



SCM – Software Configuration Management at Acme Widget Company, NJ Division

Local Developers' Guide & FAQ:

ClearCase and Release Management

Tim Gill, ClearCase Administrator

Preface

Purpose: *This document is written to aid AWC-NJ developers in the move to ClearCase for their application source code and to a release management process for deploying those applications. It is the primary mechanism I will use to convey the information they need for that work.*

This document contains longer explanations of material to help developers. For a quick summary of the most used commands and the easiest way to get started, get the 1-page document "ClearCase Quick Reference Sheet, for Unix" (filename ClearCase-NJ-QuickRef.doc).

Audience: *Application developers, and their managers, in AWC-NJ.*

Revisions: *This document will be updated frequently. Each published version, available as a PDF file, has a version number in the footer of each page, which is incremented by .1 for each minor revision. If you are using a printed copy, check the location for the published version, http://AWOnline/Content/8_Projects/cc_cq/7_Deployment/FAQ.pdf, to see if it has been updated since you printed it.*

This document is now v2.x, reflecting the changes in direction that have happened in Q3- and Q4-2003. Much material from version 1.x has been removed in this 2.x version.

Terminology: *Until I learn any common terms used locally, I will use:*

AWC-NJ:	<i>the System Component Manufacturing organization/facility in Neptune, NJ.</i>
SF	<i>the Development Services ClearCase group in San Francisco, which provides some of the ClearCase and release management tools, process, and support for the NJ effort</i>
VOB or vob	<i>a ClearCase versioned object base (I can't use capitalization consistently!)</i>

Table of Contents

- 1. Preparing for ClearCase and Release Management..... 4**
 - 1.1 Preparing for the Move of Your Code to ClearCase..... 4
 - 1.2 The Use of “Test” VOBs 4
 - 1.3 Preparing for Release Management 5
 - 1.4 Preparing to Get ClearCase Installed 6
- 2. Using ClearCase 7**
 - 2.1 Documentation and Tutorials..... 7
 - 2.2 Basics of the ClearCase Permissions Model..... 8
 - 2.3 On Various Platforms 8
 - 2.3.1 On Your Unix Workstation.....8
 - 2.3.2 On Your Windows PC10
 - 2.3.3 On an SCO Open Server Development Machine10
 - 2.3.3.1 Background 10
 - 2.3.3.2 AWC-NJ Conventions for Exported Views..... 10
 - 2.3.3.3 Using ClearCase Files on the SCO Machine 10
- 3. Development Models & Policies with ClearCase 12**
 - 3.1 The Model to Be Used for AWC-NJ Applications..... 12
 - 3.1.1 Two Common Approaches to Code Changes12
 - 3.1.1.1 Main Line *code change model*..... 12
 - 3.1.1.2 Lead Programmer *code change model*..... 12
 - 3.1.2 Naming Standards12
 - 3.1.2.1 Branch Names..... 13
 - 3.1.2.2 View Names 13
 - 3.1.2.3 Label Names 13
 - 3.1.3 A Deployment Process to Production13
 - 3.2 How LOD Will Use This Model 14
 - 3.2.1 Summary of LOD Model14
 - 3.2.2 Visual Example of LOD Model.....15
- 4. Commonly Asked Questions 16**
 - 4.1 Basics 16
 - 4.1.1 How do I reference a manual page for each *cleartool* subcommand?16
 - 4.1.2 How do I get and use a view?16
 - 4.1.3 How do I do check-outs and check-ins of files?16
 - 4.1.4 I don’t want my checked out file anymore. What do I do to undo it??17
 - 4.1.5 How do I see the version of the files in my current directory, in my view?17
 - 4.1.6 I have made changes to a copy of a file and now I want to check-it in. How do I do it?17
 - 4.1.7 How do I add a file or directory to ClearCase, or move one, or remove one?17
 - 4.1.8 A directory I just created is not writable by others. Why not, and how do I fix it?17
 - 4.1.9 You say I should be in the group *vob_two* when working in ClearCase. What’s an easy way to do this?18
 - 4.1.10 How do I see the changes I’ve made to my checked-out file?18

4.1.11	How do I see all the changes made recently in this directory? Or the comments made with previous check-ins on a file?.....	18
4.1.12	How do I see what files are checked out?.....	18
4.2	There are no files in my VOB!-	18
4.3	What Does the Message “checked out but removed” Mean? Or, Why Has My Checked-out File with All My Changes Disappeared?	18
4.4	When I run make in a view, my build fails because touch fails to update my .c files.....	20

1. Preparing for ClearCase and Release Management

ClearCase is a sophisticated *software configuration management* system into which all AWC-NJ application source code will be migrated. This process requires identifying all the source code that needs to be moved, agreeing on a logical mapping of the applications and sub-systems or components to ClearCase VOBs (*versioned object bases*), doing the migration of the code, and formalizing the cut-over to using ClearCase instead of MKS-based or home-directory-based source code management. You must also learn the basics of using ClearCase to get your work done, and get ClearCase installed on your PC and/or workstation.

Release management is the term used to describe the new processes involved in getting an application released to QA for testing and deployed to Production for use, from its location in ClearCase. There will be a number of aspects to release management in AWC-NJ, but the first two you must concern yourself with are:

- how your application code is structured in its move to ClearCase vobs from its current location in MKS Source Integrity or in your personal directories;
- how your application can be built easily by someone other than yourself—the release management staff (you must provide a build script for them to use, one that works in a *clean build environment*).

1.1 Preparing for the Move of Your Code to ClearCase

You may be involved in deciding how the code in your applications is logically structured into VOB's when it is migrated to ClearCase. Two general things to keep in mind as the move of your code is planned are:

- A good rule of thumb is that code that is developed and/or deployed together as a unit should reside in a single VOB. (This practice has a number of benefits: one of them is that the process of labeling versions of code for the *release management* activity is easier.) Although in 2003 the approach to the AWC-NJ code might have resulted in, say, 10-20 VOB's, in the new approach under Tom Moreton, there will be far fewer, and perhaps only 1 “logical” VOB for the entire front end system (sometimes called ETS, or Execution and Trading System).
- Common code—code that is shared by more than one application—should be in its own VOB. Such code can include:
 - ◆ header files
 - ◆ libraries
 - ◆ 3rd party tools, both source and binary

If you are aware of such code, make sure that it is addressed in the VOB planning, as it is much harder to move it out of an application VOB and into a common VOB at a later time.

