第三章 git分支

分支简介

- Git 的分支，其实本质上仅仅是指向提交对象的可变指针。Git 的 master 分支并不是一个特殊分支。
- 名为 HEAD 的特殊指针，指向当前所在的本地分支

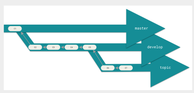
分支的新建与合并

- 分支创建：git branch <branch name>，在当前所在的提交对象上，创建了一个可以移动的新的指针。
- 分支切换：git checkout <branch name>切换到一个已存在的分支。一是使 HEAD 指回 master 分支，二是将工作目录恢复成 <branch> 分支所指向的快照内容，实际上就是把指定分支在仓库中对应的所有文件检出来覆盖当前工作区，最终表现就是切换了分支。
- 创建新分支的同时切换过去：git checkout -b <newbranchname>
- git log --oneline --decorate --graph --all ，它会输出你的提交历史、各个分支的指向以及项目的分支分叉情况。
- 分支的合并git merge <branch name>，branch name是即将被合并的分支。
  - 快进（fast-forward）： 如果顺着一个分支走下去能够到达另一个分支，那么 Git 在合并两者的时候， 只会简单的将指针向前推进 （指针右移），因为这种情况下的合并操作没有需要解决的分歧——这就叫做 “快进（fast-forward）
  - Recursive：，Git 会使用两个分支的末端所指的快照以及这两个分支的公共祖先，做一个简单的三方合并。
  - 删除被合并的分支：git branch -d iss53
- 遇到冲突时的分支合并
  - 遇到合并冲突时，Git 会暂停下来，等待解决合并冲突。 git status 命令查看那些含合并冲突而处于未合并 （unmerged） 状态的文件。
  - 利用vim来编辑冲突的文件
  - git mergetool，该命令会为你启动一个合适的可视化合并 工具，并带领你一步一步解决这些冲突
  - git status 来确认所有的合并冲突都已被解决。
  - 如果你在两个不同的分支中，对同一个文件的同一个部分进行了不同的修改，Git 就没法干净的合并它们。

分支管理

- git branch 不加参数，获得分支列表，分支前的 * 字符：它代表现在检出的那一个分支（也就是说，当前 HEAD 指针所指向的分支）
- git branch -v，要查看每一个分支的最后一次提交
- git branch --merged，列出已合并到当前分支的分支
- git branch --nomerged，列出未已合并到当前分支的分支
- git branch -d/D 删除(强制删除)掉
- 你总是可以提供一个附加的参数来查看其它分支的合并状态而不必检出它们。例如，尚未合并到 master 分支的有哪些？ git checkout testing ; git branch --no-merged master

分支开发工作流



- 长期分支
  - master 分支上保留完全稳定的代码（有可能仅 仅是已经发布或即将发布的代码。）
  - develop 或者 next 的平行分支，被用来做后续开发或者 测试稳定性——这些分支不必保持绝对稳定，但是一旦达到稳定状态，它们就可以被合并入 master 分支了。
- 主题分支
  - 主题分支是一种短期分支，它被用来实现单一特性或其相关工作。比如，改bug,实现某一个特性。

