Git基本使用

易剑波　2016-06-27

Git是什么(或者说，版本控制是什么)?

什么叫大神 —> 为了管理别人提交的Linux源代码，Linus两周时间用C语言写出一个版本管理系统—Git!

没有版本控制之前:
毕业论文版本1.doc
毕业论文版本2.doc
毕业论文版本3.doc
毕业论文版本4.doc
……
毕业论文版本N.doc
毕业论文版本N+1.doc
……
遗书

使用版本控制:

版本	用户	说明	日期
1	张三	完成了背景介绍	2016-05-01
2	张三	国内外研究现状	2016-05-05
3	张三	完成了研究方法和步骤的部分	2016-05-25
4	张三	国内外研究现状修改	2016-05-27

……..

Git vs. SVN(分布式 vs. 集中式)

集中式：版本库是集中存放在中央服务器的，而干活的时候，用的都是自己的电脑，所以要先从中央服务器取得最新的版本，然后开始干活，干完活了，再把自己的活推送给中央服务器。集中式版本控制系统最大的毛病就是必须联网才能工作。

分布式： 分布式版本控制系统根本没有“中央服务器”，每个人的电脑上都是一个完整的版本库，工作的时候不需要联网，因为版本库就在你自己的电脑上。既然每个人电脑上都有一个完整的版本库，那多个人如何协作呢?比方说你在自己电脑上改了文件A，你的同事也在他的电脑上改了文件A，这时，你们俩之间只需把各自的修改推送给对方，就可以互相看到对方的修改了。最大的优势是安全性，因为每个人电脑上都有一个完整的版本库。

Git操作(1)　基础命令

1. **git init**：　新建版本库（.git文件就是当前工作区的版本库，此文件很重要)
2. **git status**：查看当前仓库状态
3. **git add <file>**: 添加文件到版本库
4. **git commit -m “有意义的说明文字”**：　提交文件到仓库
5. **git diff <file>**：查看文件的改动部分

Git操作(2)　版本回退

在git中，每一次commit可以看做一个“快照”，可以从某个快照来恢复。

1. **git log [—pretty=online]**：　查看历史记录(pretty前面是两条短横线)
2. **git reset —hard HEAD^**　（**git reset —hard <commit-ID>**）：回退到某一个版本(注：HEAD表示当前版本，HEAD^表示上一个版本，HEAD^^表示前两个版本，…，HEAD~100表示前100个版本)
(注：HEAD内部是一个指针，每次提交，相当于在一个链表中新增一个节点，同时把指针移到对应的节点处，所以版本修改实际上是在移动指针，速度很快！)
3. **git reflog**: 回退到旧版本，后悔了，又想回到新版本？通过该命令来查看操作记录，其中包含了每次的commitID，跳到指定CommitID处就行咯

上述命令都是针对提交(commit)之后的操作，下面来看看已经修改但未添加(add)和提交(commit)之前的撤销操作。

Git操作(3)　管理修改与撤销修改

1. **git checkout — <file>** : 修改了文件但未add和commit时，撤销修改
2. **git reset HEAD <file>**: 已经add了，但未commit时，撤销修改,使用HEAD表示退到最新版本，此时退到了1(暂存区已经clean，但工作区有改动)，再使用**git checkout — <file>**来撤销工作区的修改

工作区、暂存区、分支：　执行查阅相关资料

Git操作(4)　删除文件

新建一个新文件，add并commit，然后在本地磁盘删除了新文件

此时你有2个选择：
(一) 确实要从版本库中把该文件删除，需要执行以下2步：
1. **git rm <file>**: 从版本库删除文件
2. **git commit** : 提交修改，使删除操作生效

(二) 磁盘上删除了，需要从版本库恢复
1. **git checkout — <file>**

Git操作(5)　远程仓库

Git实际使用情况，找一台电脑充当服务器的角色，每天24小时开机，其他每个人都从这个“服务器”仓库克隆一份到自己的电脑上，并且各自把各自的提交推送到服务器仓库里，也从服务器仓库中拉取别人的提交。