1.2 The Use of “Test” VOBs

For some migrations of code, I will create “test” VOBs for developers to work with. These vobs serve several purposes:

- Developers get a chance to work with ClearCase on code that they own and know before having to do it officially—a period of time to get comfortable, if they want to use it (they should!).
- The ClearCase administrator can tune the scripts used to exclude binary and backup files usually found mixed into source trees; this often has to be tuned specifically to the nature of each particular application source tree, and so doing a test vob gets that process done ahead of time.. (See [more information](#) on this below.)

Developers can also recognize any missing files that might have been excluded by the administrator scripts in the import operation, thus helping the ClearCase administrator to tune those scripts.

- Developers can test the needed changes to their `makefiles` in the test vobs, getting them to work both there and in the current non-ClearCase environment with minimal differences (see the [section](#) on this below).
- The lead developer for a vob can test the script needed for Release Management, so that it is ready as soon as the official vob is created.

- The ClearCase administrator can prepare and test the assembly of scripts used to import code for each vob, so that the official import can be done in the least possible time.

Note these points about all test vobs:

- All test vobs will be destroyed soon after the official vobs are created.
- All changes you make to code in a test vob are destroyed. If you need those changes (e.g., your makefile changes), you must copy them somewhere else.
- When a vob is officially created, the import is done from the original source code location (in MKS or in your home directory), and not from the test vob.

Test vobs are typically created with a vob tag that is identical to the expected name of the official vob, with a prefix of TEST_. For example, the vob that will eventually be known by the tag /vobs/Middleware will have a test vob with the tag /vobs/TEST_Middleware.

1.3 Preparing for Release Management

The persons who will perform the release management role in AWC-NJ need a simple, consistent way to build your application components. Since the release management process being adopted (modeled on one used by AWC-NJ in SF) requires the release manager(s) to build the software within ClearCase and in a “clean” environment, it is the responsibility of the developer to change the `makefile` as needed and to provide the build script for use by the release managers. To simplify release management, the script will be named the same across all applications: `buildScript_RM`.

From experience gained in doing the first two migrations, of the TIM and VISTA Server applications to ClearCase, your current build process (outside of ClearCase) might have some or all of these characteristics that have to be changed before the code can be built in ClearCase and by the release managers:

- x* The build assumes that certain directories are in the `PATH` variable, because it inherits the `PATH` from your personal shell environment.
- x* The build assumes that certain environment variables are set (e.g., `OS`, `ROOTDIR`, and `MAKESTARTUP` for builds using the MKS `make` utility), because it inherits its environment variables from your personal shell environment.
- x* The build assumes that you are running on the right machine: that it is, say, a SunOS 5.6 machine, or that it has the `/xyz` file system mounted from some other machine.
- x* The makefile or build script you currently use is not in MKS (i.e., not in the source tree with the source code it builds), but in your `HOME` directory.
- x* The knowledge about the order in which different components have to be built is in your head, and not in a script.

When the release managers build your application, they will be working in a “clean environment”. At a minimum, this means that the shell in which your build script and `makefile` are run will:

- have only basic environment variables set, like `PS1`, `HOME`, and `LOGNAME`;
- have a minimal setting for the `PATH` variable, probably just `PATH=/usr/bin:/usr/ccs/bin`.

So your script must ensure that your environmental setup requirements are met, allowing a build in such a clean environment to succeed, or fail gracefully.

To setup a “clean environment”: for `ksh` users: you could create a sub-shell to do your testing with the command

```
PS1='[clean subshell]: ' PATH=/usr/bin:/usr/ccs/bin ksh
```

This has a “clean” `PATH` but has inherited all environment variables. I don’t know a built-in `ksh` command to unset all environment variables, so I use my own `unset_environment` function, which you are free to use:

```
$ export FPATH=/dist/share/ClearCase/adm/kshFunctions
$ unset_environment
Changing current shell to a "clean" environment...
.
.
clean env $
```

1.4 Preparing to Get ClearCase Installed

The ClearCase administrator will install ClearCase on your Unix workstation or your Windows PC, or both. To make this as easy as possible for both you and me, you need to do several things to prepare:

- **Ahead of time:**
 - a) Let me know if your Unix ID and your Windows ID are different.
 - b) Let me know the machine ID for both your Sun workstation (if any) and your Windows PC.
 - c) You should have the VNC remote access software installed on your PC—if you do not, ask Windows/NT support to install it for you. Let me know the password if you have changed it from the default. I will use this to install your PC with ClearCase, and to help you whenever you have problems.
- **The day I install ClearCase on your machine:**
 - d) Sun workstation installations: I can install ClearCase remotely from an `xterm` window on my own machine, and no reboot is required. So I can usually do this even when you are using it. But if you have issues about this, let me know. Otherwise, I will just do it at my convenience.
 - e) Windows PC installations: The Windows installation is a little more intrusive (you cannot be working on it while I do it, a reboot is needed, and you need to enter your password after the reboot). It is usually done in 2 parts, and your machine will not be available for your use for about 30 minutes (assuming all goes well).
 - For the first part, I do several things that I don't need access to your PC for: ensure you are in the `vob_two` group in the Windows domain, ensure you have a TAS user ID mapping if your Unix and Windows ID's are different, setup a view for you and synchronize it to the ClearCase Windows region.
 - For the second part, I install ClearCase remotely on your PC using VNC. I take control of your machine, so you should need to make an Outlook appointment with me for a half hour when you can do other things and still speak to me on the speakerphone, if needed (and enter your password when your PC reboots). I configure your environment with variable settings, VOB and view file system mounts, and making your view accessible to your machine. You then verify that you have access to your VOB.

Subsequent issues with the Windows machine will most easily be handled via a VNC connection to your PC while I speak with you on the speakerphone.

