
配置管理

学习 CLEARCASE 补充读物 软件技术文档

V0.4

Status: Draft

Last Change: 14-Jan-2004

SydongSun@hotmail.com

Copyright by SYDONGSUN



作者: GuoYu
邮件: Sydongsun@hotmail.com
网站: www.askguoyu.com

作者的个人技术交流网站: <http://www.askguoyu.com> 从这个网站上你可以了解到有关软件配置管理相关工具的一些学习资料。

特别感谢 Green，只有我才可以感受到来自于 Green 的莫大的鼓励和支持。

Copyright © Sydongsun 2004. All rights reserved. For internal use only.

目录表

目录表	3
配图列表.	4
1 说明	6
1.1 阅读目标	6
2 一些基本概念	7
2.1 VOB 的概念.	7
2.2 VIEW 的概念	8
2.3 VIEW 的 Config Spec	10
2.4 Check In 和 Check Out	10
3 使用 CLEARCASE 的基本流程	11
3.1 首先安装 CLEARCASE Client 端软件	11
3.2 创建一个视图.	11
3.3 Mount VOB	13
3.4 元素受控, CHECH IN, CHECH OUT, Version Tree 等等.	14
3.5 VIEW Config Spec 和工作空间	20
3.6 标签和分支	25
3.6.1 标签 (label).	26
3.6.2 分支 (branch)	28
3.7 分支和归并	31
3.8 工作空间管理	34
4 进一步的内容	40
4.1 几个要注意区分的命令 (rmname/rmver/rmelem/mv)	40
4.2 进一步理解 -- 目录元素的版本	43
4.3 自我检查 ClearCase 客户端出现的问题.	44
4.4 进一步的培训	44

Copyright © Sydongsun 2004. All rights reserved. For internal use only.

配图列表

Figure 1:	VOB 和 VOB 中的元素	7
Figure 2:	VIEW 和 VOB 的关系	8
Figure 3:	VOB、VIEW 和 VIEW Config Spec 的关系	10
Figure 4:	创建视图的过程——1	11
Figure 5:	创建视图的过程——2 选择动态视图	12
Figure 6:	创建视图的过程——3 给视图命名, 指定盘符 (一般缺省)	12
Figure 7:	创建视图的过程——4 选择存放路径 (必须要是本计算机共享的一个目录)	12
Figure 8:	Mount VOB 的过程——1	13
Figure 9:	Mount VOB 的过程——2	14
Figure 10:	视图下的 VOB 的目录结构	14
Figure 11:	将视图下的文件受控, 存放为 VOB 的元素	15
Figure 12:	将 VOB 下的元素的版本 Check Out --- 取得修改该版本的修改权 ...	15
Figure 13:	Reserved Check Out --- 具有优先的 Check In, 产生新版本权限 ..	16
Figure 14:	Check In --- 为元素产生新的版本	16
Figure 15:	Check In --- 和上一个版本没有变化, 也产生一个版本	17
Figure 16:	查看元素的图形化的版本树信息	17
Figure 17:	版本树图 (请读者和图 1 对照, 进一步理解 VOB 存放有版本纪录的元素)	18
Figure 18:	CLEARCASE Explorer 的使用 (比 Windows Explore 更好)	19
Figure 19:	视图的 Config Spec	20
Figure 20:	不同的开发者进行并行开发	22
Figure 21:	了解附带版本信息的文件名	22
Figure 22:	基于某个开发基础, 生成的适用不同国家的产品开发分支, 做并行开发	23
Figure 23:	一个典型的元素 MyFile.c 的版本树图	24
Figure 24:	浏览 VOB 中存放的元数据	25
Figure 25:	给元素的视图中下显示的版本附加标签	26
Figure 26:	指定 label 的范围是 Global 的 (Branch type 有类似的属性设定)	30
Figure 27:	Merge 操作过程 ----1	31
Figure 28:	Merge 操作过程 ---- 2	32
Figure 29:	Merge 操作过程 ---- 3	32
Figure 30:	例子项目 ShenZhou10 的 COM_VOB 中的分支类型	35
Figure 31:	开发人员在 dev_branch 分支上进行开发活动	35

Figure 32:	需求跟踪系统或者代码审核人员在 major_branch 分支上进行活动 ...	36
Figure 33:	反映开发、集成测试、发布版本等活动的一个元素的版本树图	37
Figure 34:	在 RELEASE1.00 的基线上进行继续开发.....	39
Figure 35:	CLEARCASE 四大功能版本控制、工作空间管理、开发流程控制、构建管理	40
Figure 36:	CC_Explorer 中的 rmname 和 mv 的图形菜单命令	42
Figure 37:	自己检查计算机上的 ClearCase 的四个后台服务是否正常运行.....	44

1 说明

本文为开发人员学习 CLEARCASE 的一个补充文档。更详细的文档请参照 CLEARCASE 软件安装目录下的 PDF 文档。为了开发人员学习 CLEARCASE 的针对性，而制作本参考文档。本文档主要聚焦于：

1. 开发人员所应该具有的对 CLEARCASE 的了解：VOB 和 VIEW 等重要的概念的理解；
2. 开发人员使用 CLEARCASE 的工作流程；开发人员如何利用 CLEARCASE 来建立自己的工作空间；VIEW Config Spec 的了解以及分支(Branch)和标签(Label)的运用；

为了进一步增加对 CLEARCASE 的了解，特别加入了某些内容，这些内容对于 CLEARCASE 的管理员以及 CM 人员所应该熟悉的内容，但并不一定需要开发人员来掌握。这些内容采用褐色字区分。

1.1 阅读目标

通过本文档的阅读，开发人员应该掌握下列内容：

一些基本的概念：

比如 MVFS；元素的版本；CHECK OUT；CHECK IN；分支；归并；标签；

一些比较关键的概念：

VOB；VIEW 和 VIEW 的 Config Spec；

掌握使用 CLEARCASE 的基本流程，以及如何创建自己的私有分支；

一些问题的解决办法。

我们最终的目标是：软件配置管理是软件工程中的重要内容。工欲善其事，必先利其器。让我们一起掌握 CLEARCASE 这样的一个可以提高我们团队开发效率、开发质量和优化开发流程控制的配置管理工具。

2 一些基本概念

开始时，需要提到一些基本概念。这些基本概念是进一步阅读的基础，但也并不意味着要完全理解，有个大体的印象就可以了。在第二部分**开发人员使用 CLEARCASE 的工作流程**中会进一步体现这些概念。读者在阅读第二部分的时候，应该自己动手去试验，争取大部分图片都能在你的试验当中能够显示类似的图片来。

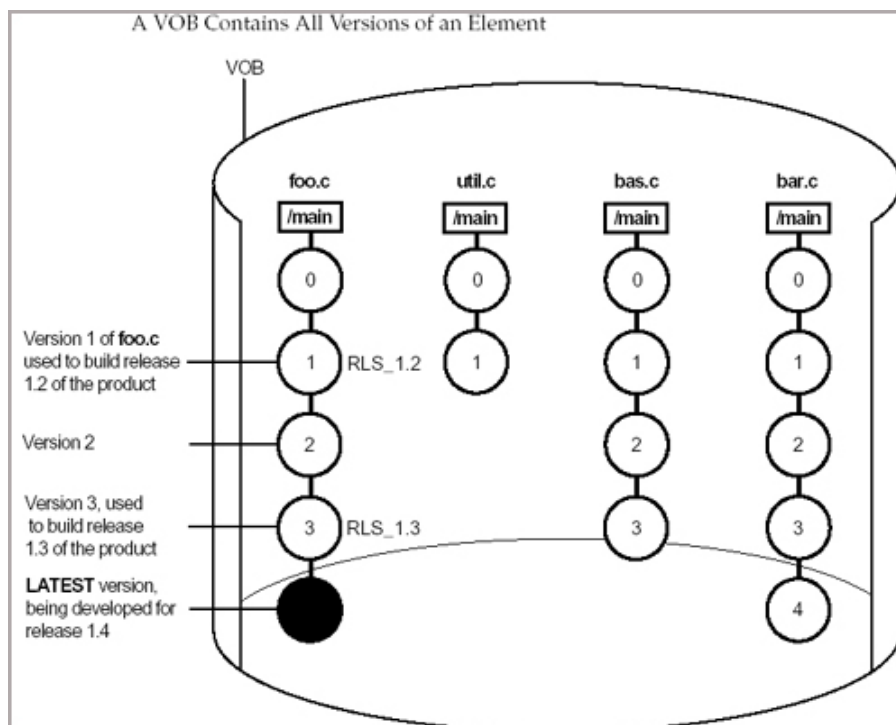
2.1 VOB 的概念

VOB 的全称是 Version Object Base,版本对象库，是 CLEARCASE 特有，特定的数据库系统，当中存放的内容具有版本的概念，保存和记录开发者的历史记录，可以让开发者回溯到任意时期，任意版本的开发阶段。VOB 库中除了用于存放这些需要**版本**历史记录的**元素**之外，还需要一些用来更好的组织，描述那些具有版本历史纪录的元素，附加于元素之上的其他内容，提供给元素更多的特性，用来支持**并行开发**，**权限控制**。

如果开发者对关系型数据库有所了解的话，可以作一个类比。比如关系型数据库除了存放最为重要的用户业务数据记录之外，也需要一些存放一些 SQL 程序，触发器等等。

图 1 说明了 VOB 库的逻辑概念：该 VOB 库中包含了四个文件元素，每个文件有个 main 的主干分支，该分支上有代表元素不同版本的数字。该图来自于 CLEARCASE 的帮助手册，但是作者还是觉得该图并不够完美(但懒于去画一个很好的)：其中只有文件元素，还没有目录元素。在 CLEARCASE 当中，元素 (Element) 是具有版本纪录的对象，包括了文件 (file) 和目录 (directory)，目录的版本就是目录下的文件增加和删除等变化。

Figure 1: VOB 和 VOB 中的元素



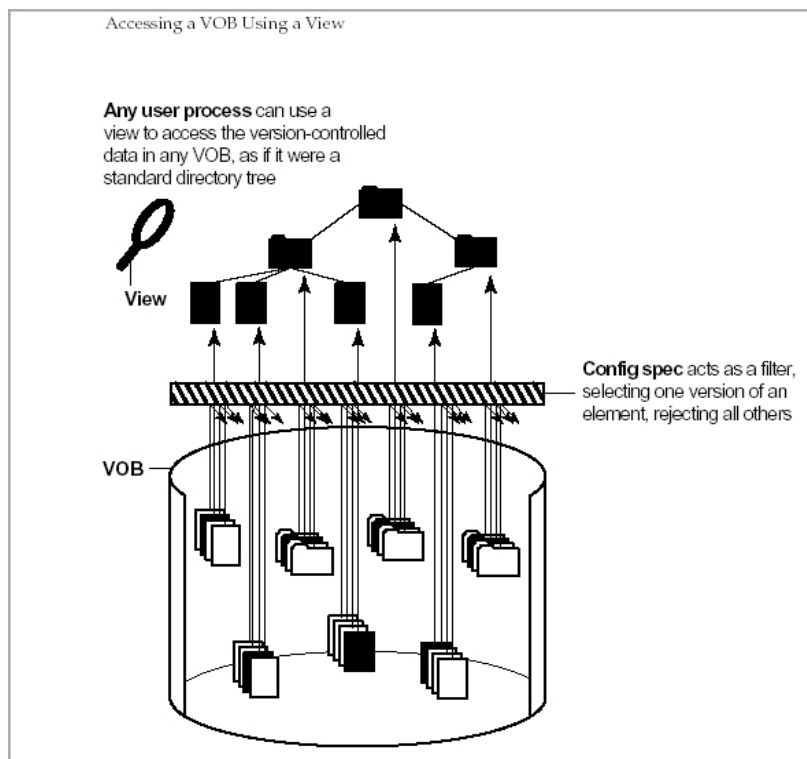
2.2 VIEW 的概念

VOB 一般是配置管理人员按照项目的配置管理计划在 VOB 服务器上创建，根据项目的大小，决定所创建的 VOB 的大小，在大的项目中，一个子系统模块对应一个 VOB。开发人员也许并不关心这些内容，还是更关注于：我如何将我的文件和代码放入到 VOB 中实现版本控制呢？我怎么样才能够观察，处理 VOB 库中的内容？

如果读者熟悉某些关系型的数据库系统，比如 Oracle 等，Oracle 提供了除了 SQL 语句之外，也提供了视图（VIEW）来帮助使用者观察数据库中的数据记录。读者也非常熟悉相机，通过调节相机的镜头（VIEW）来观察景物。类似的，CLEARCASE 中通过 VIEW 来观察、操作 VOB 中的内容。

图 2 形象的说明了 VIEW 工作时的逻辑含义：

Figure 2: VIEW 和 VOB 的关系



VIEW 通过某些规则来获取 VOB 中元素中的某个版本，并组织成操作系统中的目录结构。在 CLEARCASE 中通过 MVFS(Multi-Version File System)在 Windows 的操作系统中会为 VIEW 创建一个虚拟的盘符（本文假定采用 Windows 操作系统平台，创建的视图为动态视图）。从 VIEW 的盘符下，我们可以观察元素的某个选定的版本，并可以将原来不在 VOB 中的文件“受控（Add to Source Control）”到 VOB 中，所以 VIEW 是开发人员的工作空间。在图 2 中，只是说明了从 VOB 选择元素特定版本的一种关系，在 VIEW 中，还可以存在视图私有文件，也就是我们还没有将这些元素受控到 VOB 中，VOB 中还不存在这样的元素，只存放在视图的存储池（View Storage）。

顺便提一下，客户端安装好 CLEARCASE 之后，在程序菜单中，就会有一个创建视图的向导。视图一般创建在用户的计算机上的一个共享目录下（View Storage Directory），因为 CLEARCASE 的用户权限直接来自于 Windows 操作系统的域用户（活动目录，这里假设在 Windows 平台上使用 CLEARCASE）。CLEARCASE 本身的进程需要一个自动在域中创建的用户身份来运行，同时也为了其他的开发者（一般会是一个组）来访问你的视图。所以该目录的共享权限至少要包括这些用户的权限，如何设定这个共享目录的权限是配置管理员的工作。

2.3 VIEW 的 Config Spec

视图的 Config Spec 是视图的一个属性，但是将它单独拿出来讲，是因为它的重要性。Config Spec 是一个“高级”的“滤镜”，说是滤镜是因为它可以帮我们选择 VOB 中元素的某个指定的版本来供我们观察和操作。说是“高级”，是因为滤镜的规格，可以由一些简单的语句编写组成，非常灵活，功能强大。缺省的规格是：

```
Element * CHECHEDOUT
```

```
Element * /main/LATEST
```

