[svn](https://www.baidu.com/s?wd=svn&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLmvfvnHD3PHm4uAm4njn10ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWTYPjTsnHTk)(subversion)是近年来崛起的版本管理工具，是[cvs](https://www.baidu.com/s?wd=cvs&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLmvfvnHD3PHm4uAm4njn10ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWTYPjTsnHTk)的接班人。目前，绝大多数开源软件都使用[svn](https://www.baidu.com/s?wd=svn&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLmvfvnHD3PHm4uAm4njn10ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWTYPjTsnHTk)作为代码版本管理软件。  
客户端的使用   
 1.Checkout Repository   
 首先要Checkout服务器端的Repository，  
 所谓的Checkout就是指获得服务器端指定的Repository存储的所有文件。  
 这个Checkout和Visual Source Safe的Checkout意义完全不一样，  
 VSS的Checkout指的是锁定某个文件，如果你以前使用过VSS，  
 在学习Subversion时这个问题一定要注意。  
 Checkout的具体方式是：  
 在客户端新建一个空目录，比如：F:\Project1   
 在该目录上单击右键，在弹出式菜单中选中[SVN](https://www.baidu.com/s?wd=SVN&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLmvfvnHD3PHm4uAm4njn10ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWTYPjTsnHTk) Checkout...，  
 之后在“URL of Repository”文本框中填入你想要连接的Repository的地址，  
 这个URL地址可以用浏览方式加入。  
 对于在本教程第二节建立的Repository，  
 URL应该是“svn://xxx/project1”  
 (xxx可以是服务器端主机名，也可以是服务器端的ip地址)。   
 然后点OK，会弹出一个认证对话框，  
 输入在教程第三节设置的用户名和密码。  
 点OK后就完成了对Repository的Checkout。  
 比如：在服务器端Repository中有一个a.txt文件，  
 那么Checkout之后F:\Project1目录下也会出现一个a.txt文件。  
 在本例中由于服务器端的Repository还未添加任何文件，  
 所以在客户端的F:\Project1下没有文件被Checkout。  
 执行Checkout除了会在F:\Project1产生Repository存储的文件及目录外，  
 还会产生了一个“.svn”的隐含目录，该目录是由subversion管理的，  
 不要删除或者手工改动其中的文件和目录。  
 现在F:\Project1中的文件和目录就叫做Repository的“Working Copy”简写“WC”  
 (这个简写...汗)。  
 以后对Repository中文件和目录的修改，添加，删除的操作，  
 都是通过对这个“Working Copy”的操作实现的。  
 Checkout执行完后，  
 会发现F:\Project1目录的图标的左下角附着了一个小的状态图标  
 (当F:\Project1目录中的文件改变时，这个状态图标也会随之变化)，  
 它表示F:\Project1是一个Repository的“Working Copy”，  
 F:\Project1内的所有文件和目录也会有类似的状态图标。   
  
 2.添加文件   
 将要添加的文件或者目录拷贝到F:\Project1下，  
 然后在该文件或目录上单击右键，TortoiseSVN->Add，点OK。  
 如果添加了不止一个文件或目录，  
 则鼠标不要在F:\Project1中点中任何文件，  
 然后单击右键，TortoiseSVN->Add，  
 就可以添加多个文件或目录。  
 这时文件的状态图标会发生变化。  
 Add命令只是告诉本地的“Working Copy”将该文件纳入版本管理，  
 并没有将这个改变提交到服务器端，  
 如果想要别人也看见你对Repository的修改，你需要   
 在F:\Project1下单击右键，SVN Commit...，  
 将你所做的修改提交到Repository。  
 文件的状态图标也会更新。  
 不管你在“Working Copy”内添加、修改、删除文件后，  
 要想其他人也看见你的修改，  
 都必须用Commit命令将所做修改递交到服务器端的Repository。   
  
 3.修改文件   
 用文本编辑器或IDE对文件修改后，  
 文件的状态图标会变化，  
 然后单击右键，SVN Commit...   
 提交修改，只有当执行Commit提交修改后，  
 你所作的修改才会反映到服务器端的Repository中。   
  
 4.删除文件   
 删除文件时，选中要删除的文件或目录，  
 单击右键，TortoiseSVN->Delete，提交修改。  
 注意千万不要用“Delete”键来删除文件，否则将无法提交你的修改。  
 这一点对目录的删除来说尤为重要。   
  
