**Git应用介绍**

**2017年1月**

目 录

1. Git 功能概述 3

2. Git for Windows软件安装 4

3. Git与远程仓库的关联 5

3.1. 本地Git与oschina的连接 5

3.2. 本地Git与oschina关联（一） 7

3.3. 本地Git与oschina关联（二） 10

3.4. 本地Git与和团队项目保持同步 11

3.5. 解决merge时的冲突 12

3.6. push修改到自己的项目上 13

4. Git 常用命令 14

# Git 功能概述

我们用Git来对这些不同的版本进行控制。还可以很方便地查看两个不同版本之间的不同之处。

使用Git，我们只保存最新的一份文件就可以了。

使用Git，我们可以把文件回退到想要的旧版本。

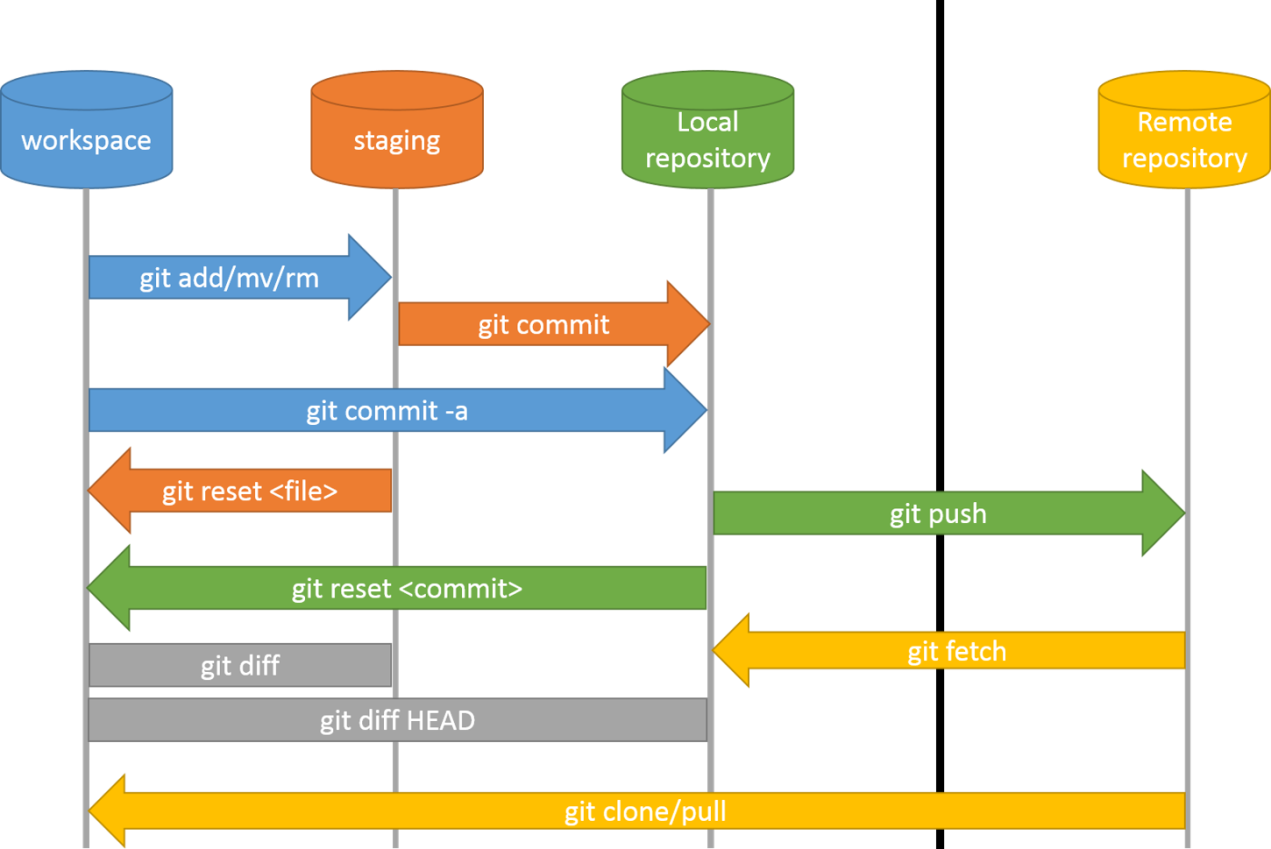
使用Git，我们可以将回退的版本再次还原版本。

Git 有三种状态，你的文件可能处于其中之一：

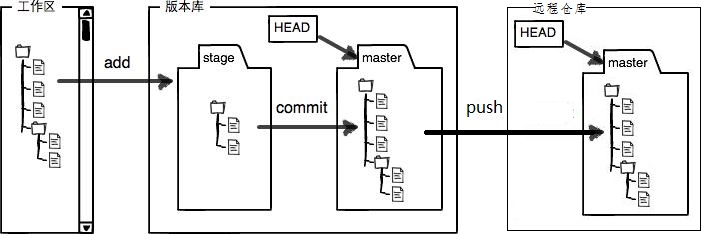
已提交（committed）、已修改（modified）和已暂存（staged）。

* 已提交表示数据已经安全的保存在本地数据库中。
* 已修改表示修改了文件，但还没保存到数据库中。
* 已暂存表示对一个已修改文件的当前版本做了标记，使之包含在下次提交的快照中。

Git本地使用与远程仓库的关系：



图片引用自：[Git introduction for CVS/SVN/TFS users](http://blog.podrezo.com/git-introduction-for-cvssvntfs-users/)



图片引用自：[工作区和暂存区 - 廖雪峰的官方网站](http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013745374151782eb658c5a5ca454eaa451661275886c6000) （做了点修改）