This procedure may be streamlined the more installations we do, at which time I will update this document.

2. Using ClearCase

Once I have installed ClearCase on your machine, and have created a *view* for you, you should be able to use it. However, there may be a few things you need to do to see your vob source files. And recall that you must be in a *view* in order to see any ClearCase files stored in a vob. (Note that we use *dynamic views* here, not *snapshot views*—you will see references to both in the documentation.)

The most important thing to remember about using a *view* is that they give you access to a single version of each file in your source tree, and the version you get is determined *dynamically* when you open the file—according to the rules of the *configuration specification* (*config spec*, for short). Remember that using ClearCase does not remove any capability you had otherwise, in Unix or Windows: it just adds the ability to see the files stored in vobs. Thus, for example, you can do a build “outside” of ClearCase that references a file stored in ClearCase, as long as you are running in a view context.

2.1 Documentation and Tutorials

There are a couple of ways to get help in using ClearCase:

- My 2-page ClearCase Quick Reference Guide contains the briefest possible summary of the information you need to know to use ClearCase at System Component Manufacturing.
- Your handouts from the training class in February 2003 could still be useful, to remind you of what you learned, and what you did in the labs.
- The supplementary handout I provided at the February 2003 class to most of you—a photocopy of pp. 1-13 of Chapter 1 of the *ClearCase Concepts* manual (from Release 3, but still valid)—is the single best short description of the concepts and power of ClearCase that I know of. I do not have an online version of this but can still provide printed copies if you need it.
- Online tutorials, if I have installed ClearCase on either your Windows or Unix machine:
 - ➔ on Unix, run the command `/usr/atria/bin/hyperhelp cc_tut.hlp`, which provides a tutorial on using ClearCase (presented using X windows graphics).
 - ➔ on Windows, run Start → Programs → Rational ClearCase → ClearCase Tutorial.

The online tutorial also provides graphical online versions of all of the ClearCase *manual pages*, with detailed coverage of command syntax and such. I strongly recommend trying the online tutorial (in either Windows or Unix) early in your use of ClearCase, to get comfortable with the tutorial and the information it can quickly provide.

- On a Unix machine with ClearCase installed, there is also `help` sub-command and a `man` sub-command:
 - ➔ `cleartool help`
 - ➔ `cleartool [-graphical] man` # `-g` option gives a graphical version

The “printed” versions of these “man pages” from the online *Command Reference* manuals in PDF format are also available from the online PDF version of this document by using RightMouseClick → Open Weblink in Browser on these links: [cleartool](#), [help](#), [man](#).

- The primary manual on ClearCase that Rational provides that might be of use to you as a developer is called *Working in Base ClearCase*, but only sections 1.2-1.5, Chapter 3, and Chapter 5. It is available as a PDF file, like all the ClearCase manuals, once ClearCase has been installed on your machine. It is available in both Windows and Unix versions:
 - ➔ on Unix, if ClearCase has been installed on your Sun workstation, it is in the file `/usr/atria/docs/hlp/ClearCase_dev.pdf` (this is the Unix version)
 - ➔ on Windows, if ClearCase has been installed on your PC, it is in the file `C:\Program Files\Rational\ClearCase\bin\ClearCase_dev.pdf` (this is the Windows version)
 - ➔ a subset of the manuals that I think might be of interest are in a directory on the `devnfs1` machine, accessible with the URL <http://devnfs1.mash.AWC-NJ.com/dist/share/ClearCase/Manuals> (the files have been renamed to make them more recognizable, from the more cryptic names they have in the Windows directory—e.g., the above manual is in the file I have named [Working-in-Base-ClearCase-unix.pdf](#)).

- There are also a pair of comprehensive *Command Reference Manuals* (about 600 pages each), for subsets *A-L* and *M-Z*. These are available graphically online as mentioned above, or as PDF files (in the locations given above).

Note that ClearCase has nearly 4,000 pages of documentation, across 12-14 manuals. You should rarely need to access any of this other than the *Working in Base ClearCase* manual or the *Reference Manuals*, as your use of ClearCase will be fairly simple. But it is there if you want it.

2.2 Basics of the ClearCase Permissions Model

The ClearCase permissions model—built on top of the Unix model originally—is rather complicated (over 50 pages in the *ClearCase Administrator's Guide* is devoted to explaining it), but you need to understand some basics to make use of it without getting into trouble. If you don't you may be unable to access files in a view or you may create files in a vob that others cannot use. For a full discussion of the permissions model, see [Chapters 3-5 of the *ClearCase Administrator's Guide*](#).

The basics that you must know can, I think, be summarized as follows:

- Unless a project or application is setup specially, all files in ClearCase vobs are readable by anyone on the system, as long as they have a *view context* set and the view's *configuration spec* yields a version of the file for viewing.
- In general, and within AWC-NJ, shared vobs are “writable by group” so that developers can collaborate on code. To maintain this for files and directories you create, your *umask* setting when working in ClearCase should be 002, or `-rw-rw-r--`. Of course, if you are working in your own view, your checked out or view-private files are not writable by others (unless you share your view).
- To do “writes” in ClearCase—to check out or check in a file, to create a `.o` file as the output of a build being done in a view, to modify the attributes of an element—a user must belong to one of the group ID's associated with a vob. In general, the special group used in AWC-NJ for all vobs is `vob_two` (admittedly a silly name, given to us by SF) and all developers here have been made a member of that group. However, most developers are not yet setup to have `vob_two` as their primary, or login, group (use the `id(1)` command on Unix to see your primary group and the `groups(1)` command to see all the groups you belong to).

This rule—alone—lets you create a new file or directory in a vob if you are a member of `vob_two` but, for example, currently have `dev` as your primary group. However, the group ID associated with the file or directory you create that way will be `dev` and its *umask* setting will be your current one—and neither of these will necessarily allow other developers to write to this file or directory.

Thus, the last rule you need to remember.

