

# 自建私有云 Git 集群仓库解决方案

作者： 熊佳乐

2020 年 12 月 20 日星期日

# 说明

该解决方案是用于解决小公司小团队代码的管理的问题，通过使用的先进的 git 的工具来实现代码的管理和控制，帮助公司和团队更好时候管理和运行。同时解决资金避免使用的重量的管理工具和官方的管理服务。

由于 git 是开源的工具，其中不涉及到任务的侵权问题。使用的系统也是 linux 或者 Centos 等开源的系统。

该解决方案包括 8 个部分：

- git 的简介
- git 的安装配置
- git 的工作原理
- 私有云本地 git 集群构建
- git 的容灾备份
- git 的具体的操作
- git 的可视化管理工具
- 其他问题以及解决方案。

欢迎大家有任何问题在博客留言或者私信问我。

CSDN 博客留言：[https://blog.csdn.net/weixin\\_41605937](https://blog.csdn.net/weixin_41605937)

个人邮箱：[18279148786@163.com](mailto:18279148786@163.com)

---

## 目 录

1git 的简介.....	1
2git 的安装配置 .....	2
2.1Linux 平台上安装.....	2
2.2Windows 平台上安装.....	2
3git 的工作原理 .....	3
3.1git 工作流程 .....	3
3.2git 的工作区 .....	3
4git 操作命令说明 .....	5
4.1 常用的命令: .....	5
5 私有云本地 git 怎么构建 .....	7
5.1 本地服务器 git 的初始化.....	7
5.2 用户的连接和使用 .....	7
5.3 用户个人的操作 .....	8
6git 的容灾备份 .....	10
7git 的可视化管理工具 .....	11
7.1GitHub for Desktop.....	11
7.2Source Tree.....	11
7.3TortoiseGit.....	13
8 其他问题以及解决方案 .....	14

## 1git 的简介

Git 是一个开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目。Git 是 Linus 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。Git 与常用的版本控制工具 CVS, Subversion 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。

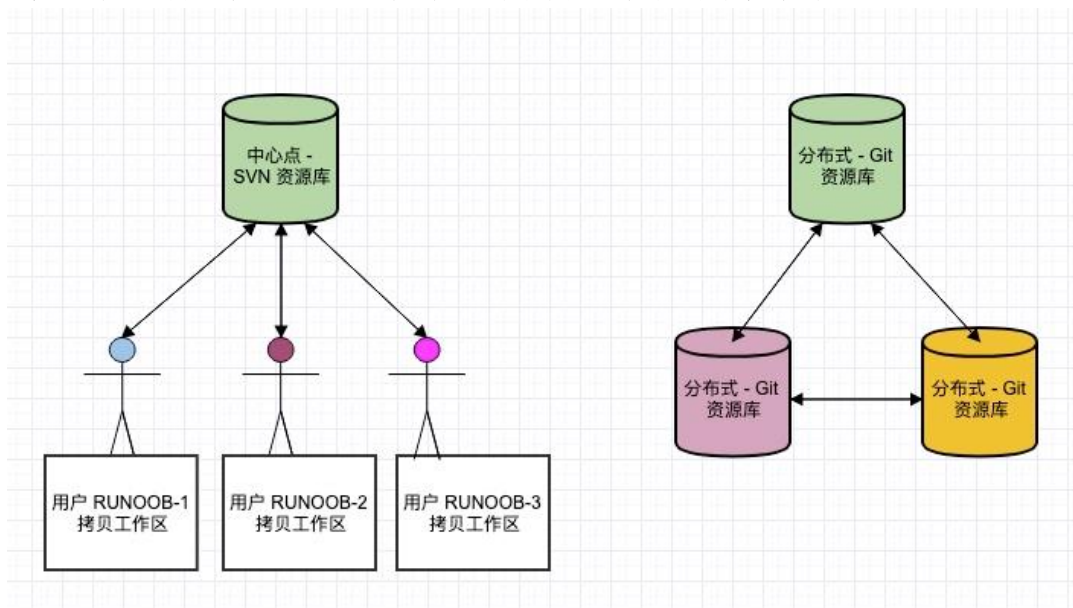
### Git 与 SVN 区别

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。

如果你是一个具有使用 SVN 背景的人，你需要做一定的思想转换，来适应 Git 提供的一些概念和特征。

### Git 与 SVN 区别点：

- 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，例如 SVN, CVS 等，最核心的区别。
- 2、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。
- 3、Git 分支和 SVN 的分支不同：分支在 SVN 中一点都不特别，其实它就是版本库中的另外一个目录。
- 4、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。
- 5、Git 的内容完整性要优于 SVN：Git 的内容存储使用的是 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。



