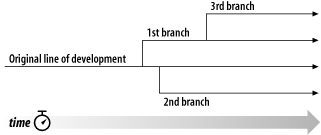
<http://blog.csdn.net/keda8997110/article/details/21813035>

先说说什么是branch。按照Subversion的说法，一个branch是某个development line（通常是主线也即trunk）的一个拷贝，见下图：



branch存在的意义在于，在不干扰trunk的情况下，和trunk并行开发，待开发结束后合并回trunk中，在branch和trunk各自开发的过程中，他们都可以不断地提交自己的修改，从而使得每次修改在repository中都有记录。

设想以下场景，如果你的项目需要开发一个新功能，而该功能可能会修改项目中的绝大多数文件，而与此同时，你的另一位同事正在进行bug fix，如果你的新功能不在branch中开发而直接在trunk中开发，那么你极有可能影响另一位同事的bug fix，他/她在bug修复中可能会遇到各种各样的问题，因为你的频繁提交代码引入了过多的不稳定因素。你可能会说，那我在开发的过程中不提交不就行了，等到我全部开发结束我再提交，是，你可以这么做，那还要版本控制干什么呢？也许等到你最后提交代码的时候（也许一周，也许两周？），你会发现有一大堆conflict等着你resolve。。。

那么，正确的做法是什么？使用branch，从trunk创建branch，然后在你的branch上开发，开发完成后再合并到trunk中。

关于branch先讲到这里，下面说说什么叫做合并。很好理解，当branch开发完成后（包括必要的测试），将branch中的修改同步到trunk中，这个过程有可能包括修改文件、增加文件、删除文件等等。

说到这里，貌似本文差不多可以结束了，不就是分支和合并么？只要再简单地说说如何建立分支和如何合并就可以收尾了，可能只需两个命令，也可能只需鼠标点几下然后键盘敲两下即可。其实事情远非这么简单，爱动脑筋的同学可能会问了，将branch的改动merge到trunk的时候，和上文说的直接在trunk中全部开发完然后提交有何区别？你最后还不是要处理一大堆conflict？

这个问题问得非常好，其实这正是本文的重点：branch和trunk在并行开发的过程中如何感知对方，branch如何才能在开发过程中不会和trunk越走越远，导致最后无法合并？试想一下，如果在你开发branch的过程中，trunk中的某个类文件已经被删除了（这可能是另外一个家伙在另一个branch上开发了两周后才合并到trunk的），而你竟然在这个类文件上做了大量修改，试问你到最后合并回trunk的时候该有多蛋疼？解决这一问题的唯一手段是，branch要不停地和trunk保持同步，你要及时地知道trunk都做了什么修改，这些修改是否会影响你正在开发的新功能，如果需要，你必须及时调整branch的代码，使之能与trunk“兼容”。

那么如何让branch和trunk保持同步？合并，从trunk合并到branch，你没听错，是从trunk合并到branch。关于TortoiseSVN的合并，有几点需要注意：

* TortoiseSVN的合并发生在本地，也即你的working copy中，你无需过多担心会对repository中的代码造成影响
* 不管是从trunk合并到branch还是最终从branch合并回trunk，在每次合并前最好先update，然后将本地的修改先全部commit，保护好现场，万一合并不理想随时都可以revert
* 合并完成后看是否能正确编译，然后测试验证，最后将合并后的改动提交到repository

下面我将step by step地演示如何一次完整的branching和merging，包括创建分支、分支开发、分支和主线同步，分支合并到主线的全过程，甚至包括如何在本地创建一个测试用的repository。

首先需要安装TortoiseSVN，我安装的版本是：TortoiseSVN 1.6.15, Build 21041 - 32 Bit , 2011/03/23 18:00:27

**1、本地Repository的创建**

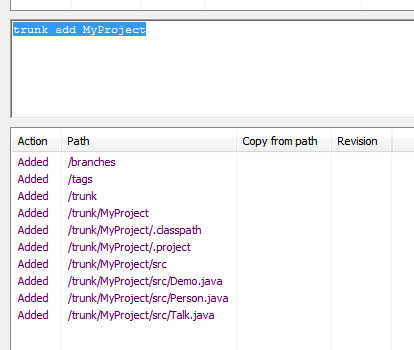
repository的创建很简单，假设我要在D:\TortoiseSVN\TestRepository目录中创建repository，只需右键TestRepository目录，依次选择"TortoiseSVN" -> "Create repository here"便完成了repository的创建。

**2、Check out**

假设要check out到D:\TortoiseSVN\TestSVN，同样很简单，在D:\TortoiseSVN目录下创建TestSVN目录，然后在该目录上右键，选择"SVN Check out..."，在弹出的窗口中的"URL of repository"中填入"file:///D:/TortoiseSVN/TestRepository"，其他默认即可，最后点击ok。