注释：第一句中 CHECHEDOUT 用来表示当 VOB 中的元素某个版本被 Check Out，取得该版本上修改权，视图应该首先选择这个版本。如果第一句得不到元素的某个版本，那就执行下一句，该语句选择元素的 main 分支上的最新版本。

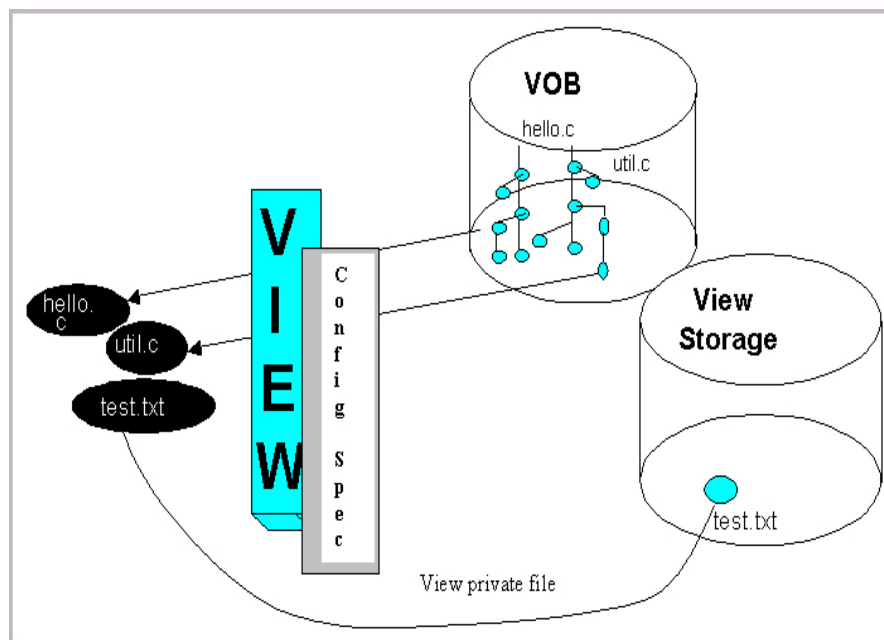
在复杂项目的开发中，这些缺省的规则是不够用的，我们要在这里两个语句中间增加其他的内容，让滤镜的功能更“高级”。更具体的精彩内容，让我们在以后的内容中提到。毕竟，一开始就展示太多的内容，显得有些急不可待了。

2.4 Check In 和 Check Out

前面提到：VOB 库中存放具有版本历史记录的元素，比如拿文件元素来说，文件将具有多个版本。这些版本当然是由开发人员来产生。在 ClearCase 中（包括其他版本管理工具），如果我们要修改其中的元素，首先要做 Check Out 的操作，意味着取得文件的修改权；当修改完成了，通过 Check In 的操作，将你的修改保存到 ClearCase 的 VOB 库中，并形成一个新的版本。

让我们用一个图来作为第一部分“一些基本概念”的结束语吧：

Figure 3: VOB、VIEW 和 VIEW Config Spec 的关系



3 使用 CLEARCASE 的基本流程

有了前面的三个重要的基本概念的介绍，具体该如何一步步的做？将是读者希望得到的答案。建议读者将本部分的内容作为一个试验手册，练习手册来对待，还不是像阅读小说一样的快速，请要求你的配置管理员提供一个试验 VOB，做为你的练习场地。

3.1 首先安装 CLEARCASE Client 端软件

一般来说，公司的配置管理人员要为开发人员建立一个关于安装的手把手的文档说明，让有关人员自己来进行操作，这样的文档也有助于减轻配置管理人员的维护工作量。同时在这些文档中也可以附带交代一些重要的注意事项。这里省略如何安装 Client 端软件的说明。

3.2 创建一个视图

在“开始”---“程序”中，找到 ClearCase 的菜单项中的 Create View 向导程序，点击后如下：这里假定以 BASE 方式使用 ClearCase，并创建动态视图为例。

Figure 4: 创建视图的过程—— 1

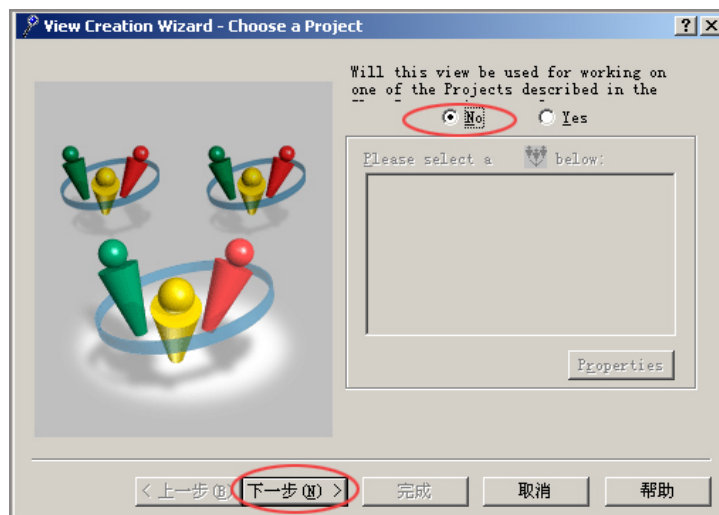


Figure 5: 创建视图的过程——2 选择动态视图

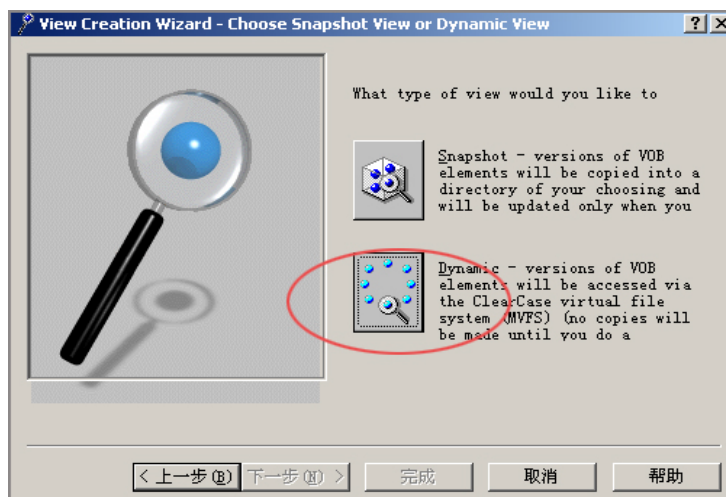


Figure 6: 创建视图的过程——3 给视图命名, 指定盘符 (一般缺省)

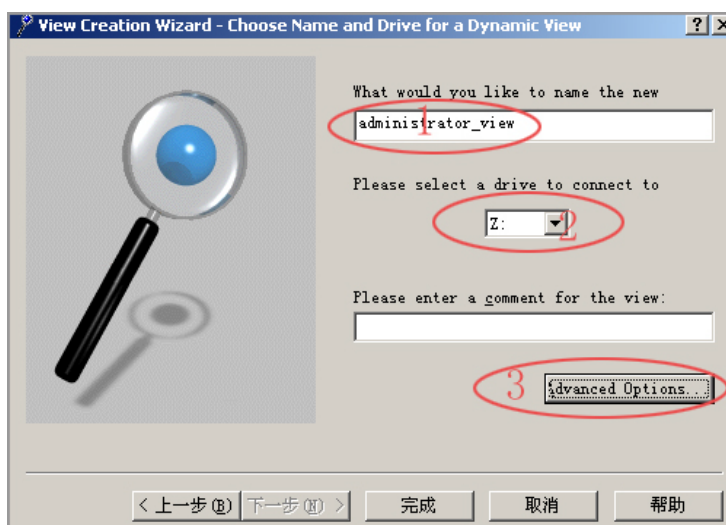
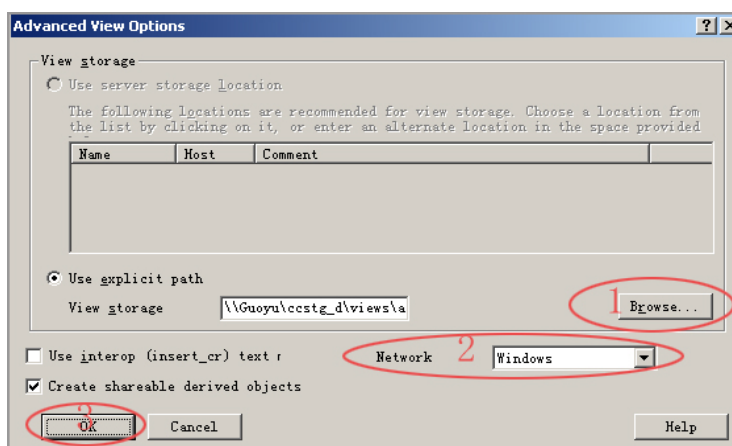


Figure 7: 创建视图的过程——4 选择存放路径 (必须要是本计算机共享的一个目录)



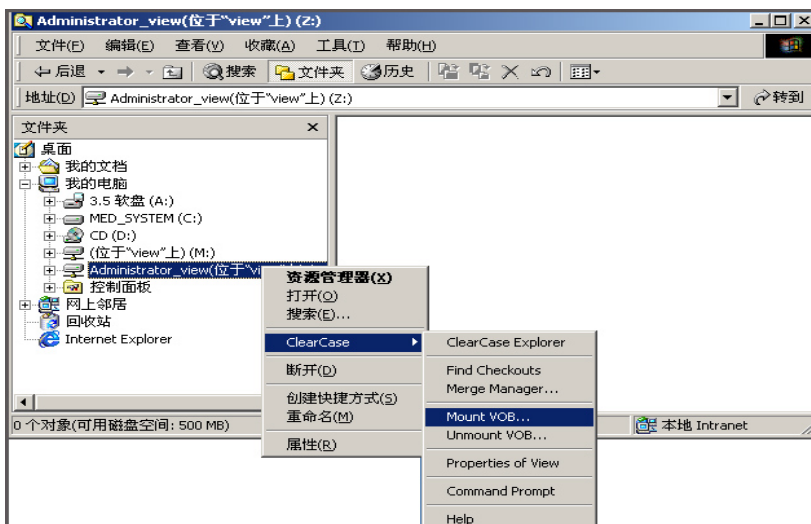
在这里顺带提一下图 7 中的第 2 个红圈标注内容，此处的 Network 是 ClearCase 中的一个特别的含义。在 ClearCase 中为了支持大规模开发活动，比如在一个公司当中同时进行不同的项目的开发，这些项目基本上是相互独立的。为了隔离这些不同的项目和对应的开发人员。在 ClearCase 中可以建立一些 Region，让不同项目的 VOB 归于不同的 Region，用户建立视图的时候，也可以在图 7 的 Network 中选择不同的 Region，那么视图 Mount VOB 的时候，只能看到该 Region 下的所有 VOB，通过 ClearCase 的 Region (Network) 来隔离不同的，不相关的项目。图 7 中使用了缺省的 windows Region。该值要咨询配置管理员，有时候，你的配置管理员发布的 ClearCase Client 安装程序会自动设置好该值。

最后就是点击图 6 中的“完成”按钮，然后在出现的一个反馈信息窗口中点击确定，直到创建视图成功。

3.3 Mount VOB

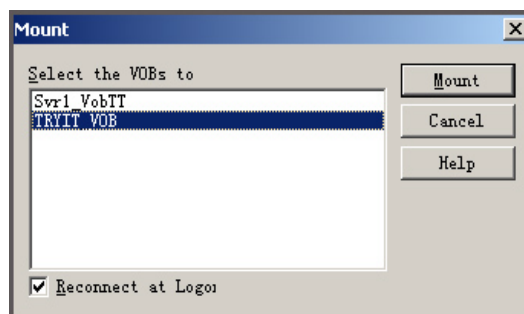
Mount VOB 意味着将 VOB Server 上的 VOB 关联到你的视图当中，通过视图来“观察”、“操作”VOB 中的内容。（补充提一下，相对应的，有 Unmount VOB，我们在做 Unmount VOB 的时候，应该要保证 VOB 的元素应该没有还处在 Check out 状态的元素）。

Figure 8: Mount VOB 的过程——1



下图中有两个 VOB，我们只 Mount 一个，在一个大的项目中，一个模块将会成为一个 VOB，在大项目中，就会有很多 VOB 组成。

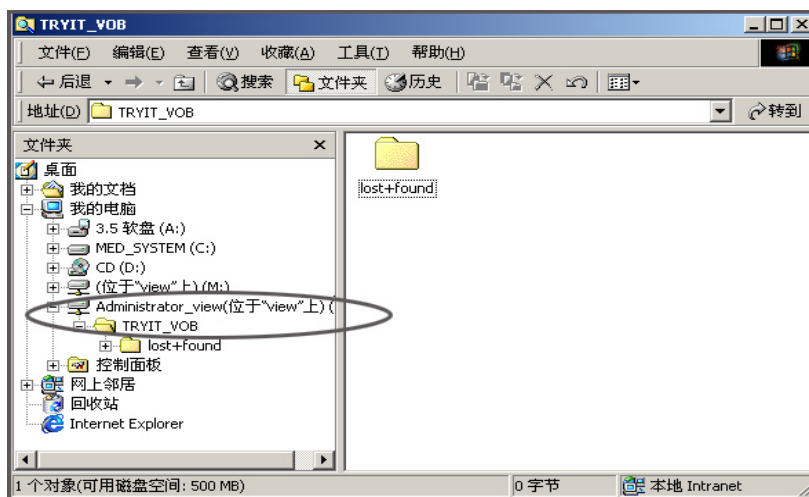
Figure 9: Mount VOB 的过程—— 2



3.4 元素受控, CHECH IN, CHECH OUT, Version Tree 等等

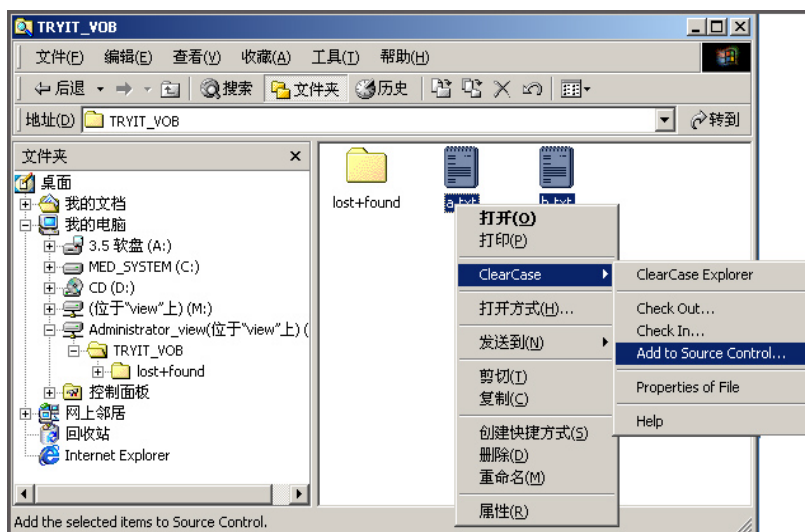
现在在视图 Administrator_VIEW 下有一 TRYIT_VOB 的 VOB 目录, 但是该 VOB 中只有一个 lost+found 的目录, 这个目录有些类似于 Windows 操作系统的 Recycle Bin 类似, 但是要记住你永远不要像 Windows 操作系统一样, 选中文件, 然后按 " Delete " 键。前面, 我们说过, VIEW 的含义是 " 取景器 ", 我们通过 VIEW 来操作和观察 VOB 的内容。和我们一般的文件系统不一样。所以, 也就意味着, 这个 VOB 当中还没有内容, 只有 lost+found 目录。

