svn服务配置和基本应用

[简介]

svn(subversion)是近年来崛起的版本管理工具，是cvs的接班人。目前，绝大多数开源软件都使用svn作为代码版本管理软件。

本文通过具体实践，讲述了如何配置和建立最初的svn服务，如何使用svn基本命令，并对svn中比较关键的概念：版本号、分支和合并进行了介绍，最后通过一个具体的案例说明了为什么使用分支合并管理可以更高效。

[主要内容]

一、搭建和配置svn服务

二、svn基本命令帮助

三、分支和合并的使用

四、典型案例

五、其它

[详细内容]

一、搭建和配置svn服务

==========================

使用svn管理代码，需要配置好svn服务器，首先需要使用"svnadmin"命令在svn服务上面创建一个svn库，然后确定好需要使用svn管理的代码，最后通过svn的"import"子命令将未版本化代码目录导入版本库(根据需要创建中介目录)。具体步骤如下：

1,创建svn库:

$mkdir -p /home/quietheart/svn\_repos/test\_repos

$svnadmin create /home/quietheart/svn\_repos/test\_repos

这样,就创建好了原始的svn库test\_repos，

查看刚刚创建的库中什么如下：

$ls /home/quietheart/svn\_repos/test\_repos

conf  dav  db  format  hooks  locks  README.txt

2,将目录svn\_study纳入版本控制：

$svn import svn\_study file:///home/quietheart/svn\_repos/test\_repos/mystudy/test -m "Initial import"

这样，就把svn\_study中的内容纳入到版本控制系统中了，实际纳入的是拷贝且原始的test\_char\_driver不变。实际版本库也可以在别的机器上面，但是访问的时候就不能用file://来指定了。

运行之后，输出如下：

新增           svn\_study/test\_char\_driver

新增           svn\_study/test\_char\_driver/00\_widget\_dev

新增           svn\_study/test\_char\_driver/00\_widget\_dev/testDemo

新增           svn\_study/test\_char\_driver/00\_widget\_dev/testDemo/main.c

新增           svn\_study/test\_char\_driver/00\_widget\_dev/widgetdrv\_simple.c

新增           svn\_study/test\_char\_driver/00\_widget\_dev/readme

新增           svn\_study/test\_char\_driver/00\_widget\_dev/Makefile

新增           svn\_study/test\_char\_driver/00\_widget\_dev/widgetdrv\_api.h

新增           svn\_study/test\_char\_driver/00\_widget\_dev/widgetdrv\_simple.h

提交后的修订版为 1。

3,查看纳入到版本库中的文件：

$svn ls file:///home/quietheart/svn\_repos/test\_repos/mystudy/test

test\_char\_driver/

这里，注意我们纳入svn\_study实际上是把svn\_study中的子目录等内容纳入了，而svn\_study并没有在svn ls中看到。

如果使用ls查看版本库目录,会发现文件系统结构实际没有变化：

$ls /home/quietheart/svn\_repos/test\_repos/

conf  dav  db  format  hooks  locks  README.txt

至此，我们已经把svn\_study下的test\_char\_driver纳入到版本库中了，。

4,使用版本库：

在导入后，原来的目录树并没转化成工作拷贝，为了开始工作，需要运行svn checkout导出一个工作拷贝。然后可以使用svn的基本命令进行相关的操作了。这里给出一个简单的演示过程：

1)本地导出：

\*导出版本库的命令如下：

$svn checkout file:///home/quietheart/svn\_repos/test\_repos/mystudy/test

这样会把test导出，查看当前目录文件内容如下：

$ ls -a

.  ..  test

$ls -a test

.  ..  .svn  test\_char\_driver

可见，test内的内容纳入了版本控制。

\*查看当前版本库的日志：

[quietheart@lv-k test]$ cd test/

[quietheart@lv-k test]$ svn log

------------------------------------------------------------------------

r1 | quietheart | 2011-01-19 15:38:58 +0800 (三, 19  1月 2011) | 1 line

Initial import

------------------------------------------------------------------------

这里，就可以看到导入时候写的日志。

之后我们可以再这个工作拷贝上进行其他svn操作了。

2)从局域网其它主机导出:

2.1)在服务端运行：

$svnserve -d -r /home/quietheart/svn\_repos/

这里，服务段就是import整个svn库所在的主机(假设ip为10.1.29.44),-d表示守护进程，-r指定把本地的特定目录作为外部访问的根目录，这样安全。实际svnserve还可以指定端口号。

2.2)在客户端运行：

$svn checkout svn://10.1.29.44/test\_repos/mystudy/test

这里，test\_repos就是服务端svn\_repos的子目录，通过刚才svnserve的-r选项把服务器的/home/quietheart/svn\_repos/变成客户端访问的根，这样隐藏了细节，增加了服务器端的安全。另外，如果是windows分区，那么可能checkout的时候提示无法更改.svn的权限等，那么使用sudo svn checkout \*\*\*即可。"svn://"方式的访问速度快，但是安全的控制不太好（没有加密）。关于访问svn的url，本地可以使用"file:///""，其他主机使用"svn://"或者"http://"或者"svn+ssh://"的等等，svn的最快安全性最低配置最为简单，http的介于两者之间，ssh的最慢安全性最高.这里讨论的是"svn://"的url。

5,配置:

权限控制

\*启动服务进程：

$svnserve -d -r /home/quietheart/svn\_repos/

这里，服务段就是import整个svn库所在的主机(假设ip为10.1.29.44),-d表示守护进程，-r指定把本地的特定目录作为外部访问的根目录，这样安全。实际svnserve还可以指定端口号。

\*修改/home/quietheart/svn\_repos/test\_repos/conf/svnserve.conf:

添加如下行(实际仔细阅读该文件注释即可知)：

anon = none

auth-access = write

password-db = passwd

authz-db = authz

这里，注意要修改成"anon = none",默认的是"anon = read",否则后面svn checkout的时候会提示"svn: 未授权打开根进行编辑操作"这个错误。

\*修改/home/quietheart/svn\_repos/test\_repos/conf/passwd:

在[users]下添加如下行：

lv-k = test

\*修改/home/quietheart/svn\_repos/test\_repos/conf/authz

添加

[test\_repos:/]

lv-k = rw

这个表示访问的时候，给lv-k开启test\_repos库的根权限。当前需要注意的是这个"[test\_repos:/]"是基于前面那个"$svnserve -d -r /home/quietheart/svn\_repos/"命令的。

\*从别的主机将svn\_repos中相应的内容导出：

$sudo svn checkout svn://10.1.29.44/test\_repos/mystudy/test --username lv-k

不知为什么，没输入密码。

二、svn基本命令帮助

==========================

使用svn工具的时候，我们可以使用"svn help"来查询我们想要使用的功能。"svn help"命令列出了当前svn所支持的所有操作命令，这里大致介绍其输出的含义，更为详细的信息需要自行阅读"svn help <command>"的输出.

1,"svn help"命令

以下是"svn help"命令的帮助输出：

Available subcommands:

   add

   blame (praise, annotate, ann)

   cat

   checkout (co)

   cleanup

   commit (ci)

   copy (cp)

   delete (del, remove, rm)

   diff (di)

   export

   help (?, h)

   import

   info

   list (ls)

   lock

   log

   merge

   mkdir

   move (mv, rename, ren)

   propdel (pdel, pd)

   propedit (pedit, pe)

   propget (pget, pg)

   proplist (plist, pl)

   propset (pset, ps)

   resolved

   revert

   status (stat, st)

   switch (sw)

   unlock

   update (up)

Subversion 是个版本控制系统的工具。

2,"svn help"命令介绍

每个命令大致解释如下

(\*)add:

将一个未纳入版本库的目录或者文件添加到svn版本管理库中，add之后需要用commit才能体现在svn版本库中。

例如：

$svn add testfile

这样，svn 会将testfile纳入版本控制。

(\*)blame (praise, annotate, ann):

这个命令显示某个文件每一行的版本号以及修改者。

例如：

$svn blame testfile

输出大致如下：

   158       lv-k #!/bin/bash

   901       lv-k ########Notation for development#########

   901       lv-k #This script is made by XXX.