---

## 2git 的安装配置

在使用 Git 前我们需要先安装 Git。Git 目前支持 Linux/Unix、Solaris、Mac 和 Windows 平台上运行。Git 各平台安装包下载地址为：<http://git-scm.com/downloads>

### 2.1Linux 平台上安装

Git 的工作需要调用 curl, zlib, openssl, expat, libiconv 等库的代码，所以需要先安装这些依赖工具。

在有 yum 的系统上（比如 Fedora）或者有 apt-get 的系统上（比如 Debian 体系），可以用下面的命令安装：

各 Linux 系统可以使用其安装包管理工具（apt-get、yum 等）进行安装：

Ubuntu Git 安装命令为：

```
$ apt-get install libcurl4-gnutls-dev libexpat1-dev gettext \
    libz-dev libssl-dev
```

```
$ apt-get install git
```

```
$ git --version
git version 1.8.1.2
```

### Centos/RedHat

如果你使用的系统是 Centos/RedHat 安装命令为：

```
$ yum install curl-devel expat-devel gettext-devel \
    openssl-devel zlib-devel
```

```
$ yum -y install git-core
```

```
$ git --version
git version 1.7.1
```

### 2.2Windows 平台上安装

在 Windows 平台上安装 Git 同样轻松，有个叫做 msysGit 的项目提供了安装包，可以到 GitHub 的页面上下载 exe 安装文件并运行：安装包下载地址：<https://gitforwindows.org/> 官网慢，可以用国内的镜像：<https://npm.taobao.org/mirrors/git-for-windows/>。

完成安装之后，就可以使用命令行的 git 工具（已经自带了 ssh 客户端）了，另外还有一个图形界面的 Git 项目管理工具。

在开始菜单里找到"Git"-">"Git Bash"，会弹出 Git 命令窗口，你可以在该窗口进行 Git 操作。

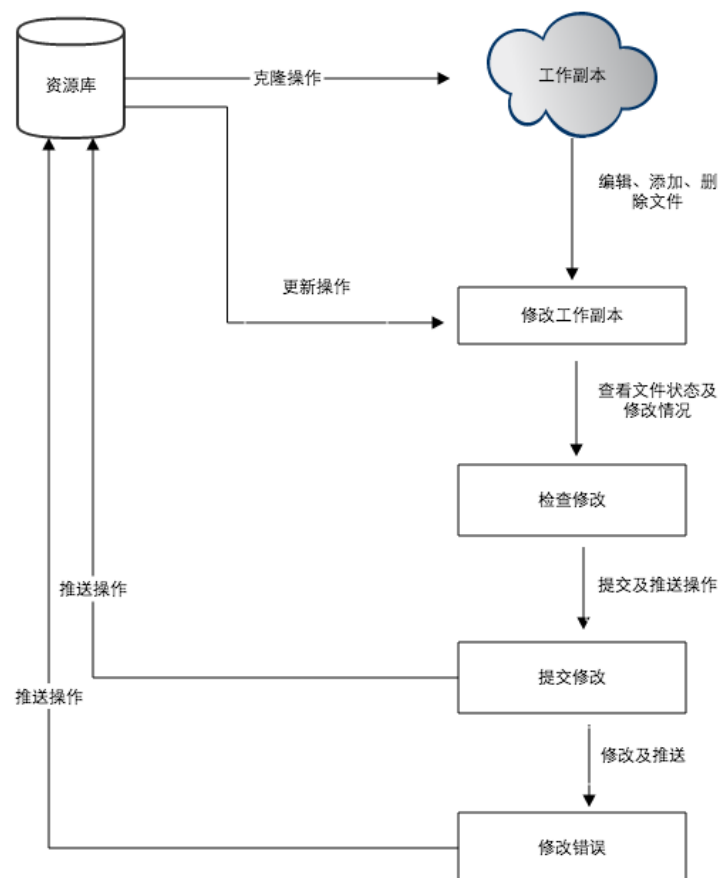
### 3git 的工作原理

#### 3.1git 工作流程

一般工作流程如下：

- 克隆 Git 资源作为工作目录。
- 在克隆的资源上添加或修改文件。
- 如果其他人修改了，你可以更新资源。
- 在提交前查看修改。
- 提交修改。
- 在修改完成后，如果发现错误，可以撤回提交并再次修改并提交。