Figure 10: 视图下的 VOB 的目录结构



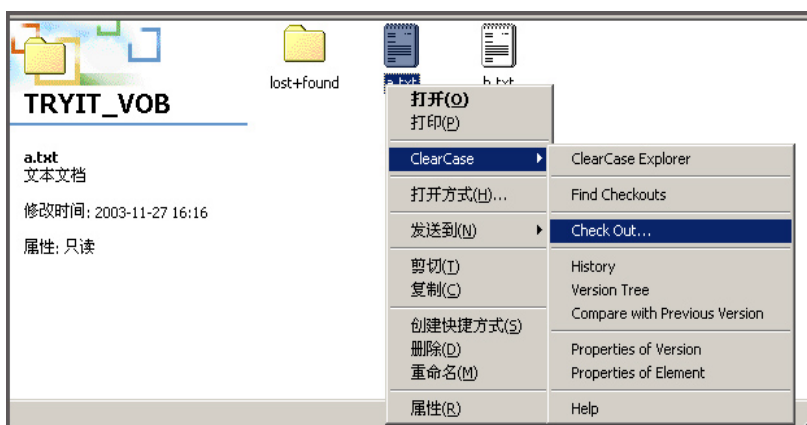
拷贝两个文件, a.txt 和 b.txt 到该 VOB 中, 然后选中它们, 受控(Add to Source Control)。受控意味着将视图下的视图私有文件保存到 VOB Server 的 VOB 库中。

Figure 11: 将视图下的文件受控, 存放为 VOB 的元素



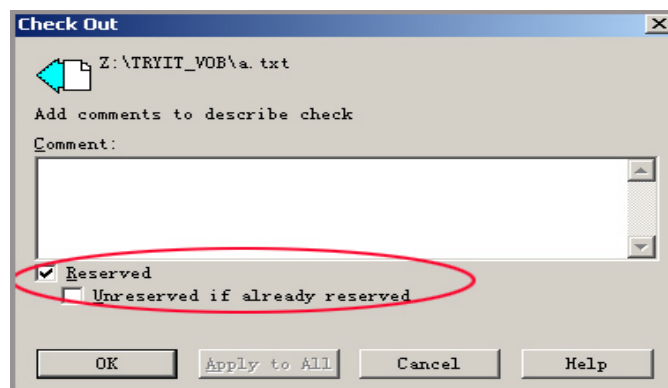
元素受控之后, 当我们右键点击某个元素, 则在 CLEARCASE 的菜单, 就会出现 check in / check out 的选项。对一个元素对 Check in 和 Check out 使该元素的状态发生变化, Checkout 后 (会去掉元素的只读属性), 意味着你可以对该元素作修改 (但是要记住, 不能做**直接改名**的操作, 但是你可以用其他同名文件覆盖。**直接改名**, 指的是使用 Windows 的重命名; VOB 中的元素的改名, 需要通过 CLEARCASE 的所提供的文本命令或者图形方式下的菜单命令)。

Figure 12: 将 VOB 下的元素的版本 Check Out --- 取得修改该版本的修改权



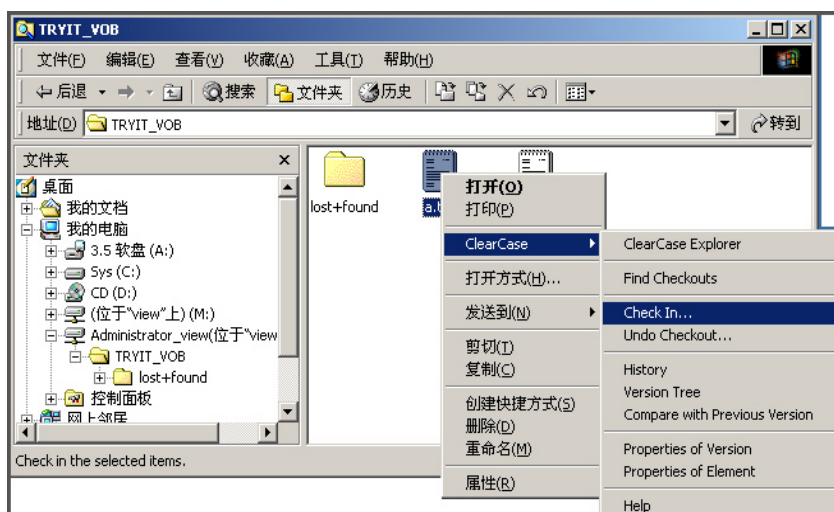
在 Checkout 元素的窗口中有个 "Reserved" 的选项, 如果选中 Reserved Checkout (保留检出), 表示当元素只有等待你 check in 之后, 别人才可以 check in, 所以你一定会最先为元素产生一个新的版本。而其他做 Check out 的时, 只能为 Unreserved Check out, 他们对该元素作修改, 但是必须要等到你做 check in 之后, 他们才可以做 check in 的操作。

Figure 13: Reserved Check Out --- 具有优先的 Check In，产生新版本权限



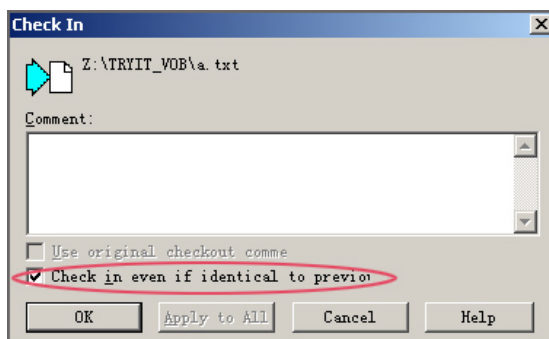
修改完成后，然后 Check In，作 Check In 的动作，将在 VOB 中将为该文件产生一个新的版本。

Figure 14: Check In --- 为元素产生新的版本



在 Check In 的动作时，有个选项 " Check in even if identical to prevision ", " 即使同前一个版本没有作改动，也可以 Check In "，这很容易理解，CLEARCASE 认为如果文件没有变化，就没有必要生成新的版本。但我们常常也为元素生成新的版本。

Figure 15: Check In --- 和上一个版本没有变化，也产生一个版本



当一个文件或者目录，经过多次 Check Out/Check In 的操作之后，就会生成很多的版本。
CLEARCASE 能够跟踪到我们元素的所有版本。这就是我们使用 **CLEARCASE 的版本管理** 的功能。下图就显示一个元素 a.txt 的版本树图，非常形象。

Figure 16: 查看元素的图形化的版本树信息

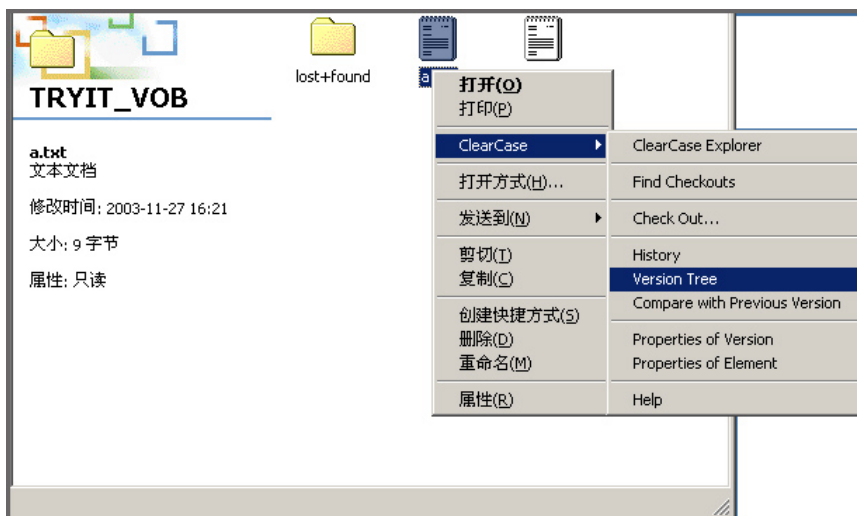
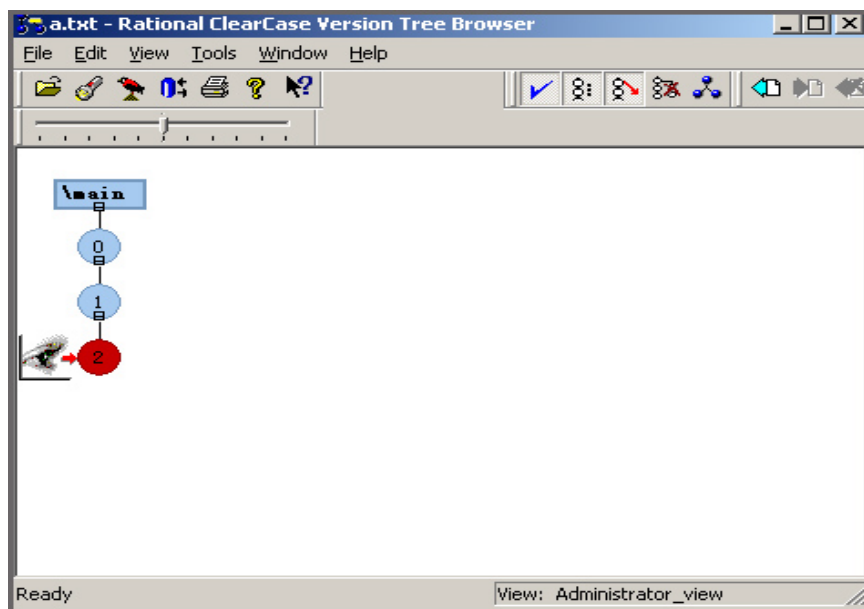
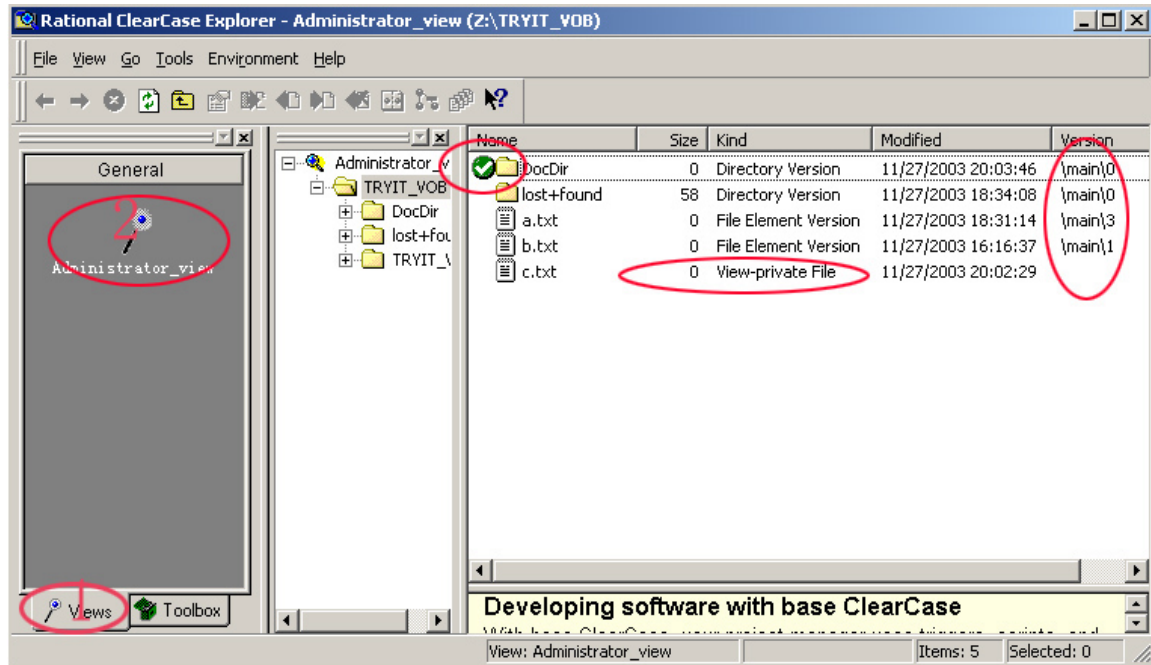


Figure 17: 版本树图 （请读者和图 1 对照，进一步理解 VOB 存放有版本纪录的元素）



在这里，补充一下 **CLEARCASE Explorer**。刚才我们使用的是 Windows Explorer。CLEARCASE 提供了具有功能更体贴的 CLEARCASE Explorer。如下图 1 8 所示：从这个浏览器，我们看到哪些元素是 VIEW-Private File(表示该文件在服务器的 VOB 中不存在)，哪些元素被 Checkout（图中勾标志为 Checkout 元素），还可以看到当前视图选择的各个元素的版本（我们的取景器定位在什么位置）。通过 CLEARCASE Explorer 来看视图，先点击 VIEWS，在 Views 的标签页中右键菜单中有 " Add VIEW ShortCut " 选项，通过向导选项，在 CLEARCASE Explorer 中增加视图的快捷方式。（建议配置管理人员在对开发者进行支持的时候，总是尽量使用 CLEARCASE Explorer，给予开发者潜移默化的影响。）