对照查看两张图：

* workspace 即工作区，逻辑上是本地计算机，还没添加到repository的状态；
* staging 即版本库中的stage，是暂存区。修改已经添加进repository，但还没有作为commit提交，类似于缓存；
* Local repository 即版本库中master那个地方。到这一步才算是成功生成一个新版本；
* Remote repository 则是远程仓库。用来将本地仓库上传到网络，可以用于备份、共享、合作。

# Git for Windows软件安装

* 在官方网站https://git-for-windows.github.io/workspace 下载最新版本；
* 安装时全部使用默认安装，默认添加环境变量。

# Git与远程仓库的关联

## 本地Git与oschina的连接

**Git命令：**

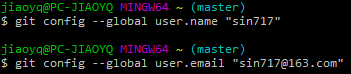
|  |  |
| --- | --- |
| **命令** | **备注** |
| git config --global user.name "你的用户名" | 设置本地用户名 |
| git config --global user.email "你的邮箱" | 设置邮箱 |
| ssh-keygen -t rsa -C "你的邮箱" | 生成ssh key |
| ssh -T git@git.oschina.net | 测试链接状态 |

1. 到oschina注册账号。
2. 本地配置用户名和邮箱

$ git config --global user.name "你的用户名"

$ git config --global user.email "你的邮箱"

