更控制是一种规程活动，它能够在对配置对象进行修改时保证质量和一致性。配置审计是一项软件质量保证活动，它有助于确保在进行修改时仍然保持质量。状态报告向需要知道关于变化的信息的人，提供有关每项变化的信息。

1. 人员组织与管理

软件开发中的开发人员是最大的资源。对人员的配置、调度安排贯穿整个软件过程，人员的组织管理是否得当，是影响对软件项目质量的决定性因素。首先在软件开发的一开始，要合理的配置人员，根据项目的工作量、所需要的专业技能，再参考各个人员的能力、性格、经验，组织一个高效、和谐的开发小组。一般来说，一个开发小组人数在5到10人之间最为合适，如果项目规模很大，可以采取层级式结构，配置若干个这样的开发小组。

在组建开发组时，还应充分估计到开发过程中的人员风险。由于工作环境、待遇、工作强度、公司的整体工作安排和其他无法预知的因素，一个项目尤其是开发周期较长的项目几乎无可避免的要面临人员的流入流出。如果不在项目初期对可能出现的人员风险进行充分的估计，作必要的准备，一旦风险转化为现实，将有可能给整个项目开发造成巨大的损失。以较低的代价进行及早的预防是降低这种人员风险的基本策略。

在选择人员的问题上，要结合实际情况来决定是否选入一个开发组员。并不是一群高水平的程序员在一起就一定可以组成一个成功的小组。作为考察标准，技术水平、与本项目相关的技能和开发经验、以及团队工作能力都是很重要的因素。在决定一个开发组的开发人员数量时，除了考虑候选人素质以外，还要综合考虑项目规模、工期、预算、开发环境等因素的影响，这些在人员管理上的重点将直接决定软件开发工作的成败。

1.3GitHub的使用说明

1. **Git简史以及介绍**

同生活中的许多伟大事件一样，Git 诞生于一个极富纷争大举创新的年代。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（1991－2002年间）。到 2002 年，整个项目组开始启用分布式版本控制系统 BitKeeper 来管理和维护代码。到了 2005 年，开发 BitKeeper 的商业公司同 Linux 内核开源社区的合作关系结束，他们收回了免费使用 BitKeeper 的权力。这就迫使 Linux 开源社区（特别是 Linux 的缔造者 Linus Torvalds ）不得不吸取教训，只有开发一套属于自己的版本控制系统才不至于重蹈覆辙。他们对新的系统制订了若干目标：\*速度 \* 简单的设计 \* 对非线性开发模式的强力支持（允许上千个并行开发的分支） \* 完全分布式 \* 有能力高效管理类似 Linux 内核一样的超大规模项目（速度和数据量）自诞生于 2005 年以来，Git 日臻成熟完善，在高度易用的同时，仍然保留着初期设定的目标。它的速度飞快，极其适合管理大项目，它还有着令人难以置信的非线性分支管理系统（见第三章），可以应付各种复杂的项目开发需求。

Git是一个开源的分布式版本控制系统，最早由Linus Torvalds最先创建开发的代码管理工具，用以有效、高速的处理各种规模的项目版本管理，它是 Linux Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件，后来得到广泛的使用。

GitHub可以托管各种git库，并提供一个web界面，但与其它像 SourceForge或GoogleCode这样的服务不同，GitHub的独特卖点在于从另外一个项目进行分支的简易性。为一个项目贡献代码非常简单：首先点击项目站点的“fork”的按钮，然后将代码检出并将修改加入到刚才分出的代码库中，最后通过内建的“pull request”机制向项目负责人申请代码合并。已经有人将GitHub称为代码玩家的MySpace。

