# wzyxidian

Git命令详解

一个中文git手册：http://progit.org/book/zh/

原文：http://blog.csdn.net/sunboy_2050/article/details/7529841

前面两篇博客 Git 版本管理工具 和 Git 常用命令详解，分别介绍了Git 基础知识和命令用法

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


< 2017年9月

| 日 | 一 | 二 | 三 | 四 |
|---|---|---|---|---|
| 27 | 28 | 29 | 30 | 31 |
| 3 | 4 | 5 | 6 | 7 |
| 10 | 11 | 12 | 13 | 14 |
| 17 | 18 | 19 | 20 | 21 |
| 24 | 25 | 26 | 27 | 28 |
| 1 | 2 | 3 | 4 | 5 |

导航
博客园
首页
新随笔
联系
订阅XML
管理

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

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 |

更多

随笔分类(323)
Git(16)
Hibernate(3)
IntelliJIDEA(1)
java集合系列(23)
JAVA小记(78)
JVM(5)
Kafka(7)
leetCode(60)
linux(4)
Markdown语法(2)
Maven(8)
MYSQL小记(32)
Restful
Spark(6)
Spring
Spring boot
Spring Cloud(3)
操作系统
多线程(38)
脚本(2)
链表(1)
面试题
设计模式(29)
树图(1)
数据库
网络(3)
微服务
栈、队列(1)

随笔档案(337)
2016年10月 (11)
2016年9月 (11)
2016年8月 (1)
2016年7月 (5)
2016年6月 (6)
2016年5月 (36)
2016年4月 (25)
2016年3月 (61)
2016年2月 (47)
2016年1月 (32)
2015年12月 (29)
2015年11月 (14)
2015年10月 (25)
2015年8月 (4)
2015年4月 (25)
2014年6月 (2)
2014年5月 (3)

文章分类
树图

最新评论
1. Re:Spring顶级项目以及Spring件
不错，希望能持续更新。

2. Re:Java Integer类分析
不错

3. Re:TCP 协议如何保证可靠传输
大神。TCP，GBN，SR可以结合一

4. Re:Hibernate的配置文件解析
涨姿势了

5. Re:Hibernate的配置文件解析
非常有用!

阅读排行榜
1. 获取当前工程路径(6608)
2. Spring顶级项目以及Spring clo(4409)
3. Git 、CVS、SVN比较(3505)

| | |
|---|---|
| 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 |

4. Git命令详解(2394)
5. Git创建一个自己的本地仓库(23

评论排行榜
1. Hibernate的配置文件解析(2)
2. Java Integer类分析(1)
3. Spring顶级项目以及Spring clo
4. TCP 协议如何保证可靠传输(1)

推荐排行榜
1. Hibernate的配置文件解析(1)
2. Java Integer类分析(1)
3. 深入剖析Java中的装箱和拆箱(1

| | |
|---|---|
| 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 |

---

原文：http://hi.baidu.com/wade_hit/item/848869db05e53af4cb0c391b

总结一下ubuntu下github常用的命令，设置部分跳过，假设repository的名字叫hello-world：

**1.创建一个新的repository**：

先在github上创建并写好相关名字，描述。

$cd ~/hello-world　　//到hello-world目录

$git init //初始化

$git add . //把所有文件加入到索引（不想把所有文件加入，可以用gitignore或add 具体文件)

$git commit //提交到本地仓库，然后会填写更新日志( -m "更新日志"也可)

$git remote add origin git@github.com:WadeLeng/hello-world.git //增加到remote

$git push origin master //push到github上

**2.更新项目（新加了文件）：**

$cd ~/hello-world

$git add . //这样可以自动判断新加了哪些文件，或者手动加入文件名字

$git commit //提交到本地仓库

$git push origin master //不是新创建的，不用再add 到remote上了

**3.更新项目（没新加文件，只有删除或者修改文件）：**

$cd ~/hello-world

$git commit -a //记录删除或修改了哪些文件

$git push origin master //提交到github

**4.忽略一些文件，比如*.o等:**

$cd ~/hello-world

$vim .gitignore //把文件类型加入到.gitignore中，保存

然后就可以git add . 能自动过滤这种文件

**5.clone代码到本地：**

$git clone git@github.com:WadeLeng/hello-world.git

假如本地已经存在了代码，而仓库里有更新，把更改的合并到本地的项目：

$git fetch origin //获取远程更新

$git merge origin/master //把更新的内容合并到本地分支

**6.撤销**

$git reset

**7.删除**

$git rm * // 不是用rm

//------------------------------常见错误----------------------------------

1.$ git remote add origin git@github.com:WadeLeng/hello-world.git

**错误提示：fatal: remote origin already exists.**

解决办法：$ git remote rm origin

然后在执行：$ git remote add origin git@github.com:WadeLeng/hello-world.git 就不会报错误了

2. $ git push origin master

**错误提示：error:failed to push som refs to**

解决办法：$ git pull origin master //先把远程服务器github上面的文件拉先来，再push 上去。

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），管理私人的文档和源代码也有很多优势（如：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