   901       lv-k #1, Comment with "#XXX" needs to be improved in the future.

   901       lv-k #2, If you "cd" to some path in sub\_function, you must "cd OLDPWD" before return.

   901       lv-k #########################################

   158       lv-k

   158       lv-k ########Global Variables#########

   901       lv-k #This value can be "develop" or "release" now.

......剩下的省略......

(\*)cat:

这个命令的功能类似shell的cat,用于查看文件的内容。

例如：

$svn cat testfile

之后会输出当前版本文件的内容。

(\*)checkout (co)

这个命令用于从版本库中取出指定版本的代码，默认为指定目录中当前版本库中最新代码。

例如：

$svn checkout -r 900 testfile

之后会从版本库中取出一个工作拷贝testfile，存放在本地目录中。取出的版本号是900，取出的内容是目录的话，会包含版本控制信息，可以在相应目录中运行特定svn命令。

(\*)cleanup

这个命令在本地文件被锁定的使用，用于清楚锁。

SVN 本地更新时，由于一些操作中断更新，如磁盘空间不够，用户取消。可能会造成本地文件被锁定的情况。一般出现这种情况可以使用cleanup来清除锁定。如果在根目录下都无法clean的话，一般采取的方法是另外找一个目录重新CHECKOUT,但有时有时SVN目录下可能有一些自己本地修改的文件，还未提交到SVN服务器，这时重新CHECKOUT需要注意本地文件的备份.

例如：

$svn cleanup

这样会清除当前锁定。

(\*)commit (ci)

这个命令用于将当前目录中本地所做修改提交到版本库中，同时生成一个新的版本。

例如:

$svn commit -m "<some information>"

这里，使用-m来指定提交的日志。

(\*)copy (cp)

这个命令用于将版本库中某个子目录进行拷贝，经常用于创建分支。这里的拷贝是廉价的拷贝，仅在版本库中保存每个分支拷贝中相应的变化而非全部，但检出的时候却可以将拷贝当做独立的工作拷贝来对待。copy动作需要commit才能提交到版本库中。

例如：

$svn copy trunk branches/tmp

(\*)delete (del, remove, rm)

这个命令类似linux的rm命令，用于将版本库中的某个文件或者目录删除。

例如：

$svn delete branches/tmp

(\*)diff (di)

这个命令查看某两个工作拷贝之间的修改。

例如：

$svn diff -r 901:902

输出内容类似如下：

Index: init.initrd

===================================================================

--- init.initrd （修订版 901）

+++ init.initrd （修订版 902）

@@ -5,7 +5,7 @@

mount -o rw -t proc proc /proc

#lvkai+{ XXX print time

-cat /proc/uptime

+#cat /proc/uptime

#lvkai+}

mount -t sysfs none /sys

@@ -23,7 +23,7 @@

mount -t vfat -o shortname=winnt /dev/sxs\_blk0 /SlotA\:

#XXX print time

-cat /proc/uptime

+#cat /proc/uptime

#XXX startup appliation

#/diablo/diabloMainCpu/diablo.bin&

这里，会比较901和902两个版本之间的不同，并把相应的修改输出到标准输出。

(\*)export

这个命令用于从版本库中取出指定版本的代码，默认为指定目录中当前版本库中最新代码,取出的信息不包含svn信息。

例如:

$svn export -r 900 testfile

这个命令之后会从版本库中取出testfile，存放在本地目录中。取出的版本号是900，取出的内容是目录的话，就如同一个没有纳入版本控制的目录一样不包含版本控制信息，不能在相应目录中运行特定svn命令。

(\*)help (?, h)