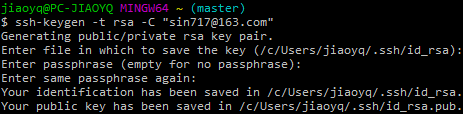
如图：



※也可以修改C:\Users\你的用户名\.gitconfig。

1. 生成ssh key

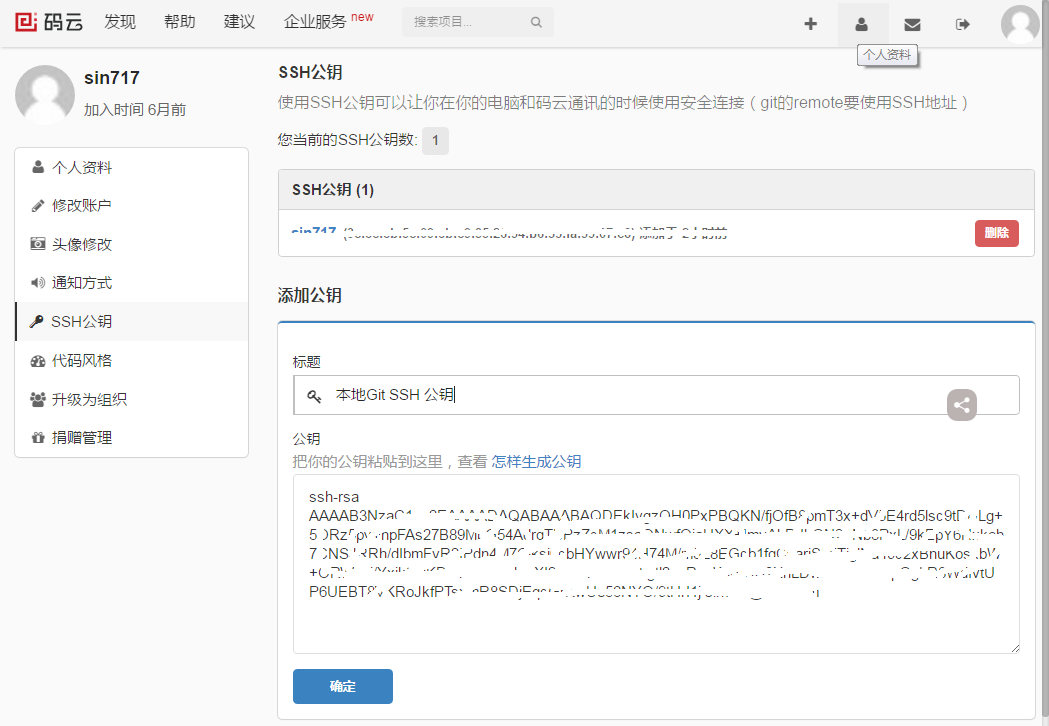
$ ssh-keygen -t rsa -C "你的邮箱"，它会有三次等待你输入，直接回车即可。



将生成的ssh key复制到剪贴板，可从上图的路径打开文件id\_rsa.pub并复制。

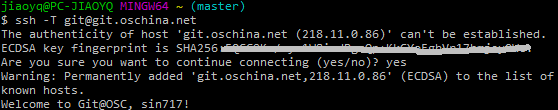
1. 打开oschina，进入个人资料修改“SSH公钥”：

将ssh key粘贴到右边的公钥里面。标题随便命名即可。



点击下面的确认，输入验证密码就添加成功了。

测试链接状态，执行$ ssh -T git@git.oschina.net：



链接成功。

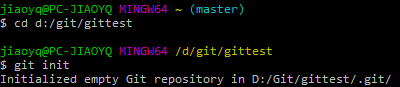
## 本地Git与oschina关联（一）

**Git命令：**

|  |  |
| --- | --- |
| **命令** | **备注** |
| git init | 初始化本地仓库 |
| git remote add origin 远程仓库地址 | 添加远程仓库地址 |
| git push -u origin master | 本地仓库上传至远程仓库并进行关联 |
| git push | 本地仓库上传至远程仓库 |