Figure 18: CLEARCASE Explorer 的使用（比 Windows Explore 更好）



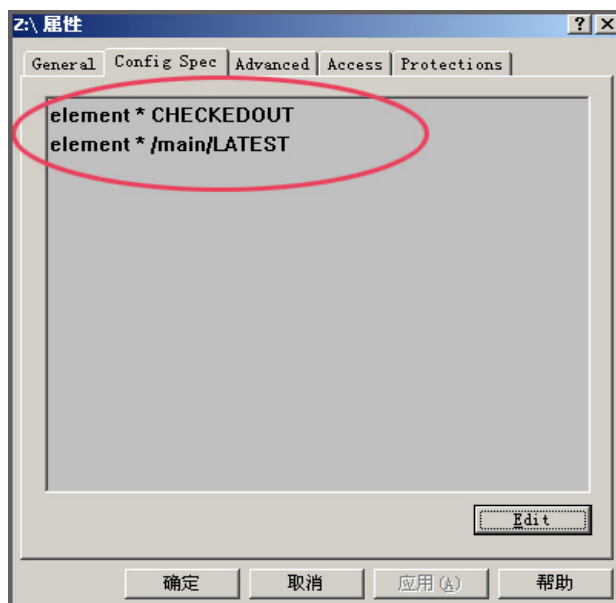
3.5 VIEW Config Spec 和工作空间

我们观察：在前面的元素 a.txt 的版本树图中（图 1 7），为什么我们的“眼睛”会停留在 a.txt 的最新的版本上呢？如果，我想看 a.txt 的第一个版本怎么办呢？你可以右键直接点击版本树图的 1 版本，然后选择“Check Out”之后，“眼睛”会停留在版本 1 上，此时，你打开 a.txt 就看到版本 1 的情况。如果仔细想想，我们好像会觉得会有某个标准在限定我们的眼睛的位置。VIEW 作为照相机的“取景器”，也有调节相机的焦距来决定我们选择景物。同样的道理。CLEARCASE 的 VIEW 也具有这样的功能，就是 VIEW Config Spec；同时 VIEW Config Spec 也提供了更多的其他功能。

视图建立在开发者的本地计算机上。不同的开发者都一一分别建立自己的视图，然后通过各自的视图的 Config Spec 来看 VOB server 计算机上的 VOB。同摄影同样的道理，众多的摄影者可以以自己的技术视角去看模特。CLEARCASE 通过视图来隔离不同开发者的工作空间。前面我们提到的 Mount VOB 和 Unmount VOB，将 VOB 和 VIEW 建立关联，也可以类比为：将模特（VOB）请到 T 台上，供镜头（VIEW）表现。

用右键点击视图，在菜单中有“Properties of VIEW”，进入可以看到如下窗口：

Figure 19: 视图的 Config Spec



上面的 Config Spec 是缺省的语句：Config Spec 中的语句，从上到下执行（废话！），如果某一句能够得到元素的版本选择结果，则停止执行，不再执行下面的版本选择语句。其中的*号，表示对所有内容都是起作用的（作用域）。

```
Element * CHECKEDOUT
Element * /main/LATEST
```

相应的简单的解释：

第一句：表示选择被 Checkout 的元素的版本；这一句总是在最前面；

第二句：表示选择元素的 main 主干分支的最新版本。当我们通过“Add to Source Control”将文件或者目录放入到 VOB 中时，就会成为 VOB 库中的元素，VOB 的含义是 Version Object Base，其中的元素是具有版本的概念，总会有一个 main 的主干，可以让大家在做 check in 的时候，在 main 上形成一个个的版，如图 1 6 所示。所以本语句可以看作是一个垃圾收集器。总是能够看到元素的 main 分支上的最新内容。

较多情况下，ConfigSpec 中的语句分为三个部分，

第一部分，元素类型选择部分，我们几乎始终使用 Element；

第二部分，作用域（或者模式匹配），我们常用“*”来匹配所有的元素路径，另外还有比较多见的是，用类似于“\CC_TEST\train_folder\...”来匹配所有的目录路径的前部分，也就是限定特定的目录下的内容。另外“*.txt”“\CC_TEST\test.txt”等等之类也是可以的，但是很少如此用。

第三部分，就是版本选择。比如当我们在元素的版本上附加了标签 REL2 后，我们可以用 REL2 来作为版本选择的依据，另外如“.../mybranch/LATEST”表示选择 mybranch 分支下的最新版本，不管该分支建立的位置情况，“...”（三个点），表示一种模式匹配，比如

/main/mybranch/LATEST

/main/testbranch/mybranch/LATEST

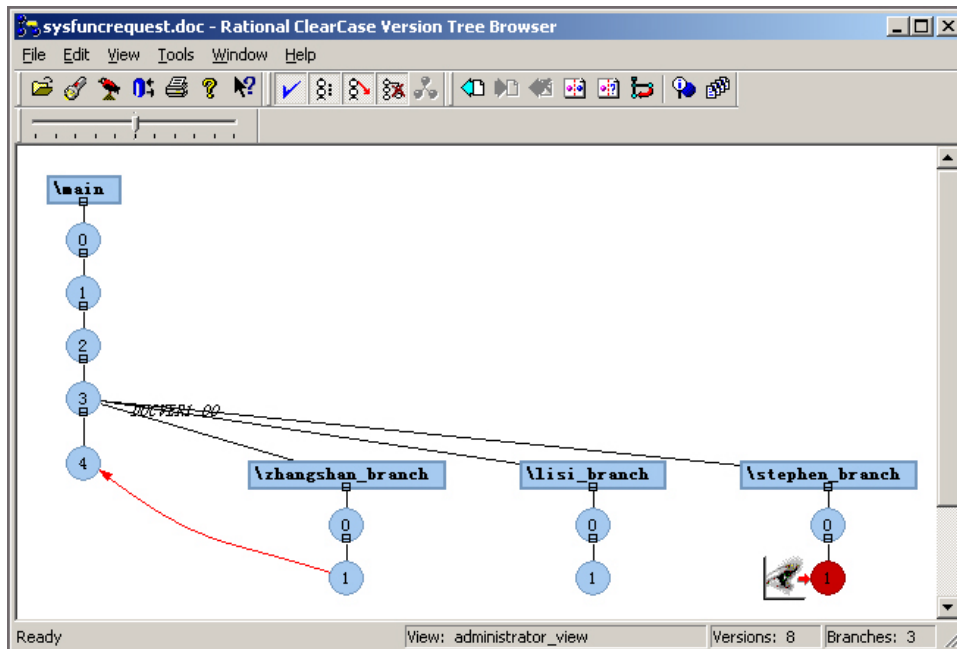
都可以用“.../mybranch/LATEST”表示和匹配。

请读者在这个地方思考一下了：假如一个文件在该文档的生命周期中，会有多个人对它进行修改，假如只有一个 main 主干上产生版本。就会比较混乱。

再比如：假如一个产品会同时进行不同版本的开发。比如通讯系统的短消息系统，这种产品的购买地区都会需要一些比较客户化的版本；再比如医疗系统的某些产品，因为不同国家和地区的技术要求和规范以及客户的不同，也需要同时进行多个版本项目的开发；而在企业管理软件中，有时候需要为一个企业客户进行特定的开发活动，同时你的公司有好几个这样的企业客户项目。你希望在同样的 VOB 库中进行不同版本的开发，你主要考虑到在开发过程中，这些产品的不同的版本在同时进行，需要随时引入其他版本的开发成果。

图 2 0 和图 2 2 也许就是你希望的答案：

Figure 20: 不同的开发者进行并行开发



对同一个文件 sysfuncRequest.doc，三个开发者分别从 DOCV1.00 版本上拉出自己的分支，各自在自己的分支上进行修改。当修改到完成后，可以归并到 main 主干上生成新的版本。上图是 stephen 的工作视图下的情况，stephen 视角正关注于 stephen_branch 上的第一个版本。下图显示了 stephen 所看到的描述版本的文件名：**sysfuncrequest.doc@@\main\stephen_branch\1** 读者此时应该理解：**sysfuncrequest.doc** 在 VOB 当中是代表一个元素的名称，而如果你要知道这个元素的内容的话，应该需要明确指定版本信息的文件名（Version-Extended Pathname）。

Figure 21: 了解附带版本信息的文件名

```

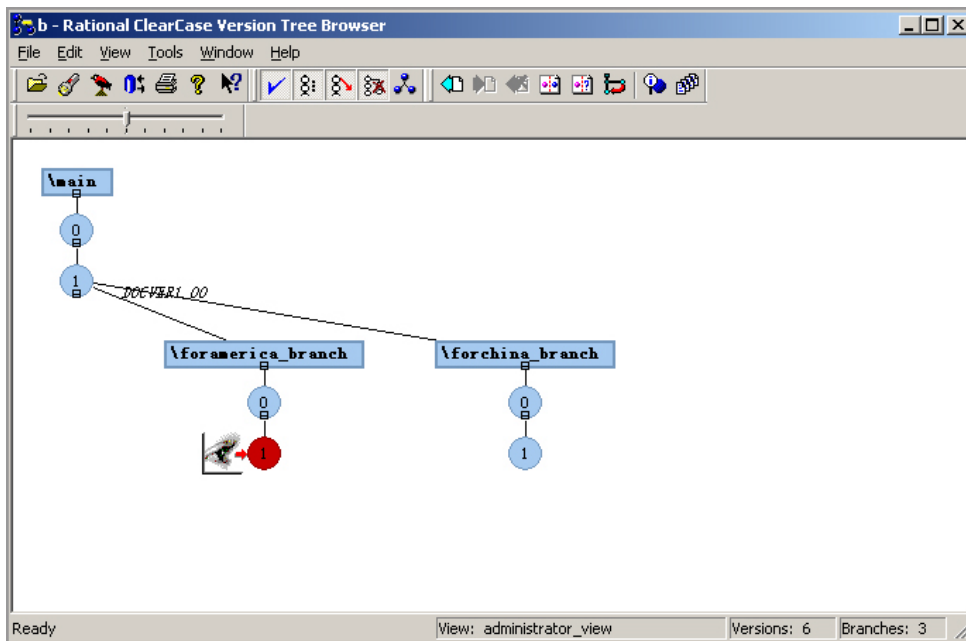
C:\WINNT\system32\cmd.exe

Z:\TRYIT_UOB>cleartool ls
a.txt@@\main\4                                Rule: DOCVER1.00 -mkbra
nch stephen_branch                             Rule: DOCVER1.00 -mkbra
b@@\main\1                                     Rule: DOCVER1.00 -mkbra
nch stephen_branch                             Rule: DOCVER1.00 -mkbra
lost+found@@\main\0                           Rule: DOCVER1.00 -mkbra
nch stephen_branch                             Rule: \main\stephen_bra
sysfuncrequest.doc@@\main\stephen_branch\1
nch\latest

Z:\TRYIT_UOB>
    
```

图 2 2 可以表示目录 b 在有关系的同一个项目的不同的版本(目录同样是 VOB 中的元素, 也具有版本, 目录的版本就是目录下的文件增加, 删除等历史)。一个版本用于中国市场, 另外一个版本用于美国市场。该图所用的视图的 Config Spec 为开发美国市场的版本的开发人员的视图。

Figure 22: 基于某个开发基础, 生成的适用不同国家的产品开发分支, 做并行开发



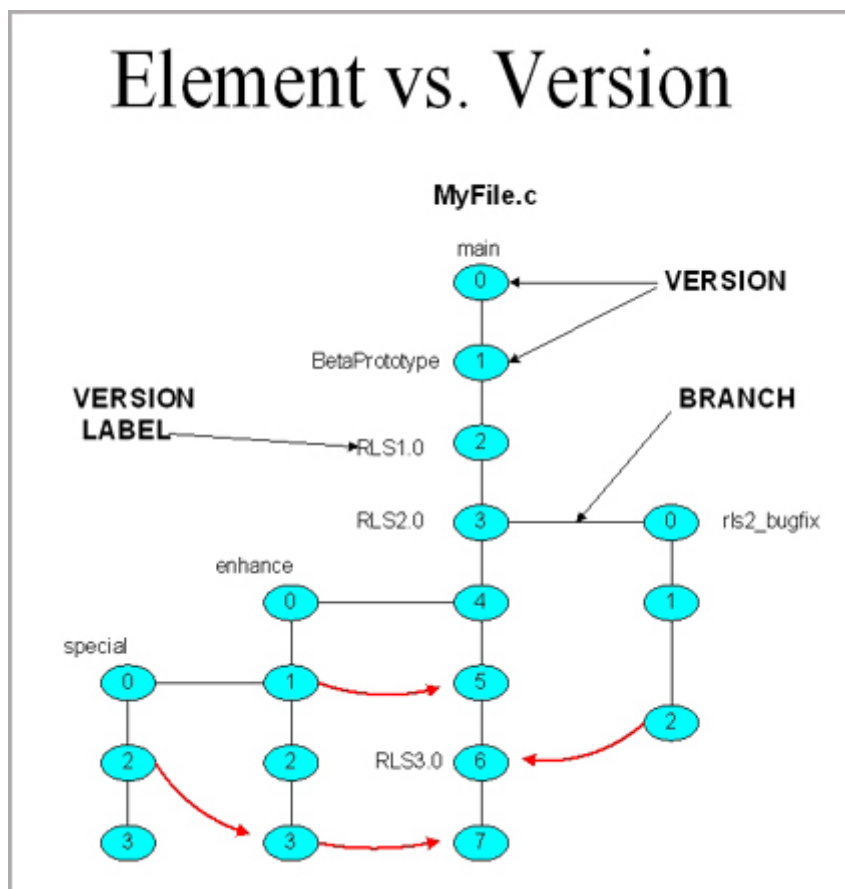
通过上述内容, 读者对于视图、视图的 Config Spec, 以及分支有了基本概念上的大致的了解。不同的开发小组或者开发人员使用不一样的 View Config Spec, 从而让他们工作在不同的分支上, 形成各自的“工作空间”, 来支持并行开发。各自的 View 的 Config Spec 决定了工作空间: 工作的文件和文件的版本(也包括目录)。有了这些基本概念的了解后, 我们来进行“如何作”, “如何更好的去做”的话题, 我们将开始一些需要集中你的注意力的内容了。主要聚焦于:

- 一. 标签和分支;
- 二. 分支和归并;
- 三. 工作空间管理和规划; (主要针对配置管理人员)

并且在这些内容当中会提及到一些用于命令行操作的 CLEARCASE 的命令。在进一步学习新的内容之前, 请先看图 2 3, 提出一些问题:

- 一. 你已经能够清楚理解“元素”和“版本”的概念么?(期望你已经能够很清楚的知道)
- 二. 你能找出图中哪些内容是“标签”? 哪些是“分支”? 能大概猜想它们的作用么?
- 三. 图中的红色曲线的会表示什么含义呢? 能给出一些你的估计么?

