Professional Expertise Distilled

IBM Rational ClearCase 7.0: Master the Tools That Monitor, Analyze, and Manage Software Configurations

Take a deep dive into extending ClearCase 7.0 to ensure the consistency and reproducibility of your software configurations

Foreword by Lars Bendix, Ph. D., ETP, Lund University, Sweden

Marc Girod Tatiana Shpichko

[PACKT] PUBLISHING enterprise
professional expertise distilled

Table of Contents

1
Using the command line

Something the previous chapter (**Teaser**) displayed already is our focus on the UNIX command line, as a primary working environment.

We see it as a means to maximize power and management over one's work. This gives access to scripting, in portable and efficient ways, as well as to accurate and relevant documentation.

We will devote a few words of justification to the command line interface. This will be followed with a few hints on Perl usage, and on tool configuration.

Rationale (pun intended)

There is a general perception that graphical user interfaces are a sign of modernity (in a positive sense). Also that they free the user from having to learn the syntax of tools, and thus bring in simplicity and help her to focus on her own tasks.

We do not think this is true. GUIs hide more than they show, and introduce discontinuities, thus making it more difficult to reverse engineer the hidden assumptions of their authors.

A possible objection is that GUIs and the command line would be addressed to and used by different groups of people, maybe in different roles. Let's stress that this only enforces our point. We do not believe anyway in such assertions, our experience being that:

- There is no task that couldn't be better carried with appropriate, possibly self-configured, command-line tools than with a GUI; we reject the idea that either would, for instance, be better or worse adapted to *advanced* tasks.
- The use of a GUI encourages a *back-seat driver's* attitude, the dispatching of one's responsibility to others, which fights the spirit and intent of **Software Configuration Management**.

- Encouraging practices that might lead groups of people, who are meant to collaborate on common goals, to ignore one another, is a dangerous idea.
- The command line is not a monolithic tool that would have to be *mastered* as a whole; it is easily extendible via scripting.

But be it one way or the other, GUI users want to use them, not to read books. Besides, the existing literature already covers them extensively. We thus propose to cover the other half of the world.

Finally, we do not intend to compete against IBM, which nowadays provides instructions on clicking on the right (or sometimes the left) button as videos.

Against intuition

GUIs aim at being *intuitive*. We do not deny them some success in this direction, which explains the satisfaction of many users. We only believe this is a misguided strategy, conflicting with the goal of *making sense* of one's, and others' work, and this for several reasons:

- Intuition doesn't scale**.** There is a limit in terms of complexity to what may feel intuitive to anyone.
- Intuition doesn't communicate: intuition is an event, and not a process.
- Intuition is hard to question or to validate.
- Intuition is only for humans; it is awkward for tools.

We'll follow Nobel Prize winner Daniel Kahneman (***Maps of Bounded Rationality: Psychology for Behavioral Economics*** in stating:

Intuition and reasoning are alternative ways of solving problems.

And we'll just opt for reasoning.

Some graphical tools such as maps are obviously very useful. But as Kahneman notes, consulting a map involves reasoning, not intuition. A large source of intuition in common GUIs is based on metaphors such as the desktop or ... the map (especially following *Google Earth*), which appeal to experiences familiar to end users, precisely because they are remote to *software*.

The continuity of reasoning

Let's take the perspective of users who intend to learn as much as they need, and not necessarily more but in a continuous, that is manageable, way. Users who intend to share their experience with others, by comparing their progress gradually, and reproduce the steps they take, at their own pace.

Reasoning, and building up one's understanding, is a process that takes time and goes through states. It is useful and safe to identify and validate these states with others: this defines a path. There may of course be many roads from one point to another, and nobody is forced to take the same road as anybody else, but it is essential to identify the roads in a way allowing one to compare them. This identification operates in practice on discrete states. It may feel paradoxical, but continuity is a result of using discrete representations instead of more *continuous* graphical ones.

Text is well-suited to record the progression of reasoning, in a way easy to trace by the next pilgrim.

