# Git教程

## 集中式（CVS、SVN）vs分布式（Git）

集中式版本控制：有一个中央服务器，每个人都从服务器取得最新版本，修改完成后再推送给服务器。且必须联网工作。

分布式版本控制：没有中央服务器，每个人都有一个完整的版本库，工作不需要联网，所谓的“中央服务器”作用只是方便大家交换自己的修改。还有极其强大的分支管理。

## 安装

**Linux系统安装git**：通过一行命令，sudo apt-get install git

**源码安装**：./config，make，sudo make install

**MacOs安装**：从AppStore安装Xcode，Xcode集成了Git，不过默认没有安装，你需要运行Xcode，选择菜单“Xcode”->“Preferences”，在弹出窗口中找到“Downloads”，选择“Command Line Tools”，点“Install”就可以完成安装了。

**Windows系统安装**：msysgit <https://git-for-windows.github.io/>

因为Git是分布式版本控制系统，所以，每个机器都必须自报家门：你的名字和Email地址。

$ git config --global user.name "Your Name"

$ git config --global user.email "email@example.com"

## 创建版本库repository

需要明确的是，所有的版本控制系统，其实只能跟踪文本文件的改动，比如TXT文件，网页，所有的程序代码等等，Git也不例外。版本控制系统可以告诉你每次的改动，但图片、视频这些二进制文件，虽然也能由版本控制系统管理，但没法跟踪文件的变化，只能把二进制文件每次改动串起来，也就是只知道图片从100KB改成了120KB，但到底改了啥，版本控制系统不知道，也没法知道。

Windows自带的记事本在每个文件开头添加了0xefbbbf（十六进制）的字符，会出现很多奇怪问题。建议使用notepad++ ，记得把Notepad++的默认编码设置为UTF-8 without BOM即可。

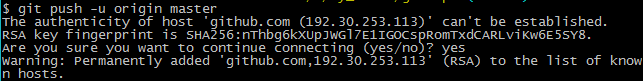
1. 新建一个空目录：mkdir repository
2. 在创建的目录下初始化使目录变成git可以管理的仓库：git init
3. 添加文件：git add <filename> //可一次添加多个文件；git commit -m “comments” //提交到仓库中。
4. 查看文件状态：git status ,要随时掌握工作区的状态，使用git status命令
5. 查看修改内容：git diff <filename> / git diff HEAD -- <filename> //查看工作区与版本库文件的区别
6. 提交修改和提交新文件是一样的两步:git add <filename> / git commit -m “comments”
7. 查看版本修改的历史记录：git log / git log --pretty=oneline //精简输出
8. 分布式版本管理，在多人操作时，很容易出现冲突，因此git会生成一个SHA1计算出来的16进制的“版本号”。
9. 在Git中，用HEAD表示当前版本HEAD^ 表示上个版本，HEAD~100表示上100个版本，回退到某个版本：git reset --hard HEAD^ / git reset --hard 3628164 //版本可以只写前几位。
10. Git记录每一次命令：git reflog //可通过显示的commit id回到当前版本。
11. **Git暂存区**：git有一个称为stage（或者叫index）的暂存区，还有Git为我们自动创建的第一个也是唯一一个分支master，以及指向master的一个指针叫HEAD。工作区即是实际目录。
12. 每次修改后，都需要add到暂存区，不然不会commit到版本库。
13. 使文件回到最近一次git commit或git add时的状态：git checkout -- filename ,一种是readme.txt自修改后还没有被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态；

一种是readme.txt已经添加到暂存区后，又作了修改，现在，撤销修改就回到添加到暂存区后的状态。

1. 命令git reset HEAD file可以把暂存区的修改撤销掉（unstage），重新放回工作区；再丢弃工作区的修改：git checkout – filename
2. git rm fielname //从版本库删除文件。

## 远程仓库

在用户住目录下有一个.ssh目录，这个目录下有两个文件：id\_rsa , id\_rsa.pub 为SSH Key的秘钥对，id\_rsa是私钥，不能泄露出去，id\_rsa.pub是公钥，可以放心地告诉任何人。

