目录

[一.Git简介 2](#_Toc470177208)

[1.1 Git 术语 3](#_Toc470177209)

[二.Git安装 3](#_Toc470177210)

[三.Git常用配置 4](#_Toc470177211)

[四.Git常用的三种操作 4](#_Toc470177212)

[五.Git删除 4](#_Toc470177213)

[六.Git版本回退 5](#_Toc470177214)

[七.文件比较差异 5](#_Toc470177215)

[八.Tag 6](#_Toc470177216)

[九.远程 7](#_Toc470177217)

[十.从远程拉取内容，提交内容到远程 7](#_Toc470177218)

[十一.暂存管理 7](#_Toc470177219)

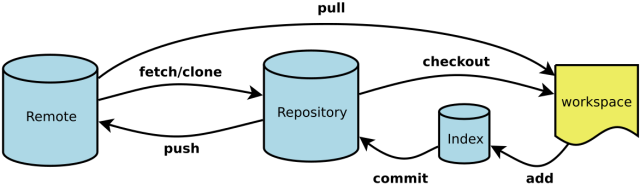
# ****一.Git简介****

Git的诞生确实是一个有趣的故事，我们知道，当年Linus创建了开源的Linux，从此，Linux系统不断发展，现在已经成为最大的服务器系统软件了(请不要傻傻分不清Linus和Linux)。

但是随着Linux的不断壮大，就需要各种版本控制了，起初Linus带着他的小弟们使用的是BitKeeper(商业版本控制系统),之后呢由于某种原因BitKeeper的公司不让他们使用了，于是Linus自己花了两周时间写出了Git并且开源了(BitKeeper已哭晕在厕所)，阿弥陀佛，幸亏BitKeeper不让Linus他们用了，要不然我们现在也不会有这么好用的Git了，博主更不会在这写这篇博文了。

之后的岁月里，渐渐有了github,coding等一些可以使用git存储的网站，Git的江湖地位变得无可替代了，如果你是个开发者却还不会使用Git那就太out了。

这里先引用一张图解释Git工作原理：

**Workspace:**工作区，执行 git add \* 命令就把改动提交到了暂存区，执行 git pull 命令将远程仓库的数据拉到当前分支并合并，执行 git checkout [branch-name] 切换分支  
**Index:**暂存区，执行 git commit -m '说明' 命令就把改动提交到了仓库区（当前分支）  
**Repository:**仓库区（或本地仓库），执行 git push origin master 提交到远程仓库，执行 git clone 地址将克隆远程仓库到本地  
**Remote:**远程仓库，就是类似github，coding等网站所提供的仓库**注：**实际操作命令和上述命令会有所不同，这里这是解释清楚命令和仓库的关系。

## ****1.1 Git 术语****

**仓库（Repository）**一个仓库包括了所有的版本信息、所有的分支和标记信息。在Git中仓库的每份拷贝都是完整的。仓库让你可以从中取得你的工作副本。   
**分支（Branches）** 一个分支意味着一个独立的、拥有自己历史信息的代码线（code line）。你可以从已有的代码中生成一个新的分支，这个分支与剩余的分支完全独立。默认的分支往往是叫master。用户可以选择一个分支，选择一个分支执行命令git checkout branch.  
**标记（Tags）**一个标记指的是某个分支某个特定时间点的状态。通过标记，可以很方便的切换到标记时的状态，例如2009年1月25号在testing分支上的代码状态   
**提交（Commit）** 提交代码后，仓库会创建一个新的版本。这个版本可以在后续被重新获得。每次提交都包括作者和提交者，作者和提交者可以是不同的人

# ****二.Git安装****

Git-for-window：https://git-for-windows.github.io/index.html

# 三.Git常用配置

--system #系统级别

--global #用户全局

--local #单独一个项目

**git config --global user.name** "xxxx" #用户名

**git config --global user.email** "xxxx@xxx.com" #邮箱

git config --global core.editor vim #编辑器  
**git config --global alias.st status** #按这种方法，配置别名

git config -l #列举所有配置

# 四.Git常用的三种操作

#将工作区的修改提交到暂存区

**git add** <file>

**git add** .

#------------------------------------------

#将暂存区的内容提交到版本库

**git commit** <file>

**git commit** .

**git commit -a** #包括git add/ git rm /git commint 这三个操作，所有一般在操作工作区的时候，直接删除了文件，而不是使用git rm的，最后提交是可以用这个，如下  
  
#git commit -am "提交信息"

# 抛弃工作区修改(使用当前暂存区的内容状态去覆盖工作区，从而达到抛弃工作区修改的作用)

**git checkout** <file>

**git checkout** .

五.Git删除  
#删除文件的几种方法（貌似Git2.0后有了变化）

#第一种直接在工作区删除

**rm your\_file** #直接在工作区删除文件

git add -u . #将有改动的都提交到暂存区（包括修改的，删除的等操作），貌似git2.0 不加 -u 参数也可以

**git commint -m** "message" #提交版本库  
#第二种方法直接在工作区删除

rm your\_file #直接在工作区删除文件

git commit -am "message" #这个在前面提过，直接可以提交版本库，-a会包括包括git add/ git rm /git commint 这三个操作  
#第三种方法使用git rm