- To ensure that files and directories you create within a vob in ClearCase, from within one of your views, are usable by others on your project and others within AWC-NJ, you should always make sure your *primary group* and *umask* setting are correct—for collaborating with others on your project—when working in ClearCase. Although not needed to read files or work “alone”, you will cause headaches for your group and work for the ClearCase administrator if you do not follow this practice.

For the above reasons, *[we highly recommend you use a personal script or shell function to ensure your group and umask are properly set when using a ClearCase view—having your group set to vob_two and your umask set to 002.](#)*

2.3 On Various Platforms

2.3.1 On Your Unix Workstation

On a Unix workstation, the following steps may need to be done the first time you try to use ClearCase after it is installed:

- A) You must add the ClearCase directory `/usr/atria/bin` to your `PATH`. If you put it first in your `PATH`, or at least before the `/usr/bin` entry, then there is a hard link of `ct` to the `cleartool` binary, and you won't need to define an alias for `ct` to get `cleartool`. (What you give up is the ability to use the executable `ct(1)` without giving a full pathname—but who dials a telephone modem anymore from the command line? ☺)

Once you have done so, try the commands `cleartool lsvob` or `cleartool lsview` to see if you recognize your vobs or your view. (Is there an asterisk (*) preceding the *tag* in this output?).

- B) [ksh users]: If you want, define an alias for the `cleartool` command in your `.kshrc` file: `alias ct=cleartool`. (I will use `ct` in the examples in this document.) This is not necessary if you put the ClearCase bin directory before the `/usr/bin` directory in your `PATH` setting (see item A above).

- C) You must *set* the *view* that has been defined for you. For example:

```
ct setview jsmith_0      # the view tag is jsmith_0
```

- D) You should change the group of your current process to `vob_two` using the `newgrp` shell built-in command, and change your `c` to `002`. For example:

```
$ id -a
uid=2068(tgill) gid=102(dev) groups=102(dev),30(vob_two)
$ newgrp vob_two
$ id -a
uid=2068(tgill) gid=30(vob_two) groups=102(dev),30(vob_two)
$
```

All ClearCase users in NJ must be a member of this group to do work in ClearCase. Eventually, this may become your default group so that you won't ever need to use the `newgrp` command. *Since setting a view invokes a sub-shell, you may need to use the `newgrp` and `umask` commands each time you set a view. See the FAQ section below for a method to do this in one step.*

NOTE: changing to the group `vob_two` is only required for official, final vobs; if you are working with a test vob in early 2003, you can ignore this step. ALSO, if you have a restricted vob, available only to certain developers, you will have your own Unix group with which to do work on that vob. But the same approach applies.

- E) If there is not an asterisk preceding the vobs you care about in the output of the `ct lsvob` command above, you will need to tell ClearCase to *mount* the vobs on your system. For example, the following fragment shows the output of the `lsvob` command before and after a `mount` command:

```
$ cleartool lsvob -s
/vobs/TIM/AutexData
/vobs/TIM/Server
/vobs/TIM/MktData
$ cleartool mount /vobs/TIM/Server
$ cleartool lsvob -s
/vobs/TIM/AutexData
* /vobs/TIM/Server
/vobs/TIM/MktData
$
```

Note that the mounting of the vob that ClearCase does is a separate, additional type of mounting to the Unix file system mounting that of course must also have occurred (and does automatically for you via the NFS automounter).

2.3.2 On Your Windows PC

On a Windows PC, there are a few aspects to using ClearCase that you should know about right away:

- Part of the installation of ClearCase done by the ClearCase administrator will be to setup 3 mounts of Unix file systems: the vobs' file system, the views' file system, and your initial view. These are mounted from the relevant Unix ClearCase server. They should automatically re-mount each time you re-login to your PC. I have set it up so that the vob file system mounts as the V: drive, the views file system as the W: drive, and your views as the Z: drive.
- An application called the ClearCase Doctor will run by default each time you login to Windows. It only takes a few seconds and might detect a problem that will impede your use of ClearCase, so I would let it run each time, check that the output is OK, and then close it. For most users in AWC-NJ, the Doctor will show a "warning error" for an element in your Windows path setting that is across a network, which it recommends you change; at this time, we are ignoring this warning.

2.3.3 On an SCO Open Server Development Machine

2.3.3.1 Background

ClearCase does not run on the SCO Open Server operating system that AWC-NJ uses for some of its applications. Thus it cannot be installed on such machines, and you cannot use ClearCase commands on such machines.

However, ClearCase includes a feature called *exported views* that allow you to access ClearCase data—files in your vobs—from non-ClearCase machines. We will be using this feature in AWC-NJ to allow us to migrate the source code for SCO applications to ClearCase and maintain it there, while still building the code on the SCO machine and deploying it there.

To allow this access, the administrator has to mark each vob and each view for which such access is desired and export the view to the SCO machine(s) that need it, and then mount the exported view as a normal NFS file system on that SCO machine. This will be done each time some SCO application is migrated to ClearCase. The one issue that will need to be worked out between the ClearCase administrator and the SCO application developers is the views that will be used for such code—they will be a little different in name and number from the other views you will be using, and probably shared by multiple developers.

2.3.3.2 AWC-NJ Conventions for Exported Views

The ClearCase administrator will establish conventions for the exported views created for the SCO code. Although we only have test vobs for now (as of early June), these conventions will probably be something like:

- The name of views that are exported will contain the name of the vob and the word "exported", such as `exported_OMS_OrdDirector`. If we need the developer's name in the view too, it might have a name like `exported_OMS_OrdDirector_skramor` instead.
- The VOB's files will be available on the relevant SCO machines in the usual place, according to our vob mount point conventions used on SunOS machines, using the mount point `/vobs`; thus the `OMS_OrdDirector` vob would be available on the SCO machine in the mounted file system `/vobs/OMS_OrdDirector`, just as it is on a SunOS machine.

For now, all SCO code in the test vobs is available through a single view, `testSCOexported`, which all developers can share. The mount points on the SCO machine use the convention above; e.g., the `TESTOMS_OrdDirector` test vob is available at the mount point `/vobs/TESTOMS_OrdDirector` on the SCO machines `dev16` and `dev27`.