这个命令用于获取svn命令的帮助信息,默认列出所有命令的列表。

例如：

$svn help commit

这样会输出commit命令的帮助信息。

(\*)import

这个命令用于将指定的目录或者文件纳入版本控制，通常在最初建立svn库的时候使用。

例如：

$svn import svn\_study file:///home/quietheart/svn\_repos/test\_repos/mystudy/test -m "Initial import"

这样，会将svn\_study这个目录下的内容纳入到file:///home/quietheart/svn\_repos/test\_repos/mystudy/test中。这里，svn库的名称是test\_repos,同时提交的日志用-m指定。

(\*)info

这个命令用于显示指定路径的版本控制信息。

例如：

$svn info test

会输出如下：

路径：test

地址(URL)：file:///home/quietheart/svn\_repos/test\_repos/mystudy/test

Repository Root: file:///home/quietheart/svn\_repos/test\_repos

档案库 UUID：4ee27b8d-b954-4796-92c7-e0d337941423

修订版：9

节点种类：目录

调度：正常

最后修改的作者：quietheart

最后修改的修订版：9

最后修改的时间: 2011-01-20 17:46:35 +0800 (四, 20  1月 2011)

(\*)list (ls)

这个命令用于查看指定目录中，已经纳入版本控制的所文件，类似linux中的ls命令。

例如：

$svn ls

这个命令输出的内容，不一定是当前目录所有文件，而是当前目录中纳入版本控制的文件。

(\*)lock

这个命令用于某个用户将某个文件进行锁定.

例如：

$svn lock test

这里，锁定动作会从当前工作拷贝中自动传播出去，不用通知别人了.锁定之前，这个文件必须是版本库中最新版本的，否则锁定失败。锁定之后，其他用户工作拷贝中无法对此文件加锁，无法提交修改，也无法解锁（即使是同一用户名）.除非加锁的工作拷贝处解锁(可用unlock命令),或者别人使用--force选项的unlock强制解锁.实践发现加锁处提交修改之后自动解锁,锁定动作不能针对目录只能单个文件。

注意，可以使用"svn lock test --force"强制把锁抢过来。

(\*)log

这个命令查看某个工作拷贝的日志信息。

例如:

$svn log test

输入之后输出类似如下：

------------------------------------------------------------------------

r17 | quietheart | 2011-01-21 16:49:53 +0800 (五, 21  1月 2011) | 1 line

------------------------------------------------------------------------

r7 | quietheart | 2011-01-20 17:36:10 +0800 (四, 20  1月 2011) | 1 line

add test

------------------------------------------------------------------------

这里，每个日志中一个log说明，就是commit 命令中的-m选项指定的。

(\*)merge

这个命令用于将某个分支在某个阶段的修改应用到指定的目标上。命令翻译为合并，实际是应用修改的意思，类似打补丁。

例如：

$svn merge -r 6:7 file:///home/quietheart/svn\_repos/test\_repos/mystudy/test/trunk /home/quietheart/test/svn\_study/test/branches/quietheart\_branch/

这样，会把主干上版本6:7上面的修改应用到分支上面。然后可以用svn commit将合并的结果提交。注意，这里目标是/home/quietheart/test/svn\_study/test/branches/quietheart\_branch/，它是本地工作拷贝的目录。

(\*)mkdir

这个命令用于为版本控制系统中增加一个目录,类似linux中的mkdir命令。

例如：

$svn mkdir test1

这里，添加的目录必须用commit提交，通知到版本库。

(\*)move (mv, rename, ren)

这个命令用于在版本库中移动文件，或者给某个文件命名，类似linux中的mv命令。

例如：

$ svn move good test2

输入之后，输出如下：

A         test2

D         good

这里，类似用add添加一个目录test2，然后把原来的目录用delete删除。必须用commit提交到版本库别人才能够看到你的动作。

(\*)propdel (pdel, pd)

这条命令???

(\*)propedit (pedit, pe)

这条命令???

(\*)propget (pget, pg)

这条命令???