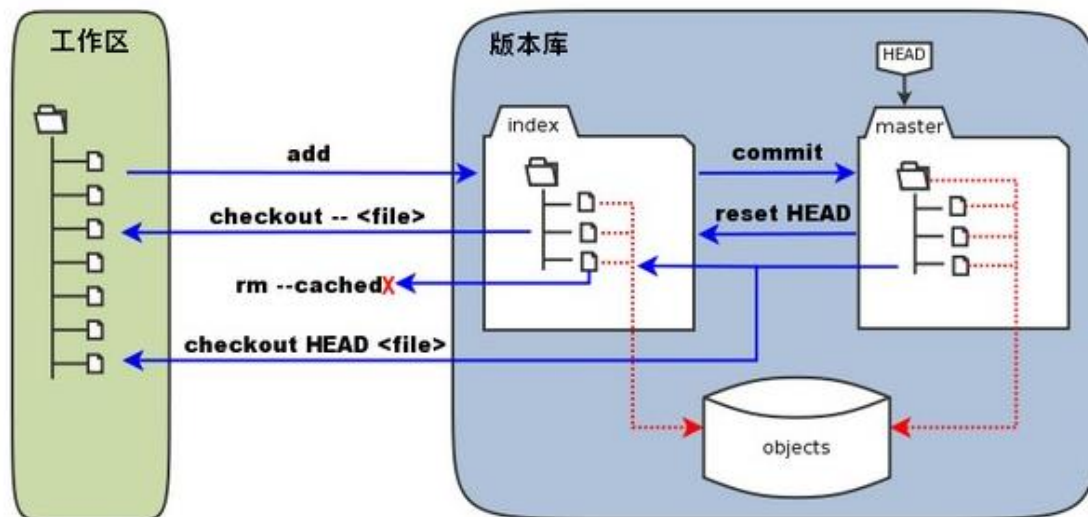
Git 工作流程



#### 3.2git 的工作区

我们先来了解下 Git 工作区、暂存区和版本库概念：

- **工作区：**就是你在电脑里能看到的目录。
- **暂存区：**英文叫 stage 或 index。一般存放在 .git 目录下的 index 文件(.git/index)中，所以我们把暂存区有时也叫作索引(index)。
- **版本库：**工作区有一个隐藏目录 .git，这个不算工作区，而是 Git 的版本库。



- 图中左侧为工作区，右侧为版本库。在版本库中标记为“index”的区域是暂存区（stage/index），标记为“master”的是 master 分支所代表的目录树。
- 图中我们可以看出此时“HEAD”实际是指向 master 分支的一个“游标”。所以图示的命令中出现 HEAD 的地方可以用 master 来替换。
- 图中的 objects 标识的区域为 Git 的对象库，实际位于“`.git/objects`”目录下，里面包含了创建的各种对象及内容。
- 当对工作区修改（或新增）的文件执行 `git add` 命令时，暂存区的目录树被更新，同时工作区修改（或新增）的文件内容被写入到对象库中的一个新的对象中，而该对象的 ID 被记录在暂存区的文件索引中。
- 当执行提交操作（`git commit`）时，暂存区的目录树写到版本库（对象库）中，master 分支会做相应的更新。即 master 指向的目录树就是提交时暂存区的目录树。
- 当执行 `git reset HEAD` 命令时，暂存区的目录树会被重写，被 master 分支指向的目录树所替换，但是工作区不受影响。
- 当执行 `git rm --cached <file>` 命令时，会直接从暂存区删除文件，工作区则不做出改变。
- 当执行 `git checkout .` 或者 `git checkout -- <file>` 命令时，会用暂存区全部或指定的文件替换工作区的文件。这个操作很危险，会清除工作区中未添加到暂存区的改动。
- 当执行 `git checkout HEAD .` 或者 `git checkout HEAD <file>` 命令时，会用 HEAD 指向的 master 分支中的全部或者部分文件替换暂存区和以及工作区中的文件。这个命令也是极具危险性的，因为不但会清除工作区中未提交的改动，也会清除暂存区中未提交的改动。

## 4git 操作命令说明

### 4.1 常用的命令：

#### 1 新建代码库：

在当前目录新建一个 Git 代码库	git init
下载一个项目和它的整个代码历史	git clone url

#### 2 配置

设置提交代码时的用户信息	git config [--global] user.name "[name]"
设置提交代码时的用户信息	git config [--global] user.email "[email address]"

#### 3 增加和删除

添加指定文件到暂存区	git add [file1] [file2]
添加指定目录到暂存区，包括子目录	git add [dir]
添加当前目录的所有文件到暂存区	git add .
删除工作区文件，并且将这次删除放入暂存区	git rm [file1] [file2] ...
改名文件，并且将这个改名放入暂存区	git mv [file-original] [file-renamed]
停止追踪指定文件，但该文件会保留在工作区	git rm --cached [file]

#### 4 代码的提交

提交暂存区到仓库区	git commit -m [message]
提交暂存区的指定文件到仓库区	git commit [file1] [file2] ... -m [message]
提交时显示所有 diff 信息	git commit -v