2.3.3.3 Using ClearCase Files on the SCO Machine

The basic model for using ClearCase files on an SCO machine is as follows:

- The developer does all required ClearCase work for the SCO code on a SunOS machine that they have ClearCase installed on: check-outs, check-ins, creation of new elements, etc. This is done in the view being used for exporting the files to the SCO machine.
- The developer then switches to the SCO machine to do the editing of the files and the builds. All operations possible with Unix on NFS-mounted files are possible on the ClearCase files in the exported

view, as long as they are writable (checked out). And view-private files can be created on the SCO machine.

This model probably works best when a developer has 2 `xterm` windows next to each other, both in the same exported view, but one window is in a real view set on the SunOS machine and the other window is in the mounted *exported view* on the SCO machine (it is not “*in the view*” in the same way, since this is not a ClearCase machine).

3. Development Models & Policies with ClearCase

Using ClearCase effectively on a software project requires the establishment of a *development model* and *development policies* to help enforce that model (some policies can be enforced via ClearCase mechanisms). Initially, projects often want to have a model that reflects how they have been working without ClearCase, and to some extent, this is possible.

3.1 The Model to Be Used for AWC-NJ Applications

As part of the overall effort in the AWC-NJ development organization to move to ClearCase-based development and to adopt a *release management* mechanism in which only a designated *release engineer* releases applications to Production, we will attempt to move all development to the same development model or variants thereof. This section describes the basics of it as it will be used for the LOD project in that project's initial use of ClearCase. A slight variation of this is being setup for the VISTA project.

The main parts of this approach will be:

- common approaches to code changes on the main line and branches—a *code change model*
- naming standards for branches, their associated views, and labels
- a deployment process to Production

3.1.1 Two Common Approaches to Code Changes

There seem to be 2 approaches used currently in AWC-NJ to do code changes, both of which can be carried over into ClearCase in a parallel manner:

- everyone works on the main code line, as they never work on the same files and don't interfere with each other's work
- a lead programmer maintains the main code line and all feature work and bug fix work is done on a branch

We will call the first approach the *main line code change model* and the second one a *lead programmer code change model*.

3.1.1.1 Main Line code change model

In the *main line code change* model, all developers work on a single code base. This usually works well only for small projects with two or three developers, or slightly larger projects when it is clear no one ever works on the same files. With this, the code is built on the mainline and deployed from there when it is tested and ready. No branches are needed in the model, and a single view with a default config spec can be used by each developer. Each developer's work is isolated in their view until they check-in their code (which goes to the main line), at which point their changes are available to and used by others. Since the project is small, this model can work because the few team members are communicating constantly and unlikely to break the other's code.

3.1.1.2 Lead Programmer code change model

In the *Lead Programmer code change* model, only the lead programmer can check-out or check-in files on the main line—all other developers must work in an isolated workspace (the lead developer typically does feature work and bug fix work on a branch as well). Outside of ClearCase, that workspace might be known as a sandbox (typically, a copy of the source code in each developer's home directory); within ClearCase, this is called a *task* and is done using a combination of a *task branch* and a *task view*—in this model, a branch and a view are always associated, and when developers speak of “that code is on the “xyz bugfix branch” they know that there is a corresponding view used to make changes to that branch. When the code for that task is done—whether it is a new feature or for bug fixes—the changes are *merged* back to the main line of code; the lead developer takes responsibility for this merge. ClearCase provides merging tools to make this process easier. In this model, the `/main` branch is usually *locked* so that only the lead developer can make changes to it, when merges are done.

3.1.2 Naming Standards

This overall model and process require naming standards for 3 objects in ClearCase: *branches*, *views*, and *labels*. The standards are driven by both ClearCase convention and local needs. Such standards make things easier in two main ways:

- ✓ Developers recognize what project/task/code they are working on from the structure of the names.
- ✓ The ClearCase administrator can have standard scripts to facilitate managing the objects so named.

3.1.2.1 Branch Names

Branches are used only in the *Lead Programmer code change* model. They are not needed in the *main line code change model*.

- Branch names are all lowercase (ClearCase convention).
- Branch names have a site prefix, such as nj_ or njpy_ or sf_ (ClearCase MultiSite convention).
- Branch names contain identifying information about the task (nj_orh_bugfix_1.11) and/or developer doing the task (nj_tgill_orh_feature_B).

3.1.2.2 View Names

- View names are lowercase or mixed case.
- View names contain identifying information about the developer (lvolchk_0), the project and task (VISTA_orh_bugfix_1.11), or the developer doing a task (VISTA_tgill_orh_feature_B).
- View names for views used on Windows will have a suffix of _NT to distinguish them from views used on Unix (SunOS or Linux).

3.1.2.3 Label Names

- ClearCase label names are all uppercase. A *label* is applied to a version of an element.
- Label names maintain a common naming scheme for application releases. I suggest a 3-level approach that allows for major releases, minor releases, and bug fix releases:

R2.0	major release (always a "0" in 2 nd digit)	frequency: every 6-12 months?
R2.2	minor release (never a "0" in 2 nd digit)	frequency: every 1-6 months?
R2.0.1	bugfix release (always a 3 rd digit)	frequency: as needed

Longer label names typically make it harder for developers and others to read the graphical version trees that developers routinely use to look at their code structure using ClearCase tools like `xclearcase`, `cleartool lsvtree -g`, and the like (see Figure 1 on the next page). And there is no need to duplicate the name of a project in the labels used for it, like the label `LOD_release_1.1.2`. This 3-level naming approach can be expanded with a letter suffix, as in `R2.0.1A`, or even a 4th digit if really needed.

3.1.3 A Deployment Process to Production

With either development model, the same deployment process to Production will be used (at least in the long term). The basics are simple:

- When the developers finish the development of the code for a release to be deployed (to QA or Production), the *release manager* takes over.
- The release manager labels the code with the *release number* for the code going to Production, builds it in his own environment in ClearCase using only the code that is labeled, and deploys it to production.

The label allows fixes to that deployed release to be made, even if the main line continues being developed before any bugs are found in production.

