**git功能整理**

# 属性配置

## 别名

### git config --global alias.co checkout

git checkout -> git co

## 连接远程仓库

### git remote add origin git@github.ocm:J0kerMh/xxx.git

添加一个远程仓库，地址是git@github.com:J0kerMh/xxx.git origin 是给这个项目的远程仓库起的名字，是的，名字你可以随便取，只不过大家公认的只有一个远程仓库时名字就是 origin ，为什么要给远程仓库取名字？因为我们可能一个项目有多个远程仓库？比如 GitHub 一个，比如公司一个，这样的话提交到不同的远程仓库就需要指定不同的仓库名字了。 git remot -v 可以查看当前项目有哪些远程仓库 如果直接git clone 的话就不用再添加了

# 分支管理

## 提交记录

### git commit

提交记录可以当作项目快照，Git 希望提交记录尽可能地轻量，因此在你每次进行提交时，它并不会盲目地复制整个目录。条件允许的情况下，它会将当前版本与仓库中的上一个版本进行对比，并把所有的差异打包到一起作为一个提交记录。

* git commit --amend

修改最近一次提交说明

## 创建分支

### git branch

git branch 是从当前节点创建分支，我们在协作开发时一般会从master上创建一个自己的分支，各自开发完后，最后在merge到master分支上

## 合并分支

### git merge

git merge 将两个分支合并，产生的结果包含两个合并分支所有差异的并集。

### git rebase

rebase的中文是垫底，git rebase 也是一种合并的方法，Rebase 实际上就是取出一系列的提交记录，“复制”它们，然后在另外一个地方逐个的放下去。Rebase 的优势就是可以创造更线性的提交历史，这听上去有些难以理解。如果只允许使用 Rebase 的话，代码库的提交历史将会变得异常清晰。

* git rebase -i

交互式操作 -i 是 --interface 的缩写 你可以在这个界面做： 调整提交记录的顺序（通过鼠标拖放来完成） 删除你不想要的提交（通过切换 pick 的状态来完成，关闭就意味着你不想要这个提交记录） 合并提交。

## 切换分支

### git checkout

切换分支，其实这里的操作在本质上是管理一个叫 HEAD的指针，他指向的位置，就是你当前所有操作的位置。它可以指向某一个单独的提交，不仅仅是你的分支。因为有 HEAD 这个指针，因此我们可以在提交树上方便的进行定位和查询。

### HEAD操作

HEAD 的基本索引符号是 ~^ ~ 指的是从HEAD向上移动， ~<num>移动num个提交 ^

## 强制修改分支位置

### git branch -f master HEAD~3

将 master 分支强制指向 HEAD 的第 3 级父提交。

## 撤销提交

### git reset

在reset后，之后 所做的变更还在，但是处于未加入暂存区状态。 git reset 相当于重置到某个提交 所以 命令为 git reset HEAD~ 是重置到上次提交

### git revert

git revert最大的特点是将撤销后的分支生成成一个新的分支，相当于这个新的分支所作的更改能够抵消掉你撤销的那个分支的更改 这个操作的作用是：分享 如果只是reset的话，那只有你本地知道，revert你就可以让其他人知道，你撤销了这个提交。

## 查看分支记录

### git log

查看提交记录的hash值 并且哈希值在真实的 Git 世界中也会更长（译者注：基于 SHA-1，共 40 位）。例如前一关的介绍中的提交记录的哈希值可能是 fed2da64c0efc5293610bdd892f82a58e8cbc5d8。舌头都快打结了吧... 比较令人欣慰的是，Git 对哈希的处理很智能。你只需要提供能够唯一标识提交记录的前几个字符即可。因此我可以仅输入fed2 而不是上面的一长串字符。

## 分支tag

因为平时的开发都是阶段性的，很容易会有一些提交，分支很容易被改变。 为了将某些重要的分支永久记录，我们用tag命令来完成这个操作。 这就像我们软件的各种版本，每次发布版本都是相当于加了一个新的tag

### git describe <ref>

描述离你最近的锚点（也就是标签） 例： git describe master

### git tag <tagName> <commit>

## 复制提交

### git cherry-pick

将某些提交复制到当前HEAD后

# 远程管理

远程仓库非常有用，主要包括以下几点 备份 分享 团队协作 这也是为什么当 B站后台代码上传到github之后，即便github再怎么进行封禁，这段代码也不可能从互联网上消失。

## 复制到本地

一般clone下来的项目的分支都是 o/master ，它一般是 HEAD分离状态，它的意思是你需要在别的地方开发。 为什么要有 o/ 远程分支的命名规范是 <remote name>/<branch name> 因此，如果你看到一个名为 o/master 的分支，那么这个分支就叫 master，远程仓库的名称就是 o。 大多数的开发人员会将它们主要的远程仓库命名为 origin，并不是 o。这是因为当你用 git clone 某个仓库时，Git 已经帮你把远程仓库的名称设置为 origin 了

### git clone

## 从远程仓库获取数据

### git fetch

git fetch的功能： 从远程仓库下载本地仓库中缺失的提交记录 更新远程分支指针(如 o/master) git fetch 并不会改变你本地仓库的状态。它不会更新你的 master 分支，也不会修改你磁盘上的文件 可以将 git fetch 简单理解成下载操作

## 更新本地仓库

### git pull

git pull 相当于两个命令 先是 git fetch 然后是 git merge

### git pull --rebase

fetch 和 rebase 的缩写

## 更新远程仓库

### git push <remote> <place>

本地和远程的分支都会更新 <remote> <place> 参数规定了提交记录的来源和去向 当远程分支和本地分支不同时，可以使用 git push origin <source>:<destination> 同样 fetch具有相同的参数，只是两者作用的方向不同，一个上传一个是下载

## 远程跟踪

因为 pull 和 push 默认的都是master分支 git 如何将 本地分支 master 和 远程分支 o/master关联起来的？ 远程跟踪 直接了当地讲，master 和 o/master 的关联关系就是由分支的“remote tracking”属性决定的。master 被设定为跟踪 o/master —— 这意味着为 master 分支指定了推送的目的地以及拉取后合并的目标。 你可能想知道 master 分支上这个属性是怎么被设定的，你并没有用任何命令指定过这个属性呀！好吧, 当你克隆仓库的时候, Git 就自动帮你把这个属性设置好了。 当你克隆时, Git 会为远程仓库中的每个分支在本地仓库中创建一个远程分支（比如 o/master）。然后再创建一个跟踪远程仓库中活动分支的本地分支，默认情况下这个本地分支会被命名为 master。 克隆完成后，你会得到一个本地分支（如果没有这个本地分支的话，你的目录就是“空白”的），但是可以查看远程仓库中所有的分支（如果你好奇心很强的话）。这样做对于本地仓库和远程仓库来说，都是最佳选择。 这也解释了为什么会在克隆的时候会看到下面的输出： local branch "master" set to track remote branch "o/master"

### git checkout -b totallyNotMaster o/master

### git branch -u o/master foo

# 历史偏离