使用github：　注册github账号；使用自己的邮箱名来生成ssh key；登录github，在账户设置中添加生成的ssh public key。（步骤略）

1. 在github上新建一个远程仓库
2. **git remote add origin <远程仓库地址>** : 添加远程仓库(origin是git默认的叫法，一看就知道是远程库，可以修改但一般不做修改)
3. **git push -u origin master**: 推送本地内容到远程仓库（由于远程库是空的，我们第一次推送master分支时，加上了-u参数，Git不但会把本地的master分支内容推送的远程新的master分支，还会把本地的master分支和远程的master分支关联起来，在以后的推送或者拉取时就可以简化命令）
4. **git clone <远程仓库地址>** : 从远程库克隆文件到本地

Git操作(6)　分支管理

1. **git branch**：查看当前所在分支
2. **git checkout -b <新分支>**：新建分支并切换到该分支，等价于：git branch <新分支>, 然后 git checkout <新分支>
3. **git checkout <新分支>**：切换到已存在的分支

HEAD指针，指向当前分支最新的一次commit，每commit一次，HEAD指针就后移一步，此时就造成master分支落后于开发分支,使用下面2个命令将开发分支的内容合并到主分支:

1. git checkout master
2. **git merge <当前开发分支>**: 合并分支（Fast-forward表示是快进模式，也就是把master指针直接指向开发分支的最新commit，所以速度很快）

以下两步是可选操作：

3. **git branch -d <当前开发分支>**：删除开发分支
4. **git push --delete origin <当前开发分支>**: 删除远程的分支

Git操作(7)　冲突解决

合并的过程往往不是一帆风顺的，通常会伴随着冲突！
准备工作，先切换到dev分支，修改文件并commit；然后切换到master，也修改同一个文件并commit，然后尝试在master上合并dev分支，此时会报冲突错误。

此时需要打开冲突文件，手工修改后，再合并提交！

git log --graph --pretty=oneline —abbrev-commit：　查看分支合并情况
git log —graph：　查看分支合并图

Git操作(8)　多人协作

1. **git remote [-v]** ：查看远程分支的信息（上面显示了可以抓取和推送的origin的地址。如果没有推送权限，就看不到push的地址。）
2. **git push origin <新分支>**: 向远程推送新分支
3. **git checkout -b dev origin/dev**：　创建远程分支到本地

假设别人修改了dev分支的内容并已经push到远程，你在push的时候可能失败，因为远程dev分支的内容比本地的要新。此时需要：

4. **git pull [origin dev]** : 抓取远程分支并试图与本地合并
如果合并出现冲突，则按冲突解决的办法先解决冲突，然后再提交。
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
5. **git branch --set-upstream branch-name origin/branch-name**：创建本地分支与远程分支的链接关系

Git操作(9)　标签管理

发布正式版本时，通常先打一个标签，这样，就唯一确定了打标签时刻的版本。将来无论什么时候，取某个标签的版本，就是把那个打标签的时刻的历史版本取出来。所以，标签也是版本库的一个快照。可以用来回退到某个线上版本。

1. **git tag <标签名称>** :打一个新标签
2. **git tag**: 查看当前标签

本应该周一打一个标签，忘了，现在周五了，怎么办?
使用 git log [—pretty=oneline] 找到周一的commit-ID，再在对应的id上打标签即可。
3. **git tag <标签名> <commit-id>** :在commit-id次提交处打标签
4. **git show <tagname>**: 查看标签信息（注意，标签不是按时间顺序列出，而是按字母排序的。）
5. **git tag -a v0.1 -m "version 0.1 released" 3628164**: 创建带说明文字的标签
6. **git tag -d <标签名>**: 删除本地标签
7. **git push origin <标签名>**:　推送某个标签到远程
8. **git push origin :refs/tags/<标签名>**: 删除远程标签(先删除本地， git tag -d XXX)

Git操作(10)　自定义git

1. 忽略某些文件

.gitignore文件

node_modules/
dist/
build/
…

2. 配置命令别名

git config —global alias.<自定义别名> <git命令>

例如:

git config —global alias.ci commit
git config —global alias.co checkout
git config —global alias.br branch