Recording achieves an essential *SCM* concern: *re-produce* rather than produce (from scratch, re-inventing the wheel) and *re-use*. We think that one ought to use an SCM tool (ClearCase) in an SCM way! One's investment will be rewarded.

Another important issue is the possibility to get access to the specific outputs of different tools, and to be able to analyze them by parsing, searching, and comparing relevant details rather than being stuck by the opacity of a pop-up window. Nowadays, the *production* of documents, be it text or not, is assisted in frightful ways, which puts the *reader* to face ever more difficult challenges. Plain text is her best chance to practice *active* reading, that is, to decide precisely what information she wants to focus upon and to actually make it emerge out of the surrounding noise. Passive readers most often have the information they need at their disposal, but fail to notice it; they are instructed by experience to avoid spending the effort necessary to read it!

The users we target are not alone, and their reading is well assisted by powerful tools—text being still significantly better supported than graphics. Not only is there a profusion of existing tools, but there is a world of scripting that allows the user to maintain her own ad hoc tools.

Illustrations

French sociologist Pierre Bourdieu explained in *On Television* that communications takes time; instant communications is no communications at all. Graphical illustrations are often misleading in this respect: they tend to overspecify a situation for the purpose of giving a representation of one single aspect. The constraint of producing an image implies to complete the useful parts with details, which otherwise would be left undetermined. We'll pay a special effort to avoid this mistake, and will therefore restrict ourselves to producing textual illustrations, that ought to be read from a beginning to an end, and therefore lend themselves to critical analysis from a careful reader.

Text, shell, and terminal

When we think of text, we think at the same time of the format of files and of tools to operate on the system: **terminals** and **shells**. The files are the natural logs of this system operation. Of course, most text editors allow us to modify text in arbitrary order, but each text file implements the convention of time—from its beginning to its end. Note how there is nothing similar with graphics (apart for cinema).

Storing commands as text files and processing text from files in the same way as from any text stream is called **scripting**. The boundary is not meant to be obvious, and we will have an example below.

This brings us to the point that *text* is not as straightforwardly and universally defined as one might naively assume; it is subject to encoding conventions, and rendering formats (for example, to support colors or fonts). The goal of preserving contents through text-to-file-and-back transformations, leads one to restrict the definition to the barest (restricted character set, single font, single color). We still have to pay attention to one crude issue: incompatible end-of-line conventions. There are actually three: UNIX (one character, *linefeed*—ASCII 10 decimal), Windows (a two character sequence: *carriage return* linefeed—ASCII 13 - 10), and Mac (carriage return, #ASCII 13, alone). One way is to take care of conversions, and the problem is to decide at what point. One option, which we'll touch in *Chapter 2*, *Presentation of ClearCase*, is to delegate it to the view. The other, clearly inferior option, is to use checkin triggers. The best is to raise the level of understanding of users, and to have them stick to tools consistent in this respect, that is, in practice avoid tools that enforce the Windows convention.

The *shell* is the program in charge of executing interactive commands (on UNIX, it is distinct from the *terminal*). It presents annoying differences between Windows and UNIX such as:

- The use of back versus forward slashes as separators in directory hierarchies (including in the format of ClearCase *vob tags*—see Chapter 2). Backslashes are used in UNIX as *escape* characters to prevent the evaluation of special characters. The result is that they get consumed by the shell, so that to preserve them, one needs to escape (`\\word`) or otherwise *quote* them (`"\word");`
- Quoting has two flavors on UNIX but only one on Windows. Double-quotes ("`word`") allow the evaluation of variables in the quoted text; on UNIX, single quotes ('word') make it possible to prevent this.
- The UNIX file systems are all *rooted* from a common directory, marked as `/`. This is not the case on Windows where the situation is more complex: while file systems (rooted in \) are usually *mapped* to single letter *drives* (followed with a colon, for example, `C:`). On the other hand, a competing syntax may be used to access named *shares*, prefixed with double backslashes and host names (for example, `\\frodo\nest`).
- Shells support the expansion of *environment variables*, with different syntaxes: `$VAR` on UNIX and `%VAR%` on Windows. The definition syntax varies even between UNIX shells.
- Less visible, but let's mention it: the separator used in list variables such as `$PATH`/`%PATH%` differs from "`;`" (UNIX) to "`;`" (Windows). We'll have an example later with `MANPATH`.