3.2 How LOD Will Use This Model

LOD, not using ClearCase yet in Jan 2004, currently has developers working on new features in a sandboxes separate from the daily bug fix work of the lead programmer (Anna) that goes into daily production. When she is ready for a new feature, the lead developer (Anna) merges the code, by hand (using her own tools) from the other developers' feature sandboxes into her production release. The daily bug fixes are done both in Anna's main code and in sandboxes of others.

The LOD project will use the *Lead Programmer code change model*, with a *shared branch* for bug fixes for each release. Since each developer's feature work is separate, this isolation to their own task branch gives them a ClearCase equivalent to the sandbox they use now. While they work on their feature over the course of some weeks or longer, bug fix work for production code can be jointly done in a *shared view* on that release's bug fix branch.

3.2.1 Summary of LOD Model

This approach to development for LOD in ClearCase can be summarized as follows:

- a. A *task branch* exists for each feature—one per feature (or related set of features), per developer, with a corresponding view.
- b. A *task branch* for bug fixes exists for each major (R2 . 0) or minor (R2 . 1) release—as well as a corresponding *shared view* in which all bug fix work for that release (by all developers) is done
- c. All code deployed to Production is built after it is labeled with the *release number*:
 - ♣ Code for major and minor releases is labeled (e.g., R2 . 0 or R2 . 1) and then deployed to Production only from the `/main` branch, after merging from branches is done by the lead developer.
 - ♣ Code for bug fixes is labeled (e.g., R2 . 0 . 1) and then deployed to Production only from the bug fix branch for that release, NOT from the `/main` branch. The bug fix branch for each release is worked in a *shared view*, by all developers who need to make fixes.
- d. All branches (for both features and bug fixes) are *based on*—*branched off of*—a *labeled* major or minor release, from the `/main` branch.
- e. Bug fix branches are done for each major or minor release, as needed, and bugs fixes for that release are done only on the branch.
- f. Merging to the `/main` branch is done by the lead programmer, and done when she is ready to release a major/minor release to Production. She must merge:
 - ♣ features from whichever feature branches are needed for the next release
 - ♣ bug fixes from the previous release's bug fix branchShe can continue to use her own tools, or learn to use the ClearCase tools.
- g. Views are removed and branches locked when they are no longer needed:
 - ♣ A feature branch is no longer needed when its code has been merged to the `/main` branch.
 - ♣ A bug fix branch is no longer needed when its corresponding release is no longer anywhere in production.Of course, the code remains on the branches, with all labels: locking simply prevents further checkouts from being made on the branch.

The next section will show this flow with an example file and its version tree.

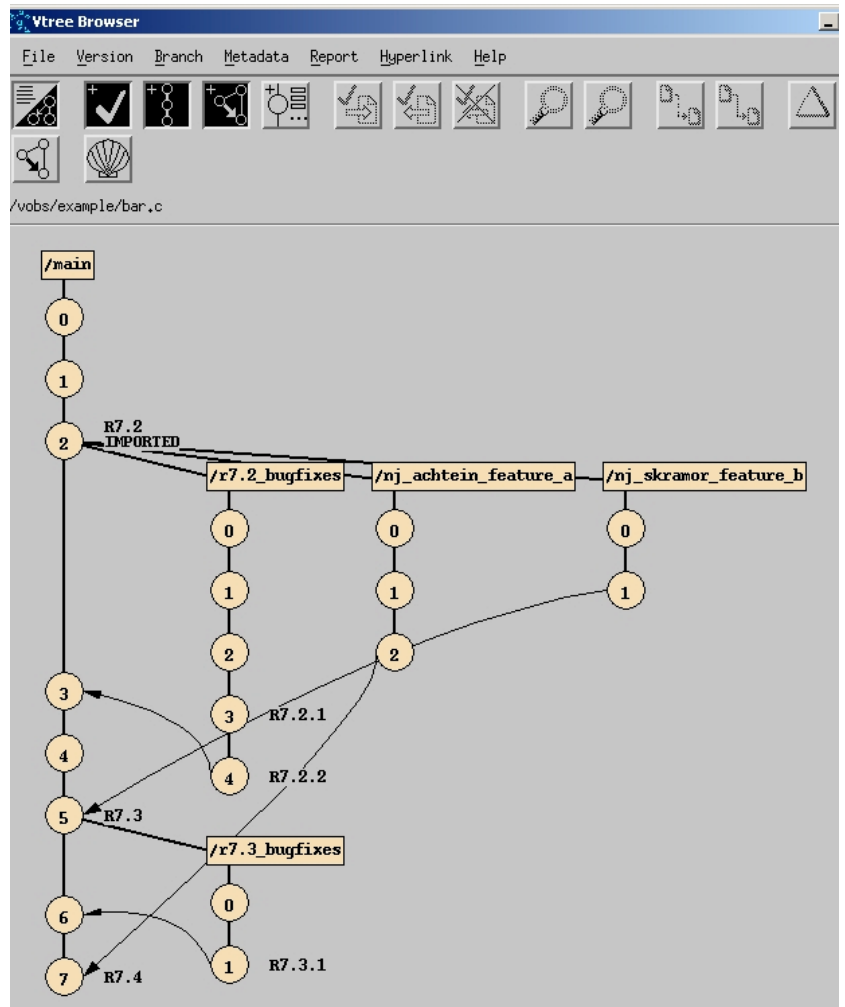
3.2.2 Visual Example of LOD Model

The flow of development using the above summary of the LOD development model can be described as follows, using the version tree for a single element, `bar.c` (see Figure 1).

- The latest code, outside of ClearCase, is, say, release 7.2 and is brought in again to ClearCase; it is labeled both `IMPORTED` and `R7.2`.
- At this point, MKS will be locked so that all future development proceeds in ClearCase only.
- From this labeled release, branches (and corresponding views) are created, for bug fixes for release 7.2 and for two new features (one being done by Anna, one by Sergei).
- Several bug fixes are made in the bug fix branch and released to production as releases `R7.2.1` and `R7.2.2`.
- When Sergei's "feature B" is ready, Anna merges both the bug fixes from release `R7.2` and the code from Sergei's `nj_skramor_feature_b` branch into the `/main` branch (at versions `/main/5` and `/main/7` respectively) and labels it as release `R7.3` before deploying to production.
- Anna's "feature A" is still not ready, so she continues working on it even after bug fixes start happening for release `R7.3`.
- When her feature is ready, its code and the changes from the bug fix branch for `R7.3` (which had 1 labeled release, `R7.3.1`) are merged to the `/main` branch and released as `R7.4`.