分布式相比于集中式的最大区别在于开发者可以提交到本地，每个开发者机器上都是一个完整的数据库。

1. **功能特性**

从一般开发者的角度来看，git有以下功能：

1、从服务器上克隆数据库（包括代码和版本信息）到单机上。这保证了在断网的情况下也能进行软件的项目管理（虽然是单机的）。

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。

以上的6、7、8点都方便分布式的开发，就是一个开源软件，在团队之间沟通不方便的情况下也能很好的进行软件项目管理。

从主开发者的需求（假设主开发者不用开发代码）看，git有以下功能：

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。

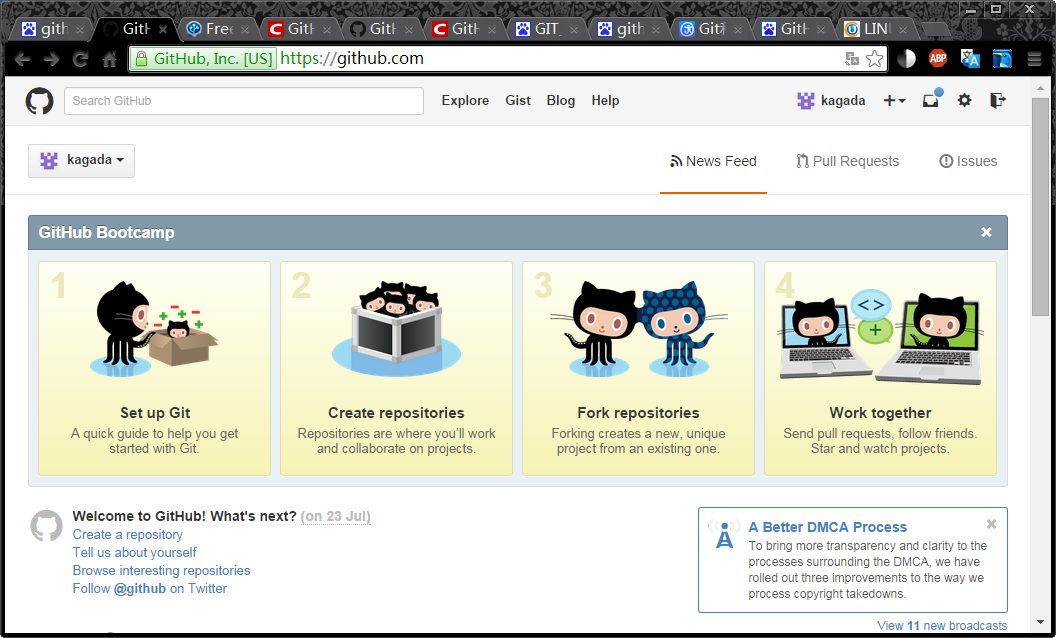
不符合常规思维。

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。

1. **安装与使用**

从前文得知，Git是这个分布式版本控制系统，而Github是Git的一个Web应用。在linux下我们可以很容易地在shell中使用Git，但是在windows下，要安装一个github的shell环境相当费事，所以我们不费事了，选择相对要简单很多的方法。使用Github+SourceTree的组合来进行软件项目的版本控制。

首先我们上Gtihub的网站，一进入就是注册，非常方便，但是我这里已经有了账号，直接登录。这里一提，github是git项目的代理托管仓库，而sourcetree相当于是git项目管理的一个客户端。



1-3-1

进来之后是导航，可以从图片中看到，这里有四个功能的导航，分别是Set up Git，创建你的Git项目，Create repositorise，创建仓库，要想创建一个Git项目，首先得创建一个仓库（repository），这样项目才有一个存放的空间。然后是Fork repository，fork是分叉的意思，在linux中是创建子进程，而在项目管理软件中，这是从一个项目中创建一个分支，这个分支相当于一个Git项目的一个备份，在Git中，这个备份会存放到我们指定的我们的仓库中。最后一个是Work together，这个是指导我们讲其他一起开发的开发人员拉入项目开发人员名单（这在创建私有项目时会有用）。

Git的用户使用指导

1. 客户端的安装与配置
2. 在windows下安装bash git客户端
3. 在Git bash中console中配置Git
4. 配置username

Git config—global user.name “haiyang”

1. 配置email

Git config—global user.email [haiyang@hotmail.com](mailto:haiyang@hotmail.com)

1. 生成SSH public key，用于在github服务端注册。每台客户端key只需要生成一次，然后备份起来起来即可。 cd ~/.ssh

ssh-keygen -t rsa -C "xuhansheng@gmail.com"//上述step 4配置的email，一路回车 打开C:\Documents and Settings\Administrator\.ssh\id\_rsa.pub,里面的内容就是SSH public key。