1. 本地库初始化

进入本地库所在文件夹，执行$ git init来初始化仓库。



它会创建一个隐藏的文件夹 **.git**

1. 创建远程仓库

首先在右上角点击“新建项目”进入创建界面：



接着输入远程仓库名：



点击“创建”就创建好了。

1. 将远程仓库和本地仓库关联起来

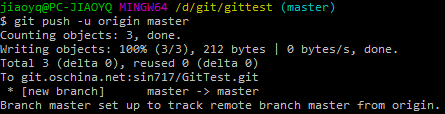
先到oschina上复制远程仓库的SSH地址：



执行$ git remote add origin 你复制的地址：

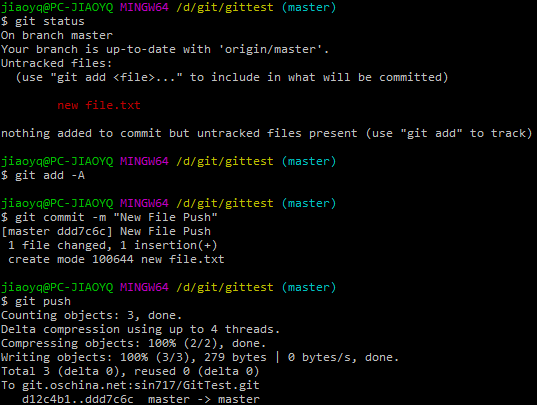


执行$ git push -u origin master将本地仓库上传至oschina的仓库并进行关联：



以上关联已经完成！

以后同步到oschina上，只要直接执行$ git push ：



在oschina查看结果：

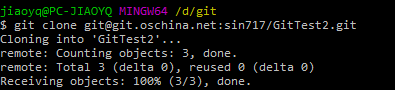


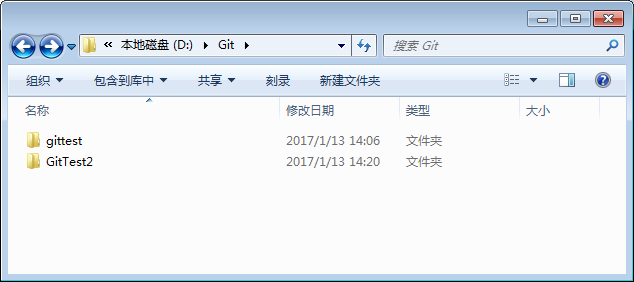
## 本地Git与oschina关联（二）

**Git命令：**

|  |  |
| --- | --- |
| **命令** | **备注** |
| git clone 远程仓库地址 | 克隆远程仓库内容，并完成初始化和关联 |

1. 在oschina创建远程仓库。
2. 复制刚才创建的仓库的SSH
3. 执行$ git clone 你复制的地址：





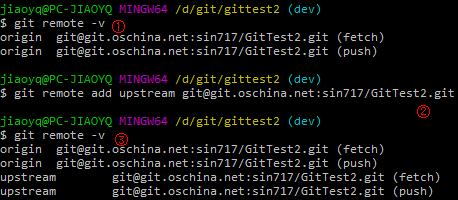
这样可以不用再做关联了。省去了上面的 init 和 git remote add origin 以及 git push -u origin master 。

## 本地Git与和团队项目保持同步

**Git命令：**

|  |  |
| --- | --- |
| **命令** | **备注** |
| git remote -v | 查看远程服务器地址和仓库名称 |
| git remote add upstream 远程仓库地址 |  |
| git fetch upstream | 获取最新版本 |
| git merge upstream/分支 |  |

1. 首先查看有没有设置upstream，使用$ git remote -v命令来查看。如下图①



1. 如果没有显示upstream，则使用$ git remote add upstream 远程仓库地址命令。如上图②
2. 接着再次使用$ git remote -v，如果如上图③，显示出了upstream，那么就设置好了
3. 开始同步。执行$ git fetch upstream获取团队项目最新版本。如下图①

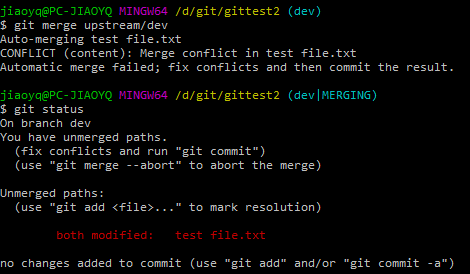


1. 此时并没有把最新版本合并到你本地的分支上，因此还需要一步。如上图②，当前分支是dev分支，执行$ git merge upstream/dev命令后，会将源分支（upstream/dev）合并到当前分支（dev）。

※merge的时候，有可能碰到冲突。需要解决冲突才能继续下面的操作。

## 解决merge时的冲突

冲突提示如下图：

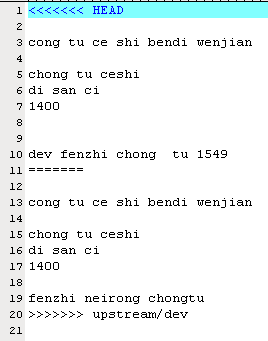


CONFLICT表示有冲突，在这一行的末尾，显示冲突文件。

使用$ git status我们可以看到test file.txt自动被合并修改了。

※当前的分支显示“(dev|MERGING)”

