Git 分支branch

几乎所有的VCS都支持分支功能，分支意味着可以从主线开发环境上分离出来，但是不会错过主线上的变更，在很多VCS工具中，这个功能是一个很复杂的功能，通常需要对源码目录进行一个克隆复制，对于大型项目来说，会花费很多时间。

很多人认为git的分支模型是git的一个关键功能点，将git融入到VCS版本控制社区中。Git处理分支非常轻量级，对分支的操作基本上是瞬间完成的，在分支之间切换和回退非常快，和其他版本控制系统不一样的是，git推荐在分支上进行开发，然后合并到主线上。

分支简介

在git基础知识中可以知道，git并不是以变化或不同来存储数据的，git是以快照的形式存储数据。当执行commit操作的时候，git会存储一个commit提交对象，包含一个快照，指向暂存的内容，也包含提交者的名字和邮箱、备注，也会包含一个指针，指向本次提交之前的commit快照点，也就是父节点，父节点可能有0个、1个或多个，比如初始化commit的时候，就没有父节点，正常的commit就只有1个父节点，当进行分支合并，涉及2个及2个以上分支的时候，就存在多个父节点。

为了让分支更加可视化的理解，假设一个目录中有3个文件，这三个文件都被添加到暂存区然后完成了commit操作，存储这些文件时会生成一个脚研制，会在git仓库中存储该版本的文件，在暂存区中增加这些脚研制。

当执行commit操作时，git会为每一个子目录去计算校验值，然后将 这些文件存储在git仓库中，git然后会创建一个commit对象，该对象中包含元数据和指针，指针指向工程的根节点，通过该指针可以随时创建该快照。

这个时候在git仓库中就会包含5个对象，3个是代表3个文件的内容，1个树形对象，列出了目录的内容和文件和blob之间的对应关系，1个commit对象，该对象中包含了一个指针，指向树形根节点和所有提交的元数据。也就是commit对象指向树形对象，树形对象指向所有的文件对象。



如果在做了部分修改之后，再进行提交，则在下一个commit对象中就会多一个父节点的指针，指向之前的快照commit对象。



在git中的分支，就是指向这些commit对象的一个轻量级的可移动指针，在git中的默认分支名为master，当开始进行commit的时候，实际上是将master分支，指向上一次的commit，每进行一次commit，master分支的指针，就会自动的向前移动。

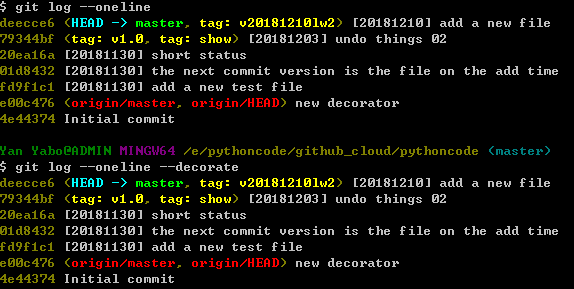
在git中的master分支，并不是一个特殊的分支，master分支和其他的分支都是一样的，唯一可能的区别就是每一个仓库在进行git init时默认创建一个，每个仓库都有，一般不会有人去改变这个事情。

创建新的分支

在创建分支的时候，实际上就是就是创建了一个新的指针，指向当前所在的commit对象上，通过命令git branch \*\*\*来创建分支。

Git是如何知道当前所处的分支是哪一个呢？在git中有一个特殊的指针，叫做HEAD，这个HEAD指针和其他VCS里面的HEAD内容是不同的，在git中，HEAD指针用来指向当前所处的分支，如果一直在master分支上，则HEAD指针则会指向master，git branch只会去创建一个分支，但是并不会切换到新创建的分支上。

通过命令git log，可以查看HEAD当前指向的分支名称，使用的命令是git log –oneline –decorate，如果是刚创建的分支，则HEAD指针会指向两个分支，也就是master和新分支。



通过命令git checkout命令，可以切换到一个已经存在的分支上。切换分支，就是将HEAD指针的指向改变。当切换了分支之后，再进行commit操作，这个时候，就会发现，新分支上的commit操作，parent是之前创建分支的commit对象点，master分支依然指向创建分支的commit点，而新分支会指向刚创建的commit对象点。



假设在当前的工程中红，已经在master分支上做了两个commit提交了，然后当前所处的commit节点C2，在这个时候，准备去解决生产上的问题，就需要创建一个新的分支，然后切换到该分支上去解决该问题，可以通过命令git checkout -b来创建分支，并进行分支切换，实际上是两条命令合并缩写，git branch \*\*\* ,git checkout \*\*\*\*。

由于已经切换到新分支上，HEAD指针现在指向的是新分支，在新分支上做一些修改操作，然后commit，新分支就会向前继续运动。这个时候新分支已经和master分支已经有了区别。



在修改新分支的时候，被告知，在生产master分支上，又有一个新的问题，需要立即修复，通过git，现在并不需要将刚才的新分支上的修改部署到生产上，也不需要将这些修改还原成之前未修改的样子，所需的只是切换回到master分支上，就能保持和生产一样的状态。

