# Git

# Git 版本管理工具（一）

[**Git**](http://book.git-scm.com/)是一个分布式版本控制工具，它的作者 [Linus Torvalds](http://en.wikipedia.org/wiki/Linus_Torvalds) 是这样给我们介绍 [Git](http://en.wikipedia.org/wiki/Git_(software))  —— The stupid content tracker（傻瓜式的内容跟踪器）

**1、 Git 背景**

Git 最初由Linus Torvalds编写，用于 Linux 内核开发的版本控制工具。

Git 与常用的版本控制工具 [CVS](http://www.nongnu.org/cvs/)、[Subversion](http://subversion.apache.org/) 等不同，它采用了分布式版本库的方式，不必服务器端软件支持，使源代码的发布和交流极其方便。

Git 的速度很快，这对于诸如 Linux kernel 这样的大项目来说自然很重要，Git 最为出色的是它的合并跟踪（merge tracing）能力。

实际上内核开发团队决定开始开发和使用 Git 来作为内核开发的版本控制系统的时候，世界开源社群的反对声音不少，最大的理由是 Git 太艰涩难懂，从 Git 的内部工作机制来说，的确是这样。但是随着开发的深入，Git 的正常使用都由一些友好的脚本命令来执行，使 Git 变得非常好用，即使是用来管理我们自己的开发项目，Git 都是一个友好、有力的工具。现在，越来越多的著名项目采用 Git 来管理项目开发，例如：[wine](http://source.winehq.org/git/)、[hiphop-php](https://github.com/facebook/hiphop-php) 等。

Git 作为开源自由原教旨主义项目，没有对版本库的浏览和修改做任何的权限限制，但通过其他工具也可以达到有限的权限控制，比如：gitosis、CodeBeamer MR。原本 Git的使用范围只适用于 Linux / Unix 平台，但逐步并成熟了在 Windows 平台下的使用，主要归功于Cygwin与msysgit环境与TortoiseGit这样易用的 GUI 工具。其实 Git 的源代码中已经加入了对 Cygwin 与 MinGW 编译环境的支持并被逐步完善，对于 Windows 使用者是个福音。

**2、 为什么选择Git**

流行的软件版本开源管理软件，有CVS、SVN、GIT版本管理工具，Git的优势在哪里呢？

Git 入门教程，请查看我的百度空间博客：[*Blog*](http://hi.baidu.com/sunboy_2050/blog/item/6d075c2e3161c1cc8b1399d1.html)

Git 和 CVS、SVN不同，是一个分布式的源代码管理工具，它很强，也很快，Linux内核的代码就是用Git管理的，它给我们带来的直接好处有：

1. 初始化，git init, git commit -a, 就完了。对于随便写两行代码就要放到代码管理工具里的人来说，再合适不过。也可以拿git做备份系统，或者同步两台机器的文档，都很方便。

2. 绝大部分操作在本地完成，不用和集中的代码管理服务器交互，终于可以随时随地大胆地check in代码了。 只有最终完成的版本才需要向一个中心的集中的代码管理服务器提交。

3. 每次提交都会对所有代码创建一个唯一的commit id。不像CVS那样都是对单个文件分别进行版本的更改。所以你可以一次性将某次提交前的所有代码check出来，而不用考虑到底提交过那些文件。（其实SVN也可以做到这点）

4. branch管理容易多了，无论是建立新的branch，还是在branch之间切换都一条命令完成，不需要建立多余的目录。

5. branch之间merge时，不仅代码会merge在一起，check in历史也会保留，这点非常重要。

**Git 的优势主要有：**

1、更方便的 Merge

分布式管理必然导致大量的 Branch 和 Merge 操作。因此分布式版本控制系统都特别注意这方面。在传统的 CVS 里面制作 Branch 和 Merge 简直就是噩梦，Subversion 作为一个用于替代 CVS 的系统，专门改进了 Branch 操作。然而似乎人们没有注意到，Branch 是轻松了，可是 Merge 呢？如果不能很方便地 Merge 回来，做 Branch 仍然是噩梦。事实上，我就经历过在开发团队里面由于队友操作不对而在 Merge 的时候把我的许多代码都覆盖掉了。当时正是使用的 subversion 。虽然源代码仍然在历史里面，但是要去一个一个地找出被覆盖掉的文件并恢复过来确实是一件很难忘的事情。

2、更方便的管理

传统的版本控制系统使用中央仓库，一些仓库相关的管理就只能在仓库上进行。赋予开发团队每一个人中央仓库的管理权限是非常不好的。但是有时候确实会比较不方便的地方。

3、更健壮的系统

分布式系统一般情况下总是比单服务端的系统要健壮，因为当服务端一旦挂掉了整个系统就不能运行了。然而分布式系统通常不会因为一两个节点而受到影响。

4、对网络的依赖性更低

虽然现在网络非常普及，但是并不是随时随地都有高速网络，甚至有时候根本没有网络可以访问。低速的网络会让人心情烦躁，有时候就呆呆地盯着屏幕上的 commit 进度，什么事情也干不了。而没有网络连接更是致命的：你无法 commit ！这表示你进行任何改动以前都必须小心翼翼，否则你可能再也找不会你曾经写的一些代码了。

5、更少的“仓库污染”

有时候你要做一个模块，它不是太大，所以没有必要为它新建一个 branch ，但是它又不是那么小，不可能一次提交就做好。于是便会提交一些不完整的代码到仓库，有时候会导致整个程序无法运行，严重影响团队里其他人的开发。大多数人在这种情况下的解决办法都是写完之后再提交。但是作为习惯了版本控制的人来说，进行不计后果的大幅修改是经常的事情，到后来突然发现自己先前的代码没有提交，就后悔莫及了。如果是分布式系统的话就不会存在这样的问题，因为本地仓库的修改不会影响到别人的仓库。当你完成并测试以后，就可以在邮件列表里面说：我已经把这个模块做好了。然后感兴趣的人就可以从你这里 pull 你的成果了。

虽然网上各种对Git的誉美之词决不止于此，但是在Git的主站上，还是尽可能客观的对Git和Subversion进行了一番比较（[GitSvnComparsion](https://git.wiki.kernel.org/articles/g/i/t/GitSvnComparison_cb82.html)）。另外，Subversion目前通过 [SVK](http://svk.bestpractical.com/view/HomePage) 也已经提供了一定程度上的源代码库分布式的管理能力，能够实现源代码的离线提交等功能。

**3、 Git 、CVS、SVN比较**

项目源代码的版本管理工具中，比较常用的主要有：CVS、SVN、Git 和 Mercurial  （其中，关于SVN，请参见我先前的博客：[SVN常用命令](http://blog.csdn.net/sunboy_2050/article/details/6187464) 和 [SVN服务器配置](http://wenku.baidu.com/view/b0d5421ba76e58fafab00337.html?st=1)）

目前Google Code支持SVN、Git、Mercurial三种方式，例如：我上传的 [linux-kernel-source](http://code.google.com/p/linux-kernel-source/source/checkout)（Git 方式）、[sdk-java](http://code.google.com/p/jdk-java/source/checkout)（SVN方式），那么它们各有什么区别呢？