(\*)proplist (plist, pl)

这条命令???

(\*)propset (pset, ps)

这条命令???

(\*)resolved

这个命令用在解决冲突之后。

例如：

$svn resolved test

这里，发生冲突的文件就是test.当你提交时候，svn会自动update一下将版本库最新版本合并到你的工作拷贝，如果有人修改了和你同样文件的同样行，那么就产生冲突，解决冲突之后(如果不删除冲突备份文件)，使用这个命令就可以继续提交了，如果删除冲突备份文件，那么不用这个命令也行。

(\*)revert

这条命令会撤消当前没提交的本地工作拷贝的修改，假设修改了一个文件，再用这个命令，就可以把修改撤消。

例如：

$svn revert test

这里，可以结合status命令进行合适的撤消操作。

注意：本子命令不会存取网络，并且会解除冲突的状况。但是它不会自动恢复被删除的目录，除非你指定了那个被删除的当前看不见的文件名称。

(\*)status (stat, st)

这条命令用于在提交之前查看当前当前工作拷贝中哪些文件被修改，添加或者删除了，哪些没在版本控制之下。

例如：

$svn status

会输出类似如下：

?      test4

D      test1/testtest

M      test3

A      test5

这里，?表示test4不在版本控制之下，D表示提交时会再版本控制库中删除testtest文件，M表示test3做了修改，A表示test5是将要被添加的文件。

如果在上述命令继续运行,输出如下：

$ svn revert \*

已恢复“test3”

跳过“test4”

已恢复“test5”

$svn status

?      test4

?      test5

D      test1/testtest

$svn revert test1/testtest

$ svn revert test1/testtest

已恢复“test1/testtest”

$svn status

?      test4

?      test5

可见revert不是智能地恢复被删除的文件。

(\*)switch (sw)

这条命令用于在主干工作拷贝，各个分支工作拷贝之间自由切换（节省了checkout一个独立的分支或者trunk的时间和空间）；以及在svn服务器改变了之后，将本地工作拷贝的服务器地址更新。

例如：

$ svn info

路径：.

地址(URL)：file:///home/quietheart/svn\_repos/test\_repos/mystudy/test/branches/quietheart\_branch

Repository Root: file:///home/quietheart/svn\_repos/test\_repos

档案库 UUID：4ee27b8d-b954-4796-92c7-e0d337941423

修订版：19

节点种类：目录

调度：正常

最后修改的作者：quietheart

最后修改的修订版：10

最后修改的时间: 2011-01-20 17:49:19 +0800 (四, 20  1月 2011)

$svn switch file:///home/quietheart/svn\_repos/test\_repos/mystudy/test/trunk

D    test

A    test1/testtest

A    test2

U    test3

$ svn info

路径：.

地址(URL)：file:///home/quietheart/svn\_repos/test\_repos/mystudy/test/trunk

Repository Root: file:///home/quietheart/svn\_repos/test\_repos

档案库 UUID：4ee27b8d-b954-4796-92c7-e0d337941423

修订版：19

节点种类：目录

调度：正常

最后修改的作者：quietheart

最后修改的修订版：19

最后修改的时间: 2011-01-21 17:12:25 +0800 (五, 21  1月 2011)

上面，原来我在工作分支中工作，现在想要切换到trunk主线上面，那么使用switch可以很快地将当前工作拷贝变成trunk，而不用另外checkout整个trunk主线了。

(\*)unlock

这条命令用于解锁lock命令锁住的文件。

例如：

$svn unlock test

(\*)update (up)

这条命令用于更新工作拷贝，将当前路径工作拷贝更新为svn库中相应路径最新版本。

例如：

$svn update

三、分支和合并的使用

==========================

比较高效的利用svn提供的各种功能，需要对svn的版本号，分支，合并有所了解。

所谓版本号，是指整个 svn 库的版本号，而不是哪个工作拷贝，那个工作目录的版本号。

所谓分支，就是一个拷贝，可以对库内任意子目录进行拷贝，然后在于主线互不影响的前提下对这个拷贝进行任意修改，需要的时候可以把任意两个时间之内的修改合并到主线上。需要注意的是：