Figure 23: 一个典型的元素 MyFile.c 的版本树图

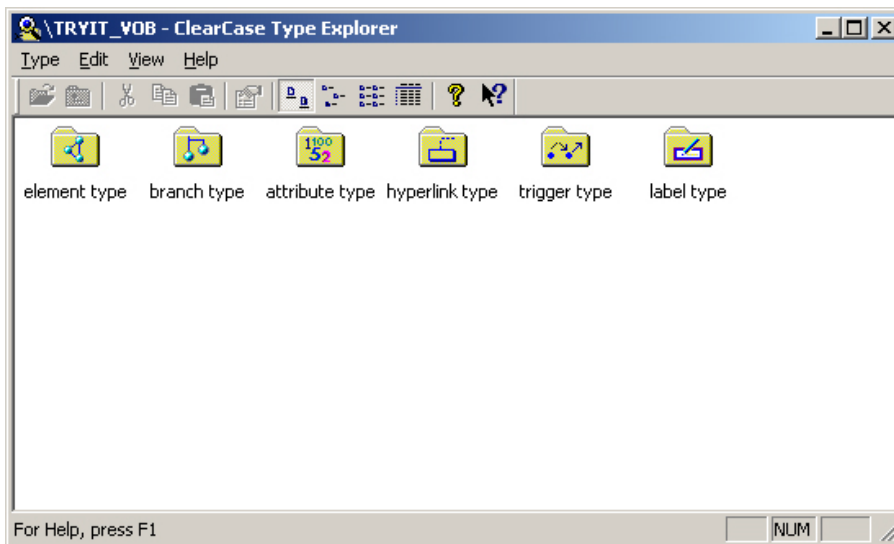


Copyright © Sydongsun 2004. All rights reserved. For internal use only.

3.6 标签和分支

用鼠标右键点击 VIEW 下的 VOB，右键菜单中的 Explore Types，将会出现下图：

Figure 24: 浏览 VOB 中存放的元数据



我们在前面说过，VOB 是版本对象库，中间存放的元素（文件或者目录）具有版本的概念。VOB 当中除了存放有版本特性的元素(Element)之外，还需要存放一些用来很好描述和识别，帮助开发者来很好的组织这些元素的附加内容，我们称之为 metadata(元数据)，用来表述数据的数据。比如，前面的几个图，就使用了分支，标签等等。

Explore Types 中的 Type 的含义，就类似于编程中的 Class；你可以到其中的 Element type 中看看，就有 directory，file 等等。请右键点击，察看属性，就有下列表述：

directory: Predefined Element type used to represent a directory.

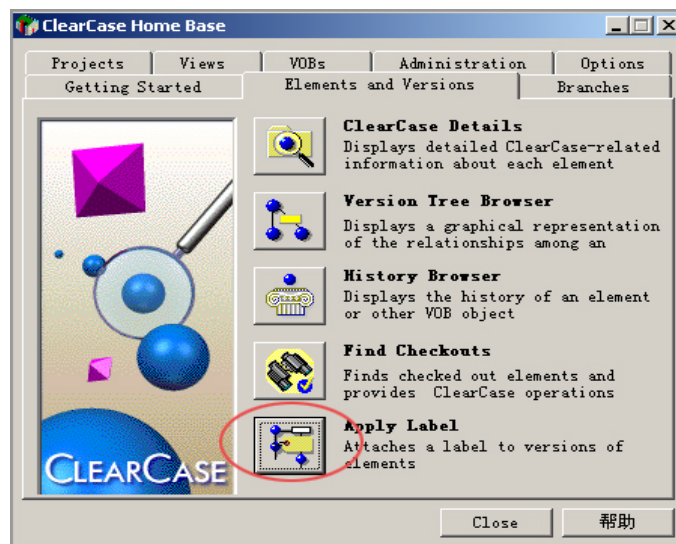
file: Predefined Element type used to represent a file.

我们存放到 VOB 库中的每一个具体的文件，都是属于 CLEARCASE 的 file 类型 (Type,Class) 的一个具体的实例，存放到 VOB 库中的目录，都是属于 CLEARCASE 的 directory 类型 (Type,Class) 的一个具体的实例。

3.6.1 标签 (label)

标签用来标识重大的开发阶段，便于检索和组织软件代码和文档。要给元素某个版本应用标签，首先要在图 2 4 中的 Label type 中创建 label 类型，然后可以通过 ClearCase Home Base 中 Apply Label 向导工具来为视图中的元素的当前版本附加 label，形成附加于元素上的 label 实例。

Figure 25: 给元素的视图中下显示的版本附加标签



利用该向导工具，可以一步步的设定为视图下的 VOB 的那些元素的当前视图显示版本，附加什么样的标签。

对于开发者来说，在图形方式下给元素的版本应用标签，总是比较方便些，先在图 2 3 中的“label type”中创建标签类型，然后在图 2 5 中“Apply Label”的向导工具来给为视图中的你要选择元素的当前版本附加标签。

有些开发者，更愿意使用命令行的方式，他们觉得命令行方式快，似乎是更喜欢：命令行比图形操作更本质一些。如果你希望在 ClearCase 所提供的命令行下进行操作的话，下面这些命令可以参考：

一个例子：

```
cleartool mklbtype -c "Version label for V2.7.1 sources" V2.7.1
cleartool mklabel -recurse V2.7.1 .
```

一些解释：

第一句，创建一个标签类型 V2.7.1，并用-c 来附加一个注释；

第二句，给当前的目录下（注意这一句的最后面有个“.”号代表当前目录）的所有元素的当前视图选择的版本应用 V2.7.1 标签，-recurse 参数表示递归运用，对该目录下的目录也有效。另外举一例：mklabel -replace V2.7.1 hello.c@@\main\4 对 hello.c@@\main\4 文件版本应用标签 V2.7.1，-replace 参数表示如果给标签已经应用到该元素的其他版本的话，则移动到\main\4 的版本上。

关于命令方式使用 CLEARCASE 的话题:

现在我们接触到 CLEARCASE 的命令行操作了。对于一般开发人员所用操作, 在 Windows 平台下 CLEARCASE 大部分都提供了基于图形方式的用户界面 (GUI), 同我们所作的开发工作一样, 如果我们在做用户界面的程序开发, 会设计易于学习, 非常直观的图形方式的用户界面; 有时候也为“高级用户”, “熟练用户”设计方便, 快捷, 具有“高级”功能的基于命令行的用户界面。某些复杂的操作, 如果设计成 GUI, 需要设计成 Step by Step 的向导程序, 如果是命令行的话, 只需要附带较多的参数。所以, 我们应该乐于学习一些能够提高我们工作效率的命令行操作, 更好的掌握 CLEARCASE 能够提高我们开发工作效率的工具。

在本文中, 我们并不讨论某个命令的具体细节信息。在 Windows 平台下的“开始”菜单的“运行”中输入 Cleartool man 可以获取 Cleartool 命令集下的所有子命令的详细帮助信息。

为了进一步让大家熟悉 CLEARCASE 的命令行操作, 我们来分析开发者的比较典型的使用 CLEARCASE 的两个场景:

场景一: 对代码的修改:

开发人员要对一个工程进行修改, 该工程所涉及到的文件分布在一个目录 (假定目录为 ProjectDir) 下的多个目录中, 也有可能处于该目录下的子目录。开发人员修改完毕后, 并对代码作了必要的测试, 然后需要将这些修改在 VOB 中形成新的版本, 并附加新的标签 (假定标签为 FOR_NightBuild), 让自动编译系统在晚上对该版本的内容作晚间编译。

首先, 将 ProjectDir 下的所有文件和目录 (包括子目录) 作 CheckOut, 这里假定 ProjectDir 目录的完整路径为 Z:\TRYIT_VOB\ProjectDir 其中 Z: 为分配给试图的盘符。参考命令为:
cleartool find \TRYIT_VOB\ProjectDir -exec "cleartool checkout -nc %CLEARCASE_PN%"

Find 命令结构类似于 find <path> <rule> <action>, 有三个部分组成, 上述命令没有使用到 rule, 在 ClearCase 的路径描述的部分, **不必附带盘符**, \TRYIT_VOB\ProjectDir 表示搜索 VOB 名称为 TRYIT_VOB 下的 ProjectDir 下内容。

修改测试完毕之后, 需要为目录 ProjectDir 下的所有元素产生新的版本:
cleartool find \TRYIT_VOB\ProjectDir -exec "cleartool checkin -identical -nc %CLEARCASE_PN%"

为 ProjectDir 下的所有文件和目录的最新版本应用标签 (这里假定标签类型 FOR_NightBuild 已经创建), 参考命令为:

```
cleartool find \TRYIT_VOB\ProjectDir -exec "cleartool mklabel -replace FOR_NightBuild  
%CLEARCASE_PN%"
```

一个同样作用但简单一些的命令是: 处在目录 ProjectDir 下, 应用命令 (下列命令最后处有代表当前目录的“.”):

```
cleartool mklabel -recurse -replace FOR_NightBuild .
```

然后你想查询一下, 那些元素被你应用了标签 FOR_NightBuild:

```
find . -name "*" -element "lbtype_sub(FOR_NightBuild)" -exec "cleartool ls -d %CLEARCASE_PN%"
```

对于标签的操作，配置管理人员有时需要为某种条件（比如已经应用了某个标签的所有元素版本）的版本应用其他的标签：比如要为TRYIT_VOB所有已经标签为V3.0的版本再附加标签，下列命令的作用是：将搜索该VOB下的所有应用V3.0标签的元素的版本再附加BaseForV4.0为名称的标签，让该版本作为开发V4.0的的基础。

```
find \TRYIT_VOB -version "(lbtype(V3.0))" -exec "cleartool mklabel BaseForV4.0  
%CLEARCASE_XPN%"
```

场景二：对文档的修改：

文档往往处于同一个目录（假定目录为 Z:\TRYIT_VOB\DocDir），某些软件，比如用于技术文档的写作的 FrameMaker，完整的一个文档往往由多个单独的文件组成。我们需要将该目录下的这些文件都作 ChoutOut，修改完成后，作 CheckIn 的操作，再给这些文件附加标签（假定为 Doc_LATEST），然后验证这些文件的标签是否运用上。

首先，在目录 Z:\TRYIT_VOB\DocDir 下运用上下文关联的命令：

上下文关联的命令，所指示的操作是：

你用鼠标右键点击某个元素，比如 Z:\TRYIT_VOB\DocDir 目录，在右键菜单中的 ClearCase 的子菜单中，选择“Command Prompt”菜单，就会出现一个命令行 Cleartool 的交互命令窗口，在这个窗口中，你可以输入 Cleartool 命令集下的子命令。

CheckOut 该目录下的所有文件：

```
checkout -nc *.*
```

修改完毕后 Check In 该目录下的所有文件：

```
checkin -nc -identical *.*
```

为新的版本应用标签 Doc_LATEST：

```
mklabel -replace Doc_LATEST *.*
```

最后，检查该目录下的文件是否应用了标签 Doc_LATEST：

```
cleartool lsvtree -g *.*
```

上面的命令，使用了代表当前目录下的所有文件的通配符 (*.*)，也称为模式匹配。也可以用 (*.doc)代表该目录下的所有 doc 扩展名的文件进行操作；用一个点 (.) 代表当前目录。了解这些可以方便我们的工作。

3.6.2 分支（branch）

在元素的版本树上，用具有一定含义的字符串来标识元素的某个特定版本，这是 label 的作用。而分支是用于在元素的某个版本拉出一个枝干，用于并行开发，或者解决一个 Bug。要给元素的某个版本创建分支实例，必须先要在 VOB 中的 Explore Types 中的 branch type 中创建分支类型。

创建分支的实例，有两种应用方式，一种是为特定的元素创建分支实例，需要 ClearCase 所提供的命令行来操作；另外一种方式在视图的 Config Spec 中使用 -mkbranch 命令，“统一”的为元素创建分支，这往往是为了并行开发的目的而这样做。

为特定的元素创建分支实例：

我们可以为特定的某个元素创建分支，比如你为了解决某个 Bug，需要修改某个文件，但是该 BUG 解决起来比较复杂，你需要记录在解决 BUG 过程中的一些文件修改版本，但你不希望在原来的分支上产生太多的版本记录。此时你就可以为特定的某些元素创建分支实例，可以在 ClearCase 的命令行中操作。例如：

一个例子：

```
mkbrtype -c "bugfixing branch" bugfix_branch
mkbranch -nc bugfix_branch hello.c@@/main/3
```

相应的解释：

第一句创建一个名为“bugfix_branch”的分支类型；

第二句在元素的 hello.c 的 main 分支上的第三个版本上创建分支 bugfix_branch；如果该语句换为“mkbranch -nc bugfix_branch *.c”，表示在当前目录中，在当前的视图选择的以.c 为后缀的所有元素的当前版本上创建 bugfix_branch 分支实例；

顺便提一下：既然可以创建分支的实例，那么自然也可以删除分支的实例（如果删除分支类型，所有该分支类型的实例将被删除）。但是我们使用 ClearCase 的主要的目的就是为了保证保留开发过程的所有的历史纪录，所有大多数的删除操作是违背我们使用 ClearCase 的目的，尽量不要删除任何内容，ClearCase 能够将“错误”的内容和“正确”的内容，都能够有条理的组织在一起，并不妨碍你的工作视线。

通过 Config Spec 语句来统一创建分支实例：

如果是为了并行开发，我们常常在视图的 Config Spec 中使用-mkbranch 的命令，而统一为视图下的所有元素创建并行开发的分支。

下面就是一些这种应用的一些视图的 Config Spec 的例子：

例子一：

```
element * CHECKEDOUT
element * /main/dev_branch/LATEST
element * /main/LATEST -mkbranch dev_branch
element * /main/LATEST
```

对应的解释：

第一句，找被 Checkout 的版本；

第二句，找 main 主干下的 dev_branch 分支上的最新版本；

第三句，找主干上的最新版本；如果元素被 Check Out 的话，会触发在元素的最新版本上创建 dev_branch。因为创建元素的分支是对元素的修改，必须要被做 Check Out 的时才会触发。然后会重新对该元素解释运行 Config Spec 的内容。此时视图将选择/main/dev_branch 分支上的被 Check Out 的版本。如果该分支上被 Check Out 的元素版本被 Check In 的话，视图将选择/main/dev_branch 分支上的最新版本，也就是第二句规则的运用结果。（顺便提醒一下，dev_branch 的分支类型应该被事先创建）。

当修改视图的 Config Spec 成例子一的情形后。可以使用 ClearCase 的复合命令来为某个 VOB 下（假定 VOB 为 TRYIT_VOB）的所有元素做一次 Check Out 的操作，然后再 Check In 将为所有元素产生 dev_branch 的分支。让使用该视图 Config Spec 在 dev_branch 分支下进行工作。参考命令为：

```
cleartool find Z:\TRYIT_VOB -exec "cleartool checkout -nc %CLEARCASE_PN%"
```