1. 服务端配置
2. 申请帐号

http://github.com/plans，免费帐号服务端有300M的磁盘空间

1. 在服务端创建项目名，本文档以StompTest为例。
2. https://github.com/account的SSH public key中增加客户端生成的key。
3. 客户端初始化项目

Sample：

mkdir StompTest

cd StompTest

git init git add .

git commit -m 'initial commit'

     git remote add origin git@github.com:xuhansheng/StompTest.git     git push origin master

1. 从服务端克隆一个存在的项目

git clone git@github.com:xuhansheng/StompTest.git

注意clone后面的参数url 需要用private里面url。Read-Only和HTTP Read-Only clone下来的代码没有修改服务端代码的权限。

1. 通常的使用流程

1) 从服务端同步代码

git pull origin master

2) 往服务端增加一个Readme的文件

git add Readme

git commit –a

  git push origin master

3) 修改 后的文件

discard

git checkout Readme

**GitHub详细操作明示：**

**克隆Git版本库**

从你的git版本服务器上clone版本库到本地开展工作，或者从github上clone一个开源项目的代码库，这时候你就需要git clone命令：

# git clone url [newname] # 克隆到当前目录 $ git clone https://github.com/octocat/Spoon-Knife.git # 在当前目录新建目录knife 将Spoon-Knife克隆到knife目录 $ git clone https://github.com/octocat/Spoon-Knife.git knife

不管是git init还是git clone都会在你的本地创建一个包含.git目录的git版本库

**提交更新**

通过git init命名或git clone命令都可以在本地创建一个git版本库，版本库创建成功后便可以在本地进行暂存文件，提交更新等操作了。

在git中可以使用git status查看文件的更改信息，但这个信息比较概要。如果想获取更为详细的更改信息，可以使用git diff命令：

# 查看未暂存文件的变化（与最近一次的暂存/提交比较） $ git diff # 查看已暂存文件的变化（与最近一次提交比较） $ git diff --cached # 查看与版本库中任一版本的变化 $ git diff 2bd094a # 查看任意两个版本间的变化 $ git diff 2bd094a 78ab3d1 # 具体到某个文件 $ git diff 2bd094a 78ab3d1 README

关于diff再补充一点内容：如果暂存区/已暂存区都不存在任何未提交的文件，那么diff将对最新版本中与上一版本进行比较。

清楚文件更改信息之后，便可使用git commit对暂存区文件进行提交操作。

如果想提交未暂存文件，可以使用git commit -a命令：

$ git commit -a -m 'all changes'

**删除文件**

**删除Git版本库中的某个文件**

可以使用git rm <file>：

$ git rm myfile

当然其实你也可以直接在文件目录中手动删除，这两者的区别在于：使用git rm命令相当于手动删除后使用add命令将更改添加到暂存区域。

**删除暂存区的某个文件**

$ git reset HEAD <file>

**回退版本**

如果想撤销某（几）次提交，回退到某个版本，可以使用git reset [--mode] <commit>：

$ git reset [--(mixed|soft|hard)] cec8506

* –mixed 默认模式，回退到某次提交后的未暂存状态
* –soft 回退到某次提交后的暂存状态
* –hard 回退到某次提交完整状态，舍弃提交后的所有修改

**文件重命名**

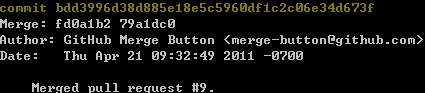
git不会跟踪在文件目录中手动的文件重命名操作，如果手动重命名了某个文件，git会认为这是一次delete-create操作。但是，你可以使用git mv命令完成重命名（文件目录亦使用该命令）：

$ git mv oldfile newfile

**查看提交历史**

这是git中使用平率非常高的一个操作，git中查看提交历史的功能也非常强大，提供各种筛选和输出格式定制功能。

最简单的，运行git log命令，你将看到一个详细的提交日志：



git-log

# 当然也可以只查看某个版本 $ git log fd0a1b2

信息内容都很好理解，重点说说第一行commit后这个40个字符的字符串，这是该次提交的对应的SHA-1值，在git中，会对提交（commit）、文件（blob）、目录（tree）、标签（tag）生成一个唯一的SHA-1值，git就是基于此来得知文件或目录的改动，因为这四类对象计算得到的SHA-1值都是唯一的，同时你也可以直接使用SHA-1值来指代相应的对象。比如：