**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

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），管理私人的文档和源代码也有很多优势（如：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

**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：*[Math Processing Error]* 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**

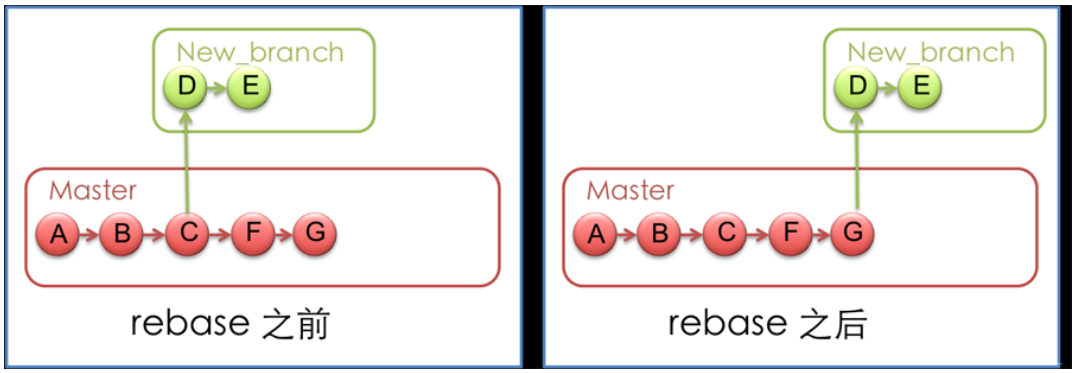
**git revert**：还原一个版本的修改，必须提供一个具体的Git版本号，例如'git revert bbaf6fb5060b4875b18ff9ff637ce118256d6f20'，Git的版本号都是生成的一个哈希值

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：

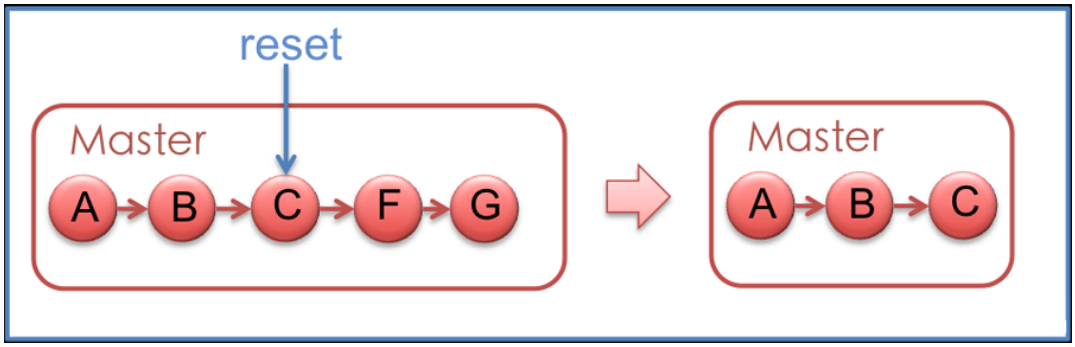
**git branch**：对分支的增、删、查等操作，例如'git branch new_branch'会从当前的工作版本创建一个叫做new_branch的新分支，'git branch -D new_branch'就会强制删除叫做new_branch的分支，'git branch'就会列出本地所有的分支

**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：

```
COMMIT_EDITMSG  HEAD            branches/       description     index
logs/           refs/
FETCH_HEAD      ORIG_HEAD       config          hooks/          info/
objects/
```

