Git服务器的搭建：在linux centos上搭建git服务器

1. git分支

几乎所有的版本控制系统都支持分支，意味着可以把工作从开发主线上分离出来，以免影响开发主线。

Git的分支是轻量级的，创建分支和切换分支几乎是瞬间完成的，git鼓励频繁的使用与合并分支，在进行commit操作时，git会保存一个提交对象，该对象包含一个指向暂存内容快照的指针，包含指向父commit对象的指针，以及提交者的名字，提交信息等。首次提交的对象没有父对象，普通提交操作产生的提交对象只有一个父对象，合并操作产生的提交对象有多个父对象。

暂存操作（add）会计算每一个文件的校验和（SHA-1），然后将当前版本的文件快照保存在git仓库中，git使用blob对象保存，最终将校验和加入到暂存区域等待提交。

一个项目有一个跟目录，其对应的tree对象的校验和是y46r5，根目录下有三个子目录（tree对象）每一个目录下的文件对应一个blob对象，

分支创建本质上是创建了一个指向提交（commit）对象的指针，

使**用git branch 分支名 创建一个分支，**创建一个指向当前提交对象的分支。该命令只是**创建了一个分支，但没有切换到该分支，通过head指针，head指针指向当前的本地分支（可以将其看作是当前分支的别名**）

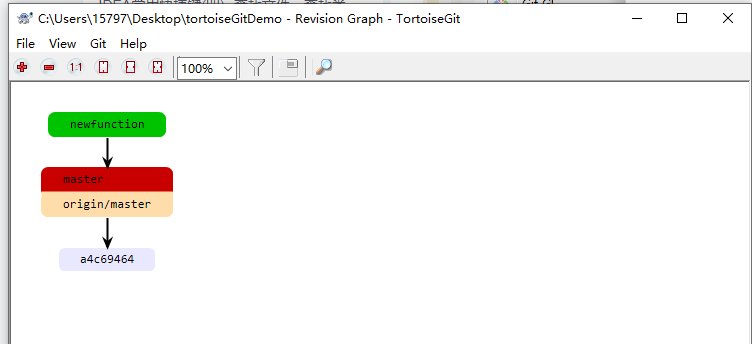
**Git log --decorate查看各分支当前所指对象**

（3）分支切换 git checkout 分支名 切换分支，本质上是将head指针指向branch-name指向的分支

如果执行 git checkout master 会执行两件事：（1）将head指向master （2）将工作去恢复为master所指向的快照内容。

TortoiseGit查看版本分支图

右键--》TortoiseGit--》Revision graph



红色部分代表这本地当前分支。比如本次分支代码将要发布。选择Master分支->右键->与工作副本对比。

Git创建**开发分支**的命令：

Git checkout -b develop master

将开发分支发布到master分支的命令：

首先要切换回master分支：

#切换回master

Git checkout master

#对develop分支进行合并

Git merge --no--ff develop

#参数--no--ff 代表不使用快速合并 ff:fast-forword

Fast-forword：表示在合并的时候直接将master分支指向develop分支

<http://www.ruanyifeng.com/blog/2012/07/git.html>

#使用--no--ff表示正常合并，在master分支上生成一个新节点，为了保证版本演进清晰，建议使用非快进式

**临时性分支:**功能分支（feature分支）；预发布分支（release）修补分支（fix-bug）

Bug-fix

(1)功能分支：是为了开发特定的功能，从develop分支上面分出来的，开发完成后，在并入develop分支

创建一个功能分支：git checkout -b feature-x develop

开发完成后将功能分支merge到develop分支

Git check develop #切换回开发分支

Git merge --no--ff feature-x

(2)预发布分支

预发布分支是指发布正式版本之前（及合并到master分支之前），可能需要有一个预发布的版本进行测试。

预发布分支是从develop分支上面分出来的，预发布结束后，必须合并进develop和master分支，

创建一个预发布分支：git checkout -b release-1.2 develop

确定没有问题后，合并到master分支：

Git checkout master

Git merge --no--ff release-1.2

#对合并的新节点，做一个标签

Git tag -a 1.2

在合并到develop分支：

Git checkout develop

Git merge --no--ff release-1.2

（3）修补bug分支

修补分支，软件正式发布之后，难免会出现bug，这时就需要创建一个分支，进行bug修补。修补bug分支是从master分支上面分出来的，修补结束后，在合并进master和develop分支

创建一个修补分支：git checkout -b fixbug-0.1 master

修补结束后，合并到master分支

Git checkout master

Git merge --no--ff fixbug-0.1

Git tag -a 0.1.1

在合并到develop分支：

Git checkout develop

Git merge --no--ff fixbug-0.1

最后删除bug分支：

Git branch -d fixbug-0.1

删除feature分支 git branch -d feature-x

总结：切换分支：git checkout master[分支名]