$ git show bdd3996 # 查看某个版本下具体某个文件 $ git show bdd3996 README

git log还有很多命令选项来定制历史记录

|  |  |
| --- | --- |
| **选项** | **说明** |
| -(n) | 仅显示最近的 n 条提交 |
| –since,–after | 仅显示指定时间之后的提交 |
| –until,–before | 仅显示指定时间之前的提交 |
| –author | 仅显示指定作者相关的提交 |
| –committer | 仅显示指定提交者相关的提交 |
| –reverse | 按时间倒序显示 |
| -p | 按补丁格式显示每个更新之间的差异 |
| –stat | 显示每次更新的文件修改统计信息 |
| –shortstat | 只显示 –stat 中最后的行数修改添加移除统计 |
| –name-only | 仅在提交信息后显示已修改的文件清单 |
| –name-status | 显示新增、修改、删除的文件清单 |
| –abbrev-commit | 仅显示 SHA-1 的前几个字符，而非所有的 40 个字符 |
| –relative-date | 使用较短的相对时间显示（比如，“2 weeks ago”） |
| –graph | 显示 ASCII 图形表示的分支合并历史 |
| –pretty | 使用其他格式显示历史提交信息。可用的选项包括 oneline，short，full，fuller 和 format（后跟指定格式） |

可以通过对上述选项进行组合定制出更为个性化的日志信息，比如：

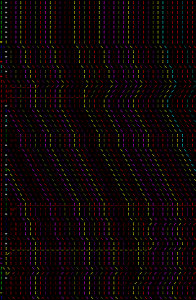
$ git log --committer 'god' --shortstat --pretty=oneline

该命令将以单行模式显示由god提交的统计信息。

$ git log -p -5

显示最近5次提交的，并显示其差异

除此之外，git log –graph也很好玩。我git log –graph了一下git项目的日志，非常壮观。

[](http://omiga.org/img/git.png)

git graph log

**–pretty=format**

单独介绍下–pretty=format选项。使用format和占位符可以定制出更为个性化的显示格式。

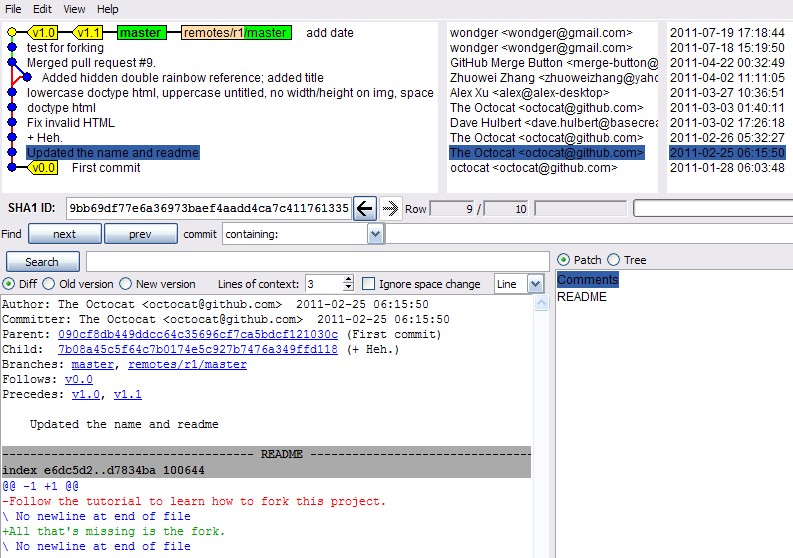
|  |  |
| --- | --- |
| **选项** | **说明** |
| %H | 提交对象（commit）的完整哈希字串 |
| %h | 提交对象的简短哈希字串 |
| %T | 树对象（tree）的完整哈希字串 |
| %t | 树对象的简短哈希字串 |
| %P | 父对象（parent）的完整哈希字串 |
| %p | 父对象的简短哈希字串 |
| %an | 作者（author）的名字 |
| %ae | 作者的电子邮件地址 |
| %ad | 作者修订日期（可以用 -date= 选项定制格式） |
| %ar | 作者修订日期，按多久以前的方式显示 |
| %cn | 提交者(committer)的名字 |
| %ce | 提交者的电子邮件地址 |
| %cd | 提交日期 |
| %cr | 提交日期，按多久以前的方式显示 |
| %s | 提交说明 |