**Git与CVS 的区别**

* 分支更快、更容易。
* 支持离线工作；本地提交可以稍后提交到服务器上。
* Git 提交都是原子的，且是整个项目范围的，而不像 CVS 中一样是对每个文件的。
* Git 中的每个工作树都包含一个具有完整项目历史的仓库。
* 没有哪一个 Git 仓库会天生比其他仓库更重要。

**Git与SVN 的区别**

Git 不仅仅是个版本控制系统，它也是个内容管理系统(CMS)、工作管理系统等。如果你曾是一个使用过SVN背景的人，那么你可以很容易的做一定的思想转换，来适应Git提供的一些概念和特征。这篇文章的主要目的就是通过介绍Git能做什么，以及它和SVN在深层次上究竟有什么不同，通过比较来帮助你更好的认识Git

1. Git是分布式的，SVN不是

这是Git和其它非分布式的版本控制系统（SVN，CVS）最核心的区别。如果你能理解这个概念，那么你就已经上手一半了。需要做一点声明，Git并不是目前第一个或唯一的分布式版本控制系统。还有一些系统如 [Bitkeeper](http://www.bitkeeper.com/), [Mercurial](http://mercurial.selenic.com/about/) 等也是运行在分布式模式上的，但Git在这方面做的更好，而且有更多强大的功能特征。

Git 跟SVN一样有自己的集中式版本库或服务器。但 Git 更倾向于被使用于分布式模式，也就是每个开发人员从中心版本库的服务器上chect out代码后会在自己的机器上克隆一个自己的版本库。可以这样说，如果你被困在一个不能连接网络的地方时，就像在飞机上，地下室，电梯里等，你仍然能够提交文件，查看历史版本记录，创建项目分支等。对一些人来说，这好像没多大用处，但当你突然遇到没有网络的环境时，这个将解决你的大麻烦。

同样，这种分布式的操作模式对于开源软件社区的开发来说也是个巨大的恩赐，你不必再像以前那样做出补丁包，通过email方式发送出去，你只需要创建一个分支，向项目团队发送一个推请求。这能让你的代码保持最新，而且不会在传输过程中丢失，一个这样的优秀案例就是： [GitHub.com](http://www.github.com/)

有些谣言传出来说subversion将来的版本也会基于分布式模式。但至少目前还看不出来。

1. Git 把内容按元数据方式存储，而SVN是按文件

所有的资源控制系统都是把文件的元信息隐藏在一个类似.svn、.cvs等的文件夹里。如果你把 .git 目录的体积大小跟.svn比较，你会发现它们差距很大。因为 .git 目录是处于你的机器上的一个克隆版的版本库，它拥有中心版本库上所有的东西，例如标签、分支、版本记录等。

1. Git 分支和SVN的分支不同

分支在SVN中一点不特别，就是版本库中的另外的一个目录。如果你想知道是否合并了一个分支，你需要手工运行像这样的命令[*svn propget svn:mergeinfo*](http://jan.baresovi.cz/dr/en/subversion-mergeinfo)，来确认代码是否被合并。所以，经常会发生有些分支被遗漏的情况。

然而，处理Git 的分支却是相当的简单和有趣，你可以从同一个工作目录下快速的在几个分支间切换。你很容易发现未被合并的分支，你能简单而快捷的合并这些文件。

1. Git 没有一个全局的版本号，而SVN有

目前为止这是跟SVN相比GIT缺少的最大的一个特征。你也知道，SVN的版本号实际是任何一个相应时间的源代码快照，它是从CVS进化到SVN的最大的一个突破。Git 可以使用SHA-1来唯一的标识一个代码快照，但这个并不能完全的代替SVN里容易阅读的数字版本号。

1. Git 的内容完整性要优于SVN

Git 的内容存储使用的是[SHA-1](http://en.wikipedia.org/wiki/SHA-1)哈希算法。这能确保代码内容的完整性，确保在遇到磁盘故障和网络问题时降低对版本库的破坏。这有一个很好的关于Git 内容完整性的[讨论](http://stackoverflow.com/questions/964331/git-file-integrity)。（英文原文参考：[diff](http://boxysystems.com/index.php/5-fundamental-differences-between-git-svn/)）

[**CVS-SVN-GIT**](http://www.cnblogs.com/sujz/archive/2011/05/12/2044379.html)**综合比较**

首先，介绍几个版本控制软件相互比较的重要依据：

**（1）**版本库模型（Repository model）：描述了多个源码版本库副本间的关系，有客户端/服务器和分布式两种模式。在客户端/服务器模式下，每一用户通过客户端访问位于服务器的主版本库，每一客户机只需保存它所关注的文件副本，对当前工作副本（working copy）的更改只有在提交到服务器之后，其它用户才能看到对应文件的修改。而在分布式模式下，这些源码版本库副本间是对等的实体，用户的机器出了保存他们的工作副本外，还拥有本地版本库的历史信息。

**（2）**并发模式（Concurrency model）：描述了当同时对同一工作副本/文件进行更改或编辑时，如何管理这种冲突以避免产生无意义的数据，有排它锁和合并模式。在排它锁模式下，只有发出请求并获得当前文件排它锁的用户才能对对该文件进行更改。而在合并模式下，用户可以随意编辑或更改文件，但可能随时会被通知存在冲突（两个或多个用户同时编辑同一文件），于是版本控制工具或用户需要合并更改以解决这种冲突。因此，几乎所有的分布式版本控制软件采用合并方式解决并发冲突。

**（3）**历史模式（History model）：描述了如何在版本库中存贮文件的更改信息，有快照和改变集两种模式。在快照模式下，版本库会分别存储更改发生前后的工作副本；而在改变集模式下，版本库除了保存更改发生前的工作副本外，只保存更改发生后的改变信息。

**（4）**变更范围（Scope of change）：描述了版本编号是针对单个文件还是整个目录树。

**（5）**网络协议（Network protocols）：描述了多个版本库间进行同步时采用的网络协议。

**（6）**原子提交性（Atomic commit）：描述了在提交更改时，能否保证所有更改要么全部提交或合并，要么不会发生任何改变。

**（7）**部分克隆（Partial checkout/clone）：是否支持只拷贝版本库中特定的子目录。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **名称** | **版本库模型** | **并发模式** | **历史模式** | **变更范围** | **网络协议** | **原子提交性** | **部分克隆** |
| CVS | Client-server | Merge | Changeset | File | Pserver,ssh | No | Yes |
| SVN | Client-server | 3-way merge, recursive  merge, octopus merge | Changeset and Snapshot | Tree | custom (svn), custom (svn) over ssh,  HTTP and SSL (usingWebDAV) | Yes | Yes |
| Git | Distributed | Merge or lock | Snapshot | Tree | custom, custom over ssh, rsync,  HTTP/HTTPS, email, bundles | Yes | No |