This not so brief review should convince us that it is challenging to write simple commands, even just to run programs, in a way that would be portable (that is, stable) across the UNIX/Windows boundary. This issue is usually crucial in almost any organization using ClearCase: a typical environment comprises a large number of servers and workstations based on a variety of platforms with a tendency to run the servers under UNIX and the end-user workstations under both UNIX and Windows. May this be our reason to introduce Perl, as an answer to the portability issue.

Perl

In this book, as already mentioned in the *Teaser*, we'll focus on Perl as a scripting language for our examples. There are many reasons for this choice.

We already mentioned portability: Perl is available on all the platforms on which ClearCase is supported, and allows (with the help of a few suitable **modules**) to factor away and to hide the platform idiosyncrasies.

A second reason to choose Perl over other scripting languages (first of which the UNIX shells) is the existence of a debugger, and the model of compiling first and running only code that has satisfied a first check. Both of these are significant advantages to cover a range of users extending to the administrator. Note that the debugger may be used interactively from the command line (`perl -d -e1`), and "one-liner" commands are possible, not to force you to save scripts to files.

Then could come the existence of a mass of experience, both via the CPAN library network (`http://cpan.org`), and the many newsgroups and forums (see perlfaq2, below), thus giving access to a culture of communications and excellence.

Finally, one must mention that Perl has been used by the developers and distributors of ClearCase. However, at this point, we would recommend that the user avoids the apparent facility of using the instance of perl that is part of the ClearCase distribution and instead maintains her own installation in a ClearCase vob (obviously sharing it network-wise with her collaborators—see *Chapter 8, Tool maintenance*, and note that we implied this in the *Teaser*).

IBM has relatively recently improved its support for Perl, with a consistent ratlperl in the `common` tools directory. However, this doesn't answer the following points:

- It is not intended for the end users: it is tailored for the needs of ClearCase tools. Remember IBM is not committed to supporting it in any way.

- It is still a relatively old version and comes with a minimum set of modules, so you'll sooner or later get into limitations (check the available modules with `egrep ^=head2 perllocal.pod`, where the `perllocal.pod` is located in the platform subdirectory, which you get with `ratlperl -V`).
- You might think of installing modules yourself, but:
    - If you need to compile them, the compiler needed to build it is not provided—you'll fail to install even ***ClearCase::CtCmd*** (which is a Perl module provided by IBM on CPAN)!
    - Your changes will be washed away at the next ClearCase installation.
    - You'd need to make the same changes locally on every host, thus to have admin rights there.
    - In fairness, there exists in *perlfaq8* advice on *How do I keep my own module/library directory*, which makes it possible to maintain modules outside of the Perl installation, forcing the users to reference the shared library path.

We want to stress that Perl is a means to open ClearCase to its users, to enable them to tailor it to their needs. This is true for every user, not only for administrators with special needs! The main need it serves is it gets one freed from the GUI.

Perl documentation

One thing Perl encourages you (the user) to do is extend the Perl documentation, and does this by lowering the threshold for producing quality documentation. Although on a complete installation, the perl documentation will be available as man pages, perl offers a format, **pod**, and a tool **perldoc** to achieve these goals. This displays another argument in favor of text: *less is more*.

Perl provides a possibility to search through its documentation using the same perldoc tool, so that one can search through Perl FAQ, functions or modules:

```
$ perldoc -q 'regexp' # Searches free text in a form of regular expression
$ perldoc -f perl_function_name
$ perldoc -m perl_module_name
$ perldoc perlfaq2
```

Windows command prompt and alternatives

For Windows, the default terminal choice is the *Windows cmd*. It has a limited functionality, for example, no logging, limited buffering, few hot keys, a clumsy (first arbitrary choice) tab completion, and so on. But as restricted as it is, to use ClearCase, it is still much more powerful, and more functional than the GUI. Even some quite basic ClearCase commands (for example, create a symbolic link in the vob root directory, change file permissions, remove labels from a version, and so on) are only available on the command line.