 5.放弃修改   
 当你添加、修改、删除文件后，决定放弃修改，  
 你可以单击右键，TortoiseSVN->Revert，  
 本地的“Working Copy”中的文件和目录会恢复到你修改前的状态。   
  
 6.获取Repository的最新版本   
 当一个团队合作开发项目时，  
 每一个人都在不断的对Repository进行更新，  
 你需要不断的更新自己的“Working Copy”，  
 以获取项目最新的文件。  
 当第一次获得最新Repository的文件时，  
 我们用Checkout命令，前面已经介绍了，  
 以后再获取最新文件时就不用Checkout了。  
 而改用Update命令。  
 接着前面的例子，这时F:\Project1已经成为一个“Working Copy”了  
 (通过执行Checkout命令)，现在其他人已经对Repository进行了修改，  
 我想将别人的修改反映到我的“Working Copy”中，  
 具体的方法是：在F:\Project1目录上单击右键，  
 SVN Update。这时F:\Project1中的文件就是最新的版本了。  
 注意，如果当你的“Working Copy”中有被修改的文件，  
 或者有被删除的文件，并且还未提交这些修改时，  
 这些文件在执行Update过程中是不会被更新的。  
 比如你修改了F:\Project1下a.txt文件，  
 还未提交修改，那么，  
 当你对F:\Project1进行Update时，  
 a.txt文件是不会更新为Repository上的a.txt文件的。  
 所以如果想放弃当前的所有修改，  
 并将F:\Project1下所有文件及目录更新到最新版本,  
 应该先对F:\Project1执行Revert命令再执行Update命令。   
  
 7.subversion的版本控制模型   
 当你用subversion进行版本控制时，  
 Subversion会记录你对Repository进行的每一次修改(包括添加，修改，删除等等)，  
 每修改一次Repository都会产生一个新的Revision(修订版本号)，  
 不同的Revision代表了不同时刻Repository的状态，  
 因此我们可以用这个Revision回朔任意时刻Repository的状态，  
 就像时间机器一样，也就是说某一Revision  
 就是Repository在某一时刻的一个“快照”。  
 注意：Revision不是针对某一个文件或者目录，  
 而是针对整个Repository而言的。  
 每修改一次Repository，Revision 都会增加1。   
 Subversion的版本控制模型是一种叫做Copy-Modify-Merge  
 (拷贝-修改-合并)的模型。  
 考虑这种情况：  
 张三和李四是公司同一个部门的同事，  
 他们共同维护一个文本文件a.txt，  
 并且对该文件进行版本控制，  
 因此他们把这个文件放到一个Repository上共同维护该文件。  
 周一上午9点，张三和李四同时想对a.txt文件进行修改，  
 于是他们同时从Repository上取得该文件的最新版本(Revision 10)，  
 然后进行修改。过了三分钟，张三首先完成了修改，  
 他在该文件的第五行修改了一个单词的拼写(将Typo改为Type)，  
 于是张三对修改后的文件执行Commit命令，  
 将修改提交到服务器端的Repository中。  
 这时Repository的Revision变为11。  
 六分钟过后，李四也完成了他的修改，  
 他修改了该文件第十行上的一个单词拼写(将He改为She)，  
 于是他也对修改后的文件执行Commit命令，  
 这时Subversion 在提交修改时会发现，  
 李四修改的文件是Revision10的a.txt文件，  
 而不是最新的Revision 11的a.txt文件。  
 于是，Subversion 提示李四在提交修改前，  
 应该先将Working Copy更新到最新版本，  
 李四执行Update命令将Working Copy更新到Revision 11，  
 这时Subversion会提示已经完成合并，  
 李四的a.txt文件的第五行的“Typo”已经变为了“Type”，  
 第十行还是“She”，就是说Subversion已经将张三的修改“合并”到李四的a.txt文件中了。  
 之后，李四再执行Commit命令，就能将他对第十行的修改(将He改为She)  
 提交到服务器端的Repository中了(生成Revision 12)。  
 但是这种合并在某些情况下会变得复杂一些，  
 比如：李四对a.txt文件的修改并不是第十行，  
 而是与张三同样修改第五行的单词，  
 李四将“Typo”改为“Typr”，并且提交修改，  
 这时Subversion会提示李四在提交修改前，  
 应该先将Working Copy更新到最新版本，  
 李四执行Update命令将Working Copy更新到Revision 11，  
 这时Subversion将Revision11的a.txt文件与  
 李四修改的a.txt文件进行合并时发现李四修改的同样是第五行，  
 于是Subversion就无法判断是李四的修改(“Tpyr”)  
 正确还是张三的修改(“Type”)正确，  
 因为他们都是在Revision10的a.txt基础上作的修改。  
 这种情况叫做Conflict(冲突)，  
 a.txt文件的图标会变成一个黄色三角。  
 这时，只能依靠李四自己去判断到底第三行应该修改为“Typr”还是“Type”。  
 当李四确定修改之后，在a.txt文件上单击右键，TortoiseSVN->Resolved   
 告诉Subversion已经解决了Conflict。  
 这时再执行Commit命令就能提交修改(生成Revision 12)。  
 Subversion 这种控制方式保证了你对文件所作的修改都是基于文件的最新版本。   
  