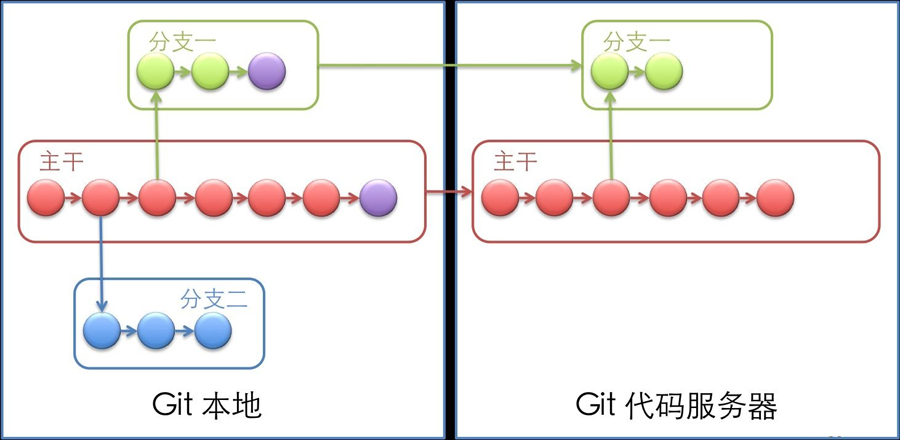
**Trunk、Branches、Tags 区别：**

Trunk：软件开发过程中的主线，开发时版本存放的目录，即在开发阶段的代码都提交到该目录上，保存了从版本库建立到当前的信息。

Branches：软件开发过程中的分支，发布版本存放的目录，即项目上线时发布的稳定版本存放在该目录中，保存了从版本库的某一特定点（不一定是版本库建立时）到当前的信息。

tags：表示标签存放的目录，tags只可读，不可写

分支主要用于在不影响Trunk其它用户情况下进行一些关于新功能的探索性或实验性的开发，待新功能完善后它也可以合并到Trunk中。



（原文，请参考我在百度空间的博客：[Git 命令参数及用法详解](http://hi.baidu.com/sunboy_2050/blog/item/b7ec1432be8d9c82a8018eff.html)）

**4、 Git 在Windows上的使用**

Git 是为Linux而生的，其最初创建人就是Linux的创始人—— Linus Torvalds

Linux环境下，使用Git 与任何Linux中的命令行工具没有什么区别，甚至在击键数上还有明显的优势。

Windows环境下，使用Git在目前看来只有两种方法：

1、使用 [**Cygwin**](http://www.cygwin.com/)（一个在Windows上运行的Linux环境）

2、使用 [**msysgit**](http://msysgit.github.com/)（Windows下提供图形界面和命令行）

Cygwin和msysgit的使用方法类似，Cygwin具有大量Linux的功能，如果只是想使用Git功能，msysgit还是最简单和快速的方法。

为了能够具备通过互联网实现与别人协作开发的能力，对于项目需要一个公开的源代码托管服务。好在，现在已经有不少可以供我们选择的，尤其是[**githost**](http://githost.com/)，更是一个中文的源代码托管服务提供方。从目前看来，在Githost上落户的项目还很少，貌似是一个新近诞生的服务提供方。如果项目对服务提供的稳定性有比较高的要求的话，还是选择较老的git源代码托管服务比较好吧。如果是在局域网内工作的小组，要使用Git做源代码管理，那就更简单了，大家安装好自己的Git，并指定一个人负责对Git版本库进行管理就好了。

（1） GitHub简介

GitHub是使用Ruby开发的，具有清爽的界面。

GitHub提供免费的源代码库托管，同时也提供付费的托管服务。通过付费私有库托管服务在财务上支持免费部分的持续运营。

GitHub提供了一套独特的代码库管理界面功能，并提供项目Wiki的能力。

GitHub提供了一系列的指南，官方网址：[GitHub](https://github.com/)

（2）Windows系统上安装Git

首先，下载并安装msysgit程序：[download](http://code.google.com/p/msysgit/)

接着，安装下载的 [Git-1.7.10-preview20120409.exe](http://code.google.com/p/msysgit/downloads/detail?name=Git-1.7.10-preview20120409.exe&can=2&q=)，可以选择最新的Git版本，以取得更好的使用效果。

安装的过程很简单，基本上可以使用默认设置。只是在设置路径的时候要注意一下，为了避免与Windows路径导致的意外情况，还是使用“Use Git Bash Only”比较安全。

Msysgit有命令行和图形UI两种使用方式，根据你的喜好选择吧，要说的是，图形UI可能不能完成所有的工作，因此在某些情况下（例如创建SSH Key），命令行还是必不可少的。Msysgit的Bash命令行对中文的支持不好，所有的中文字符都显示成了“？”。因此，为了避免麻烦，最好避免使用中文的文件名、目录名和用户名等

Msysgit 图形界面如下：



（原文，请参考我在百度空间的博客：[Git 界面GUI和命令行Command两种操作方式](http://hi.baidu.com/sunboy_2050/blog/item/76f0b39bb08e28286f068ce2.html)）

（3） 设定GitHub

要使用GitHub，首先需要创建SSH Key，SSH将用来加密本机与远端服务器之间的通信，同时也是识别你对代码所做的变更的方法。

SSH Key可以使用Git命令行来产生，如果你已经有一个SSH Key，那么在这里也可以直接使用。

要使用Git创建SSH Key 首先需要打开Git Bash 命令行，输入命令：

ssh-keygen  -C  "username@email.com"  -t  rsa

说明：username@email.com 需要更换成你自己的email地址

程序将提出一些问题，接受文件默认存放位置，当要求输入pass phrase时，如果本机安全没有问题，也可以不输入。找到当时制定的文件存储位置中id\_rsa.pub文件，这就是在GitHub上申请帐户时需要使用的SSH公钥文件。

在github.com的register中选择Free account，在后续的界面中按照要求填入相应的内容即可完成注册，很简单的。

**5、 Git 服务相关**

1、  建立Git远程服务器：

目前貌似还没有在Windows上建立Git服务器的，足见Linux在开源社区里强大的优势啊！^\_^

Hosting Git repositories, The Easy (and Secure) Way ：[gitosis](http://scie.nti.st/2007/11/14/hosting-git-repositories-the-easy-and-secure-way)

2、  基于Git的源代码托管

[**Gitorious**](http://gitorious.org/)

Gitorious is another free hosting site with a custom web interface, supporting multiple repositories per project, local installations and with open source

[**repo.or.cz**](http://repo.or.cz/)  
repo.or.cz is the oldest hosting site, accommodating many hundreds of projects, with open-sourced infrastructure and aimed at open source software. It provides full push features as well as simple mirroring mode and gitweb interface with various enhancements.

[**GitHub**](https://github.com/)  
GitHub provides both free hosting for smaller projects and paid options for private hosting and large-sized projects. It uses a custom web interface including a wiki hosting and puts emphasis on social networking of project developers

3、  关于Git的有用的联结：

Git-scm：[go url](http://book.git-scm.com/)

Git Reference：[go url](http://gitref.org/)

Git - SVN Crash Course：[go url](http://git.or.cz/course/svn.html)

Everyday GIT With 20 Commands Or So：[go url](http://bbs.chinaunix.net/thread-2035258-1-1.html)

Git User's Manual (for version 1.5.3 or newer) ：[go url](http://schacon.github.com/git/user-manual.html)

Getting Started with Git and GitHub on Windows：[go url](http://kylecordes.com/2008/git-windows-go)

**注：**本文原文，请见我在百度空间的博客  [Windows环境中使用版本管理工具Git](http://hi.baidu.com/sunboy_2050/blog/item/e42001d3411790f3562c84a1.html)

# Git 常用命令详解（二）

Git 是一个很强大的分布式版本管理工具，它不但适用于管理大型开源软件的源代码（如：[linux kernel](http://git.kernel.org/)），管理私人的文档和源代码也有很多优势（如：[wsi-lgame-pro](http://code.google.com/p/wsi-lgame-pro/)）