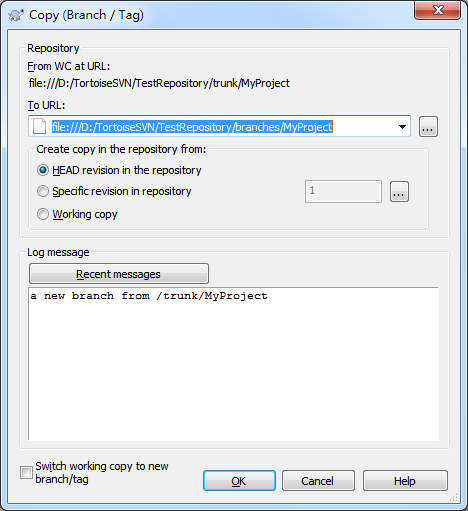
**3、trunk创建新项目MyProject**

相当简单就不赘述了，只列出本次操作所作出的修改：



**4、创建branch**

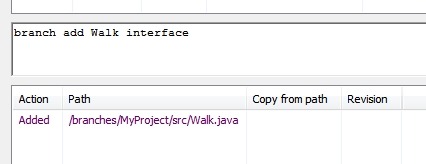
在/trunk/MyProject目录上右键，依次选择"TortoiseSVN" -> "Branch/tag..."，在弹出窗口的"To URL"中填入分支的地址，在这里目标revision选择HEAD revision，如下图所示，添加log后点击ok分支便建立了。这个操作速度非常快，新建的branch在repository中其实只是一个指向trunk某个revision的软连接而已，并没有真的复制文件。



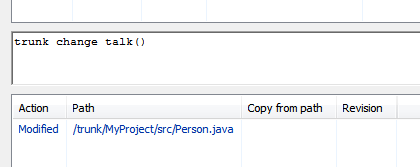
**5、Check out分支**

右键TestSVN目录选择"TortoiseSVN Update"即可将刚刚建立的分支下载回本地。进入/branches/MyProject目录下你会发现其文件结构和/trunk/MyProject一模一样。

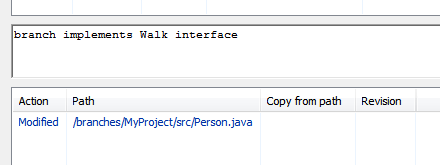
**6、branch提交一个新文件**



**7、trunk紧接着提交一个修改**



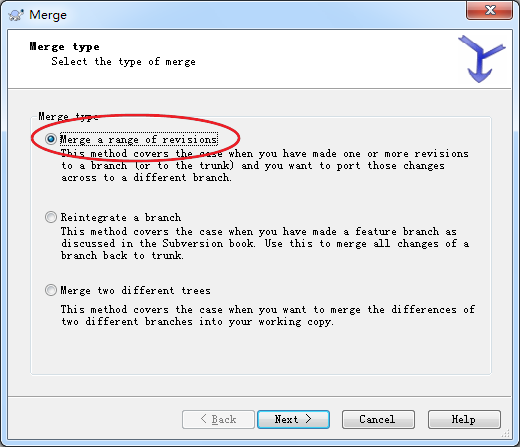
**8、branch再次提交一个修改**



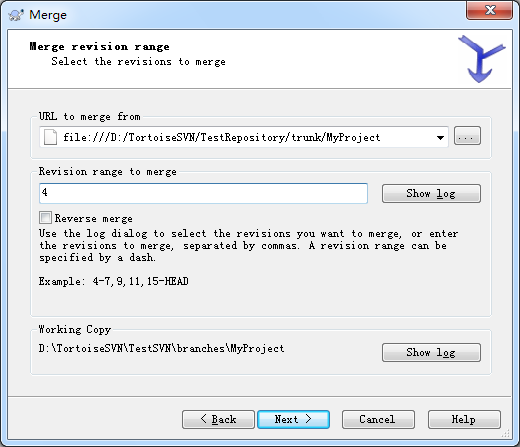
**9、将trunk中的修改同步到branch**

6-8演示的是branch和trunk在独立、并行地开发。为了防止在“错误”的道路上越走越远，现在branch意识到是时候和trunk来一次同步了（将trunk合并到branch）。

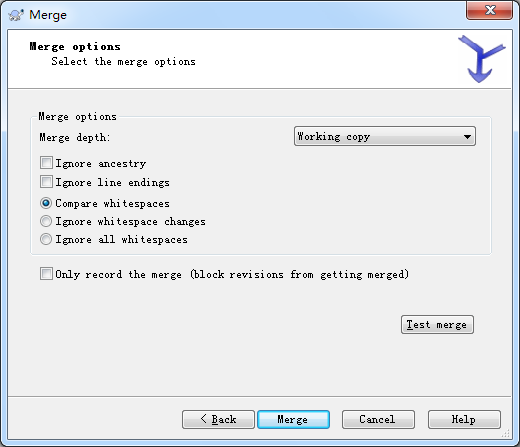
首先，在本地trunk中先update一下，有冲突的解决冲突，保证trunk和repository已经完全同步，然后在/branches/MyProject上右键，依次选择"TortoiseSVN" -> “Merge...”，在弹出的窗口中选择第一项"Merge a range of revision"，这个类型的Merge已经介绍得很清楚，适用于将某个分支或主线上提交的多个revision间的变化合并到另外一个分支上。