远程分支

- 基本概念
  - 远程引用是对远程仓库的引用（指针），包括分支、标签等等。过 git ls-remote <remote> 来 显式地获得远程引用的完整列表 git ls-remote git@github.com:YuanZehu88/Bio.git，或者通过 git remote show <remote> 获得远程分支的更多信息。git remote show git@github.com:YuanZehu88/Bio.git
  - 远程跟踪分支：是远程分支状态的引用。它们是你无法移动的本地引用。一旦你进行了网络通信， Git 就会为你移动它们以精确反映远程仓库的状态。
  - 远程跟踪分支命名：以 <remote>/<branch> 的形式命名。
  - 如果要与给定的远程仓库同步数据，运行 git fetch <remote> 命令， 从中抓取本地没有的数据，并且更新本地数据库，移动 origin/master 指针到更新之后的位置。
- 推送
  - 本地的分支并不会自动与远程仓库同步-你必须显式地推送想要分享的分支。 运行 git push <remote> <branch> ，branch 的全称：refs/heads/serverfix:refs/heads/serverfix，意思是推送本地的分支来更新远程仓库上的 serverfix 分支。可以通过这种格式来推送本 地分支到一个命名不相同的远程分支。 如果并不想让远程仓库上的分支叫做 serverfix，可以运行 git push origin serverfix:awesomebranch 来将本地的 serverfix 分支推送到远程仓库上的 awesomebranch 分支。
- 跟踪分支
  - 从一个远程跟踪分支检出一个本地分支会自动创建所谓的“跟踪分支”（它跟踪的分支叫做“上游分支”）。跟踪分支是与远程分支有直接关系的本地分支。
  - git checkout -b <branch> <remote>/<branch>
  - git checkout --track origin/serverfix
  - 如果你尝试检出的分支 (a) 不存在且 (b) 刚好只有一个名字与之匹配的远程分支，那么 Git 就会为你创建一个跟踪分支：git checkout serverfix
  - 设置已有的本地分支跟踪一个刚刚拉取下来的远程分支，或者想要修改正在跟踪的上游分支， 你可以在任意时间使用 -u 或 --set-upstream-to 选项运行git branch 来显式地设置。git branch -u origin/serverfix
  - 当设置好跟踪分支后，可以通过简写 @{upstream} 或 @{u} 来引用它的上游分支。 所以在 master 分支时并且它正在跟踪 origin/master 时，如果愿意的话可以使用 git merge @{u} 来取代 git merge origin/master。
  - 查看设置的所有跟踪分支，可以使用 git branch 的 -vv 选项。需要重点注意的一点是这些数字的值来自于你从每个服务器上最后一次抓取的数据。 这个命令并没有连接服务器，它只会告诉你关于本地缓存的服务器数据。 如果想要统计最新的领先与落后数字，需要在运行此命令前抓取所有的远程仓库。 可以像这样做：$ git fetch --all; git branch -vv
- 拉取
  - 当 git fetch 命令从服务器上抓取本地没有的数据时，它并不会修改工作目录中的内容。 它只会获取数据然 后让你自己合并。 然而，有一个命令叫作 git pull 在大多数情况下它的含义是一个 git fetch 紧接着一个git merge 命令。 如果有一个像之前章节中演示的设置好的跟踪分支，不管它是显式地设置还是通过 clone或 checkout 命令为你创建的，git pull 都会查找当前分支所跟踪的服务器与分支， 从服务器上抓取数据然 后尝试合并入那个远程分支。
- 删除远程分支
  - git push origin --delete serverfix

变基

- 原理
  - 变基（rebase）：是首先找到这两个分支的最近共同祖先，然后对比当前分支相对于该祖先的历次提交，提取相应的修改并存为临时文件， 然后将当前分支指向目标基底， 最后以此将之前另存为临时文件的修改依序应用。
- 意义
  - 为了向远程分支推送时能保持提交历史的整洁——例如向某个其他人维护的项目献代码时。
- 基本操作
  1. git checkout experiment
  2. git rebase master
  3. git checkout master
  4. git merge experiment
- 更有趣的变基例子
  - 在对两个分支进行变基时，所生成的“重放”并不一定要在目标分支上应用，你也可以指定另外的一个分支进行应用。
  - git rebase --onto master server client，取出 client 分支，找出它从 server 分支分歧之后的补丁， 然后把这些补丁在master 分支上重放一遍，让 client 看起来像直接基于 master 修改一样。
  - git checkout master
  - git merge client
  - git rebase <basebranch> <topicbranch> 命令可以直接将主题分支 （即本例中的 server）变基到目标分支（即 master）上。 这样做能省去你先切换到server 分支，再对其执行变基命令的多个步骤。git rebase master server； git checkout master； git merge server
- 变基的风险
  - 如果提交存在于你的仓库之外，而别人可能基于这些提交进行开发，那么不要执行变基。
- 用变基解决变基
  - 有人推送了经过变基的提交，并丢弃了你的本地开发所基于的一些提交那种情 境，如果我们不是执行合并，而是执行 git rebase teamone/master
  - 在本例中另一种简单的方法是使用 git pull --rebase 命令而不是直接 git pull。 又或者你可以自己手动 完成这个过程，先 git fetch，再 git rebase teamone/master。
- 变基 vs. 合并
  - 仓库的提交历史即是 记录实际发生过什么。 它是针对历史的文档，本身就有价值，不能乱改。
  - 为提交历史是 项目过程中发生的事。 没人会出版一本书的第一版草稿，软件维 护手册也是需要反复修订才能方便使用。
  - 总的原则是，只对尚未推送或分享给别人的本地修改执行变基操作清理历史， 从不对已推送至别处的提交执行变基操作，这样，你才能享受到两种方式带来的便利