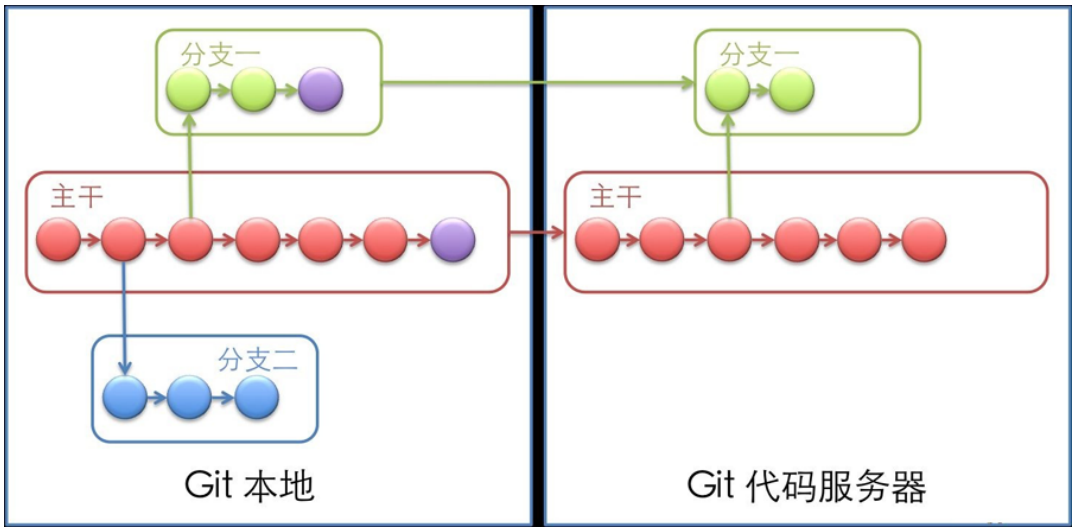
有 几个比较重要的文件和目录需要解释一下：HEAD文件存放根节点的信息，其实目录结构就表示一个树型结构，Git采用这种树形结构来存储版本信息，那么 HEAD就表示根；refs目录存储了你在当前版本控制目录下的各种不同引用（引用指的是你本地和远程所用到的各个树分支的信息），它有heads、 remotes、stash、tags四个子目录，分别存储对不同的根、远程版本库、Git栈和标签的四种引用，你可以通过命令'git show-ref'更清晰地查看引用信息；logs目录根据不同的引用存储了日志信息。因此，Git只需要代码根目录下的这一个.git目录就可以

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

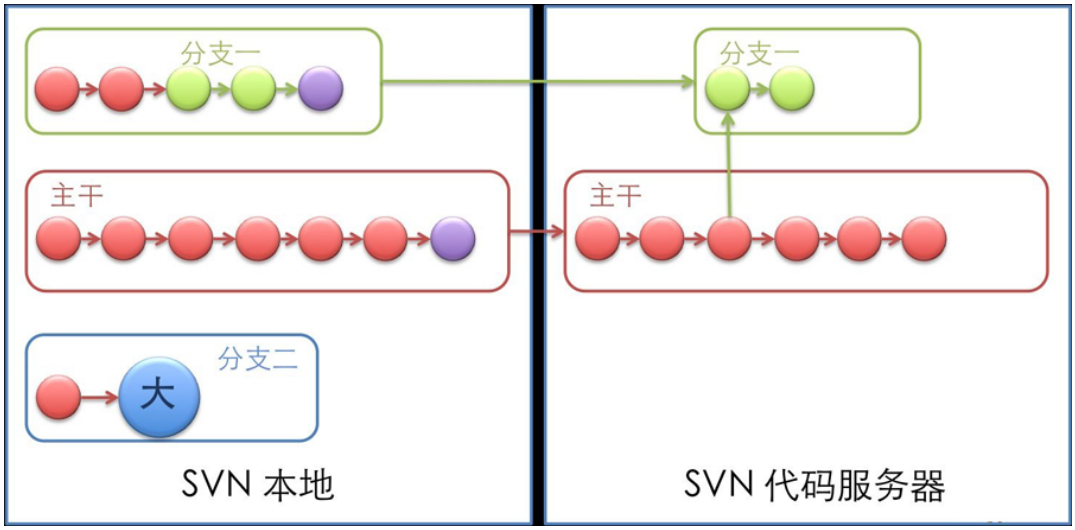
**四、 Git 与SVN 比较**

**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。

分类：Git

标签：git



wzyxidian
关注 - 17
粉丝 - 11
+加关注

0 0

« 上一篇：Git的分支与合并
» 下一篇：Git 、CVS、SVN比较

posted on 2016-05-23 11:53 wzyxidian 阅读(2394) 评论(0) 编辑 收藏

» 更多新闻...



**最新知识库文章：**

· Google 及其云智慧
· 做到这一点，你也可以成为优秀的程序员
· 写给立志做码农的大学生
· 架构腐化之谜
· 学会思考，而不只是编程
» 更多知识库文章...