Here are some useful hints on configuring the Windows cmd to make it more usable:

- Enable the Quick edit mode. Locate `HKEY_CURRENT_USER` -> `Console` -> `QuickEdit` in Registry Editor (`Regedit`) and set its value to 1. This enables easy copy-pasting (helpful in the absence of any other text handling utilities/keys): select with the mouse and press *Enter* for copying, and right-click to paste. This may also be set and unset in the command prompt window, in the **Properties** pane of the menu, by right-clicking the menu bar.
- Set appropriate *Screen Buffer Size* and *Window Size*. This also affects the ease to copy and paste!

Here is a small example of what one can do in Windows command line prompt:

```
> cleartool startview myview
> cleartool mount \myvob
> cd m:\myview\myvob\dir1
> cleartool ln -s ../dir2/file my_symlink
> cleartool mklbtype -nc LBL
> cleartool mklabel LBL ../dir2/file
> cleartool rmlabel LBL_1 ../dir2/file
```

This transcript creates a symbolic link, then creates a label type, applies a label to a versioned file, and removes another label from the same version.

A far better choice of a terminal for Windows would be **Cygwin**. Even the default Cygwin terminal window uses GNU bash shell, with all the expected bash features supported (such as Emacs key bindings, bash profile, and so on). Installing the Cygwin `ssh` package, one could use it conveniently from the same Cygwin terminal.

The terminal we use ourselves for both UNIX and Windows is GNU Emacs shell mode. Originally a text editor, GNU Emacs has developed, by the virtue of the genericity of text, into a full-blown integrated environment.

In Windows, one of the options is again to obtain Emacs as a Cygwin package, and to use it in Cygwin/X mode.

Emacs' ability to combine shell, file editing, man, and info modes, powerful multi-window and multi-frame system, unlimited customizing options, powerful logging, text-and-command search, and the ability to edit remote files via tramp/ssh, makes it a powerful choice, even for "normal" users, not to mention administrators. The pattern of using Cygwin Emacs is not without minor pitfalls such as the absence of the out of the box tab completion and `Ctrl+C` process signaling over ssh.

Let's compare Cygwin with a UNIX *terminal*.

Here is an example of what one can do in a UNIX terminal:

```
$alias ct=cleartool
$export DO='"do@@--04-11T14:50.3375"'
$ for i in $(ct lsvob -s); do if ct des lbtype:LBL@$i >/dev/null 2>&1; \
  then echo mkattr -rep ConfigRecord \'$DO\' lbtype:LBL@$i; fi; \
 done | ct
```

This UNIX shell code goes through all the ClearCase vobs, checking whether a label type `LBL` exists there. Every instance of `LBL` gets an attribute of type `ConfigRecord` and value `do@@--04-11T14:50.3375`.

This demonstrates the use of *pipes*, that is, a model of text as a stream, produced by one process and consumed by another—here cleartool. It also shows the use of basic programming language constructs such as *loops* and *conditionals* as part of the (here *bash*) shell, as well as the definition and use of an environment variable, together with a quoting issue (the `mkattr` function of cleartool requires that string values are enclosed in double quotes, which themselves have thus to be preserved from the shell). Note that the conditional tests the return code of a cleartool function (`describe`), both stream outputs (*standard* and *error*) of which are ignored by being sent to a special device, `/dev/null`, with a file interface. Note the use of `ct` as a shorthand for cleartool—we shall keep using it! Aliases need to be marked as expanded with the `shopt` command in order to work in non-interactive mode, as in a pipeline.

Now we can illustrate the annoying differences in Windows and UNIX shells explained earlier (in the *Text, shell, and terminal* section of the chapter). The UNIX transcript shown above would not work as such on Cygwin. The vob tags returned by the (Windows ClearCase) `lsvob` function would be rooted in backslashes, which Cygwin bash would evaluate as escape characters and thus remove. We would have to restore them, for example, as follows:

```
for i in $(ct lsvob -s); do if ct des lbtype:LBL@\\$i >/dev/null 2>&1;
then echo mkattr -rep ConfigRecord \'$DO\' lbtype:LBL@\\$i; fi; done | ct
```

This may seem as a tiny issue at first glance, but errors like this are enough to break a script or a build system, and chasing them soon takes enormous efforts in implementation, debugging, and testing.