Git 的更多介绍，请参考我的上一篇博客：[Git 版本管理工具](http://blog.csdn.net/sunboy_2050/article/details/7527877)

**一、 Git 命令初识**

在正式介绍Git命令之前，先介绍一下Git 的基本命令和操作，对Git命令有一个总体的认识

示例：从Git 版本库的初始化，通常有两种方式：

1）git clone：这是一种较为简单的初始化方式，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

上面的命令就是将'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

2）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

上面的命令就会增加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

**二、 Git 常用命令  
  
1) 远程仓库相关命令**

检出仓库：        $ git clone git://github.com/jquery/jquery.git

查看远程仓库：$ git remote -v

添加远程仓库：$ git remote add [name] [url]

删除远程仓库：$ git remote rm [name]

修改远程仓库：$ git remote set-url --push [name] [newUrl]

拉取远程仓库：$ git pull [remoteName] [localBranchName]

推送远程仓库：$ git push [remoteName] [localBranchName]

\*如果想把本地的某个分支test提交到远程仓库，并作为远程仓库的master分支，或者作为另外一个名叫test的分支，如下：

$git push origin test:master         // 提交本地test分支作为远程的master分支

$git push origin test:test              // 提交本地test分支作为远程的test分支

**2）分支(branch)操作相关命令**

查看本地分支：$ git branch

查看远程分支：$ git branch -r

创建本地分支：$ git branch [name] ----注意新分支创建后不会自动切换为当前分支

切换分支：$ git checkout [name]

创建新分支并立即切换到新分支：$ git checkout -b [name]

删除分支：$ git branch -d [name] ---- -d选项只能删除已经参与了合并的分支，对于未有合并的分支是无法删除的。如果想强制删除一个分支，可以使用-D选项

合并分支：$ git merge [name] ----将名称为[name]的分支与当前分支合并

创建远程分支(本地分支push到远程)：$ git push origin [name]

删除远程分支：$ git push origin :heads/[name] 或 $ gitpush origin :[name]

\*创建空的分支：(执行命令之前记得先提交你当前分支的修改，否则会被强制删干净没得后悔)

$git symbolic-ref HEAD refs/heads/[name]

$rm .git/index

$git clean -fdx

**3）版本(tag)操作相关命令**

查看版本：$ git tag

创建版本：$ git tag [name]

删除版本：$ git tag -d [name]

查看远程版本：$ git tag -r

创建远程版本(本地版本push到远程)：$ git push origin [name]

删除远程版本：$ git push origin :refs/tags/[name]

合并远程仓库的tag到本地：$ git pull origin --tags

上传本地tag到远程仓库：$ git push origin --tags

创建带注释的tag：$ git tag -a [name] -m 'yourMessage'

**4) 子模块(submodule)相关操作命令**

添加子模块：$ git submodule add [url] [path]

如：$git submodule add git://github.com/soberh/ui-libs.git src/main/webapp/ui-libs

初始化子模块：$ git submodule init  ----只在首次检出仓库时运行一次就行

更新子模块：$ git submodule update ----每次更新或切换分支后都需要运行一下

删除子模块：（分4步走哦）

1) $ git rm --cached [path]

2) 编辑“.gitmodules”文件，将子模块的相关配置节点删除掉

3) 编辑“ .git/config”文件，将子模块的相关配置节点删除掉

4) 手动删除子模块残留的目录

**5）忽略一些文件、文件夹不提交**

在仓库根目录下创建名称为“.gitignore”的文件，写入不需要的文件夹名或文件，每个元素占一行即可，如

target

bin

\*.db

**三、 Git 命令详解**

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：

**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，该功能类似于SVN的**add**

**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'，该功能类似于SVN的**rm、del**

**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，该功能类似于SVN的**commit**

**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，该功能类似于SVN的**log**

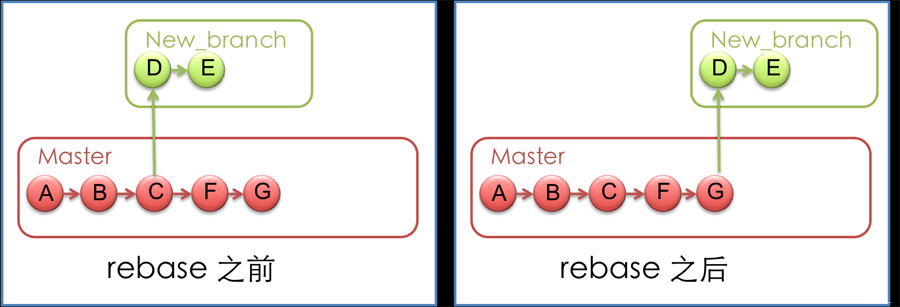
**git revert：**还原一个版本的修改，必须提供一个具体的Git版本号，例如'git revert bbaf6fb5060b4875b18ff9ff637ce118256d6f20'，Git的版本号都是生成的一个哈希值

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：

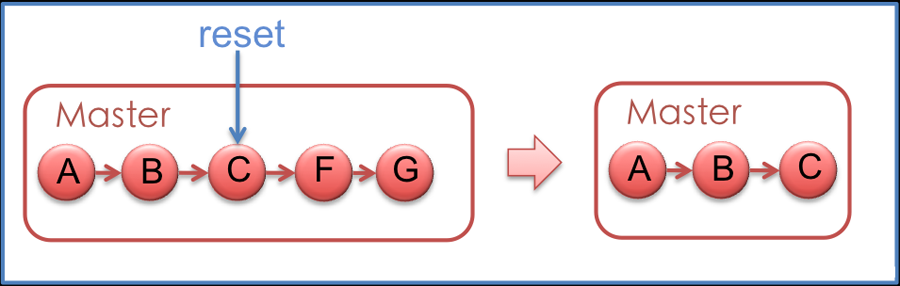
**git branch：**对分支的增、删、查等操作，例如'git branch new\_branch'会从当前的工作版本创建一个叫做new\_branch的新分支，'git branch -D new\_branch'就会强制删除叫做new\_branch的分支，'git branch'就会列出本地所有的分支