$ git log --pretty=format:'%h by %ce at %cd'

该命令将以“简短SHA-1 by 提交者 at 提交时间”的格式显示日志

**图形化日志界面gitk**

gitk命名会启用图形化的日志界面

[](http://omiga.org/img/gitk.jpg)

gitk

**撤销操作**

前面已经介绍过使用[git reset](http://omiga.org/blog/archives/1896#git-reset)来撤销暂存区的文件，以及回退整个版本。但如果只想恢复某个文件，则需要使用checkout — <file>命名：

$ git checkout -- README

它只有在修改文件还没有暂存的情况下，使用最近的提交版本进行恢复。如果文件已经暂存，则需要先使用git reset HEAD <file>从暂存区删除文件，再使用该命令。

**修改最后一次提交**

git commit –amend命名允许你对最后一次提交信息重新编辑。相当于重新进行一次提交，覆盖掉上一次提交。

**远程版本库的使用**

尽管git在本地也可以方便地进行版本管理，但是多人协作，或者是多地操作时，总需要使用到远程仓库来进行版本维护。在前面创建版本库的内容中其实使用clone命名就已经是在与远程版本库进行交互了，clone远程库后便会自动创建一个名为origin的远程库，可以使用git remote -v命名查看远程库的详细信息。

$ git remote -v

而在实际工作中，我们可能需要频繁地与某一个或几个远程库交互，那么更好的办法是使用一个别名把远程库保存起来。git添加远程库的方法很简单：

# $ git remote add <name> <remote-url> $ git remote add pro-git https://github.com/progit/progit.git

这样便将https://github.com/progit/progit.git这个远程git版本库添加并命名为pro-git，后续你只需要使用pro-git便可以指代progit的远程库。

**push**

当你完成本地工作，并将改动提交到本地版本库后，你便可以使用push将本地提交推送到远程仓库了：

# $ git push <remote-name> <branch-name> $ git push pro-git master

默认会使用origin和master作为远程仓库和本地分支的名称。

当然你也可以将本地分支推送到远程仓库作为一个分支：

# $ git push <remote-name> <local-branch>:<remote-branch> $ git push pro-git master:git-branch

当<local-branch>为空时，会尝试删除远程分支：

$ git push pro-git :git-branch

上述命令会删除远程仓库中的”git-branch”分支

**fetch与pull**

fetch与pull命令都会将一个远程仓库抓取到本地，不同的是fetch仅仅是将远程仓库抓取到本地，以供进行后续操作；pull除了将远程仓库抓取到本地，还会试图与本地当前分支进行合并。

他们与clone不同的是，clone会copy一份版本仓库到本地，如果本地已存在版本仓库，则会被clone后的仓库替换。而fetch和pull都需要在已有本地仓库的条件下操作，不能作为创建本地仓库的方法，即是必须先git init或是git clone后才能使用fetch和pull。

**删除远程仓库**

$ git remote -d <remote-name>

准确一点说，这只是删除远程仓库在本地的别名，而不是真正删除远程服务器上的git仓库。

**重命名远程仓库**

$ git remote rename oldname newname

**Git别名**

虽然git中很多命名都简单易记，但每次都手动输入这些命令确实会浪费不少时间，而且也有那么些命令选项非常冗长，这时就可以使用别名来简化命令的输入了。

别名属于配置项内容，所以需要使用git config命令，如可以为“checkout -b”命令配置别名“cob”

$ git config --global alias.cob 'checkout -b'

为“commit -a -m”配置别名“cam”:

$ git config --global alias.cam 'commit -a -m'

为单行图像化显示log命令“log –pretty=oneline –graph”配置别名“lol”:

$ git config --global alias.lol 'log --pretty=oneline --graph'

**Git的基本命令：**

**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，下面是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。

**对于Git，最基本的使用要求：**

1. 先更新，再提交
2. 2.  按最小功能（或是单功能）提交代码  ，每次提交以完成一项功能为宜，添加一个小特性或修复一个 bug 。   尽量不要按时间为基础，比如到下班的时候提交一次，不管这个功能是否完成。在开发功能模块的时候，可以为每个小功能的测试通过后，进行提交一次。       我们提倡多提交，也就能多为代码添加上保险。

3. 不要提交未完成的代码 ， 代码在提交之前，首先要确认自己已经检查无误。

     项目经理在需要确保开发小组成员在签出代码之后能够在统一的环境中进行开发。

4. 每次提交必须书写明晰的注释

     提交时可以使用一下方式，尽量说明你的代码实现了什么功能：       add a feature.       ﬁx a bug.  remove a feature.

      尽量不要使用一下的方式, 这些方式没有说明你修改的代码实现什么功能：       add some ﬁles       delete some ﬁles   merge 085bb3bc and a11bef06a

 5. 提交时注意不要提交本地自动生成的文件,配置忽略配置文件

      例如 editplus 生成的 .bak  ，Windows 生成的缩略图 Thumbs.db  ，Smarty 生成  的 templates\_c 文件夹中的缓存文件等等。

 6. 不要提交自己不明白的代码, 需要看懂以后写上注释       7 不要放入与代码无关的文件

     版本控制系统主要是管理代码的有序更新，方便程序员能够在发现错误的时候，回到

现错误以前。所以与代码无关的文件尽量不要放进来，如编辑器软件，非项目本人的

发文档，非文本格式的文档，上传的图片或文件。

**Git、SVN、CVS、VSS的比较：**

**Git与SVN的比较**

**1.GIT是分布式的，SVN不是：**

这一点绝对是最最关键的重点，与原作者不同，我要特别强调的是这一点对身处大陆的码农们的重要性。为什么？因为人家老外下全套android源码要几个小时，而我们要十几二十个小时，甚至好几天，还会不停断线！

分布式的最大好处，在于当你要切换不同的提交，不同的分支时，不再需要联网。试想一下，要是google的android源码用svn（这个是纯假设，实际上是完全不可行的，后面会提到），本来在大陆下一次代码就够慢的了，要是每天再切个几次分支和提交，那就别活了。所以在这一点上，我不说git完爆svn，应该说是分布式完爆非分布式。

**2. GIT把内容按元数据方式存储，而SVN是按文件：**

说实话，我没看出来原文列第二点出来是为了说明什么。这确实是个区别，但一般用户是不需要了解的。

看到评论中有人提到了提交冲突的问题。真发生了大规模冲突，不管git还是svn，都是挺让人感到头疼的，两者对冲突的解决办法也是差不多的。个人见解，冲突是靠良好的团队管理和项目分工来尽力避免的，但真发生了，解决起来都差不多，也没见哪个版本控制系统是以解决冲突效率高作为卖点的。

**3. GIT分支和SVN的分支不同：**

分支确实是一个重点，只不过原文没说到点子上，不过看文章是挺久之前的，可能那时候git还没现在这么普及（当然现在国内也未必有多普及），也情有可原吧。

这里我就说最关键的一点，你可以看完整的android代码，如果没有，可以用git clone个linux的完整代码库，然后用git branch -r命令看看，你会发现有几十个甚至近百个分支，如果你只用svn的话，你一定完全无法想象。但这就是git分支的优势，因为git是基于差异来管理分支的，其分支的代价极小，再结合第一点，所以其切换分支也极为方便和快捷。这一点上我认为git是优于svn的，之所以我不说完爆，是因为这一点要结合项目需求，你的项目不是android，不是linux，没有大量分支并行开发、合并的需求，其实用svn也是可以的。但用git，你可以为一个单独的小功能拉分支，为一轮完整的测试拉分支，为你的每一个客户单独拉分支，等等等等，基本上是想拉就拉，这一点svn应该是比不上的。

