## Git简介

#### 集中式版本控制系统：

版本库是集中存放在“中央服务器”。电脑每天要从中央服务器获取最新的版本，然后开始干活，干完了，再把自己的活推送给“中央服务器”。

#### 分布式版本控制系统

根本没有“中央服务器”，每一个参与的开发者的电脑上都有一个完整的版本库。多个人写作的时候只需要把各自的修改推送给对方，就可以互相看到对方的修改了。

和集中式比起来，分布式的安全性要高很多，因为每个人电脑里都有完整的版本库。分布式的最大毛病就是必须联网才能工作。

当然，Git最强大的还是它的分支管理。

## 2获取与创建项目命令

#### 2.1创建版本库

版本库又名仓库repository.可以简单理解为一个目录，这个目录里面的所有文件都可以被Git管理起来，每个文件的修改、删除，Git都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以还原

### git init 将这个目录变成Git可以管理的仓库

在目录中执行git init,就可以创建一个Git仓库

例如创建runoob项目：

$ mkdir runoob

$ cd runoob/

$ git init

#### Git add把文件添加到仓库

Git add 文件前出来的状态是红色的，git add文件后，文件进入暂存区出来的状态是绿色的，文件放入暂存区后执行命令 git commit –m “修改描述”

#### Git commit把文件提交到仓库

$ git commit –m “This is my First commit”

Git commit命令后面的-m后面输入的是本次提交的说明，在项目中必须输入！输入的信息为有意义的！如果在commit的时候写message写错了，使用$ git commit –amed可以修改最近一次提交的message.

Git 为你的每一个提交都记录你的名字与电子邮箱地址，所以第一步需要配置用户名和邮箱地址

### Git clone 拷贝

Git clone [url]

例如克隆github上的项目

$ git clone [git@github.com:schacon/simplegit.git](mailto:git@github.com:schacon/simplegit.git)

Cloning into ‘simplegit’…

克隆完成后，在当前目录下会生成一个simplegit目录

### Git status查看项目的当前状态

$ touch README

$ touch hello.php

$ ls

README        hello.php

$ git status -s

?? README

?? hello.php

$ git add README hello.php

添加完成，再执行git status,就可以看到这两个文件已经加上去了

$ git status –s

A README

A hello.php

修改README文件 $ vim README，修改保存退出，再执行git status时，README前会有AM标志，“AM”状态的意思是，这个文件在我们将它添加到缓存之后又有改动，改动后我们再执行git add命令将其添加到缓存中

Git status 以查看在你上次提交之后是否有修改

加-s参数，获得简短的结果输出，如果没加会详细输出内容

#### Git diff 查看文件修改的内容

git diff 命令显示已写入缓存与已修改但尚未写入缓存的改动的区别。git diff 有两个主要的应用场景。

尚未缓存的改动：git diff

查看已缓存的改动：git diff –cached

查看已缓存的与未缓存的所有改动：git diff HEAD

显示摘要而非整个diff:git diff –stat

Git status显示你上次提交更新后的更改或者写入缓存的改动，而git diff一行一行地显示这些改动具体是啥。

#### Git log查看提交日志

$ git log –pretty=oneline逐条显示从最近到最远的提交日志

#### 如何回退版本？

。在 Git 中，使用 HEAD 表示当前的版 本，也就是最新提交的版 本，上一个版本为 HEAD^，在上一个则在加一个 ^ 。那前100个版本呢。则用 HEAD~100 来表示。HEAD 就像一个指针，总是指向当前的版本。 要从现一个版本回退到上一个版本。使用如下命令**回到过去**。

$ gir reset –hard HEAD^

从过去回到未来：只要你知道那个未来版本的commit id即可回去，如果不记得，可以用$ git reflog查看commit的版本号

$ git reflog

5c21c1a HEAD@{0}: reset: moving to HEAD^

3958ac1 HEAD@{1}: commit (amend): append GPL

**回到未来**

$ git reset –hard 3958ac1

#### 2.2 Git中的工作区和暂存区

工作区：其实就是电脑里能看到的目录。 版本库：工作区中有一个隐藏目录 .git，这个不算工作区，而是Git的版本库。Git版本库 中存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自 动创建的第一个分支master，以及指向master的一个指针（HEAD） 上面说的两步： 第一步：git add 就是把文件修改添加到暂存区。 第二步：git commit 提交更改，实际就是把暂存区的所有内容提交到当前分支

