Git分支(branch)

在代码版本控制工具中，都会有branch的概念。刚开始建立版本仓库的时候，我们只有一个主分支（master branch），我们不可能把日常的新功能开发、代码优化以及bug修复等概念工作全都放在主分支上，这样会使主分支很难维护。这就是为什么会有branch。

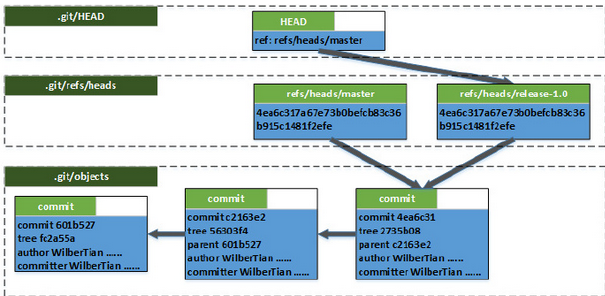
# 分支的创建

在Git中，branch的创建很简单，我们可以通过下面的命令创建一个"release-1.0″的branch。然后通过"checkout"命令来切换branch。

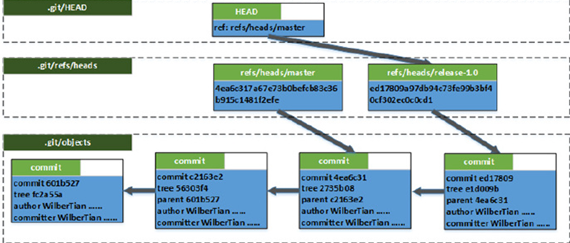


在Git中，我们可以通过"git checkout -b"命令来达到同时创建并切换branch的效果。

我们进入".git/refs/heads"目录，发现有"HEAD"和"release-1.0″两个文件，并且两个文件包含的哈希值相同，根据"git log"可以知道这个哈希值代表master上最新的提交。所以，创建分支后我们会得到下面的关系图，从值张图中可以看到，branch的切换对应HEAD引用值的改变。



有了新的branch之后，我们就可以分别在不同的branch上工作了。假设我们现在更新"app.py"，并且在release-1.0 branch上面提交，重新查看对象关系图。



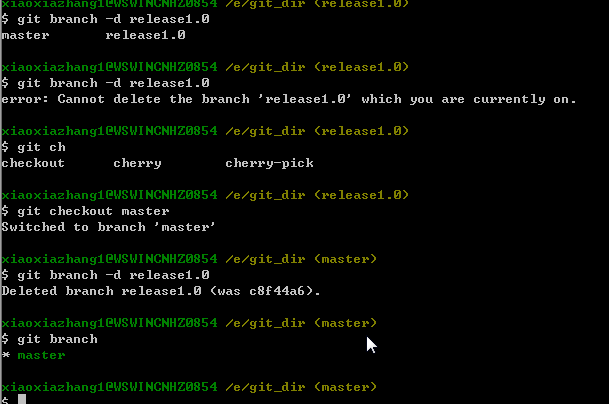
根据上面的对象图，可以分析出以下信息：

* "ref/heads/release-1.0″文件内的哈希值将更新为release-1.0 branch上最新的更新
* release-1.0 branch上面的更新不会体现在master branch

经过查看，"ref/heads/release-1.0″文件的内容确实是release-1.0 branch上最新的更新。并且，当我们切换到master branch的时候，所有master上的东西都没有变化。