至于原文提到的合没合并的问题，svn我不常用，git下用git log命令，加上--graph参数，再配合--oneline和--color，我觉得是很方便直观的。

**4. GIT没有一个全局的版本号，而SVN有：**

原文承认这一点不如svn，我也承认，但要补充说明一下，git每次提交产生的40位（160bit）哈希值，是完全可以等同于svn全局id的作用的，之所以说不如，只是因为不好记而已。

**5. GIT的内容完整性要优于SVN：**

完整性这一点git要优于svn这是事实，虽然我个人不论git还是svn，都没碰到过这类问题。

**CVS和SVN比较：**

**1 存储类型格式**

CVS是个基于RCS文件的版本控制系统。每个CVS文件都不过是普通的文件，加上一些额外信息。这些文件会简单的重复本地文件的树结构。因此，不必担心有什么数据损失，如果必要的话可以手工修改RCS文件。

SVN是基于关系数据库的(BerkleyDB)或一系列二进制文件的(FS\_FS)。一方面这解决了许多问题 (例如，并行读写共享文件)以及添加了许多新功能(例如运行时的事务特性。)。然而另一方面，数据存储由此变得不透明。

**2 速度**

CVS比较慢。

整体而言,由于架构实现的不同, SVN的确比CVS快很多。在网络上它只传输很少的信息并支持更多的离线模式的功能。但这也是有代价的。速度的代价就是巨大的存储（完全备份所有的工作文件）。

**3 标志&分支**

SVN把采用标志和分支而抛弃了其他三件东西，实际上这意味着他们把这个概念替换为在档案库内部复制文件或目录以便保存日志。这样一来，无论标志创建还是分支创建都只是仓库内部的文件复制了。对分支而言：现在分支不过是在仓库内部的一个单独的目录而已了，不像早期还有些什么交错。对标志而言：已经不能对代码加标志了。在某种程度上说，SVN全文件编号补足了这个缺陷，SVN里整个仓库都有版本号，但不是针对单个文件。

**4 元数据**

　CVS只允许存储文件。

　SVN允许一个文件有任意多的可命名属性，功能十分完全。

**5 文件类型**

CVS最初是为文本文件存储而设计的。因此其他文件类型（二进制，统一码）文件的支持几乎没有，如需要的话则要有其他信息，并且客户端服务器端都要调整。

　　SVN会关心所有的文件类型，不需要你来手工操作。

**6 回滚**

　　CVS允许任意的回滚，在任意一个已递交的版本上，尽管这要花些时间（所有的文件都要分别处理）。

　SVN不允许递交后回滚。建议把版本库里好的状态版本加到末尾，覆盖掉损坏的版本。而损坏的版本无论如何也是会存在数据库里的。（SVN的滚回操作实际上是merge操作）

**7 事务**

　CVS中的“零或一”事务原则根本没有实现。如果检入几个文件的话（加到服务器上），很有可能部分文件完成了，而另几个没有。作为一个潜规则，手工纠正这些并且对余下的文件 (而不是所有文件)一一重复检入。这样这些文件将在两阶段中被检入。SVN的确支持“零或一”事务原则，这是SVN的一大优势。

**Vss与Svn 的对比**  
　　**1. 支持重命名，这对 Java 开发来说非常重要。**　　为了得到更好的代码，开发中需要经常进行重构，重构就经常涉及到文件的重构名，而重命名中 VSS中是不被支持的。  
　**2. 开发的时候不一定要锁定。**　　VSS一方面导致重构不方便，另一方面，不能离线开发，使用 SVN就不同，可以带回家继续开发，回来后，提交就行了。  
　　**3. 多平台。**　　SVN可以支持多个平台下的操作  
　**4. 更好的客户端支持。**　　Eclipse 中的 VSS Plugin 不如它的 SVN Plugin 好用。一个在 Windows 下用的 SVN 客户端 TortoiseSVN 也比VSS 的客户端好用（VSS 只有微软提供的一个 GUI 客户端）。  
　　**5. 更好地与外围工具集成。**　　VSS各种各样的外围工具（主要是服务器端），满足多种需要。如果有需要，也可以自己写插件或管理脚本，开放的架构，允许我们这样做。