This cycle continues. As it becomes stable, effort can be put into other aspects of the overall goals Tom has for the development environment:

- deployment to Production done only by the *release manager*
- building applications without reference to arbitrary file systems for libraries and include files
- building the whole ETS as a single system



4. Commonly Asked Questions

These questions are for now just in the order that they seem to get asked. This section may be re-organized at a later time.

4.1 Basics

The most basic questions are in this section. But please take 20 minutes and use the ClearCase tutorial and help facilities to learn (re-learn) the basics:

- on Unix, run the command `/usr/atria/bin/hyperhelp cc_tut.hlp`
- on Windows, run `Start → Programs → Rational ClearCase → ClearCase Tutorial`.

For a fuller introduction to using views, read [Chapter 3, “Working in a View,” of the *Working in Base ClearCase manual*](#).

4.1.1 How do I reference a manual page for each *cleartool* subcommand?

[Unix answer]:

```
$ cleartool man mv                # ASCII version
$ cleartool man -g[raphical] mv   # graphical version
```

Or you can access the *Command Reference* manuals, in PDF format, on the machine `devnfs1`, in the directory <http://devnfs1.mash.AWC-NJ.com/dist/share/ClearCase/Manuals>.

4.1.2 How do I get and use a view?

Views are created by the ClearCase administrator, and typically when you first are told that ClearCase has been installed on your machine, he will let you know that you have a view as well. Usually, the view name (called a tag) of your first view (you can have several) is your Unix ID followed by the 2 characters “_0” as in `tgill_0` or `dsagoo_0`. However, there is a command you can use to find out the names of your views:

[Unix answer]:

```
$ cleartool lsview -s | grep $LOGNAME
```

4.1.3 How do I do check-outs and check-ins of files?

[Unix answer]:

```
$ cleartool checkout -comment 'this is the comment' file.c
$ cleartool co -c 'this is the comment' file.c           # same as above
```

If you don't supply the `-comment` operand, ClearCase will prompt you for a multi-line comment, which you end with a “.” (period).

```
$ cleartool checkin -comment 'this is the comment' file.c
$ cleartool ci -c 'this is the comment' file.c           # same as above
```

If you don't supply the `-comment` operand, ClearCase will prompt you for a multi-line comment, which you end with a “.” (period); but if you supplied one with the `checkout` command, it will be used if you don't give one.

4.1.4 I don't want my checked out file anymore. What do I do to undo it??

[Unix answer]:

```
$ cleartool uncheckout file.c
```

4.1.5 How do I see the version of the files in my current directory, in my view?

[Unix answer]:

```
$ cleartool ls -l
```

4.1.6 I have made changes to a copy of a file and now I want to check-it in. How do I do it?

ATTENTION: This is a bad practice and a developer error—DO NOT DO IT!

The whole point of doing a “check-out” is to inform the ClearCase database that you are now going to make changes to a file—so always check-out a file before you make changes to it (in ClearCase, the file is not even writable unless you check it out—this should tell you something!).

A check-out allows ClearCase to prevent other developers from checking out the same file in a reserved manner, and to force one or the other of you to merge your changes when you do the check-in, since more than 1 set of changes was being made at the same time.

The above information applies to developers working on the same branch, typically the /main branch. On larger projects with more developers, each developer might work on a personal or task branch, and this problem would not happen; instead, the versions on different branches would have to be merged. Initially, all AWC-NJ applications will just be using a /main branch, however.

4.1.7 How do I add a file or directory to ClearCase, or move one, or remove one?

Since ClearCase manages versions of directories, not just files, you must check-out a directory to make a change to it. And since adding a file to a directory, or moving files between directories, or removing a file from a directory, changes the contents of the directory, these operations require that the corresponding directories be checked out first.

[Unix answer]:

```
$ cleartool co . # check-out current directory
$ cleartool mkelem newfile.c # create new file element
$ cleartool mkdir newdir # create new directory element
$ cleartool mv oldname.c newname.c # rename an element
$ cleartool rmname junk.c # remove an element
$ cleartool ci . # check-in current directory (make changes permanent)
```

4.1.8 A directory I just created is not writable by others. Why not, and how do I fix it?

The file permissions on a directory element created with the `cleartool mkdir` command are governed by the *primary group* and the *umask setting* of the creating process. Typically, at AWC-NJ, the primary group for developers is `mks`, and most developers do not have a `umask` setting allowing group write permissions; the default behavior, then, is that directories in ClearCase created by a developer will have the wrong permissions: even though others can check out the directory to make changes, they cannot write to it (e.g., a view private file like a `.o` file created during a build). You can use the `cleartool protect` command to fix this.

[Unix answer]:

```
$ cleartool protect -chmod 775 -chgrp vob_two <pathname>
```

The group `vob_two` is the one required in AWC-NJ for all ClearCase users to have common write access.

4.1.9 You say I should be in the group `vob_two` when working in ClearCase. What's an easy way to do this?

The easiest way to ensure you have the group `vob_two` as your primary group when you work in ClearCase is to create a shell function that invokes the `newgrp` command with the correct `-exec` option to change the group of the new shell.

[Unix answer for ksh users]:

```
# put this function definition in your .profile or .kshrc file:
function ve { ENV=~/.kshrc cleartool setview -exec 'newgrp vob_two' ${1:?};}
```

With this defined in your interactive shell, you can simply enter the command

```
ve { ENV=$HOME/.kshrc cleartool setview -exec 'newgrp vob_two' ${1:?};}
```

and you will enter the sub-shell with your view context set, with `vob_two` as your primary group, and your `.kshrc` file invoked to define your aliases and functions. Alternatively, you could define a function just to put you in the new group, anytime:

```
function ng { ENV=~/.kshrc newgrp -l vob_two;}
```

I prefer this method as it ensures my function and alias environment is always intact.