**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

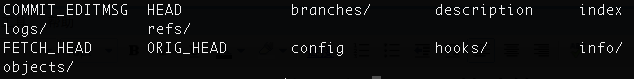


**git stash：**将当前未提交的工作存入Git工作栈中，时机成熟的时候再应用回来，这里暂时提一下这个命令的用法，后面在技巧篇会重点讲解

**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：

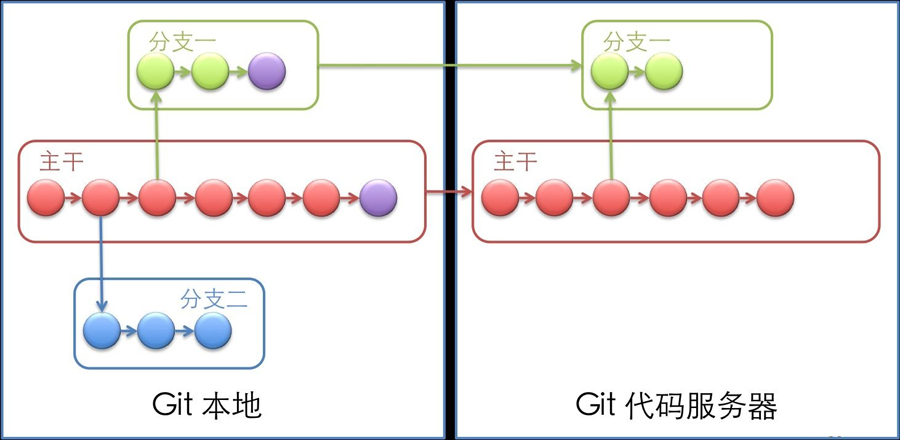


有几个比较重要的文件和目录需要解释一下：HEAD文件存放根节点的信息，其实目录结构就表示一个树型结构，Git采用这种树形结构来存储版本信息，那么HEAD就表示根；refs目录存储了你在当前版本控制目录下的各种不同引用（引用指的是你本地和远程所用到的各个树分支的信息），它有heads、remotes、stash、tags四个子目录，分别存储对不同的根、远程版本库、Git栈和标签的四种引用，你可以通过命令'git show-ref'更清晰地查看引用信息；logs目录根据不同的引用存储了日志信息。因此，Git只需要代码根目录下的这一个.git目录就可以记录完整的版本控制信息，而不是像SVN那样根目录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

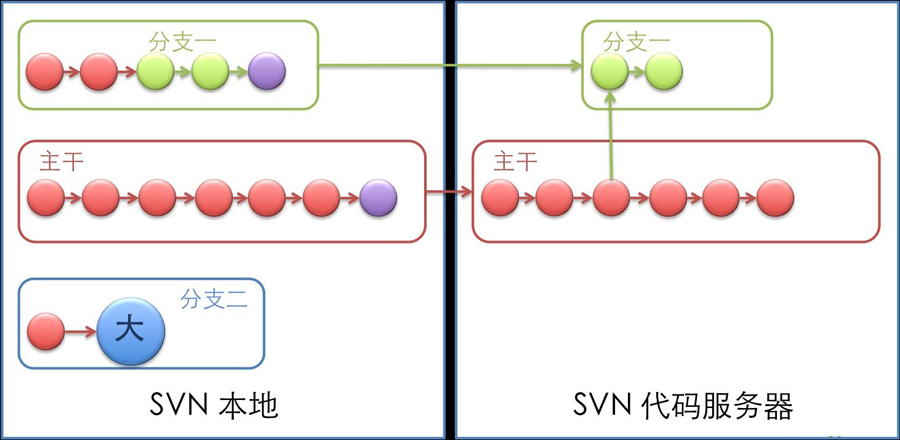
**四、 Git 与SVN 比较**

**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：



------------



1）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：

使用SVN作为版本控制工具，当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。

这时候假如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。

2）分布式提交

Git 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

3）日志查看

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

例如：代码服务器在美国，当每次查看几年前所做的工作时，日志下载可能需要十分钟，这不能不说是一个痛苦。但是如果迁移到Git上，利用Git日志在本地的特性，查看某个具体任务的所有代码历史，每次只需要几秒钟，大大方便了工作，提高了效率。

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。

**五、 总结**

上面简单介绍了Git 的基本概念、一些常用命令和原理，大家也可以尝试动手，在Google Code 或 GitHub 上创建一个自己的开源项目

Git 创建与使用示例：

在Google Code 上，我创建了Git 项目：[linux-kernel-source](http://code.google.com/p/linux-kernel-source/source/checkout)

在GitHub上，我引用的Git 项目：[hiphop-php](https://github.com/facebook/hiphop-php)

在osChina上，我分享的项目：[gcc-4.5.2](http://www.oschina.net/code/explore/gcc-4.5.2)

# Git 常用命令速查表（三）

本文将对Git 命令，做一下全面而系统的简短总结，整理成简洁、明了的图表结构，方便查询

**一、 Git 常用命令速查**

git branch 查看本地所有分支  
git status 查看当前状态   
git commit 提交   
git branch -a 查看所有的分支  
git branch -r 查看远程所有分支  
git commit -am "init" 提交并且加注释   
git remote add origin git@192.168.1.119:ndshow  
git push origin master 将文件给推到服务器上   
git remote show origin 显示远程库origin里的资源   
git push origin master:develop  
git push origin master:hb-dev 将本地库与服务器上的库进行关联   
git checkout --track origin/dev 切换到远程dev分支  
git branch -D master develop 删除本地库develop  
git checkout -b dev 建立一个新的本地分支dev  
git merge origin/dev 将分支dev与当前分支进行合并  
git checkout dev 切换到本地dev分支  
git remote show 查看远程库  
git add .  
git rm 文件名(包括路径) 从git中删除指定文件  
git clone git://github.com/schacon/grit.git 从服务器上将代码给拉下来  
git config --list 看所有用户  
git ls-files 看已经被提交的  
git rm [file name] 删除一个文件  
git commit -a 提交当前repos的所有的改变  
git add [file name] 添加一个文件到git index  
git commit -v 当你用－v参数的时候可以看commit的差异  
git commit -m "This is the message describing the commit" 添加commit信息  
git commit -a -a是代表add，把所有的change加到git index里然后再commit  
git commit -a -v 一般提交命令  
git log 看你commit的日志  
git diff 查看尚未暂存的更新  
git rm a.a 移除文件(从暂存区和工作区中删除)  
git rm --cached a.a 移除文件(只从暂存区中删除)  
git commit -m "remove" 移除文件(从Git中删除)  
git rm -f a.a 强行移除修改后文件(从暂存区和工作区中删除)  
git diff --cached 或 $ git diff --staged 查看尚未提交的更新  
git stash push 将文件给push到一个临时空间中  
git stash pop 将文件从临时空间pop下来  
---------------------------------------------------------  
git remote add origin git@github.com:username/Hello-World.git  
git push origin master 将本地项目给提交到服务器中  
-----------------------------------------------------------  
git pull 本地与服务器端同步  
-----------------------------------------------------------------  
git push (远程仓库名) (分支名) 将本地分支推送到服务器上去。  
git push origin serverfix:awesomebranch  
------------------------------------------------------------------  
git fetch 相当于是从远程获取最新版本到本地，不会自动merge  
git commit -a -m "log\_message" (-a是提交所有改动，-m是加入log信息) 本地修改同步至服务器端 ：  
git branch branch\_0.1 master 从主分支master创建branch\_0.1分支  
git branch -m branch\_0.1 branch\_1.0 将branch\_0.1重命名为branch\_1.0  
git checkout branch\_1.0/master 切换到branch\_1.0/master分支  
du -hs

git branch 删除远程branch

git push origin **:**branch\_remote\_name

git branch -r -d branch\_remote\_name  
  
-----------------------------------------------------------

初始化版本库，并提交到远程服务器端  
mkdir WebApp  
cd WebApp  
git init 本地初始化  
touch README  
git add README 添加文件  
git commit -m 'first commit'  
git remote add origin git@github.com:daixu/WebApp.git 增加一个远程服务器端

上面的命令会增加URL地址为'git@github.com:daixu/WebApp.git'，名称为origin的远程服务器库，以后提交代码的时候只需要使用 origin别名即可