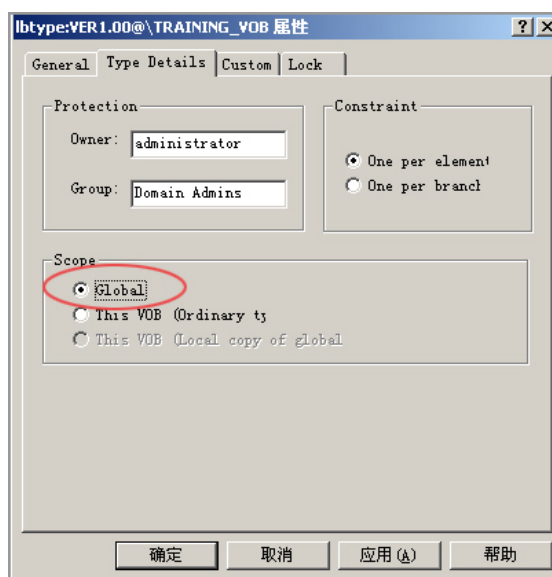
例子二:

```
element * CHECKEDOUT
element * /main/forchina_branch/LATEST
element * VER1.00 -mkbranch forchina_branch
element * /main/LATEST
```

对应的解释:

和例子一相比, 这里拉出分支的版本并不是在为元素创建分支时的元素的 main 主干上的最新版本, 还是基于特定的 label 所识别的版本, 这里需要注意的一点就是: 如果你的视图下有好几个 VOB, 你要保证这些 VOB 中都有 VER1.00 的 label type (VOB 中没有 label 类型, 无法该 VOB 中的元素附加该 label 实例), 在创建 VOB 的时候, 我们可以指定 label 是 Global 的范围, 属于同一个管理 VOB 下的所有子 VOB 将存在这样的 label。如图:

Figure 26: 指定 label 的范围是 Global 的 (Branch type 有类似的属性设定)



例子三:

```
element * CHECKEDOUT
element \\TRYIT_VOB\\... .. /sydongsun_branch /LATEST
element \\TRYIT_VOB\\... /main/LATEST -mkbranch sydongsun_branch
element * /main/LATEST
```

对应的解释:

第二句和第三句中, 并不是像前面两个例子一样是针对视图下的所有 VOB 所起作用。限定了表示 VOB 的目录。也就是说, 对其他 VOB, 是不使用这两条规则的。

3.7 分支和归并

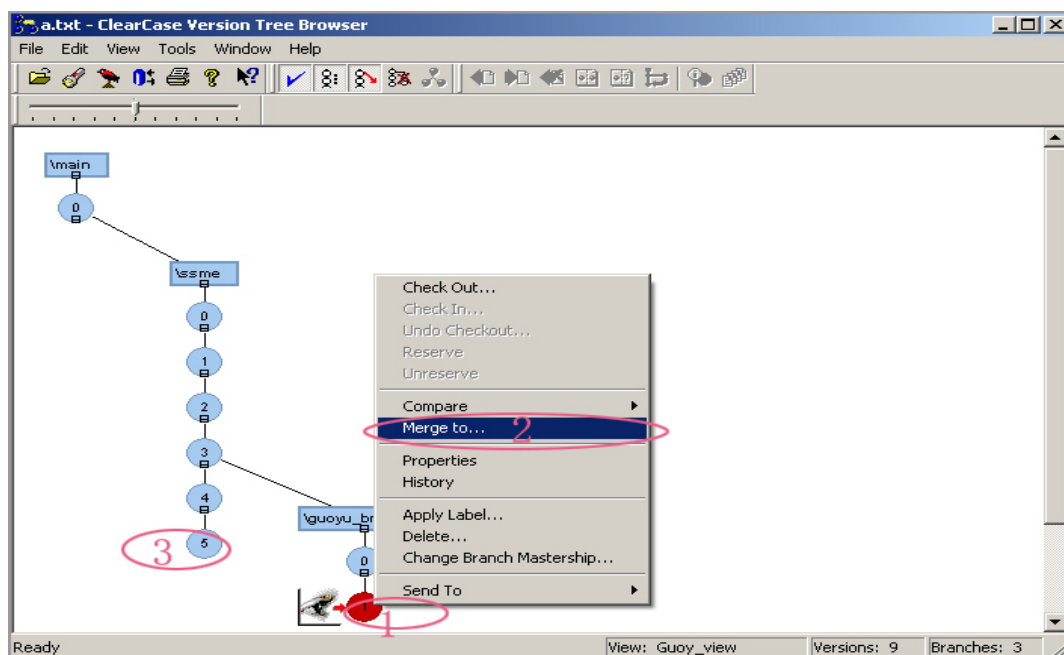
归并（Merge）是和分支相对的一个概念，我们为了并行开发（或者为了解决某个 BUG）等各种目的，我们为元素创建分支。在并行开发过程中，一个分支上的开发有可能需要引入另外一个分支上的开发成果，或者是为了解决某个问题还创建的一个 bugfix_branch 分支，当问题解决完成后需要回归到正常的 dev_branch 分支上，都需要用到归并（Merge）。

前面提到的图 2 3：一个典型的元素 MyFile.c 的版本树图中，其中的红色箭头曲线表示从一个分支下的版本归并（Merge）到另外一个分支下的版本的情形。

下面三个图表示了对一个文件元素的 Merge 过程的具体操作。

下图表示：先选中要 Merge 的元素的一个版本，在鼠标右键菜单中使用“Merge to”，然后选择要 Merge 到的目的版本。

Figure 27: Merge 操作过程 ----1



这里省略了一个进行 Merge 操作的配置设定的一个窗口，我们选择用图形方式查看 Merge 的具体。然后可以看到类似于如图 2 8 窗口，该图中上半部分表示 Merge 后所产生的结果文件内容，如果在两个版本之间有内容冲突，我们可以通过图中用红圈 1 标注处按钮来决定你所希望得到的 Merge 的结果版本。最后保存。

Figure 28: Merge 操作过程 ---- 2

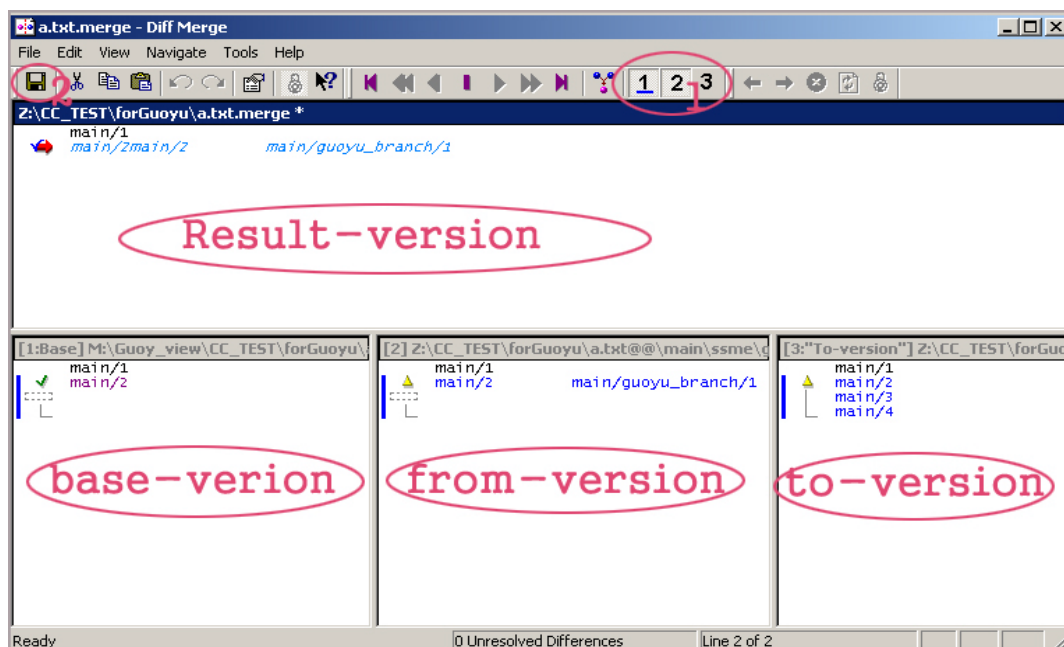
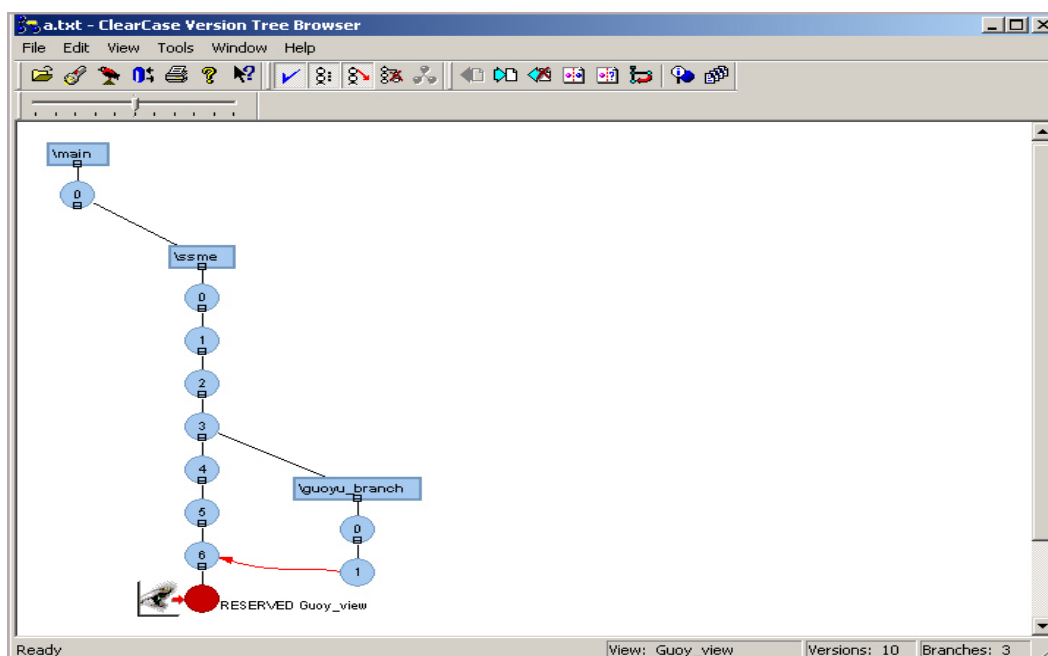


图 2-9 是 Merge 后，作 Check In 操作，产生了 Merge 的结果版本 main\ssme\6，并作 CheckOut 操作后的情形。

Figure 29: Merge 操作过程 ---- 3



Merge 的操作，通过练习几次就可以有很好的体会。请你去看看在图 2 4：浏览 VOB 中存放的元数据 中的元素类型，其中有 file、directory、html....等等预定义的元素类型。请你分别对一个目录（directory type）和 Html 文件(html type)（作者假定你熟悉 Html 文件语法，如果不熟悉，就只是对目录进行练习）创建分支，并在不同的分支上作一些修改（注意目录的修改是目录下的文件的变化），然后做从一个分支下的最新版本 Merge 到另外一个分支上的最新版本。

在并行开发环境中，有时候，需要将一个开发小组所在的工作分支的工作成果归并到另外一个小组所在的工作分支上去。此时涉及到较多文件和目录。这些工作一般由配置管理人员来做，可以采用基于 GUI 界面的 Merge Manager 工具（在“开始”----“程序”----“ClearCase”----“Merge Manager”）；也可以采用 findmerge 命令。在下面“2.8 工作空间管理”内容中将会提供一个例子。

3.8 工作空间管理

开发人员的工作空间主要是指：通过视图的 Config Spec 所定义的规则，来决定：

1. 哪些元素，元素的哪些版本反映到你的视图当中；
2. 你工作在元素的哪个分支上；

请读者复习图 2 VIEW 和 VOB 的关系 和 图 3 VOB、VIEW、View ConfigSpec 的关系以及前面和分支有关的内容；来理解工作空间的含义。而工作空间管理主要考虑到整个项目的不同的成员角色和不同的工作任务（也包括一些不拿薪水的忠实的工作伙伴：比如自动编译系统，需求跟踪系统，自动测试系统），给他们安排相应的工作空间，这往往是由配置管理人员定义一些适合不同成员的视图 Config Spec 给他们使用。从这里可以看出，工作空间的管理主要还是配置管理人员的工作任务，但是为什么还需要在面向开发人员的读物中提及这些内容呢？主要有下面几个理由：

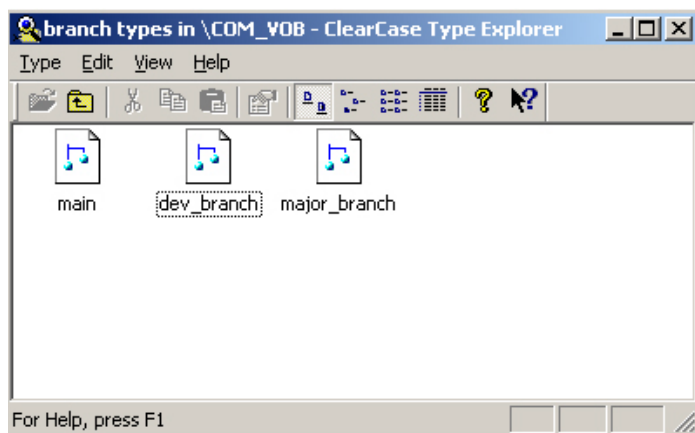
- 一. 你也许会说，配置管理人员如果熟悉你的编程工作的话，你和他沟通起来就方便多了；同样的道理，配置管理人员也希望你能尽可能理解他的工作内容的含义，他和你沟通起来也容易多了。
- 二. “Working On a Team” 是我们的工作模式，团队内部需要协作，开发人员的工作成果是测试人员，自动编译系统的输入内容。为了他们能够更好的为你服务，你要理解他们工作在什么 Config Spec 上，也就是工作空间，为了他们的工作的顺利进行，你需要做些什么？（有了 CLEARCASE，就不需要采取将代码编译后打包发送给他们的原始的方法了（毕竟现在已经是 21 世纪了！），这中间会引入太多的差错因素，在大型复杂系统的开发中这是一个让人担忧的方法）。
- 三. 在软件工程中，配置管理也是其中重要的一个环节，在国内的一部分软件开发企业中，这个环节没有得到应有的重视。在这个部分中，会列举一个典型的特意安排的开发场景，这个场景会涉及到开发流程控制、以及在 CLEARCASE 中是如何体现的，通过这样的例子，对于想成为 Team Leader 的开发人员，或者兼任配置管理工作的开发人员都是有益的。