 8.“.svn”目录   
 在客户端Working Copy的每一层目录中都会有一个“.svn”目录，  
 该目录是Subversion进行管理用的目录。  
 不要手动修改其中的文件。  
 该目录存储了Working Copy的一个副本  
 (实际存储副本的地方是F:\project1\.svn\text-base目录)，  
 比如：F:\Project1是一个Working Copy，  
 该目录下有两个文件a.txt和b.txt还有一个子目录ccc，  
 子目录ccc中还有一个d.txt文件。  
 “.svn”目录中存储的是你最近一次执行完Update或者Commit命令之后当前目录中文件的副本，  
 比如：F:\project1\.svn\text-base中存储的a.txt和b.txt  
 是最近一次执行完Update或者Commit命令之后F:\project1下的a.txt和b.txt的拷贝。  
 也就是说你所作的修改都是基于“.svn”目录存储的那些文件。  
 这种机制可以让我们在不连接网络的情况下，  
 将Working Copy中的文件恢复到修改之前的状态。  
 Subversion的Revert命令就是利用了这种机制来实现的。  
 比如你修改了F:\project1\a.txt文件，  
 这时你又改变了主意想放弃对该文件的修改，  
 你可以单击右键，TortoiseSVN->Revert，  
 修改过的F:\project1\a.txt文件  
 就会被F:\project1\.svn\text-base中a.txt文件的副本所替代，  
 使得a.txt恢复到修改前的状态。   
 Working Copy中每一个子目录下都会有一个“.svn”目录，  
 并不是只有最上层目录才有“.svn”目录。  
 所以，F:\project1\ccc下也有一个“.svn”目录，  
 该目录存储的是F:\project1\ccc\d.txt的副本  
 (d.txt的副本位于F:\project1\ccc\.svn\text-base)。  
 也就是说每个“.svn”目录只存储同级目录中的“文件”副本，  
 而不存储“目录”副本。“.svn”目录存有许多重要的内容，  
 所以前面说在删除文件或目录时，  
 必须用TortoiseSVN->Delete，  
 而不能用“Delete”键来删除文件或目录，尤其是对于目录的删除。   
  
 9.混合版本   
 Subversion的Working Copy被设计成一种能够包含不同版本的文件共存的形式。  
 比如F:\Project1是一个Working Copy，  
 该目录下有两个文件a.txt和b.txt。  
 执行Update命令，将Working Copy更新到最新版本（Revision 24）。  
 这时，a.txt和b.txt的Revision都是24  
 (其实对于单个文件来说并不存在Revision，  
 Revision是对于整个Repository而言的，  
 这里所指的是Repository的Revision24所存储的a.txt和b.txt，  
 但为了方便而采用这种描述方式，请注意，下同）。  
 之后，你的同事修改了a.txt，并且提交了修改，  
 这时Repository的Revision就变成25了。  
 注意，这时你没有再次执行Update，  
 因此你的Working Copy的Revision还是24。  
 这时你修改了b.txt文件，并提交修改。  
 因为Revision25并没有对b.txt文件进行修改，  
 因此你对b.txt文件的修改是基于b.txt文件最新的版本，  
 所以不会出现Conflict。  
 当你提交b.txt的修改后，产生Revision26。  
 这时你会发现你的Working Copy中的a.txt文件并不是Revision25中的a.txt文件，  
 它还是Revision24的a.txt文件，而你的b.txt文件是Revision26的b.txt文件。  
 也就是说当你Commit时，你的Working Copy中只有你提交的那些文件是最新版本，  
 而其他没有修改的文件并不会更新为最新版本。  
 这样就造成了你的Working Copy由不同的Revision文件所组成  
 (Revision24的a.txt文件和Revision26的b.txt文件)。  
 前面说过在提交修改前必须保证你是在文件的最新版本基础上修改，  
 如果在这种混合版本的情况下，  
 怎样才能知道当前Working Copy中的文件是否为最新版本？  
 在前面所说的“.svn”目录中有一个文件名为“entries”的文件，  
 该文件记录了当前Working Copy中的每一个文件的Revision，  
 因此当你Commit时，Subversion会从该文件中取得你提交文件的Revision，  
 再与Repository的最新Revision一比较就可以知道你修改的文件是否基于该文件的最新版本。   
  
