# 一、Git 分支管理和冲突解决

## 1、创建分支

**git branch**

　　没有参数，显示本地版本库中所有的本地分支名称。

　　当前检出分支的前面会有星号。

**git branch newname**

　　在当前检出分支上新建分支，名叫newname。

**git checkout newname**

　　检出分支，即切换到名叫newname的分支。

**git checkout –b newname master**

　　这个命令将上面两个命令合并：在master分支上创建分支newname分支并检出到该分支。

## 2、合并分支间的修改（Merge）

　　合并操作将两条或多条分支合并到一起，实际上有好几种分支合并方法，下面介绍主要的三种：

### 1.直接合并(straight merge)：

　　把两条分支上的历史轨迹合并，交汇到一起。

　　比如要把dev分支上的所有东东合并到master分支：

　　首先先到master分支： **git checkout master**

　　然后把dev给合并过来：**git merge dev**

　　注意没参数的情况下merge是**fast-forward**的，即Git将master分支的指针直接移到dev的最前方。

　　换句话说，如果顺着一个分支走下去可以到达另一个分支的话，那么Git在合并两者时，只会简单移动指针，所以这种合并成为**快进式(Fast-forward)**。

### 2.压合合并(squashed commits)：

　　将一条分支上的若干个提交条目压合成一个提交条目，提交到另一条分支的末梢。

　　把dev分支上的所有提交压合成主分支上的一个提交，即压合提交：

**git checkout master**

**git merge --squash dev**

　　此时，dev上的所有提交已经合并到当前工作区并暂存，但还没有作为一个提交，可以像其他提交一样，把这个改动提交到版本库中：

**git commit –m “something from dev”**

### 3.拣选合并(cherry-picking)：

　　拣选另一条分支上的某个提交条目的改动带到当前分支上。

　　每一次提交都会产生一个全局唯一的提交名称，利用这个名称就可以进行拣选提交。

　　比如在dev上的某个提交叫：321d76f

　　把它合并到master中：

**git checkout master**

**git cherry-pick 321d76f**

　　要拣选多个提交，可以给git cherry-pick命令传递-n选项，比如：

**git cherry-pick –n 321d76f**

　　这样在拣选了这个改动之后，进行暂存而不立即提交，接着可以进行下一个拣选操作，一旦拣选完需要的各个提交，就可以一并提交。

## 3、冲突处理

　　当两条分支对同一个文件的同一个文本块进行了不同的修改，并试图合并时，Git不能自动合并的，称之为冲突(conflict)。解决冲突需要人工处理。

　　比如当前在master分支，想把dev分支merge过来，结果产生了一个冲突，打开文件内容可以看到这么一个冲突：



## 4、删除分支

　　有些分支没有必要长期保存，比如分支中的代码已经打了标签并已发布，或者实验分支已经成功完成工作或中途废弃等等。

**注意**：打了标签的分支，Git在删除该分支时，从版本树起始到此标签间的全部历史轨迹均会保留，此时删除分支操作只是删除分支本身的名称，因此可以说该分支没有必要长期保存。

　　而在其他版本控制工具中，删除分支通常意味着删除分支上的所有历史轨迹，所以不能因为打了标签就认为其没有必要保存。

　　删除一个分支dev2：

**git branch –d dev2**

　　注意不能删除当前所在分支，需要转到别的分支上。

　　如果要删除的分支已经成功合并到当前分支，删除分支的操作会直接成功。

　　如果要删除的分支没有合并到当前所在分支，则会出现提示，如果确定无须合并而要直接删除，则执行命令：

**git branch –D dev2**

　　进行强删。

## 5、分支重命名

　　重命名分支：

**git branch –m oldname newname**

**-m**不会覆盖已有分支名称，即如果名为newname的分支已经存在，则会提示已经存在了。

　　如果改成**-M**就可以覆盖已有分支名称了，即会强制覆盖名为newname的分支，这种操作要谨慎。