先描述一下这个场景：

- 一. 项目和角色：我们假定这个项目的名称为 ShenZhou10，其中有开发人员，测试人员，配置管理人员，项目管理人员，质量审核者（比如代码质量审核，或者需求跟踪系统），另外还有自动编译系统（或者还有自动测试系统等等）；
- 二. 基本工作流程：在开发人员在开发分支上进行开发（假定 VOB 为 COM_VOB），生成新的版本，打上标签（假定为 FOR_NIGHTBUILD）；晚间自动编译系统进行编译，生成编译结果，保存在一个名为 DO_VOB 的 VOB 中，并打上日期标签；配置管理人员根据编译结果制作光盘（对于嵌入式系统，烧入 ROM 中），然后将交付给测试人员进行测试。

我们将定义三个主要的分支，一个 Main 分支（自动产生的），该分支主要用来发布面向用户的版本，一个 major_branch 分支，这个分支上的内容，由配置管理人员集成开发人员的里程碑的开发成果，一个 dev_branch 分支，开发人员在这个分支上进行开发工作。配置管理人员先在该 VOB 的 ClearCase Type Explorer 中创建分支类型 major_branch 和 dev_branch。如图：

Figure 30: 例子项目 ShenZhou10 的 COM_VOB 中的分支类型



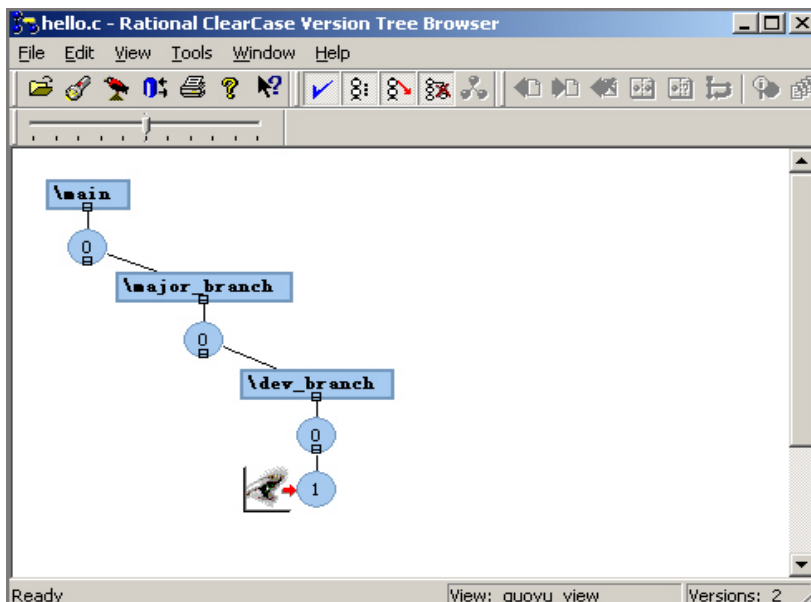
开发人员的工作空间:

下面是一个用于开发人员的视图的 Config Spec:

```
element * CHECKEDOUT
element * .../major_branch/dev_branch/LATEST
element * .../major_branch/LATEST -mkbranch dev_branch
element * /main/LATEST -mkbranch major_branch
element * /main/LATEST
```

当开发人员将代码受控到 ClearCase 当中，代码的版本上会产生 major_branch 分支和 dev_branch 分支，视角并盯住 dev_branch 分支。如图所示：

Figure 31: 开发人员在 dev_branch 分支上进行开发活动



开发人员在每天下班之前，应该对开发工作的代码进行 Check In 的操作，然后对这些新的版本附加用于自动编译的浮动标签 FOR_NIGHTBUILD，如何给较多元素应用标签的命令行操作的方法可以参看 2.6.1 标签 (label) 中的内容。

自动编译系统的工作空间：

用于自动编译系统的计算机上会建立一个视图，该视图的 Config Spec 为：

```
element * FOR_NIGHTBUILD
element * ../major_branch/dev_branch/LATEST
element * ../major_branch/LATEST
element * /main/LATEST
```

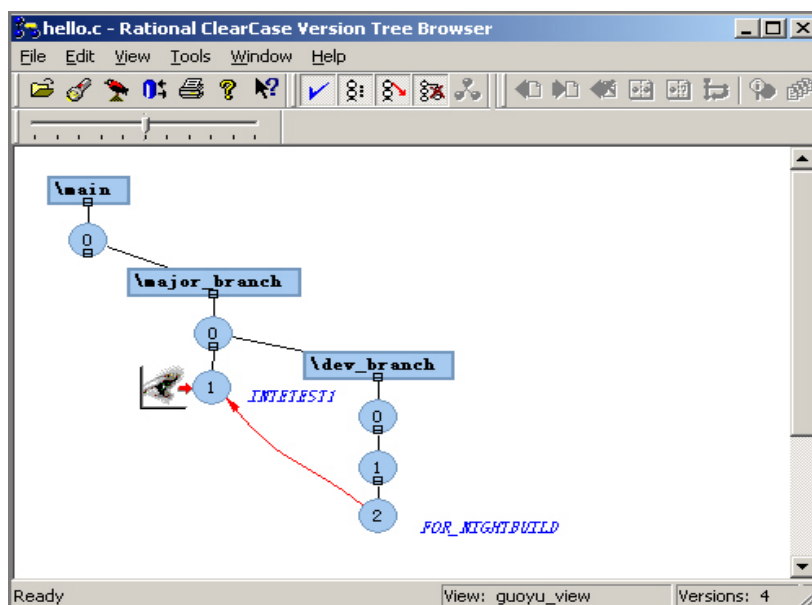
上面的视图规则，第一句表示，选择附加标签 FOR_NIGHTBUILD 的元素的版本，如果元素没有应用上述标签，则是用下面的语句规则。开发人员每天移动 FOR_NIGHTBUILD 标签到他们最新的代码上。从而保证自动编译系统的每晚编译总是使用开发人员的最新代码，构成了自动编译系统的工作空间，自动编译系统产生的结果将存放在 DO_VOB 中。

代码审核人员或者需求跟踪系统的工作空间：

当项目开发到一定阶段（里程碑），比如集成测试阶段。配置管理员将 dev_branch 分支上的代码冻结（Freeze），并 Merge 到 major_branch 分支上，并附加 INTETEST1(第一次集成测试)标签。代码审查人员或者需求跟踪系统将对这些代码进行审查。对于这些人员或者需求跟踪系统将使用下面的视图 Config Spec：

```
element * INTETEST1
element * /main/major_branch/LATEST
element * /main/LATEST
```

Figure 32: 需求跟踪系统或者代码审核人员在 major_branch 分支上进行活动



Merge 操作的具体做法:

先确定你使用视图的 Config Spec 让你工作在 major_branch 分支上, 也就是代码审核人员所有的视图 Config Spec。然后: 用鼠标右键某个 VOB 目录, 在弹出菜单中选择 “ClearCase---Command Prompt” 在命令中输入:

```
cleartool> findmerge -all -element (brtype(dev_branch)) -fver \main\major_branch\dev_branch\LATEST -merge -whynot -c "for merger intergate testing"
```

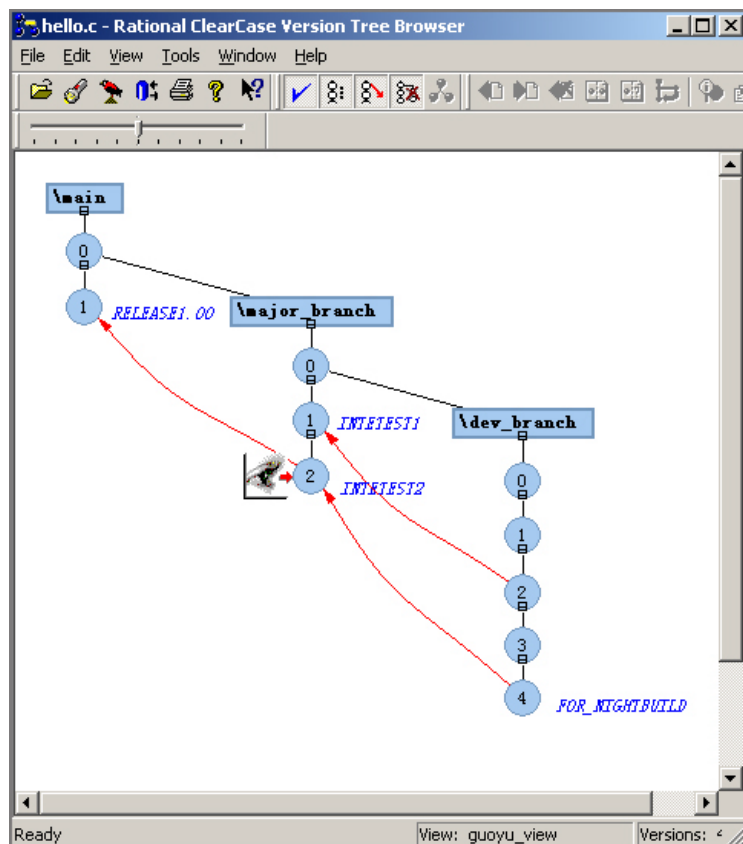
上述命令表示: 查找该目录下 (默认是递归的) 所有包含 dev_branch 分支的元素, 从 \main\major_branch\dev_branch\LATEST 的版本归并到 major_branch 的分支的最新版本上。参数 - whynot 表示对不需要作归并操作的元素则输出原因, 通过 -c "for merger intergate testing" 添加注释;

然后, 需要对在归并后生成的新的版本作 CheckIn 操作;

最后对 major_branch 分支上的新的版本附加新的标签, 比如 INTETEST1 标签, 具体命令可以参考前面 2.6.1 标签 (label) 的内容;

如果在这个阶段, 有关需求没有实现, 或者在第一次集成测试中发现的所有的 BUG, 将会提交一个 BUG 修改任务表 (有些可能在缺陷跟踪 (变更) 系统中进行) 给开发者, 开发者在自己的 dev_branch 上进行 BUG 的修正, 经过自己的测试, 然后代码 Freezed, 由配置管理人员 Merge 到 major_branch 分支上, 进行第二次集成测试...如果某次集成测试获得通过, 则从 Merger_branch 分支上 Merge 到 main 分支, 作为 RELEASE1.00 的发布版本。如下图:

Figure 33: 反映开发、集成测试、发布版本等活动的一个元素的版本树图



以上内容主要提到了三个工作空间：开发人员的工作空间（主要工作在 `dev_branch` 分支上），代码审核人员和需求跟踪系统所使用的工作空间（主要工作在 `major_branch` 分支上），也有自动编译系统的工作空间（主要选择使用了 `FOR_NIGHTBUILD` 标签的版本）。分支和标签都可以帮助我们选择所需要的版本。在图 3-3 中，也可以不使用 `major_branch` 分支，而直接在 `dev_branch` 分支上的相应的版本上附加相应的标签（`INTETEST1`、`INTETEST2`...）。但而加上 `major_branch` 分支好像可以更“清楚”的显示集成测试这样的活动阶段，在大型复杂的项目中，这样做是有好处的。

另外开发人员也可以自己为解决某个 BUG，在要修改的那些文件上产生你自己的分支。所使用的命令请参看：**2.6.2 分支（branch）中间的为特定的元素创建分支实例**的有关内容。但是要记住，这个分支是你个人的“幽径”，别人不容易知道的，也就是你的**私有分支（私有工作空间）**。所以你修正了 BUG 后要记得（Merge）回到 `dev_branch` 这个开发人员的“大道”上。

分支用来支持并行开发，并能够形象的、清晰的组织元素的版本。每个创建的分支应该有明确的意义，不要创建过多和不必要的分支，太多的分支也许反而让人迷惑。过多的分支也自然可能需要做较多地 Merge 操作。

关于基线（BASELINE）

当我们开发经过需求分析，功能规格说明，代码详细设计，多次的集成测试，然后发布正式的 `RELEASE1.00` 版本后，并提交给用户测试使用。进一步收集用户使用意见和新的需求，也包括一些在用户使用当中发现的 BUG。也许我们需要做进一步开发，希望发布一个 `RELEASE2.00` 的版本。此时 `RELEASE1.00` 将作为下一步开发基础（**也就是基线**）对于开发人员来说，需要从这个版本进行下一个版本的开发。此时开发人员的工作空间需要做下列变化：

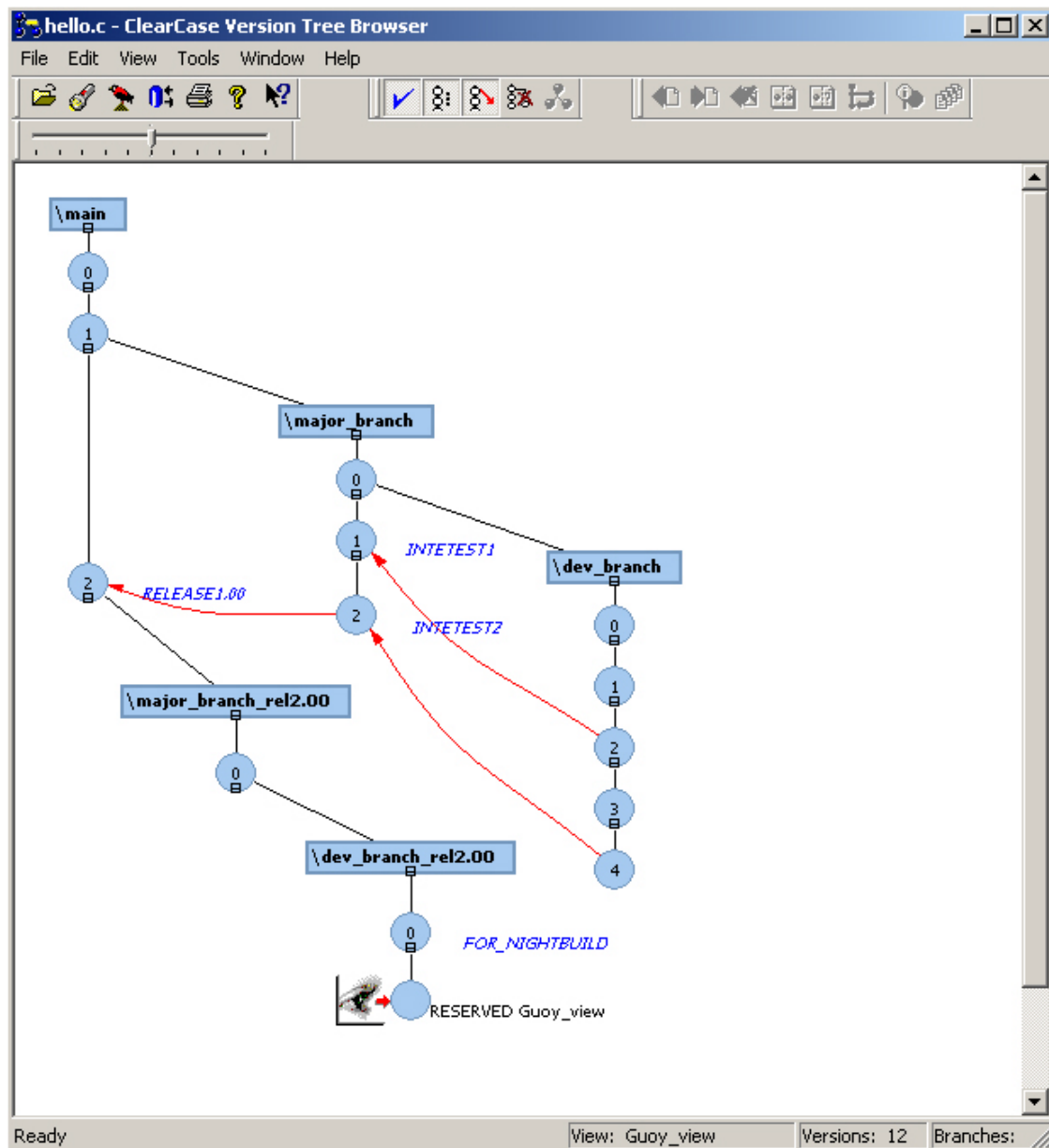
```
element * CHECKEDOUT
element * ../major_branch_rel2.00/dev_branch_rel2.00/LATEST
element * ../major_branch_rel2.00/LATEST -mkbranch dev_branch_rel2.00
element * RELEASE1.00 -mkbranch major_branch_rel2.00
element * /main/LATEST -mkbranch major_branch_rel2.00
element * /main/LATEST
```