#### 5 分支命令

列出所有本地分支	git branch
列出所有远程分支	git branch -r
列出所有本地分支和远程分支	git branch -a
新建一个分支，指向指定 commit	git branch [branch] [commit]
切换到指定分支，并更新工作区	git checkout [branch-name]
切换到上一个分支	git checkout -
合并指定分支到当前分支	git merge [branch]
选择一个 commit，合并进当前分支	git cherry-pick [commit]
删除分支	git branch -d [branch-name]
删除远程分支	git push origin --delete [branch-name]
删除远程分支	git branch -dr [remote/branch]

#### 6 标签

列出所有 tag	git tag
新建一个 tag 在当前 commit	git tag [tag]
新建一个 tag 在指定 commit	git tag [tag] [commit]

#### 7 查看信息

显示有变更的文件	git status
显示当前分支的版本历史	git log

#### 8 撤销命令



---

恢复暂存区的指定文件到工作区	git checkout [file]
恢复某个 commit 的指定文件到暂存区和工作区	git checkout [commit] [file]
恢复暂存区的所有文件到工作区	git checkout .

```

Git init                //创建本地的 git 的仓库
Git clone               //复制项目
Git add filename        //添加某一个文件
Git add -all            //添加所有文件
Git rm filename         //删除文件
Git commit -m           //提交到本地仓库中的
Git reset               //回退历史版本
Git reset -hard HEAD^   //回退到历史的版本
git remote add origin git://127.0.0.1/abc.git 这样就增加了远程仓库 abc。
git remote remove origin //移除远端仓库
git pull                //从远端库更新内容到本地（相当于 svn 的 update），
git reset HEAD~2 --hard // 回撤 2 步
git merge 分支名        // 把该分支的内容合并到现有分支上
git branch -d 分支名    // 删除分支
git branch -D 分支名    // 强制删除 若没有其他分支合并就删除 d 会提示 D 不会
git push -u origin 分支名 // 将本地分支推送到 origin 主机。

```

## 5 私有云本地 git 怎么构建

### 5.1 本地服务器 git 的初始化

#### 1 安装 git 并修改 sshd 配置文件

- RSAAuthentication yes
- PubkeyAuthentication yes
- AuthorizedKeysFile .ssh/authorized\_key

#### 2. 添加用户并修改仓库属组和属主

# useradd git # 添加用户 (clone 和 push 的时候都是必要的), 注意这里要和你未来 web 服务器的名字要相同, 如果不是做 web 服务器用的可以随便取名字

# passwd git# 添加密码 (设置每一个人登入的密码)

# chown -R git:git /path/to/projectDir/ # 修改仓库的属组和属主 (一定要设置 否则不能 push 后面是的项目的路径)

#### 3. 选定一个目录作为 git 仓库假定是/home/test.git, 在/home 目录下输入命令:

```
git init --bare sample.git
```

4

### 5.2 用户的连接和使用

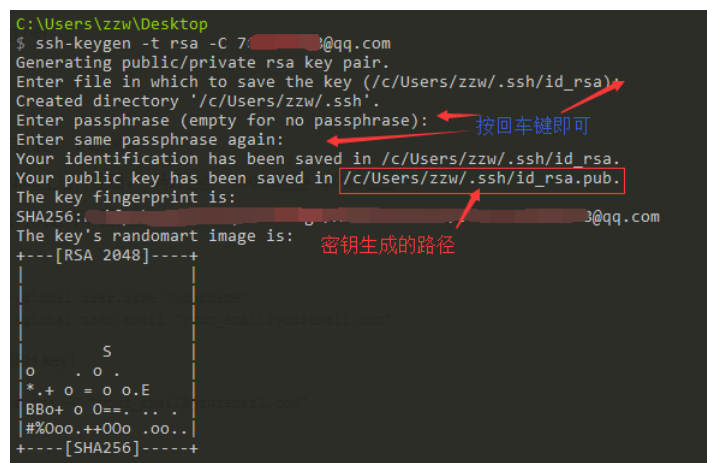
#### 1. 设置用户名和邮箱(--global 为全局参数, 表明本地所有 Git 仓库都会使用这个配置)

```
git config --global user.name "yourname"
```

```
git config --global user.email "your_email@youremail.com"
```

#### 2. 生成密钥 (SSH key)

```
ssh-keygen -t rsa -C "your_email@youremail.com"
```



```
C:\Users\zzw\Desktop
$ ssh-keygen -t rsa -C 7@qq.com
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/zzw/.ssh/id_rsa):
Created directory '/c/Users/zzw/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/zzw/.ssh/id_rsa.
Your public key has been saved in /c/Users/zzw/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:
The key's randomart image is:
+---[RSA 2048]-----+
|
| o . S
| o . o .
| * . + o = o o . E
| BB o + o O = . . .
| # % O o o . + + O O o . o o .
+---[SHA256]-----+
```