 10.文件的锁定   
 前面说过Subversion的版本控制模型是一种叫做Copy-Modify-Merge  
 （拷贝-修改-合并）的模型。  
 该模型在对文本文件进行版本控制时工作的很好，  
 但是有些需要进行版本控制的文件并不是文本文件，  
 比如说图像文件，这种模型在这种情况下就不能正常工作了，  
 因为文本文件可以合并，而二进制文件则无法合并。  
 所以Subversion从1.2开始支持一种叫Lock-Modify-Unlock  
 （锁定-修改-解锁）的版本控制模型。  
 在Windows下最常用的版本控制软件Visual Source Safe(VSS)就是采用这种模型。  
 这种模型要求在对一个文件修改前首先要锁定这个文件，  
 然后才能修改，这时，别人将无法对该文件进行修改，  
 当修改完后再释放锁，使其他人可以对该文件进行锁定，然后修改。  
 锁定文件的方法是：TortoiseSVN->Get Lock...再点OK按钮，  
 这时就完成了对文件的锁定。  
 这时，如果其他人想对文件进行锁定时，  
 Subversion会对他提示该文件已经被别人锁定。  
 当你修改完文件后，然后单击右键，SVN Commit...，  
 将修改提交，默认情况下，提交的时候就会对该文件解锁，  
 如果你想仍然锁定该文件，请在commit时弹出的对话框中选中keep lock复选框。   
  
 11.文件的附加属性   
 在Subversion中，每个文件可以拥有一种叫做附加属性的东西。  
 附加属性描述了该文件所拥有的一些特性。  
 Subversion已经预定义了一些附加属性  
 （这里只是指Subversion已经定义了一些附加属性的“名称”，  
 并不是指已经将这些属性附加在文件上了，  
 比如默认情况下文本文件一开始不含任何属性，  
 直到人为的对该文件添加附加属性），  
 并且你可以对文件添加自定义的属性。  
 Subversion对待附加属性就像对待文件内容一样，  
 当修改了一个文件的附加属性（添加，改变，删除附加属性），  
 即使没有对文件的内容进行修改，  
 同样可以Commit该文件，就像更改了文件内容那样，  
 Repository也会生成新的Revision，  
 所以从某种意义上来说，  
 Subversion不区别对待文件的附加属性的修改和文件的内容的修改，  
 文件的附加属性可以看成是一种特殊的文件内容。  
 Subversion预定义了若干个附加属性，  
 这里只讨论“svn:needs-lock”属性，  
 因为它与我们上面的文件锁定会产生的一个问题有关。  
 其他的属性可以参考Subversion自带的帮助文档。  
 考虑这种情况，  
 张三和李四同时想对一个图片文件a.jpg作修改，  
 张三在修改时先将该文件锁定，然后进行修改，  
 同时李四也开始对该文件进行修改，  
 但李四忘记了对非文本文件进行修改时应该先锁定该文件。  
 张三首先对该文件修改完毕，于是张三向服务器提交了他的修改。  
 之后，李四也完成了修改，当他提交修改时，  
 Subversion提示李四的文件版本不是最新的，  
 在Commit之前应先更新a.jpg到最新版本，  
 由于图片文件无法合并，  
 这就意味着张三和李四之间必定有一个人的修改会作废。  
 应用“svn:needs-lock”属性可以避免这个问题。  
 当一个文件拥有“svn:needs-lock”属性时，  
 该文件在没有锁定时，文件的图标是灰色的，  
 表示该文件是一个只读文件（该文件的Windows只读属性的复选框为选中），  
 这个灰色的图标就会提醒想对该文件进行修改的人，  
 在修改该文件之前应该首先锁定该文件。  
 锁定该文件之后，文件的只读属性就会去掉了，  
 一旦释放掉锁，文件的图标又会变成灰色，  
 文件也会变成只读的了。  
 李四在这种情况下就会避免在没有锁定文件时对文件进行修改。  
 对非文本文件添加“svn:needs-lock”  
 属性应该在将该文件第一次添加到Repository时就设置，  
 当然，一个文件可以在任意时刻添加附加属性，  
 这样做是为了减少李四所遇到的那个问题发生的几率。  
 具体的方法是：  
 首先将a.jpg文件拷贝到Working Copy中，  
 然后在该文件上单击右键，  
 TortoiseSVN->Add，告诉Subversion要将该文件纳入版本控制，  
 接着在该文件上单击右键并选中属性，  
 在弹出的属性对话框中选中Subversion页。  
 在下拉框中选中“svn:needs-lock”，  
 并在下面的文本框中填入“\*”  
 （其实这里填什么都无所谓，只要文件有“svn:needs-lock”附加属性就行），  
 之后点Set按钮，“svn:needs-lock”附加属性就设置好了。  
 然后执行Commit命令提交修改。  
 这时当其他人执行Update时，  
 a.jpg就会添加到他们的Working Copy中，  
 并且文件的附加属性也会随文件一起被得到。  
 可以看到a.jpg此时的图标就是灰色的，  
 文件的Windows属性也是只读的。   
  