**git rm** <file> #不仅在工作区将文件删除，同时将该删除操作提交到暂存区

git commint -m "message" #提交版本库  
#关于git rm的其他补充

**git rm --cached** <file> #从暂存区中除去该文件，git将不再跟踪该文件的变更，但仍然在工作区内，在需要.gitignore时经常用到

# 六.Git版本回退

#------------------------------------------

#改变暂存区的修改（其实是重置HEAD，将指定版本库的内容状态去覆盖暂存区，从而达到暂存区的改变）

**git reset** <file> #从暂存区恢复到工作区（不指定版本id，则默认为最后一次提交的版本id）

**git reset .** #从暂存区恢复到工作区

**git reset $id** # 恢复到指定的提交版本，该$id之后的版本提交都恢复到工作区

**git reset --hard $id** #恢复到指定的提交版本，该$id之后的版本提交全部会被抛弃，将不出现在工作区

#注：如果不小心使用了错误的HEAD重置，会发现HEAD指向了重置的版本id，该版本之后的版本提交都不见了，使用git log也无法找到，那么怎么恢复呢？使用下面两个命令

**git reflog**  #会显示所有的版本记录

**git reset --hard $id** #重新重置，至于--hard，请根据你时候将改变的内容放到工作区还是直接抛弃进行选择

#------------------------------------------

#恢复某次提交（其实是某提提交的回滚操作，不影响其他的提交，所产生的效果创建一个新版本提交去回滚将指定的提交删除，包括产生的差异文件不会出现在工作区，而是直接被抛弃）

git revert <$id>

git revert HEAD

#这里有一个很好的讲解revert与reset的差异：git reset 是把HEAD向后移动了一下，而git revert是HEAD继续前进，只是新的commit的内容和要revert的内容正好相反，能够抵消要被revert的内容。

# 七.文件比较差异

**git diff**

**git diff** <file> #比较工作区与暂存区文件的差异

**git diff --cached** # 比较暂存区和版本库差异

**git diff <$id1> <$id2>** # 比较两次提交之间的差异

**git diff <branch1>..<branch2>** # 在两个分支之间比较

分支

**git branch -r** #查看远程分支

**git branch new\_branch\_name** #新建一个分支

**git branch --merged** #查看已经被合并到当前分支的分支

**git branch --no-merged** #查看未被合并到当前分支的分支

**git checkout branch\_name** #切换分支

**git checkout -b branch\_name** #创建分支并切换

**git branch -d branch\_nam**e #删除分支

**git branch -D branch\_name** #强制删除分支

git push origin :branch-name #删除远程分支（先在本地删除该分支），原理是把一个空分支push到server上，相当于删除该分支。

# 八.Tag

**git tag v1.0.0** [SHA]

#打一个轻量级的tag，只是一个commit的指向引用,[SHA]是可选择值（某个commit的SHA），指定为哪个commit打tag，如果没写则直接为最后一个commit打tag

**git tag -a v1.0.0 -m** "你的附注信息" [SHA]

#一个带附注信息的tag，不是一个简单的引用，而是单独的一个对象，[SHA]是可选择值（某个commit的SHA），指定为哪个commit打tag，如果没写则直接为最后一个commit打tag

**git tag** #列出所有的tag

**git show v1.0.0** #打印指定tag的信息

**git tag -d v1.0.0** #删除本地指定tag

**git push origin :refs/tags/v1.0.0** #删除远程tag

# 九.远程

**git remote -v** # 查看远程服务器地址和仓库名称

**git remote show origin** # 查看远程服务器仓库状态

**git remote add origin** git@github:robbin/robbin\_site.git # 添加远程仓库地址

git remote set-url origin git@github.com:robbin/robbin #修改远程地址

git remote rm #删除远程创库地址

# 十.从远程拉取内容，提交内容到远程

**git pull** #=git fetch + git merge

git fetch #拉取

**git merge** #合并

**git push** # push所有分支

git push origin master # 将本地主分支推到远程主分支

**git push -u origin master** # 将本地主分支推到远程(如无远程主分支则创建，用于初始化远程仓库)

git push origin <local\_branch> # 创建远程分支， origin是远程仓库名

git push origin <local\_branch>:<remote\_branch> # 创建远程分支

git push origin :<remote\_branch> #先删除本地分支(git br -d <branch>)，然后再push删除远程分支

# 十一.暂存管理

**git stash** #将工作区做的修改暂存到一个git栈中

git stash list #查看栈中所有暂存

git stash apply <暂存编号> #回复对应编号暂存到工作区，如果不指定编号为栈顶的，注意：这些暂存还在栈中

**git stash pop** #将栈顶的暂存，恢复到工作区，并从栈中弹出

git stash clear #清空暂存栈