* 创建SSH Key：ssh-keygen –t rsa –C [ldjwyyx@163.com](mailto:ldjwyyx@163.com)
* 第2步：登陆GitHub，打开“Account settings”—>“SSH Keys”🡪“Add SSH Key”，填上任意Title，在Key文本框里粘贴id\_rsa.pub文件的内容。
* 为什么GitHub需要SSH Key呢？因为GitHub需要识别出你推送的提交确实是你推送的，而不是别人冒充的，而Git支持SSH协议，所以，GitHub只要知道了你的公钥，就可以确认只有你自己才能推送。当然，GitHub允许你添加多个Key。假定你有若干电脑，你一会儿在公司提交，一会儿在家里提交，只要把每台电脑的Key都添加到GitHub，就可以在每台电脑上往GitHub推送了。最后友情提示，在GitHub上免费托管的Git仓库，任何人都可以看到喔（但只有你自己才能改）。所以，不要把敏感信息放进去。
* 关联远程仓库（使用SSH）：git remote add origin [git@github.com:gitgogo/hello-world.git](mailto:git@github.com:gitgogo/hello-world.git) Git支持多种协议，包括https，但通过ssh支持的原生git协议速度最快。
* 本地内容全部推送到远程库：git push -u origin master //第一次push需加-u ，若执行失败git pull origin master //将远程仓库的文件与本地合并后再推送至远程仓库。
* 克隆版本库：git clone [git@github.com:gitgogo/gitskills.git](mailto:git@github.com:gitgogo/gitskills.git)

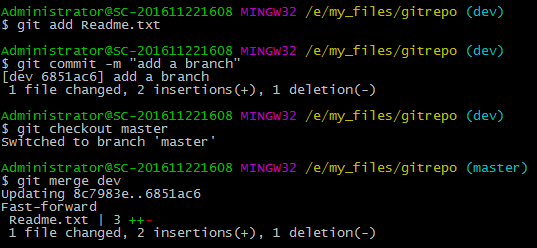
## 分支管理

一开始的时候，master分支是一条线，Git用master指向最新的提交，再用HEAD指向master，就能确定当前分支，以及当前分支的提交点；当我们创建新的分支，例如dev时，Git新建了一个指针叫dev，指向master相同的提交，再把HEAD指向dev，就表示当前分支在dev上。

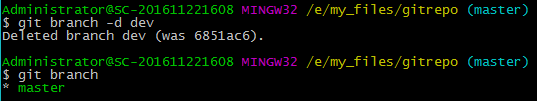
* **创建分支dev**：git checkout –b dev //相当于两条命令：git branch dev git checkout dev

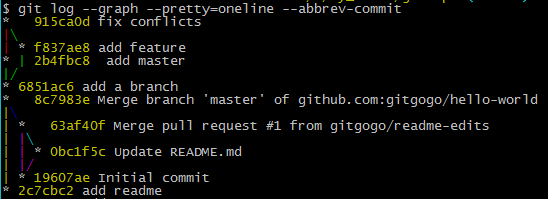


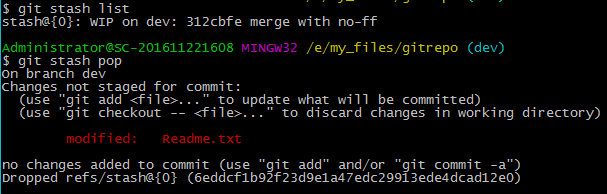
* 编辑本地文件，提交，切换至master分支，合并分支：git merge dev