4.1.10 How do I see the changes I've made to my checked-out file?

This is an instance where the *version-extended pathname* mechanism in ClearCase is particularly helpful. With such a pathname, you can simply use the `diff` command (or the `cleardiff` command, the ClearCase diff'ing utility) to compare the latest version with the one you are editing. And this is something for which it is worthwhile defining a shell function in your `.kshrc` file for repeated use.

[Unix answer]:

```
$ diff foobar.c foobar.c@/main/LATEST # ad hoc
$ function diffcc { diff $1@/main/LATEST $1 | pg;} # define function
$ diffcc foobar.c # same result using function
```

4.1.11 How do I see all the changes made recently in this directory? Or the comments made with previous check-ins on a file?

[Unix answer]:

```
$ cleartool lshistory -recurse -since Monday
$ cleartool lshistory foobar.c
```

4.1.12 How do I see what files are checked out?

[Unix answer]:

```
$ cleartool lsco # all check-outs in current directory
$ cleartool lsco -recurse # all check-outs in current directory & sub-directories
$ cleartool lsco -user tgill # all check-outs in current directory by user tgill
```

4.2 There are no files in my VOB!-

[Unix answer]: If you `cd` to the root directory of your VOB and find there are no files there, the most likely reason is that you forgot to mount the VOB. To access any files in a VOB, you must **both** have a *view context* and have the *VOB mounted*. See the earlier [section](#) on the basics of using ClearCase in Unix). Typically, all public vobs are mounted when your machine boots, but sometimes this may not happen and you have to mount them yourself.

4.3 What Does the Message “checked out but removed” Mean? Or, Why Has My Checked-out File with All My Changes Disappeared?

The designation of “checked out but removed” for a file, on the right side of the output from doing a `cleartool ls -long` command, means that the file has been checked out from ClearCase (using `cleartool co`) but has been removed (with the `rm` command?) or moved. If you do not remember removing it, the perhaps

you made a common mistake in an MKS-going-to-ClearCase environment: using the MKS command `ci` instead of the ClearCase command `cleartool ci`.

If you have made changes to a checked out file in ClearCase and try to check it in, but you type

```
$ ci foo.c
```

instead of the command

```
$ cleartool ci foo.c
```

then you will be executing the MKS command for checking in a file (`/mks/dist/mkssi/sol2/ci` probably), instead of the ClearCase command you wanted. What happens is that the MKS `ci` command asks for comments (you're used to seeing that), and then creates an RCS archive file in the location `./rcs/foo.c` and then removes the `foo.c` file in the current directory. Since it was a checked out ClearCase file, you get the "checked out but removed" message. If you look closely at your output from the `cleartool ls -long` command, you will see there is a view-private directory named `rcs` in your current directory, and it contains a file with the name you are looking for (but this file is an RCS archive file and cannot be used directly). For example:

```
$ cleartool ls -long
version          bar.c@@/main/1          Rule: element * /main/LATEST
version          car.c@@/main/3          Rule: element * /main/LATEST
version          dog.c@@/main/1          Rule: element * /main/LATEST
version          foo.c@@/main/CHECKEDOUT from /main/4  [checkedout but removed]
version          mot.c@@/main/2          Rule: element * /main/LATEST
view private object rcs
version          zoo.c@@/main/5          Rule: element * /main/LATEST
$ cleartool ls -l rcs/foo.c
view private object  rcs/foo.c
$ head rcs/foo.c | nl          # show that it is an RCS archive file
1  head 1.1;
2  branch ;
3  access ;
4  symbols p16:1.1 p14:1.1;
5  locks ; strict;
6  comment @@;
7  1.1
8  date 2002.01.23.20.03.05; author tsmith; state dev;
$
```

TO FIX THIS:

- a. Use the MKS command `co` to get the file out of the archive and into the current directory.
- b. Edit the file again to make sure it has the changes you want.
- c. Remove the RCS archive file (which was created as a view-private file in your view):
`/bin/rm -fr rcs`
- d. Do the ClearCase check-in you wanted:
`cleartool ci foo.c`

TO PREVENT THIS:

The easiest way to prevent this is probably to define some aliases (assuming you use Korn shell):

```
alias ci='cleartool ci'
alias CI=/mks/dist/mkssi/sol2/ci
```

to force you to remember to use the CAPITALIZED version, `CI`, when you want the MKS version. You could do the same with short scripts in your `$HOME/bin` directory and put it first in your `PATH`.

4.4 When I run make in a view, my build fails because touch fails to update my .c files

A problem that has occurred for some AWC-NJ developers has to do with a technique used in some of the `c` used here. Some `makefile`'s assume that the build can update files in the current directory, e.g. by using a `touch` command within the `makefile`. But this is *an incorrect assumption* in a ClearCase environment. Keep in mind that under ClearCase, files can only be accessed in a view context, and vob files in that view are read-only unless they have been checked out. But in a release management build process, no files will be checked out—one of the requirements for a build is that all files are checked in!

The first example seen of this is a `makefile` technique that uses a rule with `touch` to ensure that a C++ file with the suffix `.cpp` is compiled when only the corresponding `.hpp` file has been updated, by touching it when the `.hpp` file is detected as having changed:

```
.hpp.cpp:
    @echo "$< has been changed..."
    touch $@
```

However, this fails in ClearCase unless the `.cpp` file is checked out, since you cannot `touch` a read-only file. To get around this problem, the following rule can be used instead—it causes the required re-build of the `.cpp` file in a ClearCase environment:

```
.hpp.o:
    @echo "Making $@..."
    $(CC) $(CCFLAG) $(INCLUDE) -c $*.cpp
```

TO FIX THIS:

Change your `makefile` to not use the `touch` technique. If the above example does not seem to apply to your case, but your builds are still failing in ClearCase because of the use of a `touch` command, talk to your local `makefile` expert and find another way to accomplish what the `touch` is doing.