**git使用分支**

# 为什么要使用分支

在开发项目的过程中使用版本控制工具，建立版本库（仓库），需要分为开发库，测试库，发布库。因为，开发人员需要不断前进完成功能，测试人员在后面紧跟测试，售后人员需要稳定版本上线。

举例说明：

如果只有一个版本库，会存在什么问题：

1. 场景1：程序员在下班时想把自己的代码，提交到版本库，但是，此代码并没有写完（开发人员自己还没有测试），如果这个时候，测试人员下载最新版本进行测试时，那么就会莫名其妙（以为，程序员提交了的功能有问题）。因为，他们是共享一个库的。开发人员只要上传了代码，测试人员立即就可以得到。如果不上传代码，那么代码就有可能会丢失。
2. 场景2：程序员A修改了自己的代码，还需要等待程序员B的代码，才能一起联调功能，而此时，程序员A上传了自己的代码，测试人员得到代码后，也会莫名其妙（以为，程序员提交了一个有问题的代码）
3. 由于没有明确的稳定版本（stable版本），导致上QA（测试库），上生产（发布库），只能采用增量更新，代码管理非常混乱，而且，测试人员的代码和开发人员的代码耦合度很高。

解决问题：

1. 分支管理策略：采用适当的分支管理策略来保证开发库、测试库、发布库的隔离。有了各自的库，开发人员随时可以放心的提交自己没有写完的代码（提交的开发库，甚至自己可以有独立的开发库）而不用担心测试人员不小心拿到了还没有写完的代码。等到，开发人员都写完后（开发人员认为功能没有问题了），再把代码放到测试库里，供测试人员进行测试，这样一来，对于测试库来说，每个版本都是可以进行测试的版本；同理，测试人员测试完成认为可以上线时，才生成发布库，这样一来，发布库的每个版本都是可以发布的。即，开发库的版本数量是大于测试部版本数，测试库的版本数大于发布库的版本数，而发布库的版本就是对外开放的版本（即，用户使用的版本）。
2. 适当引入每日编译、持续集成、Code Review（代码评审）等敏捷开发的最佳实践
3. 采用自动化脚本完成上QA库、上发布库的部署工作，避免人工失误

在项目开发中，经常使用的三种版本管理策略是：不稳定主干策略、稳定主干策略、敏捷发布策略。

* 不稳定主干策略：使用用主干作为新功能开发主线，分支用作发布。
  + 使用用主干作为新功能开发主线，分支用作发布。
  + 被广泛的应用于开源项目。
  + 比较适合诸如传统软件产品的开发模式，比如微软的office等。
  + bug修改需要在各个分支中合并。
  + 新代码在主干上开发，因此如果主干不能达到稳定的标准，就不可以进行发布。
  + 这种策略的好处是没有分支合并的工作量，因此比较简单。
* **稳定主干策略**
  + 使用主干作为稳定版的发布。
  + bug的修改和新功能的增加，全部在分支上进行。
  + 不同的修改或新功能开发以分支隔离。
  + 分支上的开发和测试完毕以后才合并到主干。
  + 主干上的每一次发布都做一个标签而不是分支。
  + 每次发布的内容调整起来比较容易。
  + 缺点是分支合并所增加的成本。
* **敏捷发布策略**
  + 敏捷开发模式的项目中广泛采用，敏捷开发的项目具有固定的发布周期。
  + 为每个task建立分支。
  + 为每个发布建立分支，每个周期内的task分支需要合并到发布分支发布。
  + 在完成发布后，发布分支的功能合并到主干和尚在进行的任务分支中。
  + 一种稳定主干策略的变体。
  + 团队成员要求较高。
* **建议方案：**