#### 3. 添加密钥 (SSH key), 并验证是否成功

添加密钥: 将上一步骤生成的密钥即 .ssh/id\_rsa.pub 中内容全部复制。在 github 的 Settings-->SSH and GPG keys-->New SSH key, key 中粘贴复制的内容 (Title 自定义)。

**注意:** 如果有多个人员参与, 每个人的公钥都要追加到 authorized\_keys 里面, ssh-copy-id 命令会自动创建 .ssh 文件夹和追加公钥到 authorized\_keys 文件里

#### 4 在 gitgu 服务器中添加本地 id\_rsa.pub

- # mkdir -p /home/git/.ssh # 由配置文件我们把认证信息放到了用户家目录下的 .ssh 文件夹中, www 为刚刚添加的用户
- • # vim /home/git/.ssh/authorized\_keys # 粘贴你刚刚复制的 id\_rsa.pub 内容
- • # chmod 700 /home/git/.ssh/ # 为了保证安全性, 需要修改权限

- • # chmod 600 /home/git/.ssh/authorized\_keys #修改文件权限
- • # usermod -s /usr/bin/git-shell git#不允许该用户登录，只能做 git 操作
- 如果想要简洁操作可以在 Windows git bash 上使用这个命令：
- ssh-copy-id -i /c/Users/用户名/.ssh/id\_rsa.pub git@11.11.12.12
- linux 也可以使用上面这条命令或者执行
- • # ssh-keygen //生成 linux 的公钥和私钥 /root/.ssh 目录下面

## 5 从私有云 git 仓库 clone 项目到本地

# git clone [git@192.168.0.246:/disk/git/test.git](#) 从服务器克隆（分配的账户@目标 IP）

## 6 push 到远端仓库

git status 查看工作目录的状态  
 git add <file> 将文件添加到暂存区  
 git commit -m "commnet" 提交更改, 添加备注信息(此时将暂存区的信息提交到本地仓库)  
 git push origin master 将本地仓库的文件 push 到远程仓库(若 push 不成功, 可加 -f 进行强推操作)

## 5.3 用户个人的操作

### 创建仓库命令

下表列出了 git 创建仓库的命令：

命令	说明
<a href="#">git init</a>	初始化仓库
<a href="#">git clone</a>	拷贝一份远程仓库，也就是下载一个项目。

### 提交与修改

Git 的工作就是创建和保存你的项目的快照及与之后的快照进行对比。

下表列出了有关创建与提交你的项目的快照的命令：

命令	说明
<a href="#">git add</a>	添加文件到仓库
<a href="#">git status</a>	查看仓库当前的状态，显示有变更的文件。
<a href="#">git diff</a>	比较文件的不同，即暂存区和工作区的差异。
<a href="#">git commit</a>	提交暂存区到本地仓库。
<a href="#">git reset</a>	回退版本。
<a href="#">git rm</a>	删除工作区文件。
<a href="#">git mv</a>	移动或重命名工作区文件。

### 提交日志

---

命令	说明
<a href="#">git log</a>	查看历史提交记录
<a href="#">git blame &lt;file&gt;</a>	以列表形式查看指定文件的历史修改记录

#### 远程操作

命令	说明
<a href="#">git remote</a>	远程仓库操作
<a href="#">git fetch</a>	从远程获取代码库
<a href="#">git pull</a>	下载远程代码并合并
<a href="#">git push</a>	上传远程代码并合并

---

## 6git 的容灾备份

假设已有 git 仓库 git@10.22.52.217:/srv/autoltp.git

### 1. 进入备份的目的机器目录

```
cd /home/aouyang/work/autoltp
```

### 2. 备份

```
git clone --mirror git@10.22.52.217:/srv/autoltp.git
```

### 3. 如果已有备份，则更新备份

```
git remote update
```

### 4. 编写定时备份

a. touch ltp-backup.sh

```
#!/bin/bash
```

```
cd /home/aouyang/work/autoltp_backup/autoltp.git/
```

```
git remote update
```

```
cd -
```

