Git

作者：*杨旭东*

时间：*8/13/2017 11:54:16 AM *

Git的背景及其介绍

1. 出现背景：Linus在1991年创建了开源的Linux，从此，Linux系统不断发展，已经成为最大的服务器系统软件了。Linus虽然创建了Linux，但Linux的壮大是靠全世界热心的志愿者参与的，这么多人在世界各地为Linux编写代码，那Linux的代码是如何管理的呢？事实是，在2002年以前，世界各地的志愿者把源代码文件通过diff的方式发给Linus，然后由Linus本人通过手工方式合并代码！因为Linus坚定地反对CVS和SVN，这些集中式的版本控制系统不但速度慢，而且必须联网才能使用。有一些商用的版本控制系统，虽然比CVS、SVN好用，但那是付费的，和Linux的开源精神不符。 不过，到了2002年，Linux系统已经发展了十年了，代码库之大让Linus很难继续通过手工方式管理了，社区的弟兄们也对这种方式表达了强烈不满，于是Linus选择了一个商业的版本控制系统BitKeeper，BitKeeper的东家BitMover公司出于人道主义精神，授权Linux社区免费使用这个版本控制系统。 安定团结的大好局面在2005年就被打破了，原因是Linux社区牛人聚集，不免沾染了一些梁山好汉的江湖习气。开发Samba的Andrew试图破解BitKeeper的协议，被BitMover公司发现了，于是BitMover公司怒了，要收回Linux社区的免费使用权。 Linus可以向BitMover公司道个歉，保证以后严格管教弟兄们，嗯，这是不可能的。实际情况是这样的：Linus花了两周时间自己用C写了一个分布式版本控制系统，这就是Git！一个月之内，Linux系统的源码已经由Git管理了！牛是怎么定义的呢？大家可以体会一下。Git迅速成为最流行的分布式版本控制系统，尤其是2008年，GitHub网站上线了，它为开源项目免费提供Git存储，无数开源项目开始迁移至GitHub，包括jQuery，PHP，Ruby等等。

2. 介绍：Git作为分布式版本控制系统。分布式的系统避免了集中式需要联网才能工作的弊端，抛弃了集中式的‘中央控制器’，每个人的电脑就是一个完整的版本库。同时分布式的系统安全性高于集中式的系统，即使团队中一个人的系统当机，随便复制其他人的即可，而集中式的中央控制器如果当掉，所有人都没法干活。越来越多的优秀开源项目在GitHub上托管，组册用户超过350万用户，所以开发运用Git会使项目更加便于团队协作。GitHub的口号：**Build software better,together!**。

Git Bash操作命令（根据：实时的个人的操作习惯）

版本库创建及其基本操作

```
手动在本地建立文件夹，文件夹内右键Git Bash，打开shell操作控制台
//创建本地的版本库（repository)
$ git init
创建成功后会在文件中出现.git文件夹（系统默认是隐藏的，手动设定隐藏文件可见）
//命令查看.get文件
$ ls -ah
//设定本机的用户名和Email
其中--global代表全局设置，如果不加，单单指定了本版本库的设定
$ git config --global user.name "Your Name"
$ git config --global user.email "email@example.com"
//提交工作区某文件至暂存区
$ git add <文件>
//提交工作区所有文件至暂存区
$ git add -A
//提交暂存区的内容到当前分支
$ git  commit -m "本次提交的注解"
//查看当前分支的状态
$ git status
//查看文件内容
$ cat <文件>
//到修改了文件，查看修改了那些
$ git diff <文件>
//查看工作区和版本库中的不同
HEAD表示版本库中的当前版本
$ git diff HEAD --<文件>
//把暂存区的修改撤销掉，重新放回工作区
$ git reset HEAD file
//丢弃工作区的修改
还原一种是文件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；
```

```
一种是文件t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。
总之，就是让这个文件回到最近一次git commit或git add时的状态。
$ git checkout --<文件>
//要从版本库中删除该文件，那就用命令git rm删掉。1.在工作区手动删除掉（或rm <文件>），2git rm <文件> ，3.git commit
**如果删错了，也可以通过git checkout --<文件>回到上一个版本
$ git rm <文件>
```

补充：
1. HEAD指向的版本就是当前版本，因此，Git允许我们在版本的历史之间穿梭，使用命令git reset --hard commit_id。 2. 用git log可以查看提交历史，以便确定要回退到哪个版本。 3. 要重返未来，用git reflog查看命令历史，以便确定要回到未来的哪个版本。

分支管理

- 作用：假设你准备开发一个新功能，但是需要两周才能完成，第一周你写了50%的代码，如果立刻提交，由于代码还没写完，不完整的代码库会导致别人不能干活了。如果等代码全部写完再一次提交，又存在丢失每天进度的巨大风险。现在有了分支，就不用怕了。你创建了一个属于你自己的分支，别人看不到，还继续在原来的分支上正常工作，而你在自己的分支上干活，想提交就提交，直到开发完毕后，再一次性合并到原来的分支上，这样，既安全，又不影响别人工作。

```
//查看分支
后面添加 '-a' 表示查看所有分支（本地分支及其远程库关联的分支）；'-r'表示查看远程库中的分支
$ git branch
//创建分支
$ git branch <分支名>
//切换分支
$ git checkout <分支名>
//创建+切换分支
$ git checkout -b <分支名>
//合并某分支到当前分支
合并前先切换到合并的分支（merge和rebase方式的合并区别查看另一个文档[./GitAndGithub.md](./GitAndGithub.md "git和GitHub"))
$ git merge <被合并的分支名>
//删除分支
$ git branch -d
```