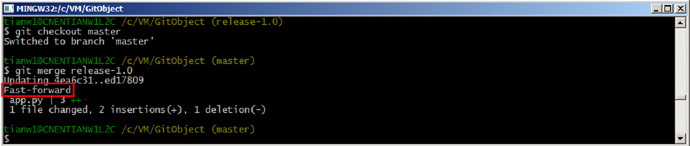
# 分支的删除

分支的删除比较简单，可以使用"git branch -d"命令，为了不影响后面的内容，这里演示新建、删除dev branch。

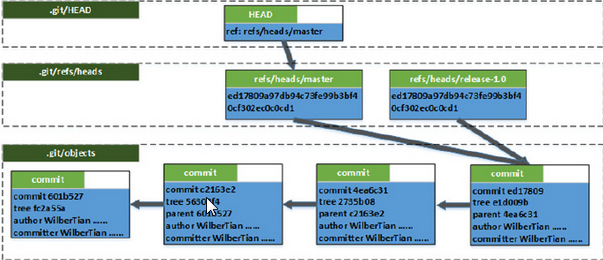


# 分支的合并

branch的创建是为了方便开发、修复bug以及保持master的稳定。但是最终branch上的内容还是要合并到master的，接下来就看看分支的合并。在Git中，可以通过"git merge"来合并分支，继续前面的例子，我们可以把"app.py"的更新merge到master branch上。

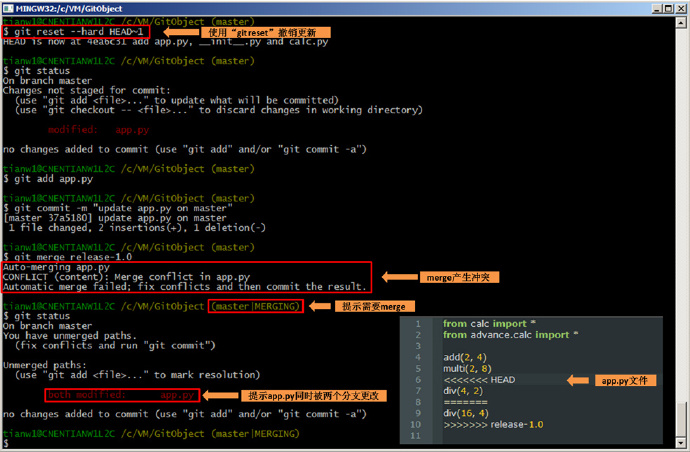


这时候，master的HEAD就被更新了，跟release-1.0内容一致了，这些就是merge命令做的事情。

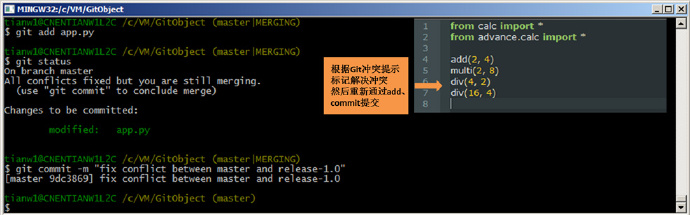


# 合并冲突

在branch的合并中，很多时候我们会遇到冲突，那么我们就需要手动解决冲突，然后再提交了。为了演示冲突合并，我们回退master到上一次提交（一定还记得git reset吧），然后同样更新"app.py"提交到master。

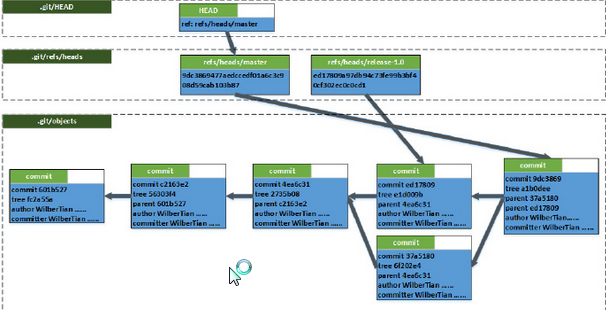


通过上面可以看到，master和release-1.0上同时修改了"app.py"，合并时引起了冲突。在Git中，用"<<<<<<<"，"======="，">>>>>>>"标记出冲突区域，我们需要根据这些符号找到所有的冲突并解决。



冲突解决之后，我们通过"git log"来产看一些结果，但是这次我们要给"git log"加一些参数。git log --graph --pretty=oneline --abbrev-commit