（１客户端检出的时候，可以把检出的分支当做独立的 svn 库来开发，不会影响开发主线。分支和主线为一共享的就是版本号，以及分支之前的开发日志。

（２分支拷贝，是廉价的拷贝，它只保存创建分支以来修改的内容（却可以当做独立的库来看待。不用的分支完全可以删除，也可在任何时候找回。

所谓合并，就是把某个分支（或者主干上，某个期间进行的修改动作（即两个版本号之间相对于该分支上的修改，应用到任意分支（或者主线上，应用的结果变成了新的版本。需要注意的是：

（１合并的时候，要保证本次合并的内容和其他人合并的内容没冲突。如果开发前安排每个开发人员所做的修改完全是在他们自己独立的模块中的修改的话，就不会出现交叉修改的情况，也不会有冲突。svn 提供的机制，可以实现冲突可以在合并之前检查出来，也可以在发生冲突之后解决。

（２合并的时候，最好记录本次应用的是合并操作，这样可以防止多次合并同一个动作。

使用svn进行代码管理，推荐使用如下的版本库布局：

/trunk

/branches

/tags

以下是前面创建好版本库的基础上，使用分支和合并来管理代码的简单例子。通过这个例子，可以对分支和合并一个简单直观的了解。

\*查看当前目录结构如下：

$pwd

/home/quietheart/test/svn\_study/test

$tree

.

|-- branches

|-- tag

`-- trunk

    |-- busybox-1.4.2.tar.bz2

    `-- test\_char\_driver

        `-- 00\_widget\_dev

            |-- Makefile

            |-- readme

            |-- testDemo

            |   `-- main.c

            |-- widgetdrv\_api.h

            |-- widgetdrv\_simple.c

            `-- widgetdrv\_simple.h

$du -sh

2.0M

$cd /home/quietheart/svn\_repos/test\_repos

$du -sh

1.8M

$cd $\_

\*创建一个分支：

$ svn copy trunk/ branches/quietheart\_branch

A         branches/quietheart\_branch

$ svn status

A  +   branches/quietheart\_branch <=====这里'+'表示是用svn copy创建的一个工作拷贝。

$ tree

.

|-- branches

|   `-- quietheart\_branch

|       |-- busybox-1.4.2.tar.bz2

|       `-- test\_char\_driver

|           `-- 00\_widget\_dev

|               |-- Makefile

|               |-- readme

|               |-- testDemo

|               |   `-- main.c

|               |-- widgetdrv\_api.h

|               |-- widgetdrv\_simple.c

|               `-- widgetdrv\_simple.h

|-- tag

`-- trunk

    |-- busybox-1.4.2.tar.bz2

    `-- test\_char\_driver

        `-- 00\_widget\_dev

            |-- Makefile

            |-- readme

            |-- testDemo

            |   `-- main.c

            |-- widgetdrv\_api.h

            |-- widgetdrv\_simple.c

            `-- widgetdrv\_simple.h

$du -sh

7.0M

$cd /home/quietheart/svn\_repos/test\_repos

$du -sh

1.8M

$cd $\_

由上可知，创建分支，虽然在本地工作拷贝中，文件系统中文件的大小变化了，但实际上，再svn库中大小没有多大变化，所以svn的创建分支是“廉价”的拷贝。

\*合并主干的内容到分支：

$svn merge -r 6:7 file:///home/quietheart/svn\_repos/test\_repos/mystudy/test/trunk /home/quietheart/test/svn\_study/test/branches/quietheart\_branch/

这样，会把主干上版本6:7上面的修改合并到分支上面。然后可以用svn commit将合并的结果提交。注意，这里目标是/home/quietheart/test/svn\_study/test/branches/quietheart\_branch/，它是本地工作拷贝的目录。