b. crontab -e

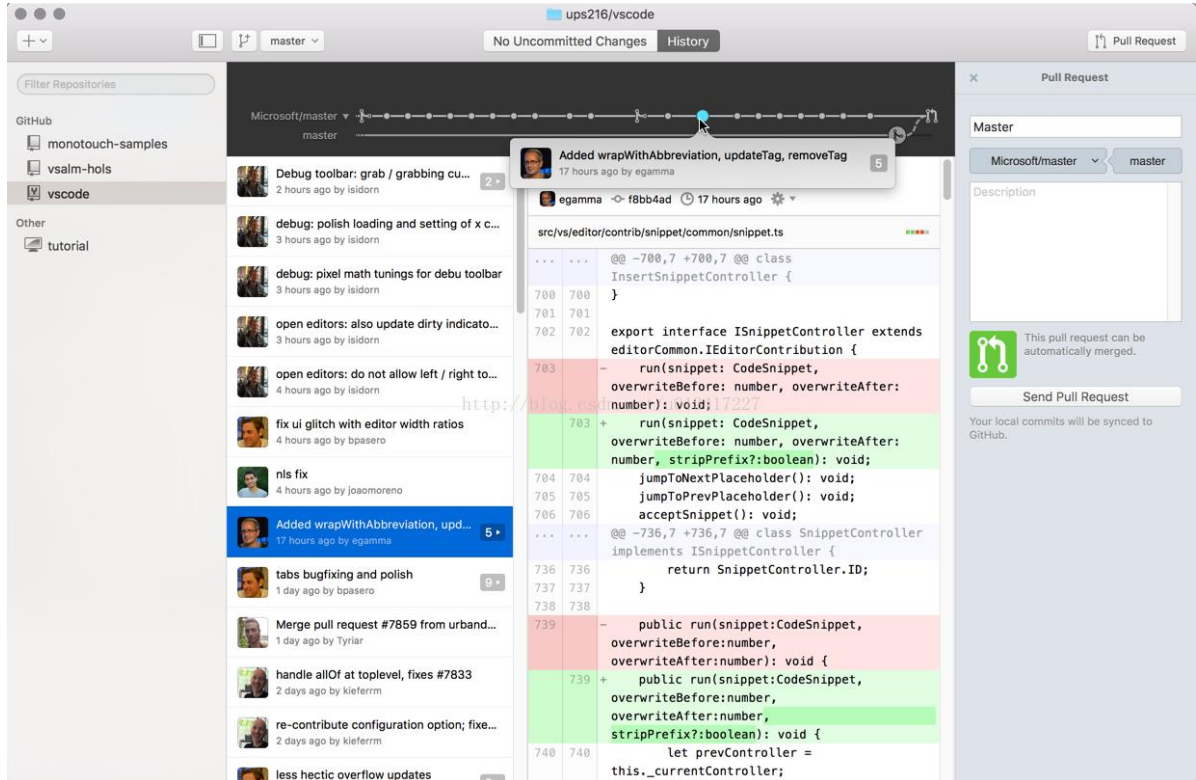
写入如下定时器任务：每周六 8 点整，定时执行 ltp-backup.sh 文件

并向多个服务集群中的进行的定时的备份工作。

## 7git 的可视化管理工具

### 7.1GitHub for Desktop

全球开发人员交友俱乐部提供的强大工具，功能完善，使用方便。对于使用 GitHub 的开发人员来说是非常便捷的工具。界面干净，用起来非常顺手，上面的这条 timeline 非常漂亮，也可以直接提交 PR。