### 3 Git如何创建并管理分支

#### 3.1 什么是分支管理

在Git中，创建、切换和删除分支，Git都在1秒钟之内完成，无论版本库中有多少文件。在Git中，每次提交，Git都把他们串成一条线，这条时间线就是一个分支。初始创建的分支为master分支。HEAD严格来说不是指向提交，而是指向master而master才是指向提交的，所以HEAD指向的是当前分支

当我们创建新的分支的时候，例如分支的名字叫 A，Git就新建了一个指针叫 A，指 向 master相同的提交，再把 HEAD指向 A，就表示当前分支在 A上了。从现在开始， 对工作区的提交和修改就是针对 A分支了！！！！ A指针往前移动一步，而 master指 针是不变的。 那如何合并分支呢？最简单的方法就是直接把 master指向 A的当前提交，就完成了合 并。

$ git branch A

$ git checkout A

Switched to branch ‘master’//可以看到A分支上修改的内容并没有显示在master上

$ git branch //查看所有分支和当前自己所在的分支

$ git merge A//合并指定分支到当前的分支

Fast-forward//合并方式：快进模式，不能每次都这么粗暴，还有其他合并方式

$ git branch –d A//删除分支

Git checkout –b A这个命令相当于两个命令 git branch A、git checkout A

#### 3.2 如何解决冲突

在合并分支的时候，如果有冲突的话，git会告诉我们：CONFLICT:…在此之后可通过git diff来查看冲突的文件。然后打开冲突文件，开始解决冲突。Git通过《《《、====和》》》》来标识不同分支的内容。修改后再次提交即可。

#### BUG分支

当遇到紧急bug的时候可以在对应分支的基础上再建一个分支（BUG分支）来进行紧急BUG的修改，在改完BUG后，可以合并到对应的分支然后删除BUG分支，重新开始之前的开发，之前开发的进程我们可以用stash功能把现场储藏起来

$ git stash//保存当前分支上工作区的状态

//方法一恢复工作区（恢复工作区但stash中的内容并不删除）

$ git stash apply

$ git stash drop(删除stash中的内容)

//方法二恢复工作区（恢复工作区并删除）

$ git stash pop

#### Feature分支

用于添加新功能的分支，在不影响主分支的情况下，添加新的功能

删除分支：

$ git branch –d branchName //删除合并了的分支

$ git branch –D branchName//删除没有合并的分支。强行删除

### 4.Git管理的是修改，而非文件

为什么 Git 比其他版本控制系统设计的优秀，因为 Git 跟踪并管理的是修改，而非文件。那 何为修改呢？新增或者删除一行，一个字符，这些就是修改。 每一次修改了文件，都需要 git add 到暂存区。如果没有的话，则无法 commit 上去。

#### 4.1 如何撤销修改

场景一：在工作区修改了某个文件的内容，在没有 git add 之前，使用命令 git checkout – filename 即可撤销修改

场景二：工作区修改了某个文件的内容，还添加到了暂存区时，想丢弃修改

$ git reset HEAD filename

//这样就回到 场景一

$ git checkout –filename

场景三：工作区修改了，添加到缓存区然后还commit到版本仓库了，就需要版本回退了

#### 4.2 如何删除文件

先本地：在工作区直接删除文件对暂存区、版本库没有任何改变，它们的文件依旧存在

$ rm README.md

再仓库：git rm删除文件会影响到工作区以及暂存区，提交以后会直接影响到版本库，但不影响提交以前版本库的目录结构。

$ git rm README.md

$ git commit -m "remove README.md"

//可以通过如下命令查看之前被删除的文件在删除前的详情

$ git ls-files --with-tree=HEAD^

$ git cat-file -p HEAD^:README.md

再远程：

$ git push origin master//origin表示远端 master表示分支

如果删除错了，因为版本库里还有，使用如下命令恢复最新版本（会丢失最近一次提交修改 的内容）

$ git checkout -- README.md

#### 4.3 如何删除一个目录呢？

$ git rm -r directory

$ git commit -m "remove directory"