As we already said, Perl allows us to bridge this gap, for example, with a cleartool wrapper as ***ClearCase::Wrapper::MGi***, itself building upon ***ClearCase::Wrapper*** and other modules. A wrapper installs itself as a Perl script—here `cleartool.plx` in `/usr/local/bin`, and may be aliased as `ct` instead of `cleartool`. It extends and corrects the cleartool behavior, falling back to the standard cleartool for the non-extended functionality.

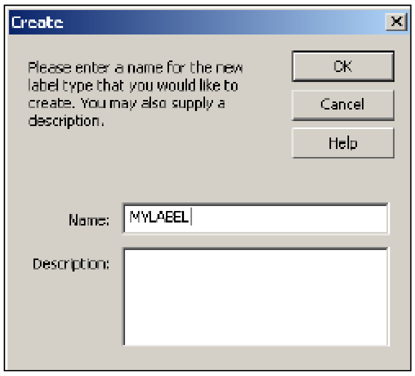
Now, with the help of ClearCase wrapper, one can run *exactly* the same original transcript on both platforms, UNIX and Cygwin Windows.

GUI versus text mode

Let's now have a small (biased) illustration on different modes of working—in ClearCase GUI and in ClearCase command line. Suppose, one would like to apply a label recursively to a directory hierarchy in ClearCase, containing a few subdirectories and hundreds of files.

First, a GUI scenario:

1. Open the Rational ClearCase Explorer (`C:\Program Files\Rational\ClearCase\bin\clearexplorer.exe`).
2. Select the view.
3. Mount the vob.
4. Click on the vob's root directory and select **Explore types**.
5. Double-click on **label type**.
6. Go to the **Type** menu and select **Create**.
7. Type label type name as **MYLABEL** in the **Name** field. Click **OK**.

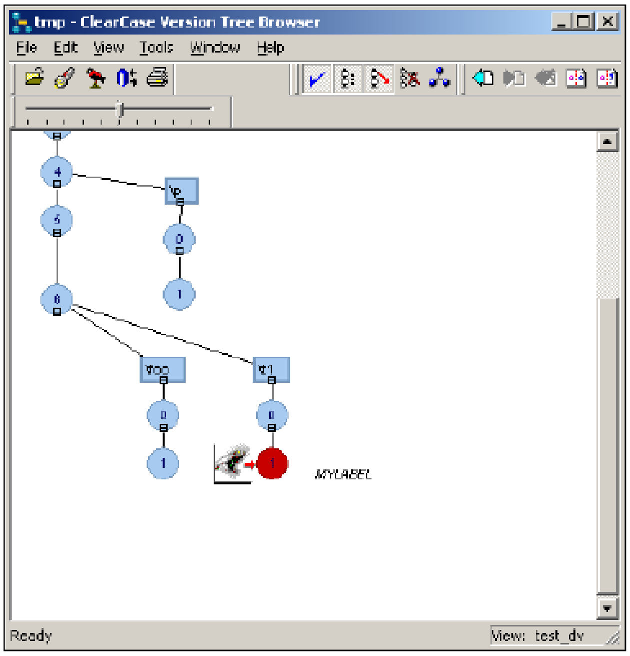


Please enter a name for the new label type that you would like to create. You may also supply a description.