**二、 Git 命令速查表**

1、常用的Git命令

| **命令** | **简要说明** |
| --- | --- |
| git add | 添加至暂存区 |
| git add–interactive | 交互式添加 |
| git apply | 应用补丁 |
| git am | 应用邮件格式补丁 |
| git annotate | 同义词，等同于 git blame |
| git archive | 文件归档打包 |
| git bisect | 二分查找 |
| git blame | 文件逐行追溯 |
| git branch | 分支管理 |
| git cat-file | 版本库对象研究工具 |
| git checkout | 检出到工作区、切换或创建分支 |
| git cherry-pick | 提交拣选 |
| git citool | 图形化提交，相当于 git gui 命令 |
| git clean | 清除工作区未跟踪文件 |
| git clone | 克隆版本库 |
| git commit | 提交 |
| git config | 查询和修改配置 |
| git describe | 通过里程碑直观地显示提交ID |
| git diff | 差异比较 |
| git difftool | 调用图形化差异比较工具 |
| git fetch | 获取远程版本库的提交 |
| git format-patch | 创建邮件格式的补丁文件。参见 git am 命令 |
| git grep | 文件内容搜索定位工具 |
| git gui | 基于Tcl/Tk的图形化工具，侧重提交等操作 |
| git help | 帮助 |
| git init | 版本库初始化 |
| git init-db\* | 同义词，等同于 git init |
| git log | 显示提交日志 |
| git merge | 分支合并 |
| git mergetool | 图形化冲突解决 |
| git mv | 重命名 |
| git pull | 拉回远程版本库的提交 |
| git push | 推送至远程版本库 |
| git rebase | 分支变基 |
| git rebase–interactive | 交互式分支变基 |
| git reflog | 分支等引用变更记录管理 |
| git remote | 远程版本库管理 |
| git repo-config\* | 同义词，等同于 git config |
| git reset | 重置改变分支“游标”指向 |
| git rev-parse | 将各种引用表示法转换为哈希值等 |
| git revert | 反转提交 |
| git rm | 删除文件 |
| git show | 显示各种类型的对象 |
| git stage\* | 同义词，等同于 git add |
| git stash | 保存和恢复进度 |
| git status | 显示工作区文件状态 |
| git tag | 里程碑管理 |

2、对象库操作相关命令

| **命令** | **简要说明** |
| --- | --- |
| git commit-tree | 从树对象创建提交 |
| git hash-object | 从标准输入或文件计算哈希值或创建对象 |
| git ls-files | 显示工作区和暂存区文件 |
| git ls-tree | 显示树对象包含的文件 |
| git mktag | 读取标准输入创建一个里程碑对象 |
| git mktree | 读取标准输入创建一个树对象 |
| git read-tree | 读取树对象到暂存区 |
| git update-index | 工作区内容注册到暂存区及暂存区管理 |
| git unpack-file | 创建临时文件包含指定 blob 的内容 |
| git write-tree | 从暂存区创建一个树对象 |

3、引用操作相关命令

| **命令** | **简要说明** |
| --- | --- |
| git check-ref-format | 检查引用名称是否符合规范 |
| git for-each-ref | 引用迭代器，用于shell编程 |
| git ls-remote | 显示远程版本库的引用 |
| git name-rev | 将提交ID显示为友好名称 |
| git peek-remote\* | 过时命令，请使用 git ls-remote |
| git rev-list | 显示版本范围 |
| git show-branch | 显示分支列表及拓扑关系 |
| git show-ref | 显示本地引用 |
| git symbolic-ref | 显示或者设置符号引用 |
| git update-ref | 更新引用的指向 |
| git verify-tag | 校验 GPG 签名的Tag |

4、版本库管理相关命令

| **命令** | **简要说明** |
| --- | --- |
| git count-objects | 显示松散对象的数量和磁盘占用 |
| git filter-branch | 版本库重构 |
| git fsck | 对象库完整性检查 |
| git fsck-objects\* | 同义词，等同于 git fsck |
| git gc | 版本库存储优化 |
| git index-pack | 从打包文件创建对应的索引文件 |
| git lost-found\* | 过时，请使用 git fsck –lost-found 命令 |
| git pack-objects | 从标准输入读入对象ID，打包到文件 |
| git pack-redundant | 查找多余的 pack 文件 |
| git pack-refs | 将引用打包到 .git/packed-refs 文件中 |
| git prune | 从对象库删除过期对象 |
| git prune-packed | 将已经打包的松散对象删除 |
| git relink | 为本地版本库中相同的对象建立硬连接 |
| git repack | 将版本库未打包的松散对象打包 |
| git show-index | 读取包的索引文件，显示打包文件中的内容 |
| git unpack-objects | 从打包文件释放文件 |
| git verify-pack | 校验对象库打包文件 |

5、数据传输相关命令

| **命令** | **简要说明** |
| --- | --- |
| git fetch-pack | 执行 git fetch 或 git pull 命令时在本地执行此命令，用于从其他版本库获取缺失的对象 |
| git receive-pack | 执行 git push 命令时在远程执行的命令，用于接受推送的数据 |
| git send-pack | 执行 git push 命令时在本地执行的命令，用于向其他版本库推送数据 |
| git upload-archive | 执行 git archive –remote 命令基于远程版本库创建归档时，远程版本库执行此命令传送归档 |
| git upload-pack | 执行 git fetch 或 git pull 命令时在远程执行此命令，将对象打包、上传 |

6、邮件相关命令

| **命令** | **简要说明** |
| --- | --- |
| git imap-send | 将补丁通过 IMAP 发送 |
| git mailinfo | 从邮件导出提交说明和补丁 |
| git mailsplit | 将 mbox 或 Maildir 格式邮箱中邮件逐一提取为文件 |
| git request-pull | 创建包含提交间差异和执行PULL操作地址的信息 |
| git send-email | 发送邮件 |

7、协议相关命令

| **命令** | **简要说明** |
| --- | --- |
| git daemon | 实现Git协议 |
| git http-backend | 实现HTTP协议的CGI程序，支持智能HTTP协议 |
| git instaweb | 即时启动浏览器通过 gitweb 浏览当前版本库 |
| git shell | 受限制的shell，提供仅执行Git命令的SSH访问 |
| git update-server-info | 更新哑协议需要的辅助文件 |
| git http-fetch | 通过HTTP协议获取版本库 |
| git http-push | 通过HTTP/DAV协议推送 |
| git remote-ext | 由Git命令调用，通过外部命令提供扩展协议支持 |
| git remote-fd | 由Git命令调用，使用文件描述符作为协议接口 |
| git remote-ftp | 由Git命令调用，提供对FTP协议的支持 |
| git remote-ftps | 由Git命令调用，提供对FTPS协议的支持 |
| git remote-http | 由Git命令调用，提供对HTTP协议的支持 |
| git remote-https | 由Git命令调用，提供对HTTPS协议的支持 |
| git remote-testgit | 协议扩展示例脚本 |

8、版本库转换和交互相关命令

| **命令** | **简要说明** |
| --- | --- |
| git archimport | 导入Arch版本库到Git |
| git bundle | 提交打包和解包，以便在不同版本库间传递 |
| git cvsexportcommit | 将Git的一个提交作为一个CVS检出 |
| git cvsimport | 导入CVS版本库到Git。或者使用 cvs2git |
| git cvsserver | Git的CVS协议模拟器，可供CVS命令访问Git版本库 |
| git fast-export | 将提交导出为 git-fast-import 格式 |
| git fast-import | 其他版本库迁移至Git的通用工具 |
| git svn | Git 作为前端操作 Subversion |