 12.回到以前的版本   
 由于Subversion会记录你对Repository的每一次修改，  
 因此能够很容易的获得Repository以前某一时刻的状态。  
 比如：现在Repository的最新Revision是56，  
 这时我想看看Repository在Revision24时的状态，  
 可以在本地的Working Copy中单击右键，  
 TortoiseSVN->Update to Revision...，  
 然后输入你想要回复到的Revision号，点OK按钮。   
 回到以前的版本还有一种情况是我想将Repository的  
 最新Revision的状态与以前某一个Revision的状态一模一样，  
 上面那种方法就不适合，  
 上面的那种方法只是将本地的Working Copy回复到以前的状态，  
 而服务器端的Repository并没有回到以前的状态。  
 将Repository的最新Revison的状态回复到以前某个Revision的状态具体的方法是：  
  
 先执行Update命令将Working Copy更新到最新的Revision，  
 然后在Working Copy中单击右键，  
 TortoiseSVN->Show Log，  
 弹出的Log Messages窗口中会显示该Repository的所有Revision，  
 选中最新的Revision，之后按住Shift键，  
 再单击你想回复到的Revision+1的那个Revision  
 （比如Repository的最新Revision是30，  
 你想将Repository的状态回复到Revision16，  
 那么就选中Revision30，再按住Shift键，  
 选中Revision17，  
 就是说选中Revision17到Revision30之间的所有Revision）。  
 然后在选中的Revision上单击右键，  
 选中“Revert changes from these revision”。  
 再点Yes按钮，就可以将Working Copy的状态回复到目标Revision。  
 注意，此时只是Working Copy回复到目标Revision，  
 之后应该用Commit提交修改，  
 这样Repository最新状态就与目标Revision的状态一样了。   
 这两种回复到以前版本的方式截然不同，  
 第一种方式是将整个Working Copy回复到某个Revision，  
 也就是说这种方式Working Copy中的“.svn”目录所存的文件副本也与目标Revision的一模一样，  
 如果这时你没有修改文件，你将不能执行Commit命令。  
 而第二种方式客户端Working Copy中的  
 “.svn”目录所存的副本始终是最新的Revision的文件副本  
 （这里我们基于一个假设：在Update之后没有其他人对Repository做修改）。  
 这种方式就像是我们自己手工将Working Copy的文件状态修改为目标Revision，  
 在修改之后提交修改一样。   
  
13.查看修改   
 有时我们对Working Copy的许多文件进行了修改，  
 这些文件位于不同的子目录，我们就可以在Working Copy的最上层目录单击右键，  
 TortoiseSVN->Check For Modifications，  
 弹出的对话框就会显示你所做的所有修改明细。   
 还有一种情况是我们的Working Copy已经很久没有执行Update命令，  
 我们想看看Working Copy中有哪些文件已经发生修改了，  
 这时就可以在Working Copy的最上层目录单击右键，  
 TortoiseSVN->Check For Modifications，  
 在弹出的对话框点击Check Repository按钮后，  
 就会显示服务器端已经修改了的文件。  
 该方法还有一个用途就是查看文件的锁定，  
 当你想锁定一个文件时，你想先看看这个文件有没有被别人锁定，  
 点击Check Repository按钮会显示服务器端Repository所有被锁定的文件，  
 如果你想锁定的文件不在这里面，那就说明该文件目前没有人锁定。