假如在当前工作目录或者是暂存区中，有未提交的变更，和当前checkout的分支有冲突，则git是不会允许切换分支的，最好是先将工作状态清空，然后再切换分支。

现在假设已经提交了所有的变更，然后切换到master分支上，git checkout master，这个时候的工作目录中的状态，就是当时刚开始处理第一个bug时候的状态，这个时候就可以集中精力处理新产生的问题，在切换分支的时候，git会将工作目录重置，变成在该分支上最后一次commit的时候的状态，会自动的增加、删除和修改相关文件，保证和最后一次commit时候一样。

而对于新的问题，建议还是再创建一个分支来进行处理，也就是这个时候的hotfix分支。



当对新分支hotfix进行测试，确定hotfix分支就是你想要的之后，就需要将该hotfix分支合并到master分支上，然后重新部署，这个时候就需要进行分支合并操作，分支合并首先是要先切换到master分支上，然后在master分支上对hotfix分支进行合并，具体的命令是：git checkout master，git merge hotfix。

这个时候合并会出现一个fast-forward的信息，因为分支hotfix指向的是C4 commit节点，比master节点指向的C2节点还要前，所以git只是单纯的将指针向前移动，换句话说，当被合并的分支，可以通过合并的分支历史到达，git合并分支只是将指针向前移动，因为并没有差别性的变更需要合并，这就被称为fast-forward。

这个合并产生的变化就是master分支指向的commit对象的指针变化，由之前的C2变成了C4，这个时候部署的代码，就会修改该问题。这个时候hotfix分支、master分支、HEAD指针都指向了当前的C4 commit对象。

这个问题已经修改完成后，就可以切换到之前的分支上开始进行继续工作，当然这个时候可以将hotfix分支进行删除，因为以后也就不再需要这个分支了。通过git branch -d \*\*\*删除分支。然后通过git checkout \*\*\*切换到之前的分支上，继续进行开发工作。

在心的分支上进行了相关commit操作之后，这个时候新分支和master分支已经不再在同一条线上了，也就是说两个分支上的文件和数据都是有差异的，如果想把master分支和这个新分支进行合并，则就需要进行数据的推送。



现在假如已经完成了对iss53的修改，然后准备合并到master分支里面去，这个时候的合并操作和之前的合并操作上差不多的，依然是切换到master分支，然后进行合并。这个时候的合并，弹出的信息是 recursive strategy，递归性策略。

由于当前的master分支，并不是被合并的分支iss53的直接祖先，git就需要做一些其他操作，这个时候的合并做了一个三步合并操作，使用了两个快照指针，有一个指向了这两个分支的共同祖先。



和之前直接fast-forward移动master分支指针不同，git通过散步合并操作，创建了一个新的快照点，同时也创建了一个新的commit对象，快照点，指向这个新创建的快照对象，这代表的是一个合并的commit操作，之所以这个比较特殊，是因为这个commit操作，拥有超过一个的父节点。



这个时候就已经完成了分支的合并，新做的变更也合并进去，iss53这个分支也就没有存在的必要了，这个时候就可以将这个分支删除，git branch -d

合并冲突

如果在两个分支上，都对同一个文件，进行了不同的更改，比如对于相同的文件file，分支A上做了重命名，分支B上删除了该文件，也就是这个意思，在将这两台分支进行合并的时候，就会出现冲突，git没法很平稳的将这俩分支合并。假如在之前的iss53分支上，修改了和hotfix分支上相同的文件，这就会出现合并冲突，体现在外围CONFICT，自动合并就会失败。

这个时候冲突产生，git没法自动的产生一个合并的commit对象，直到冲突解决了，才能进行合并，如果想看看在产生冲突的时候，哪些文件没办法合并进去，可以通过命令git status来查看。

所有存在合并冲突的文件，在没有得到解决之前，是没法合并的，git会为这些文件添加一个标准的冲突解决标志，用户可以手动的打开文件进行解决。



在上面的输出中，HEAD表示当前的master分支，以==为分隔符，上面的部分是master分支上的内容，下面的部分是iss53分支上的内容，可以自己选择其中一个，或者再修改。当冲突解决了之后，分隔符就被移除掉了，然后就可以通过git add将之前的冲突文件，添加，标志为已经解决了冲突，添加到暂存区。

如果想通过一个图形化界面来解决这些冲突问题，可以通过git mergetool，会弹出一个可视化的工具，来引导冲突。

默认的合并工具是opendiff。

当退出合并工具之后，git会进行询问，合并是否成功，如果回答成功，则git会将这些冲突文件标记为冲突已经解决，然后可以通过git status来确认所有的冲突已经解决。

当所有的冲突都已经解决之后，可以通过git commit来结束合并操作。在这个commit操作会有默认的提示信息，如果觉得这个提示信息对其他人有用，可以修改这个备注提示信息，来解释如何去面对冲突，解决冲突的。