实际最后一个参数可以省略，这样会以当前路径作为合并目标，运行如下：

$cd /home/quietheart/test/svn\_study/test/branches/quietheart\_branch

$svn merge -r 6:7 file:///home/quietheart/svn\_repos/test\_repos/mystudy/test/trunk

四、典型案例

==========================

举一个例子，假设某一个项目使用svn进行管理。

１，原来的管理状况是：

1.1,svn库的分部情况如下：

1）建立一个用于release的svn库（本地）。

2）建立一个用于develop的svn库（本地）。

3）一个对客户的release库（在客户的主机上）。

1.2,使用这个体系，开发的过程大致如下：

1）平时开发的时候，一切动作都在develop库上面做。

2）发布的时候，各个模块组手动把develop库上面的验证好用的修改动作“复制”到本地release库上面。

3）release担当手动把各个模块放到本地release库上面的本次修改动作，再次“复制”到客户的release库上面。

4）为了保持本地develop库和本地release库严格一致（以便以后发布时差异太大不好复制），release担当还要对比着把本地release库“回盖”到本地develop库。

1.3,问题是：

1）这个步骤手动做，工作量比较大，容易出错，容易导致遗漏（遗漏才会增加４这个步骤防止不一致性）。

2）这个步骤需要复制到客户release上面，工作量也不小，也许是不可避免的。

3）这个步骤完全是为了防止２这个步骤会导致develop库与release库不一致，额外增加的，工作量也不小，且它本身也容易出错。

另外，需要维护多个独立的svn库，占据不少的空间，还不方便跟踪历史记录。

２，改进的方案：

2.1,svn库的分部情况如下：

1）建立一个svn库，其开发的主线（trunk）就等价于前面的本地release库。（本地）

2）为svn库建立一个分支，例如(branches/develop)，其作用就相当于前面的本地develop库。（本地）

3）建立一个对客户的release库（在客户的主机上，这一点和前面不变）

2.2,使用改进的方案，开发的大致过程如下：

1）平时开发的时候，都从分支branches/develop上面导出代码，然后每个组基于这个本地develop分支进行开发。

2）发布的时候，各个模块组把develop分支上的相对trunk主线的修改“merge“到trunk上。

3）release担当手动把各个模块放到trunk主线上面的本次修改动作再“复制”到客户的svn上面。

4）这个步骤是可选的，为了保持develop库和release库严格一致，可以把brahches/develop删掉，然后再给trunk主线重新创建一个新的branches/develop分支。

2.3,改进之处是：

1）这个步骤，如果每个模块修改的地方相互独立，那么完全可以自动化地merge到trunk上；如果相互重叠，也容易查出哪里修改重叠了，并且相应手动修改（手动修改的内容很少）。

2）这个步骤仍然不可避免。

3）这个步骤，也不用release担当动做了，当2）这个步骤做完，担当可以什么都不用管（或者只检查一下即可），让以后继续再这个develop分支上面开发；如果不确定，担当完全可以删掉这个develop分支，重新给trunk创建一个develop分支。

另外，

\*只需要维护一个svn库，便于历史跟踪，或者创建其他分支等管理动作。

\*根据分支的概念，每次开发者和以前一样当做独立的开发库来看待从分支检出的代码，绝对不会影响release开发主线的代码。

\*开发者可以完全不考虑改进的变化，完全按照改进之前的开发过程进行。

\*虽然可能会创建不少分支，但是由于svn内部的实现机制，这些分支本身几乎不会占用空间,比使用本地release库和本地develop库，两个独立的开发库省下不少空间。

\*完全保留了所有的历史记录（包括可以找出曾经删除的分支）,便于跟踪。

以上绝大多数所动作，svn都有专有的命令来实现，同时也可以选择自己手动进行。

五、其它

==========================

(暂无)

一个好的参考资料：http://i18n-zh.googlecode.com/svn-history/r734/www/svnbook-1.4/index.html

作者：QuietHeart

Email：quiet\_heart000@126.com

时间：2011年6月27日