我们以test file.txt为例，解决冲突

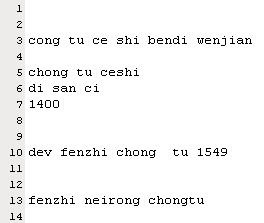


红框中的“=======”是冲突的分割线。

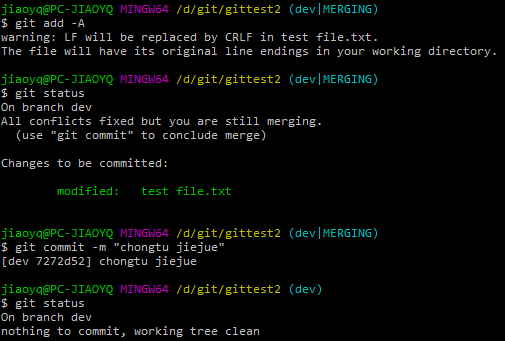
“<<<<<<< HEAD”和分割线之间的是本地的文本，分割线和“>>>>>>> upstream/dev”之间的是远程分支的文本。

冲突内容解决后，只需将冲突符号删除。

解决冲突后如下图所示：

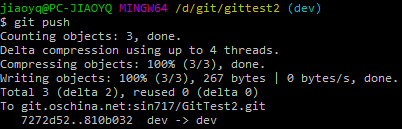


git提交修改内容，当前的分支显示“(dev)”



## push修改到自己的项目上

如有冲突，解决冲突后，使用$ git push命令将本地的修改同步到远程仓库上。



※在当前所在分支使用push，会push到与这个分支相关联的远程仓库分支。这里dev分支与origin/dev关联，因此push到GitHub上的dev分支。

# Git 常用命令

**个人本地命令：**

|  |  |
| --- | --- |
| **命令** | **备注** |
| git init | 初始化本地仓库 |
| git clone 远程仓库地址 | 从网络上某个地址拷贝仓库(repository)到本地，并完成初始化和关联 |
| git status | 查看当前仓库的状态。碰到问题不知道怎么办的时候，可以通过看它给出的提示来解决问题 |
| git add <file> | 文件提交暂存区。在commit前要先add |
| git add -A | 将分支内所有工作区文件提交暂存区 |
| git commit -m "备注" | 将暂存区的文件提交到本地库 |
| git rm <file> -f | 将在暂存区的文件直接删除 |
| git rm <file> --cached | 将在暂存区的文件删除，但不删除文件  相当于git reset <file> |
| git reset <file> | 从暂存区恢复到工作区 |
| git reset -- . | 将所有暂存区文件恢复到工作区 |
| git reset --hard <版本> | 恢复指定版本的内容，放弃该版本后的所有修改内容，不输入版本号则恢复最近一个版本 |
| git diff <file> | 比较当前文件和暂存区文件差异 |
| git diff <版本1> <版本2> | 比较2个版本间的差异 |
| git clean -xf | 删除当前目录下所有没有操作痕迹的文件 |
| git log | 查看当前版本及之前的commit记录 |
| git remote -v | 查看远程服务器地址和仓库名称 |
| git reflog | 查看HEAD历史记录 |
| git checkout <branch> | 切换分支 |
| git checkout -b <New branch> | 创建新分支，并且切换过去 |
| git checkout <版本> -b <New branch> | 以某次历史记录创建分支，但无log信息 |
| git branch | 查看分支信息 |
| git branch -d <branch> | 删除分支 |

**个人远程仓库命令：**

| **命令** | **备注** |
| --- | --- |
| git config --global user.name "你的用户名" | 设置本地用户名 |
| git config --global user.email "你的邮箱" |  |
| ssh-keygen -t rsa -C "你的邮箱" | 生成ssh key |
| ssh -T git@git.oschina.net | 测试链接状态 |
| git push -u origin master | 本地仓库上传至远程仓库并进行关联 |
| git push | 本地仓库上传至远程仓库 |
| git clone 远程仓库地址 | 克隆远程仓库内容，并完成初始化和关联 |
| git remote -v | 查看远程服务器地址和仓库名称 |
| git remote add origin 远程仓库地址 | 添加远程仓库地址 |
| git remote add upstream 远程仓库地址 | 添加远程仓库upstream地址 |
| git fetch upstream | 获取最新版本 |

|  |  |
| --- | --- |
| git merge upstream/分支  git pull | 以本地库为标准合并远程库的最新版本  获取最新版本并merge到本地 |