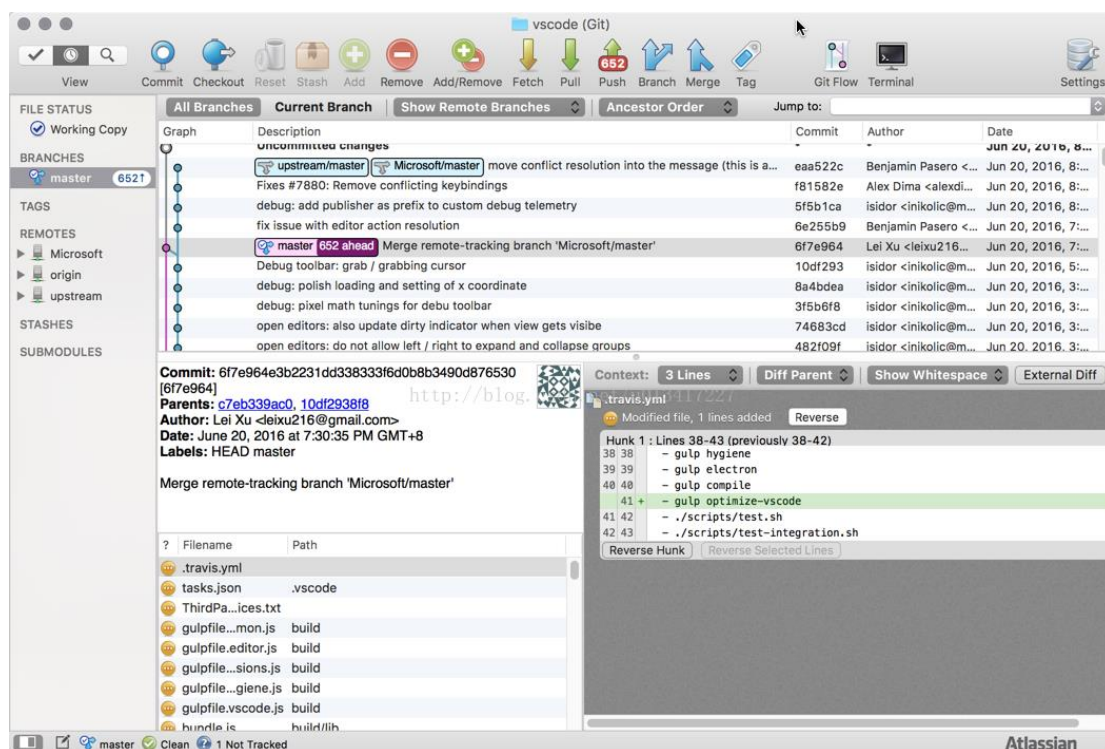


让我失望的是 GitHub for Desktop 不带三方合并工具，你必须自己手动解决冲突才可以。

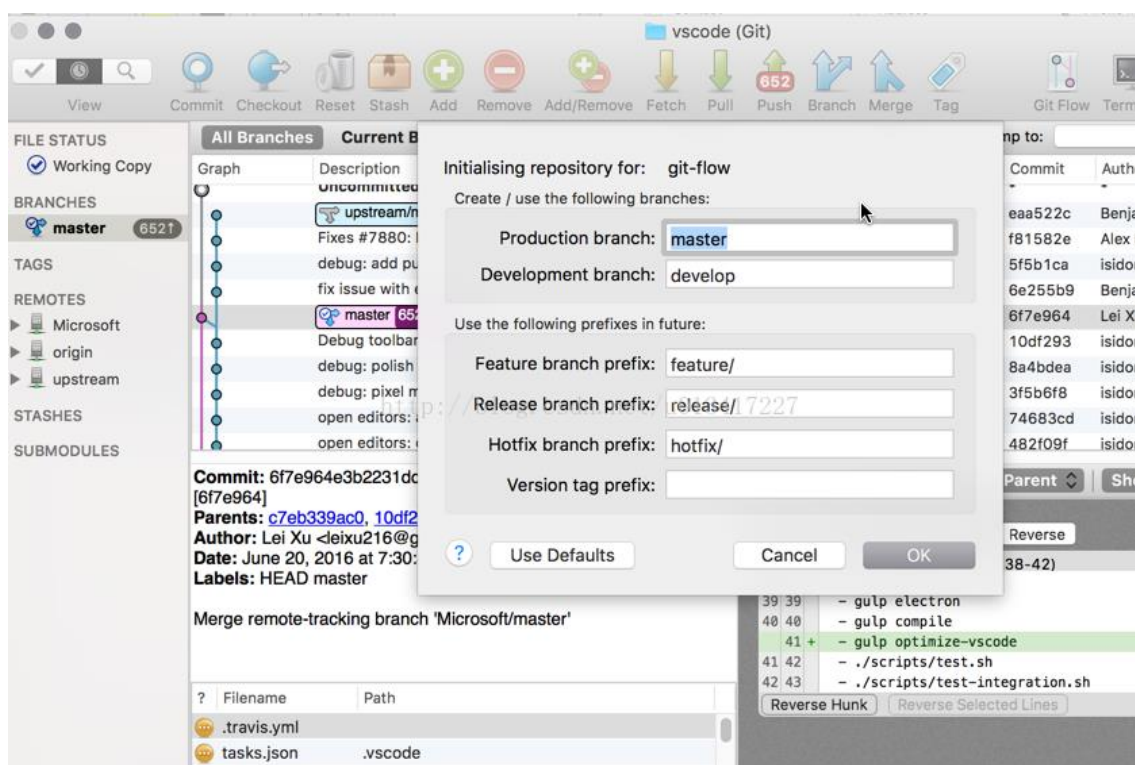
1. 免费
2. 同时支持 Windows 和 Mac：对于需要经常在不同的操作系统间切换的开发人员来说非常方便。
3. 漂亮的界面：作为每天盯着看的工具，颜值是非常重要的
4. 支持 Pull Request：直接从客户端提交 PR，很方便
5. Timeline 支持：直接在时间线上显示每次提交的时间点和大小
6. 支持 git LFS：存储大文件更加节省空间和高效
7. 不支持三方合并：需要借助第三方工具才行

### 7.2Source Tree

SourceTree 是老牌的 Git GUI 管理工具了，也号称是最好用的 Git GUI 工具。我的体验是确实强大，功能丰富，基本操作和高级操作都设计得非常流畅，适合初学者上手。