9、合并相关的辅助命令

| **命令** | **简要说明** |
| --- | --- |
| git merge-base | 供其他脚本调用，找到两个或多个提交最近的共同祖先 |
| git merge-file | 针对文件的两个不同版本执行三向文件合并 |
| git merge-index | 对index中的冲突文件调用指定的冲突解决工具 |
| git merge-octopus | 合并两个以上分支。参见 git merge 的octopus合并策略 |
| git merge-one-file | 由 git merge-index 调用的标准辅助程序 |
| git merge-ours | 合并使用本地版本，抛弃他人版本。参见 git merge 的ours合并策略 |
| git merge-recursive | 针对两个分支的三向合并。参见 git merge 的recursive合并策略 |
| git merge-resolve | 针对两个分支的三向合并。参见 git merge 的resolve合并策略 |
| git merge-subtree | 子树合并。参见 git merge 的 subtree 合并策略 |
| git merge-tree | 显式三向合并结果，不改变暂存区 |
| git fmt-merge-msg | 供执行合并操作的脚本调用，用于创建一个合并提交说明 |
| git rerere | 重用所记录的冲突解决方案 |

10、 杂项

| **命令** | **简要说明** |
| --- | --- |
| git bisect–helper | 由 git bisect 命令调用，确认二分查找进度 |
| git check-attr | 显示某个文件是否设置了某个属性 |
| git checkout-index | 从暂存区拷贝文件至工作区 |
| git cherry | 查找没有合并到上游的提交 |
| git diff-files | 比较暂存区和工作区，相当于 git diff –raw |
| git diff-index | 比较暂存区和版本库，相当于 git diff –cached –raw |
| git diff-tree | 比较两个树对象，相当于 git diff –raw A B |
| git difftool–helper | 由 git difftool 命令调用，默认要使用的差异比较工具 |
| git get-tar-commit-id | 从 git archive 创建的 tar 包中提取提交ID |
| git gui–askpass | 命令 git gui 的获取用户口令输入界面 |
| git notes | 提交评论管理 |
| git patch-id | 补丁过滤行号和空白字符后生成补丁唯一ID |
| git quiltimport | 将Quilt补丁列表应用到当前分支 |
| git replace | 提交替换 |
| git shortlog | 对 git log 的汇总输出，适合于产品发布说明 |
| git stripspace | 删除空行，供其他脚本调用 |
| git submodule | 子模组管理 |
| git tar-tree | 过时命令，请使用 git archive |
| git var | 显示 Git 环境变量 |
| git web–browse | 启动浏览器以查看目录或文件 |
| git whatchanged | 显示提交历史及每次提交的改动 |
| git-mergetool–lib | 包含于其他脚本中，提供合并/差异比较工具的选择和执行 |
| git-parse-remote | 包含于其他脚本中，提供操作远程版本库的函数 |
| git-sh-setup | 包含于其他脚本中，提供 shell 编程的函数库 |

附：[Git 命令速查表](http://www.cnblogs.com/1-2-3/archive/2010/07/18/git-commands.html)

整理集合：[Git 命令学习文档](http://download.csdn.net/detail/sunboy_2050/4272038)

**参考推荐：**

[Git 版本管理工具](http://blog.csdn.net/sunboy_2050/article/details/7527877)

[Git 常用命令详解](http://blog.csdn.net/sunboy_2050/article/details/7529022)

[Git 常用命令](http://hi.baidu.com/sunboy_2050/blog/item/34cc9c29fe1a22e1e7cd406a.html)（图表）

[SVN常用命令](http://blog.csdn.net/sunboy_2050/article/details/6187464)

[14款Web开发速查表](http://blog.csdn.net/sunboy_2050/article/details/7385723)

# Git 操作实战示例（四）

前面三篇介绍了关于 Git 的概念、命令使用、命令查询等理论知识

1）[Git 版本管理工具](http://blog.csdn.net/sunboy_2050/article/details/7527877)

2）[Git 常用命令详解](http://blog.csdn.net/sunboy_2050/article/details/7529022)

3）[Git 常用命令速查表](http://blog.csdn.net/sunboy_2050/article/details/7529841)

本文主要介绍在使用Git 过程中遇到的几个问题及其解决方案

**一、Git 界面GUI和命令行Command两种操作方式**

Git 进行代码版本管理时，有两种操作方式：图形界面GUI 和 命令行Command

图形界面GUI：有平台限制，目前仅支持Windows系统，Linux（Unix）都是命令行模式，并且图形界面每次提交好像Display显示文件数有限制（如：最大显示20000个文件）

命令行Command：没有平台限制，Windows、Linux（Unix、Mac OS X）下都支持，通过 gid add dir/files 添加文件，没有文件数限制

下面详细介绍 Git 管理和操作的两个方式：

**1、图形界面GUI**

首先，下载Windows下的Git图形界面工具：[msysgit](http://code.google.com/p/linux-kernel-source/downloads/list)  或 [官方下载](http://code.google.com/p/msysgit/downloads/list)

接着，安装 msysgit ，如下载最新的 [Git-1.7.10-preview20120409.exe](http://code.google.com/p/linux-kernel-source/downloads/detail?name=Git-1.7.10-preview20120409.exe&can=2&q=) 后安装，操作方式有Git GUI 和 Git Bash两种，默认为 Gui Bash Only

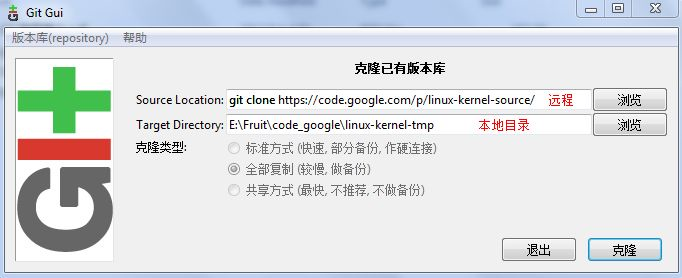
然后，安装完后，在开始 —> 所有程序 —> Git 中，有 Git Bash 和 Git Gui 两个图标，点击启动 Git Gui 界面如下：



msysgit 图形工具，可以创建新的版本库（本地）、克隆已有版本库（远程）、打开已有版本库（本地或远程）

例如：克隆已有的版本库（远程），如 克隆下载 [linux-kernel-source](http://code.google.com/p/linux-kernel-source/source/checkout) 步骤：

a、 点击“克隆已有版本库”，依次输入远程版本库（xxx）和本地目录（xxx），如下图



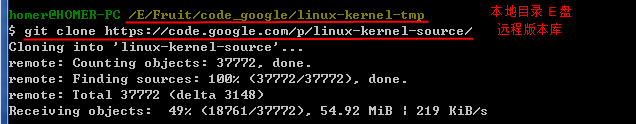
**2、命令行Command**

a、Git 远程下载

git  clone  git://git.kernel.org/pub/scm/  project\_path

1）例如：下载 [linux-kernel-source](http://code.google.com/p/linux-kernel-source/source/checkout)，Git 命令如下

git  clone  https://code.google.com/p/linux-kernel-source/ /E/Fruit/code\_google/linux-kernel-tmp