！！！！！分支冲突（远程仓库和本地版本库合并冲突性质一样）重要

我用远程分支和本地分支合并产生冲突去搭建场景，本地分支合并冲突类推。

- 场景: 当团队开发过程中，分支合并冲突难以避免。 合并冲突：A和B两个人同时进行团队开发项目，A上传文件到远程库，B此时也在上传文件，此时系统会中断后上传的一方，提示远程库超出当前分支一个提交（远程库中的当前文件和之前拉下来的**原文件**不匹配），所以需要在本地进行手动整合，在进行add，commit，push等操作！！！（系统提供的merge或者rebase是针对远程库中的当前文件和之前拉下来的**原文件**一致的前提下才整合的）。

```
//参看日志（会显示整合日志）
$ git log --graph --pretty=oneline --abbrev-commit
```

合并分支时，

```
//加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并。
默认是快速整合方式
$ git merge --no-ff -m "commit信息" <被合并的分支名>
```

补充： 其实 很多时候，程序员不需要刻意关注合并的方式[rebase还是merge(--no-ff或者fast forward)],更多时候遇到的事分支的冲突，需要我们手动何必，如果用到系统何必推荐merge --no-ff。至此我们能在log中看到。

当前分支的存储

- 场景:当工作进行了一半，没办法**提交**还需要做其他工作，可以用到stash语句去存储当前的工作现场，去做其他的事，回来再恢复接着完成余下工作。

```
//将当前的分支存储（"冷藏"）起来
$ git stash
//切换到原分支查看被存储的工作现场
```

```
$ git stash list
恢复现场一是用git stash apply恢复，但是恢复后，stash内容并不删除，你需要用git stash drop来删除；
另一种方式是用git stash pop，恢复的同时把stash内容也删了：
//恢复制定的工作现场
$ git stash list
//恢复第一个工作现场
$ git stash pop
//当某分支提交了事务，但是想删除该分支。使用git branch -d <分支名>是删不掉的,用以下语句删除：
$ git branch -D <分支名>
```

标签管理

- tag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。方便去管理。默认标签是打在最新提交的commit上的。

```
//添加一个标签
$ git tag <标签名>
//给制定的commitId打标签
$ git tag <标签名>
//查看标签（标签的排序是按字母排序的）
$ git tag
//显示标签下的详细信息
$ git show <标签名>
//给标签加说明信息
$ git tag -a <标签名> <-m "说明信息">
//删除指定标签
$ git tag -d <标签名>
//推送某个标签到远程，使用命令
$git push origin
//一次性推送全部尚未推送到远程的本地标签
$ git push origin --tags
//删除远程的标签
1. 先删除本地的标签
2. 再删除远程的标签
$ git push <远程名> : refs/tags/<删除的标签名>
```

忽略特殊文件

有时候，你必须把某些文件放到Git工作目录中，但又不能提交它们，比如保存了数据库密码的配置文件等等，每次git status都会显示Untracked files ...，Git考虑到了大家的感受，这个问题解决起来也很简单，在Git工作区的**根目录**下创建一个特殊的.gitignore文件，然后把要忽略的文件名填进去，Git就会自动忽略这些文件。不需要从头写.gitignore文件，GitHub已经为我们准备了各种配置文件，只需要组合一下就可以使用了。所有配置文件可以直接在线浏览：https://github.com/github/gitignore

```
//创建.gitignore文件(也可以手动创建)
$ touch .gitignore
```

- 配置语法：

```
1. 以斜杠“/”开头表示目录；
2. 以星号“*”通配多个字符；
3. 以问号“?”通配单个字符；
4. 以方括号“[]”包含单个字符的匹配列表；
5. 以叹号“!”表示不忽略(跟踪)匹配到的文件或目录；
```

此外，git 对于 .ignore 配置文件是按行从上到下进行规则匹配的，意味着如果前面的规则匹配的范围更大，则后面的规则将不会生效；

- 示例：
  - 规则： fd1/*
    - 说明：忽略目录 fd1 下的全部内容；注意，不管是根目录下的 /fd1/ 目录，还是某个子目录 /child/fd1/ 目录，都会被忽略；
  - 规则：/fd1/*
    - 说明：忽略根目录下的 /fd1/ 目录的全部内容；
  - 规则：
    - /*
      !.gitignore

!/fw/bin/
!/fw/sf/

- 说明：忽略全部内容，但是不忽略 .gitignore 文件、根目录下的 /fw/bin/ 和 /fw/sf/ 目录；

总结

我列举一些我觉得比较用的上的命令，不常用的我就没有整理。里面很多我混入了我的理解，我都是思考再三下的笔，但是毕竟我也是一个行者，也是路上奔波的人，所以难免我也会有错，如果那里理解有错希望大家指出，共同进步。我个人感觉Git就是开源的管理，我们也应该秉着一颗开源的心去学习工作，所以构建Git服务器我就没有整理，你们要是感兴趣自己去学习一下。这里我列出我整理参考的博客：

> 主参考：廖雪峰的官方Git教程 https://www.liaoxuefeng.com/wiki/001373951630592960 6dd18361248578c67b8067c8c017b000
> rebase和merge的区别：http://blog.csdn.net/wh_19910525/article/details/7554489
> .gitignore文件的部署： http://blog.csdn.net/kongjiea/article/details/52412634
> merge中fast_fast-forward和--no-ff的区别：http://blog.csdn.net/pzhtpf/article/details/52151320

--->End(完)