此方案已稳定主干策略为主结合了一些敏捷发布策略的思路，具体实施方案如下：

1、主干时刻处于稳定状态，随时可以发布。设SCM人员对主干代码进行管理，普通开发人员只读。

2、SCM为开发任务建立开发分支。常规的可以以小组为单位建立分支，较大的任务可以建立专门的分支。

3、在发布日，从主干复制一个测试分支，需要在本发布日发布的各开发分支向此测试分支合并。

4、对测试分支代码进行测试，出现bug在测试分支上更改，无误后发布。

5、测试分支代码发布后，合并入主干，并在主干上进行标记。

6、对紧急修复（Hotfix）的情况，可以从主干复制出测试分支，在测试分支上进行紧急修改，并在测试后发布，发布后同样将代码合并会主干，做标记。

7、 Hotfix仅限于可以很快解决的小问题，如果更改时间过长，则需采用常规方法完成。

8、如果在测试分支测试过程中需要hotfix工作，则在复制一个新的测试分支进行hotfix，测试后发布。然后同时合并入原测试分支和主干，并在主干上做标记。此过程未在上图中画出。

9、测试分支发布后，开发分支可以删除；测试分支合并入主干后，测试分支可以定期删除。

**方案的优缺点**

方案优点

1、解决了没有实施分支策略时，代码不能经常签入的问题。

2、主干代码始终处于稳定的状态随时可以发布，降低了风险。

3、可以基于一个完整的测试分支进行测试及发布，而不是以口口相传的方式增量更新。

方案缺点

1、建立分支、合并分支增加了工作量。考虑实际情况，以及版本控制工具的辅助，增加的工作量应该可以接受。

2、如果某些开发分支工期跨越多个发布周期，修改过于剧烈，合并分支时可能工作量较大。可以考虑分解任务，避免过大的任务出现。

3、在同一时间最好只有一个测试分支，因此建立测试分支的权限需要限制，除hotfix场景外应当避免。

# Git中使用分支

~~Git的分支比起svn的分支，git创建分支，切换分支，合并分支，删除分支都非常快（因为，git使用的是指针的方式切换分支的）~~

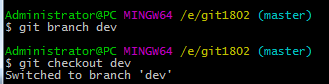
在没有使用分支之前，git会默认有一个分支，就是主分支（master分支，还记得 git push –u origin master这个命令吗？）

* 1. 创建分支（分支名为dev。）

Git branch dev

* 1. 切换当前分支到dev

Git checkout dev



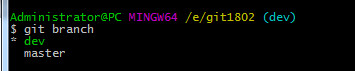
此后的add和commit最终是提交到了dev分支。如果切换到master分支，那么，修改时不能看到的，因为，修改时在dev分支上进行的。

* 1. 可以一条命令完成创建并切换到新分支（-b:表示创建并切换）

Git checkout -b dev

* 1. 查看所有分支（当前分支前面会有星号\*）

Git branch



* 1. 把dev分支的内容合并到当前分支（如：master分支）里。

1)、首先确保当前分支是master分支（用命令切换：git checkout master）

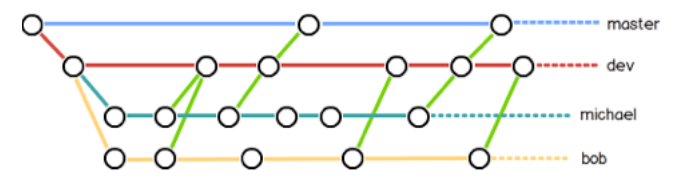
2)、命令合并 git merge dev

* 1. 删除分⽀：git branch -d dev

注意当前分支一定不能是要删除的分支（dev）

# 分支策略

分⽀策略 在实际开发中，我们应该按照⼏个基本原则进⾏分⽀管理： ⾸先，master分⽀应该是⾮常稳定的，也就是仅⽤来发布新版本，平时不能在上⾯干活； 那在哪干活呢？干活都在dev分⽀上，也就是说，dev分⽀是不稳定的，到某个时候，⽐如 1.0版本发布时，再把dev分⽀合并到master上，在master分⽀发布1.0版本； 你和你的⼩伙伴们每个⼈都在dev分⽀上干活，每个⼈可以都有⾃⼰的开发分⽀，时不时地往dev分 ⽀上合并就可以了。 所以，团队合作的分⽀看起来就像这样：



每个开发人员有自己的分支，完成后，放到dev分支。每次修改也可以临时建立一个分支，修改完成后，合并即可。