一些简单的解释：

第四句表示从 `RELEASE1.00` 这个标签所定的基线产生 `major_branch_rel2.00` 分支，第三句表示从 `major_branch_rel2.00` 产生开发人员所工作的分支 `branch_rel2.00`。另外第五句的作用是，如果一个元素（文件或者目录）在开发 `Release2.00` 的版本过程中受控后，这个元素将不会存在 `RELEASE1.00` 的版本，在这个元素的 `main` 的最新版本上产生 `major_branch_rel2.00` 分支。

如果开发者 CheckOut 原来的 `hello.c` 文件，得到的版本树：

Figure 34: 在 RELEASE1.00 的基线上进行继续开发

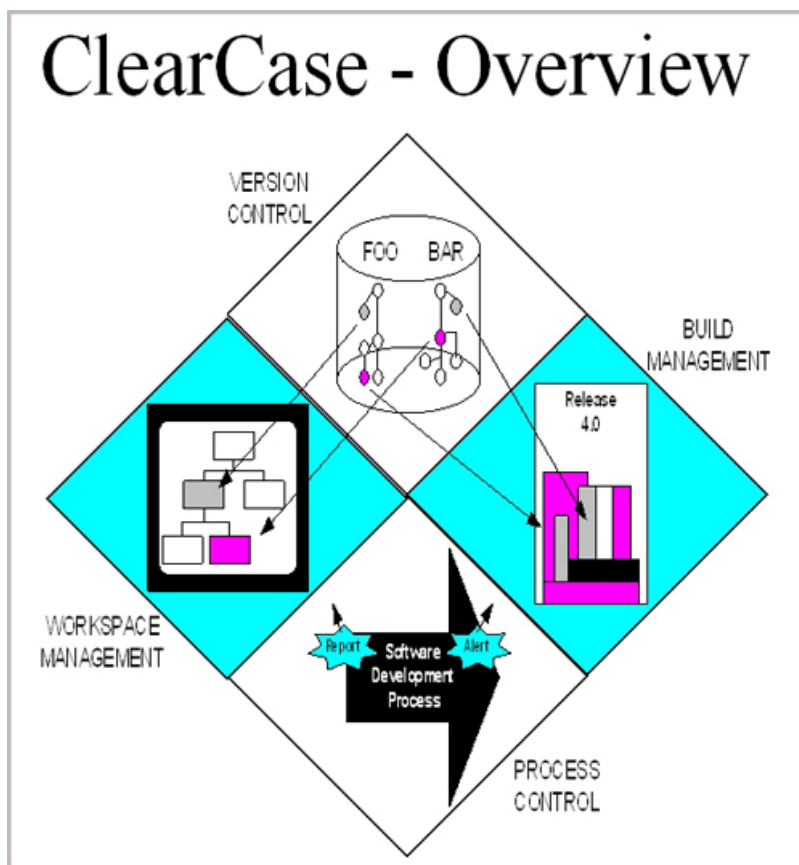


从这个 ShenZhou10 的虚拟的项目中，我们可以得到一些简单的概念：项目开发流程控制在配置管理软件 CLEARCASE 中如何体现的。为了说明的方便我们在图 3 4 中使用了较多的分支。在开发当中，我们也可以在整個项目开发中主要使用 main 和 dev_branch 分支，多应用标签，可以避免太多的分支和归并操作，版本树图看起来也较清楚。

4 进一步的内容

在前面关于 CLEARCASE 的内容的阅读基础下，结合相应的练习，我们会对 CLEARCASE 的版本管理、工作空间管理的概念有所了解，并熟悉基本的操作。在这里顺带提一下 CLEARCASE 所提供的四大块功能：

Figure 35: CLEARCASE 四大功能版本控制、工作空间管理、开发流程控制、构建管理



Copyright © Sydongsun 2004. All rights reserved. For internal use only.

显然，开发流程控制和构建管理是项目管理人员或者配置管理人员所应关心的内容。本文面向一般的开发者，所以并不提及它们。但是，在以后使用 CLEARCASE 的过程中，作为开发人员，下列内容的了解，可以让 CLEARCASE 更好的帮助我们的开发活动。

4.1 几个要注意区分的命令 (rmname/rmver/rmelem/

mv)

大部分配置管理人员并不趋向于让开发人员知道这些命令的作用，因为这些命令，和我们使用 CLEARCASE 的目的——完整的保证开发过程中的历史记录——相违背。如果你确实有某些“特殊”的要求，比如删除某些元素，可以将你的要求告诉配置管理人员，他会分析你的要求，采用合理的命令。在这里，我假设你是属于这样一类开发者：喜欢刨根问底，喜欢掌握更多的东西，当然这是优秀的开发者所具有的特质。向你介绍这些命令，我也希望你能够：你能够精确了解这些命令的作用和差别，你能够合理的有限度的使用，还不是滥用这些命令。

前三个命令都有 **rm** 前缀，也就是删除，删除什么内容在后面部分指定。

rmname

Removes the name of an element or VOB symbolic link from a directory version

A name can be removed from a directory only if that directory is checked out, Rmname does not delete elements themselves, only references to elements. Use rmelem (very carefully) to delete elements and all their names from their VOBs.

如果你在视图下的某个目录中使用命令行：**rmname c.txt** 那么将在该目录的版本中将不会出现 c.txt 文件，还该目录对象的以前的版本会有该文件。使用该命令之前，该命令必须在目录被 checkout 的情况下才可以。**rm** 是 **rmname** 的缩写，建议大家在使用时，还是明确的使用 **rmname**。在 clearcase 中，目录和文件的地位同等，都具有版本的概念，如果使用 **rmname c** 表示在当前视图的目录中删除目录元素 c，同时 c 目录下的元素也变得不可访问。

rmver

Removes a version from the version tree of an element

rmver deletes one or more versions from their elements, his command destroys information irretrievably. Using it carelessly may compromise your organization's ability to support old releases.

如果你在视图的目录中使用命令：**cleartool rmver c.txt** 那么将直接删除当前视图中 c.txt 文件的最新的一个版本。如果你要删除某个对象（文件或者目录）的中间的某个版本，可以使用 **rmver -version \main \1 c.txt** 这个语句表示删除 c.txt 文件对象的 \main\1 的版本。一般情况，强烈建议开发人员不要使用 rmver 命令，因为对象的所有版本保存在数据库中没有任何坏处。万一有一天你发现这个版本是有用的呢，或者对于别人是有用的呢。

rmelem

Removes an element or symbolic link from a VOB

The rmelem command completely deletes one or more elements or symbolic links, This command destroys information irretrievably. Using it carelessly may compromise your organization's ability to support old releases. In many cases, it is better to use the rmname command.

该命令表示从 vob 库中删除元素对象（文件或者目录）。如果你发现有一天，你却需要这个文件，或者目录的时候，将无法找回。一般的开发人员应该禁止使用这个命令，我们使用 Clearcase 的目的之一，就是为了能够在我们需要的时候，能够追索历史版本。而如果被你删除了元素，那附属于元素的所有版本也将不再存在。配置管理员应该小心使用的命令，而开发人员请不要使用。rmname 已经足够满足你的要求了。所以配置管理人员应该应用 Trigger 来限制一般的开发者使用这个命令。

mv

Moves or renames an element or VOB link

元素的重命名或者移动，元素的版本不会受到影响。

在 ClearCase Explorer 中，点击元素的右键菜单中的“Delete”菜单是 **rmname** 命令的 GUI 表示。
“Rename”菜单是 **mv** 命令的 GUI 表示。要加以区分。如下图所示：

Figure 36: CC_Explorer 中的 rmname 和 mv 的图形菜单命令



当然，这些涉及到删除操作的命令，是有权限要求的（配置管理员在配置管理计划中会做好必要的规划），如果你是 clearcase 的系统组成员，或者 Vob 的 owner，或者元素的 owner（往往创建者自动会成为 owner），加上其它的权限管理，比如 group, other group, Trigger 等等。但是我们还是建议大家对这些和开发人员有关的命令有个清楚地认识。大家可以到实验 VOB 中作一些实验，加强体验。

Copyright © SydongSun 2004. All rights reserved. For internal use only.

4.2 进一步理解 -- 目录元素的版本

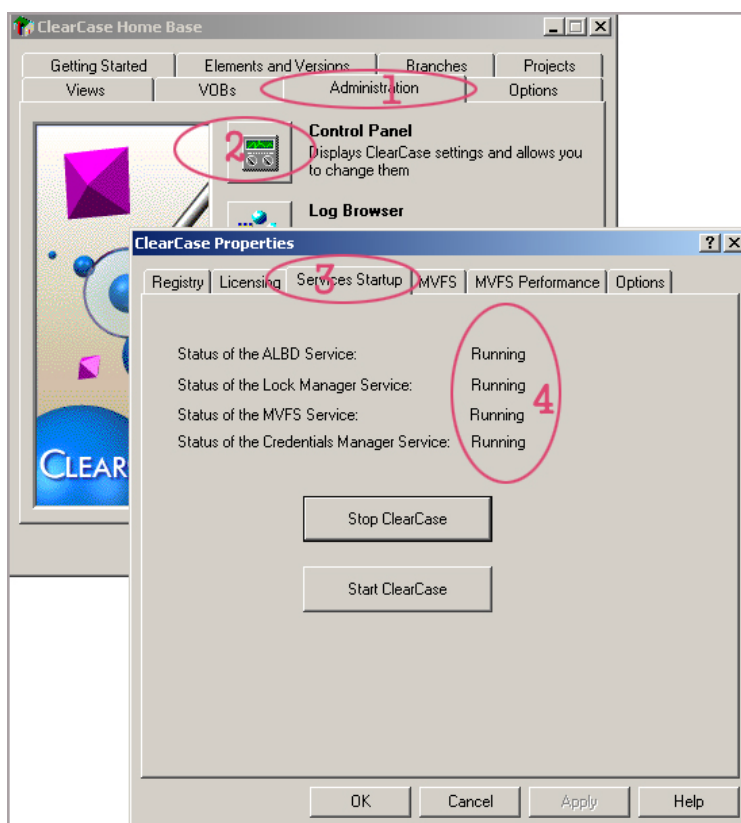
在本文的开头，我们提到 ClearCase 主要存贮的内容是具有版本概念的元素，主要包括文件和目录元素（也可以包括 ClearCase 管理员自己定义的其他元素类型，**可以参看图 2 4：浏览 VOB 中存放的元数据**，其中的 `element type` 中包含了一些其他预定义的元素类型，不过我们一般很少用到他们），也就是说，目录元素也具有版本的概念。目录元素的版本的变化就是其中包含的文件的增加或者删除。从道理上，这很容易理解，但是更多的需要你完成下列练习，让你有更实际的体会。

1. 在一个 VOB 中创建一个目录 TestDir，并受控；
2. 在这个目录下增加几个文件，然后受控；（在 Windows 图形界面中，当元素受控时候，该元素所在的目录的版本自动变化，无需 CheckOut/CheckIn 操作）；
3. 然后再增加几个文件，然后受控；请观察 TestDir 的版本树信息，请看每一个版本的说明信息。
4. 然后请在图 3 6 所示的 ClearCase Explorer 中，选中该目录下的某个文件，用右键菜单的“Delete”菜单命令（在 Clearcase 命令行上中就是 `rmname` 命令）。
5. 你可以看到 TestDir 目录下将不存在步骤 4 中被删除的文件。请察看目录 TestDir 的版本树信息。
6. 在目录 TestDir 的版本树图上，选中最新的版本的上一个版本，作 CheckOut 操作，然后刷新，请察看目录 TestDir 中是否出现了步骤 4 中被删除的文件？如果是的话，你的试验成功。你是否对于目录的版本有了更清楚地认识？

4.3 自我检查 ClearCase 客户端出现的问题

在某些情况下，如果你发现 ClearCase 出现不能正常使用，请检查 ClearCase 运行所需要的四个服务是否在正常 Running 状态，如果不正常，请停止，并重新启动这些服务（可以多点击两次，直到所有服务 Running 状态，如果还不行，则**重新启动计算机**（不要小看，**重新启动计算机**也算解决问题的方法），再检查，如果继续存在问题，请求你的配置管理员的帮助）。因为某些偶然的情况，比如网络问题，系统启动，或者其他应用程序冲突使 ClearCase 的四个服务不能正常启动。这种问题占了客户端的问题较大比例。所以你通过下图的说明，自己也可以解决很多 ClearCase 使用上遇到的问题哦。☺☺

Figure 37: 自己检查计算机上的 ClearCase 的四个后台服务是否正常运行



4.4 进一步的培训

本文包含**配套**的高质量（图像清楚，效果较好）的**视频音频教材**；
如果你阅读后，请到实验 VOB 中作相应的实验。如有问题，请联系 CM 人员。

Copyright © Sydongsun 2004. All rights reserved. For internal use only.

Copyright © Sydongsun 2004. All rights reserved. For internal use only.