2）例如：下载 Linux 内核 [git.kernel.org](http://git.kernel.org/)

git  clone  git://git.kernel.org/pub/scm/   /home/homer/Linux-kernel-3.3.4

b、 Git 本地上传

添加新文件：git add dir/files

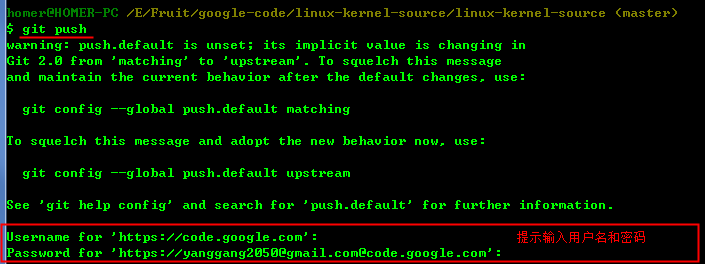
提交文件：git commit -m 'add xxx.txt'

上传文件：git push

Git命令更多实例，请见我在google code上传的两个开源项目：[linux-kernel-source](http://code.google.com/p/linux-kernel-source/source/browse/) 和 [android-source-homer](http://code.google.com/p/android-source-homer/source/browse/)

**二、Windows下google code git的无密码登录**

每次使用git push代码到google code的时候总要输入用户名及密码，很是麻烦：



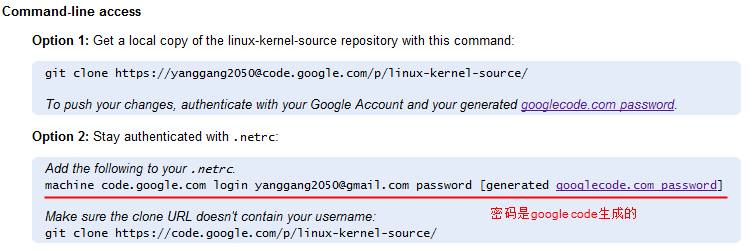
**说明：**上文提示可以看到，由于Git版本升级，push.default默认参数值从“matching”改成了“upstream”，并且给出了两种修改方案：

1） git config --global push.default matching —— 维护先前的行为（matching）

2） git config --global push.default upstream —— 引进新的行为（upstream）

以上两条命令，根据自己需求设置一条即可，以后此提示信息就不会再弹出。

幸运的是，google code在项目管理页面提供了保存认证信息的方法：



**Option 2** 即为无密码登录的配置。不过这个配置是针对Linux系统的。那么windows下有没有类似于.netrc的配置呢？

答案是肯定的。完成windows下.netrc的配置需要两个步骤：1.新建HOME环境变量；2.新建\_netrc文件，并输入配置信息。

**1. 环境变量**

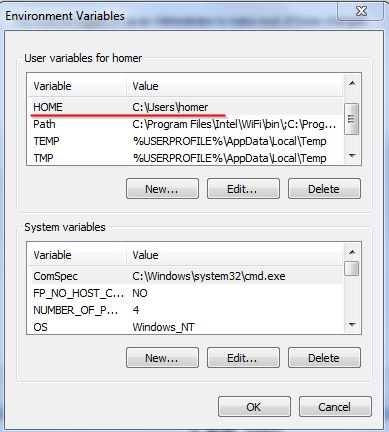
 If you are using Windows 7

run the cmd type this:

setx HOME %USERPROFILE%（**注意：%**USERPROFILE% 才是当前默认的用户目录）

and the %HOME% will be set to 'C:\Users\"username"'

例如：用户名为 homer，则 %HOME% will be set to 'C:\Users\**homer**'



**2. 配置\_netrc**

windows下的 **.netrc**文件要命名为 **\_netrc**  具体步骤：

首先，在【1、环境配置】指定的%HOME%目录（C:\Users\homer）中新建一个名为 **\_netrc** 的文件

然后，在新建的  **\_netrc** 的文件中，输入以下内容：

machine  code.google.com  login  xxx@gmail.com  password  xxx [generated googlecode.com password]

其中，password 是code.google.com自动生成的，在 Profile —> Settings —> GoogleCode.com Password（<https://code.google.com/hosting/settings>）

**Git Push 不需要输入密码上传效果图：**



推荐参考：[StackOverflow](http://stackoverflow.com/questions/6031214/git-how-to-use-netrc-file-on-windows-to-save-user-and-password/6031266#6031266)

**三、 google code托管开源项目https://code.google.com/hosting/settings被墙的解决方法**

**方法一：**

很多人在使用 code.google.com时， 都需要一个code.google.com的密码， 这个密码是从<https://code.google.com/hosting/settings> 网址获得， 但大多时候您会发现这个网址不能打开，通过实践分析的原因可能是以下原因：

1. 国家屏蔽了[www.google.com](http://www.google.com/) 的网址IP地址

2. 由于国家屏蔽， google.com被迫更改了自己的IP地址。

解决办法：

1.首先在命令行 ping [www.google.com](http://www.google.com/) ，查出当前google.com的IP地址。

2.修改c:\windows\system32\drivers\etc\hosts 文件

74.125.153.104 为你当前Ping时看到的IP。

追加以下内容：

74.125.153.104 code.google.com   
74.125.153.104 encrypted.google.com   
74.125.153.104 suggestqueries.google.com   
74.125.153.104 mail.google.com   
74.125.153.104 groups.google.com   
74.125.153.104 groups.google.com.hk   
74.125.153.104 docs.google.com   
74.125.153.104 docs0.google.com   
74.125.153.104 docs1.google.com   
74.125.153.104 spreadsheets.google.com   
74.125.153.104 spreadsheets0.google.com   
74.125.153.104 webcache.googleusercontent.com   
74.125.153.104 sites.google.com   
74.125.153.104 talkgadget.google.com   
74.125.153.104 clients1.google.com   
74.125.153.104 clients2.google.com   
74.125.153.104 clients3.google.com   
74.125.153.104 clients4.google.com

**方法二：**

刚开始的code.google.com都可以访问到，几乎所有页面都可以访问，唯独查看svn账号密码的页面(<https://code.google.com/hosting/settings> )就总是“连接被重置”

因为众所周知的原因，我朝大约是不想让子民向美帝提交代码，所以封杀了这关键性的一页。通过百度查询，发现他们的解决方法虽然都是改hosts（方法一），但具体怎么改，说的不太对，我自己验证了一下，发现下面的方法比较靠谱

如果你已经有了翻方法，那翻访问就行了，我的方法是针对不翻的情况下的。

网上有人说让我们先ping一下www.google.com和code.google.com然后去拿这个结果中的IP去设置hosts，可能会不行！！！

为什么呢？因为我们的ping也被墙给阉割过了，比如说ping www.google.com，显示的结果可能是Ping code.l.google.com的，被篡改了。

我们要得到和美帝一样的ping结果怎么办呢？

很简单，先在google搜索online ping，找到第一个或者你看着顺眼的一个网站打开，用里面的在线ping工具来ping code.google.com，相当于用了国外的服务器来ping，所以结果是靠谱的。

**步骤总结：**

1、去google搜online ping，在英文google搜；或直接打开 [ping.eu/ping](http://ping.eu/ping/)

2、在提供在线ping的国外网站上，输入：ping code.google.com ，取得ip

3、改hosts文件，将刚才ping到的ip设置和code.google.com配对。（最好清空一下dns缓存，命令行输入ipconfig /flushdns)