* **删除分支**：git branch –d dev

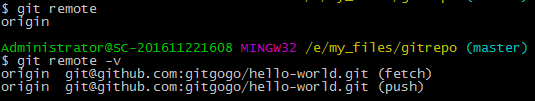


* **解决冲突**：当Git无法自动合并分支时，就必须首先解决冲突。解决冲突后（直接查看被修改文件的内容，git自动标识冲突的地方），再提交，合并完成。可查看合并log：git log --graph --pretty=oneline --abbrev-commit
* 禁用fast-forward模式：通常，合并分支时，如果可能，Git会用Fast forward模式，但这种模式下，删除分支后，会丢掉分支信息。git merge –no-ff –m “merge with no-ff” dev
* **分支策略**：master分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；干活都在dev分支上，也就是说，dev分支是不稳定的，到某个时候，比如1.0版本发布时，再把dev分支合并到master上，在master分支发布1.0版本；你和你的小伙伴们每个人都在dev分支上干活，每个人都有自己的分支，时不时地往dev分支上合并就可以了。
* **Bug分支：**当你正在自己的分支dev上开发东西时，需要临时去解决主线上的bug，首先git stash //把dev的现场保留，便于bug fix后继续开发；切换到master，git checkout master🡪git checkout –b issu-112 //新建bug分支；修复完成后提交至主线；删除issu-112分支；切换到dev分支🡪git checkout dev；查看stash列表🡪git stash list；还原dev🡪git stash pop；继续开发。 可多次stash，恢复指定的stash：git stash apply stash@{0}



* 用git stash apply恢复，需要git stash drop删除
* 开发新feature最好新建分支feature-xxx，当feature开发停止时，git branch –D feature-xxx

## 多人协作：

* 查看远程版本库：git remote -v
* **推送分支**：git push origin master //推送master分支到远程版本库；git push origin dev //推送dev分支；**master分支**是主分支，因此要时刻与远程同步；

**dev分支**是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步；

**bug分支**只用于在本地修复bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个bug；

**feature分支**是否推到远程，取决于你是否和你的小伙伴合作在上面开发。

* **多人协作工作模式**：1）首先，可以试图用git push origin branch-name推送自己的修改；2）如果推送失败，则因为远程分支比你的本地更新，需要先用git pull试图合并；3）如果合并有冲突，则解决冲突，并在本地提交；4）冲突解决掉后，再用git push origin branch-name推送就能成功！5）如果git pull提示“no tracking information”，则说明本地分支和远程分支的链接关系没有创建，用命令git branch --set-upstream branch-name origin/branch-name

## 标签管理

tag就是一个让人容易记住的有意义的名字，它跟某个commit绑在一起。

* 切换到需要打标签的分支；git tag <tag\_name>，git tag //查看所有tag
* 默认标签是打在最新提交的commit上的，在之前的commit上打标签：git tag v1.0 commit\_id
* Git show <tag\_name> //查看标签信息；$ git tag -a v0.1 -m "version 0.1 released" 3628164 //-a指定标签名，-m指定说明文字
* 还可以通过-s用私钥签名一个标签：$ git tag -s v0.2 -m "signed version 0.2 released" fec145a，签名采用PGP签名，因此，必须首先安装gpg（GnuPG），如果没有找到gpg，或者没有gpg密钥对，就会报错。
* 删除标签：git tag –d v1.0
* 推送某个标签到远程，使用命令git push origin <tagname> ，一次性推送全部尚未推送到远程的本地标签：git push origin –tags
* 要删除远程标签：先删除本地标签 git tag –d v1.0 ；再删除远程标签：git push origin :refs/tags/v1.0
* 在GitHub上，可以任意Fork开源仓库； 自己拥有Fork后的仓库的读写权限； 可以推送pull request给官方仓库来贡献代码。

## 自定义git

* 显示颜色：git config –global color.ui true
* 忽略特殊文件：新建 .gitignore 文件，配置需要忽略的文件格式。

忽略文件的原则是：

* 忽略操作系统自动生成的文件，比如缩略图等；
* 忽略编译生成的中间文件、可执行文件等，也就是如果一个文件是通过另一个文件自动生成的，那自动生成的文件就没必要放进版本库，比如Java编译产生的.class文件；
* 忽略你自己的带有敏感信息的配置文件，比如存放口令的配置文件。
* 可以对忽略的文件强制添加：git add –f file\_name ；检查.gitignore文件的配置详情：git check-ignore –v file\_name
* 配置别名：git config –global alias co checkout；常用别名：ci—commit；co—checkout；br—branch；unstage—‘reset HEAD’ ；**--global**参数是全局参数，也就是这些命令在这台电脑的所有Git仓库下都有用。git config --global alias.last 'log -1' //显示最近一次的提交
* git config --global alias.lg "log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"