Name: MYLABEL

Description:

8. Get back to the vob and locate the directory you want to label.
9. Right-click on the selected directory and select **Version tree**.
10. Right-click on the version selected by the view config spec and marked by the "eye" icon. Select **Apply label**.
11. Select **MYLABEL** from the list. Click **OK**.
12. Repeat the three last steps for each element under the selected directory (you might have to repeat the steps a hundred times…).


tmp - ClearCase Version Tree Browser
File Edit View Tools Window Help
4
5
6
\p
0
1
foo
r1
0
1
0
1
MYLABEL
Ready
View: test_dv

Now, let's look at the command-line scenario:

```
$ ct setview myview
$ ct mount /vob/myvob
$ cd /vob/myvob/mydir
$ ct mklbtype -nc MYLABEL
$ ct mklabel -rec MYLABEL
```

And we are done (feel the difference)!

One might object that it is possible to add a functionality such as `mklabel -recurse` to the suitable menus of the ClearCase Explorer. The tool to perform this is the ClearCase Context Menu Editor.

Of course, one might also object that our examples are not fair, and they are not. We stand for our point—it will always be easier to find yet a better hammer to hit any given nail using a command line than using a GUI (be it merging files or applying labels). Beyond the fact that finding a *wizard* might be non-obvious and disruptive of the user's flow of mind, the magic this one resorts upon will fall short eventually and require then reverse engineering knowledge. The user is, in fact, only buried one step deeper.

ClearCase documentation

ClearCase has a comprehensive documentation set in the form of so-called **ClearCase man pages**.

ClearCase shares, in a way, the same philosophy as Perl concerning its man pages. A ClearCase man page refers basically to a file coming along with the ClearCase installation, residing on a local machine and containing a relatively small piece of ClearCase documentation, dedicated to a single command or topic. Reading ClearCase man pages may be achieved in a standard UNIX way, using the `man` utility (or any other similar tools, for example, as part of GNU Emacs). It may also happen using the **man** function of the cleartool utility. On Windows platforms, this will result in requests to one's browser to display equivalent HTML pages.

Using UNIX man for accessing the ClearCase man pages requires little configuration. The man pages reside under `/opt/rational/clearcase/doc/man`; one needs thus to add this directory to the `MANPATH` environment variable. There is a special prefix to name the ClearCase man pages: `man ct_<ClearCase command name>` (sometimes with an abbreviation).

```
$ MANPATH=$MANPATH:/opt/rational/clearcase/doc/man
$ man ct_co
$ man ct_describe
```

These commands would display the man pages for the `checkout` (`co`) and `describe` commands respectively.

Alternatively, as mentioned earlier, and just like with the Perl `perldoc` command (which also performs as an alternative to `man`), one can use `cleartool man` (`MANPATH` setup not needed):

```
$ cleartool man co
$ cleartool man des
```

The distribution (both on UNIX and Windows) also offers a set of useful ClearCase books in PDF or HTML format (also available for reference purposes from the IBM website). They are located at `/opt/rational/clearcase/doc/books` in UNIX and at `<installation drive>:\Program Files\Rational\ClearCase\doc\books` in Windows.

There is finally a useful cleartool functionality (only on UNIX): **cleartool apropos**. It displays a brief summary of cleartool man pages matching a query to be specified as a regexp:

```
$ ct apropos 'label.*version'
mklabel Attaches version labels to versions of elements
rmlabel Removes a version label from a version
```

ClearCase does not support the users in extending this documentation (as Perl does with the *pod* markup language).

ClearCase online help is also available at the IBM Rational information center: `http://publib.boulder.ibm.com/infocenter/cchelp/v7r0m1/index.jsp`.

Note that there are separate resources there for each ClearCase version; this particular one is for the Rational ClearCase 7.0.1 family, although it is not as specific as one might think, and applies to other versions as well. We once submitted an RFE requesting to allow the user to assert changes between these docs, which was rejected.

Here are some ClearCase-related Internet resources and forums:

CM Crossroads Wiki
`http://www.cmwiki.com`

CM Crossroads ClearCase Forum
`http://www.cmcrossroads.com/forums?func=showcat&catid=31`

IBM Rational ClearCase forum
`http://www.ibm.com/developerworks/forums/forum.jspa?forumID=333`

Summary

We have reviewed some issues faced by the team player in trying to record her work and experience in a way that might be reproducible or even comparable by others. With all this set up, we are better equipped to dive into our actual topic: ClearCase.