这个工具很有特色的一个功能就是支持 Git Flow，你可以一键创建 Git Flow 的工作流。Git Flow 是非常高效的团队协作模型和流程，Git 的一大特色就是灵活轻量的分支，但如何在自己的团队中用好这个功能来匹配自己的研发流程是个问题。内置 Git Flow 让那些不太熟悉的开发人员也可以很快上手，并且将研发的业务流程固化在工具中，可以说是非常贴心的设计。



1. 免费
2. 功能强大：无论你是新手还是重度用户，SourceTree 都会让你觉得很顺手。对于非常重

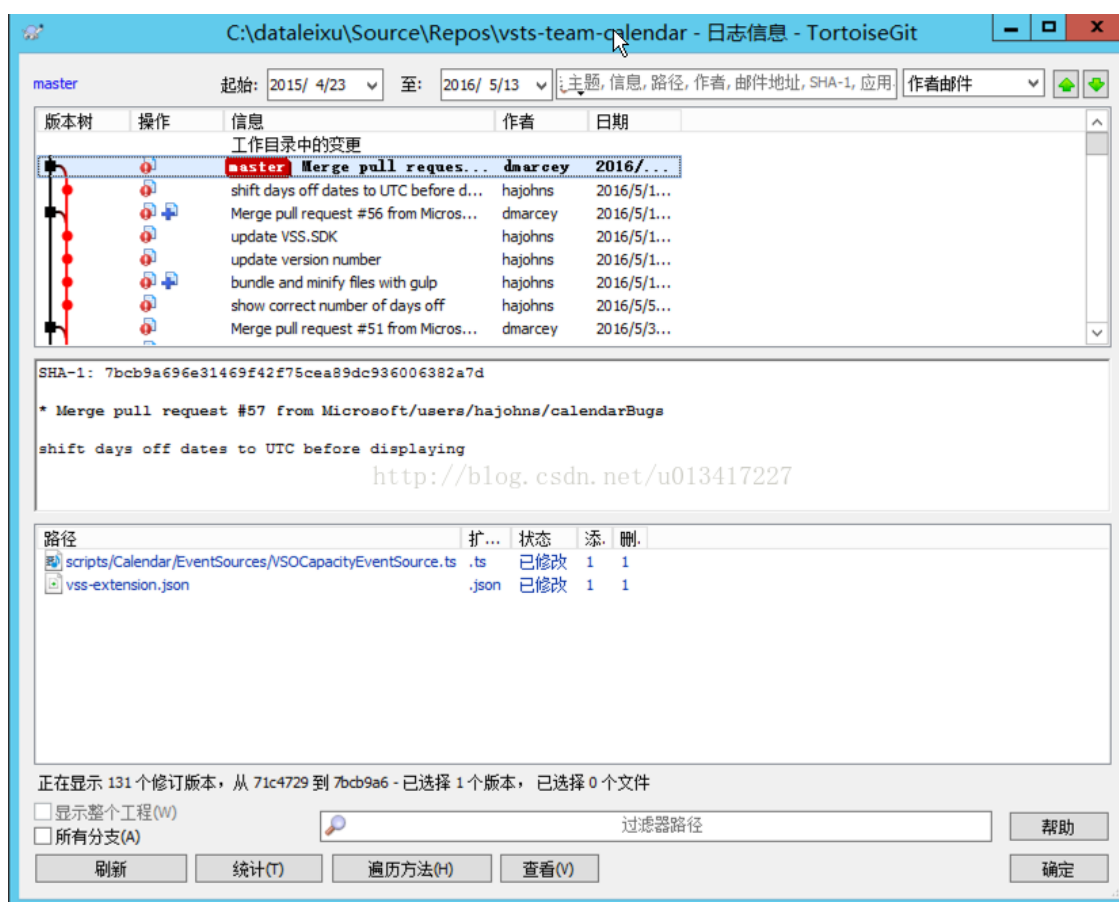


度用户，Source Tree 还支持自定义脚本的执行。

3. 同时支持 Windows 和 Mac 操作系统
4. 同时支持 Git 和 Mercurial 两种 VCS
5. 内置 GitHub, BitBucket 和 Stash 的支持：直接绑定帐号即可操作远程 repo

### 7.3 TortoiseGit

对这只小乌龟估计没有开发人员会不认识，SVN 的超广泛使用也使得这个超好用的 Svn 客户端成了几乎每个开发人员的桌面必备软件。小乌龟只提供 Windows 版本，提供中文版支持的，对于中国的开发者来说绝对是福音。



免费

只支持 Windows 操作系统：与文件管理器的良好集成

中文界面

与 TortoiseSVN 一脉相承的操作体验



---

## 8 其他问题以及解决方案