点击next后，出现如下窗口：

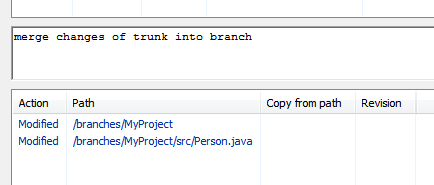


由于是要从trunk合并到branch，理所当然这里的"URL to merge from"应该填trunk的路径，"Revision range to merge"很好理解，就是你要将trunk的哪些revision所对应的变化合并到branch中，可以是某一连串的revision，比如4-7，15-HEAD，也可以是某个单独的revision号。由于在r4中，trunk修改了Person.java中的talk()方法，所以这里的revision只需填4即可。点击next后出现下图：



在这里只需保留默认设置即可。在点击Merge按钮前你可以先Test merge一把，看成功与否，以及merge的详细信息。点击Merge按钮后trunk所做的修改将同步到branch中。

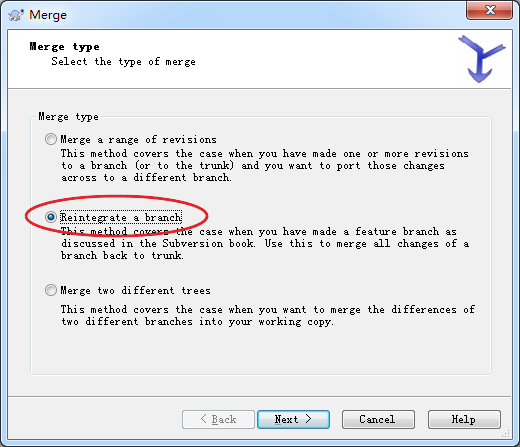
**10、提交合并后的branch**



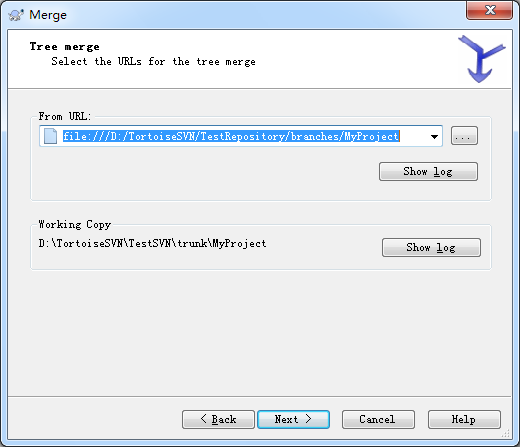
至此，branch已经完全和trunk同步，branch和trunk的代码相处很融洽，没有任何冲突，如果branch已经开发结束，那是时候将branch合并回trunk了，当然，如果branch还要继续开发，那你将不断地重复6-10这几个步骤。

**11、将branch合并回trunk**

在/trunk/MyProject上右键（注意是在主线的目录上右键），依次选择"TortoiseSVN" -> "Merge..."，在弹出的窗口中，Merge type选择第二项"Reintegrate a branch"，这种类型的合并适合在分支开发结束后将所有的改动合并回主线。



点击next后出现如下窗口：



在这里，"From URL"选择/branches/MyProject，无需选择revision号，Reintegrate会将branch上所有修改合并到trunk。后面的步骤和上文第9步中的一样，不再啰嗦了。如无意外，branch将成功合并到trunk，你需要做的只是将合并后的trunk赶紧commit！

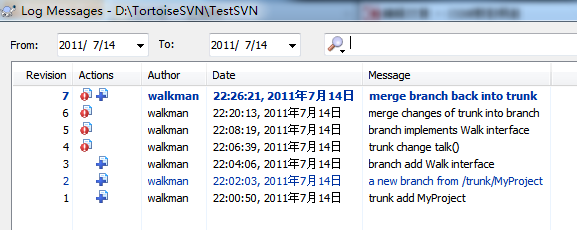
**12、提交合并后的trunk**

so easy...

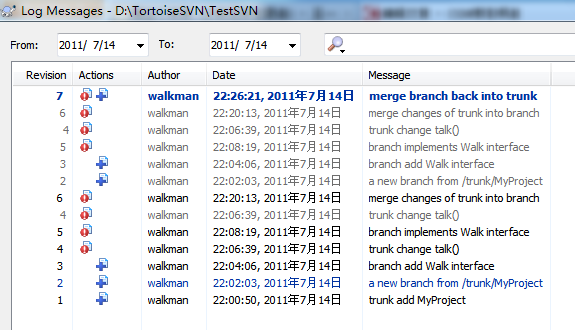
**13、删除branch**

如果你认为你新加的功能已经开发完成了，你可以删除你的分支

到这里，我已经给你演示完了整个过程，我一身的汗也下来了，我想罢工了，不过最后我们还是看看所有的log信息吧，通过log能发现我们干的所有事情：



r1-r7正是我上文在干的事情，从Message中你能发现我对trunk和branch都干了什么，另外，在Log Messages窗口的左下角勾选了"Include merged revisions"你还能看到额外的Merge information：



图中灰色的是和merge相关的log，共发生了两次merge，第一次是在r6，在r6中，branch合并了trunk在r4时提交的变化；第二次是在r7，在r7中，trunk合并了branch从r2到r6的所有变化。