同时，这里给出最新的对象关系图：

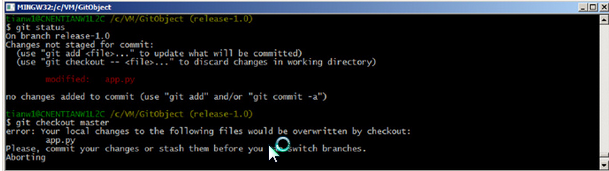


# branch相关操作

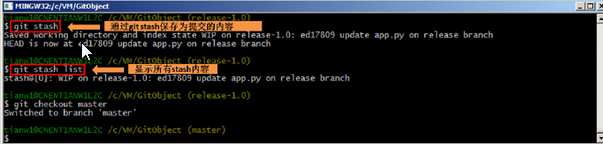
## stash

在Git中，stash是个很有用的命令，可以保存我们做到一半的工作，可以理解成一个未完成工作的保存区。假如我们在release-1.0 branch做了一些更新，但是想做的事情还没有全部完成，不能提交，这是我们又要切换到master branch，这是Git会禁止branch切换。

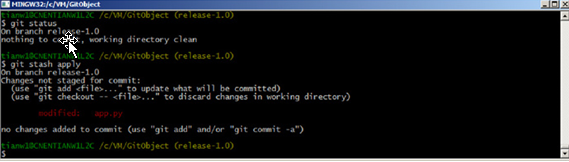
比如，例子中我们从app.py中删除"div(16, 4)"。



这个时候我们就需要使用stash来保存未完成的工作了。



如果需要继续前面的工作，我们可以通过"git stash apply"来还原被保存的工作状态。



stash空间就像是一个栈空间，每次通过stash保存等内容都会被压入stash栈。命令不仅仅是支持简单的list、apply操作，接下来我们看看更多的stash命令。

* git stash save：可以通过自定义的信息来描述一个stash
* git stash apply stash@{n}：通过这个命令，我们可以选择stash栈中的stash，从而恢复到特定的状态；"git stash apply"使用栈顶来还原WorkSpace。
* git stash pop：就像"git stash apply"使用栈顶来还原WorkSpace，但是同时会从stash栈中删除当前的栈顶stash。

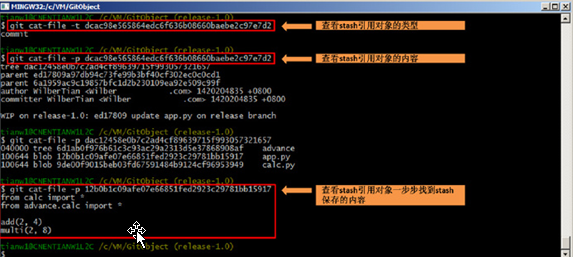
stash工作原理：

相信大家都看到了stash的强大，下面我们来看看stash的工作原理。在使用过stash保存之后，我们会发现.git目录中出现了两个新文件".git/refs/stash"和".git/logs/refs/stash"。两个文件内容分别如下：

.git/refs/stash：dcac98e565864edc6f636b08660baebe2c97e7d2

.git/logs/refs/stash：0000000000000000000000000000000000000000 dcac98e565864edc6f636b08660baebe2c97e7d2 WilberTian <Wilber\*\*\*com> 1420201804 +0800    WIP on release-1.0: ed17809 update app.py on release branch

根据我们在对象关系模型那篇文章中学到的内容，我们可以得到以下信息，所以可以看到，stash存放内容也是可以根据对象关系模型一点点找出来。



## branch之间的diff

在前面的文章中我们通过diff比较过同一个分支上的内容在WrokSpace、stage和repo中的差别。同样diff可以支持分支之间的比较。

git diff branchName：把当前branch跟branchName进行比较，也可以使用"git diff branchNameA branchNameB"

git diff branchName — fileName：比较两个branch的fileName文件差异