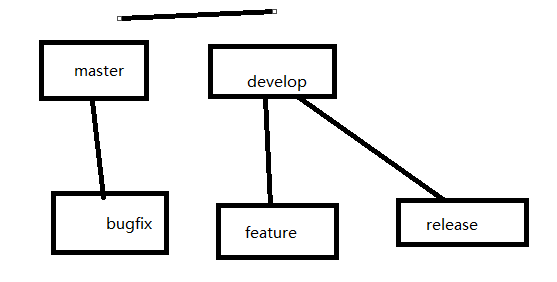
创建一个分支要指定是从哪里分出来的分支：

Git check -b 新分支名 父分支名

合并分支：git merge --no--ff 新分支名

删除分支 git branch -d 新分支名

不同的分支会给他们打上一个tag



总结：

Master分支和develop分支是并列走的

临时分支：fixbug分支是master上实际应用的时候发现的错误，需要将该分支合并到master和develop

Feature是从develop上出来的，合并到develop

Release是从develop上产生的，合并到master和develop分支上。

****本地分支，追踪分支和远程分支****

        这里有三个概念，本地分支就是我们可以通过git branch查看到的分支，也就是我们自己git仓库所拥有的分支，我们都可以利用。远程分支是对远程仓库的分支的索引，它其实也是本地分支，只是我们无法移动它，必须要在和中心服务器交互根据服务器更新到本地来的代码移动的，远程分支的作用就是我们上次和中心服务器交互更新得到的最新版本，它也是个指针。追踪分支比较难理解，它也是一个本地分支，只是它对应了一个远程分支，如果我们本地的某个分支对应了一个特定的远程分支，那么它就是追踪分支，比如我们最初的master分支就是一个追踪分支，它对应远程分支origin/master，这里origin是远程仓库名，当我们在master分支里执行更新(fetch，pull)或是推送(push)，在不指定分支的情况下，默认就是从origin/master分支更新来或者提交到origin/mster分支。

<https://blog.csdn.net/susu_xi/article/details/82025379>

更新远程代码到本地有两个命令，fetch和pull，fetch是将远程代码更新到本地，但是不会执行合并操作，需要自己查看，解决冲突什么的，然后自己再执行merge将更新来的代码合并到我们自己制定的分支，但是pull就将这两个操作合成了一步，直接更新服务器代码更新并合并到到本地指定分支，当然遇到冲突也必须要自己解决。所以我们一般都使用fetch来实现更新，虽然麻烦了点，但是不容易出问题。

将本地代码推送到远程仓库，也就是中心服务器，一般我们推送数据都是git push origin master:master，这里指定远程仓库名，本地分支名和远端分支，也就是将我们本地master分支的数据推送到远程仓库origin的master分支。如果本地的master分支是追踪分支，那么在不指定的情况下，它会自己找到远程仓库中对应的分支来推送数据。或者我们直接进行git push origin操作，只指定远程仓库名，那么git会根据我们目前所在分支和它所对应的远程仓库的分支来实现数据推送，前提是我们目前所在分支必须是追踪分支。当然如果是git push origin :master，这里本地分支名是空的，这个操作就是将空分支推送到远程仓库的master分支，结果就是将master分支删除。

既然追踪分支这么好用，那么我们怎么建立追踪分支呢，有两种方式，第一种方式是根绝远程分支创建追踪分支，如果不指定该追踪分支的名字，默认和远程仓库的分支名字一样：git checkout --track origin/test，这样我们就建立了一个名为test的追踪分支，如果重新指定追踪分支的名字：git checkout -b name origin/test，这样我们就创建了一个名为name的追踪分支，它对应远程仓库的test分支。第二种方式是已经存在某个本地分支，要让它来对应某个远程分支来成为追踪分支，也有两个命令可以用，git branch --set-upstream test origin/test 或者git branch -f --track test origin/test 这里我们就让我们本地已经存在的test分支来追踪远程的test分支。

通过 git ls-remote (remote) 来显式地获得远程引用的完整列表，

Git鼓励大量使用分支：

查看分支：git branch

创建分支：git branch <name>

切换分支：git checkout <name>

创建+切换分支：git checkout -b <name>

合并某分支到当前分支：git merge <name>

删除分支：git branch -d <name>

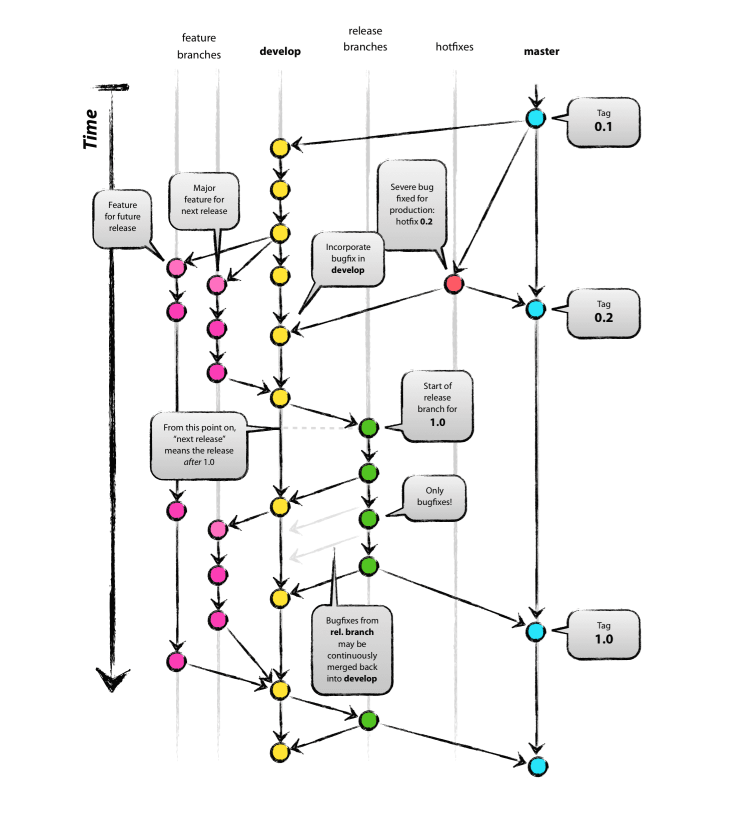
<https://www.runoob.com/git/git-install-setup.html>

<https://www.cnblogs.com/matengfei123/p/8252128.html>

<https://www.cnblogs.com/chucklu/p/4748394.html>

<https://www.cnblogs.com/wangwenhui/p/10595918.html>

<https://blog.csdn.net/qq_15037231/article/details/79137753>



https://nvie.com/posts/a-successful-git-branching-model/