$ git push orgin master

### 5. Git中的标签管理

#### 5.1 如何创建标签

首先切换要打标签的分支上。然后使用如下命令打标签

$ git branch

$ git checkout master

$ git tag v1.0

$ git tag

默认情况下，Tag是打在最新提交的commit上。如果想在之前的commit上打标签：

$ git log –pretty=oneline –abbrev-commit

标签不是按时间顺序列出的，而是按字符排序的。可以使用git show tagName来查看标签的信息

5.2 打标签是常用的几个命令

$ git tag tagName //在最新的commit上打标签

$ git tag tagName commitId //在指定的commit上打标签

$ git tag –a v0.1 –m “verson 0.1 released” 3628164 //在指定的commit上打标签并附上说明信息

$ git tag –s v0.2 –m “signed version 0.2 release” fec145a//用PGP签名标签

$ git push origin v0.1 //推送指定标签

$ git push origin –tags//推送所有未推送的标签

#### 5.2 如何操作标签

创建的标签只存储在本地，不会自动推送到远程。在本地可安全删除打错的标签

$ git tag –d v0.1

如果标签已经推送到远端了，可以使用如下命令

$ git push –d v0.9

$ git push origin :rerfs/tags/v0.9

To git@github.com:michaelliao/learngit.git

- [deleted] v0.9

### 6. GIT与远端仓库

自己创建一个工程，使用git进行版本管理。上传在github之前，需要对上传的文件做过滤， 因为在工程实现过程中，会生成一些中间文件，或者在项目中的部分文件是不需要进行版本 管理的。对于这些文件对于gitub来讲应该是透明的。这个时候就需要在版管理的根目录下 （与 .git 文件夹同级）创建一个 .gitignore 文件。

$ git init

$ touch .gitignore

$ gvim .gitignore

这样就创建并开始编辑了一个.gitgnore文件

#### Gitignore中的格式规范

表示注释：#

使用！取反，标识不忽略这些文件

可以使用shell所使用的正则表达式来进行模式匹配

远程仓库的使用

本地新建一个仓库，如何与github上的仓库关联起来

在 github 上创建一个空的仓库（创建完成后会有提示） 然后在本地 git remote add origin <github-repository-url>和 git push -u origin maste

已经在github或者gitlab上有仓库了

直接git clone即可

#### 查看远程仓库

1. $ git remote
2. Origin
3. $ git remote -v
4. origin http://gitlab.thunisoft.com/FY\_PRD\_shuzifayuan/40\_source.git (fetch)
5. origin http://gitlab.thunisoft.com/FY\_PRD\_shuzifayuan/40\_source.git (push)

git remote –v显示需要读写远程仓库使用的GIT保存的简写和它对应的URL

使用如下命令查看一个远程仓库的更对信息： git remote show [remote-name]

它同样会列出远程仓库的 URL 与跟踪分支的信息。 这些信息非常有用，它告诉你正处于 master 分支，并且如果运行 git pull，就会抓取所有的远程引用，然后将远程 master 分支合 并到本地 master 分支。 它也会列出拉取到的所有远程引用

#### 添加远程仓库

运行如下命令可以添加一个新的远程GIT仓库（统一个项目），同时指定一个可以轻松引用的简写

$ git remote add pb https://github.com/paulboone/ticgit

$ git remote –v

#### 从远程仓库中fetch和pull

$ git fetch [remote-name]

$ git pull [remote-name]

区别：fetch会访问远程仓库，从中拉去所有你还没有的数据。执行完成后，你将会拥有那个远程仓库中所有分支的引用，可以随时合并或查看。

如果使用clone命令克隆一个仓库，命令会自动将其添加为远程仓库并默认以origin为简写，所以，git fetch origin 会抓取克隆（或上一次抓取）后新推送的所有工作。必须注意 git fetch命令会将数据拉取到你的本地仓库-它并不会自动合并或修改你当前的工作，当你准备好时必须手动将其合并入你的工作。

Pull命令会抓取然后合并远程分支到当前分支

可以理解为pull=fetch+merge

#### 推送到远程仓库

Git push [remote-name] [branch-name]

$ git push origin master