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

10 Administrative Concerns

Administration is too important to be left to administrators.

Some insight is necessary for end users as well, even if only to investigate and narrow down problems, and then make useful requests to administrators.

One may consider two approaches to **administration**:

- **Top-down proactive**, concerned with resource planning, installation, and monitoring
- **Bottom-up** reactive, driven by investigation and problem solving, and virtually involving anybody

The latter is often more challenging, as under error conditions the system does not exhibit the same level of coherence as under normal ones. We'll mention a few such cases. What is important is that there is always a balance of the two.

We have actually already dealt with quite a lot of administrative concerns, especially in the previous chapter:

- Vob scrubbing (but so far we have left pools scrubbing)
- Structure of source containers for text files
- Construction of cleartext containers
- Tradeoffs in compression (existence of cleartext or not)
- Use of `ct dump`

We shall push to the next chapter issues relating to MultiSite.

Finally, we must note that administration is a well-documented part of ClearCase, partly because of the ***Administrator's manual***, but also because of the profusion of technotes available from the ***IBM website***. The reason for such a profusion of information is clear: administration confronts the multitude of the interfaces with a huge variety of systems, as well as a variety of the situations. It is not reasonable for us to attempt to compete against this; we'll as usual attempt to avoid repeating things available already, but try to focus on useful insights left from existing literature and documentation.

What does this suggest to us as a driver for this chapter? Maybe:

- First a review of a few top-down topics:
    - License and registry monitoring
    - Multiple regions and their synchronization
    - Client monitoring
    - The location broker
    - Adding logs and scheduler jobs
    - Storage and backup concerns
    - Vob and pools size
    - Authentication
    - Importing data to the vobs
    - Duplicating and registering vobs
    - Configuring Apache to access vobs
- To leave space for a couple of bottom-up case analyses, without any attempt at exhaustiveness:
    - Albd account and its password
    - Changing types
    - Dbids and the underlying database
    - Protecting vobs
    - Cleaning lost+found

Top-down

The goal is not to squeeze away any top-down concerns: some basic understanding is needed to set up the scene and as a context to filling in some holes as we promised.

License and registry

As discussed in our presentation (Chapter 2), ClearCase servers are split both into functions and hosts. The former may be mapped onto the latter, so that a standalone installation is possible, where all functions, including the role of client, are played by one single host; and on the contrary, an installation in which every function is held by a different host, and some functions sometimes by several, is possible. The functions have an order, which is reflected in the order in which one must upgrade the servers—with the exception of the license server, which is somehow nevertheless the top of the hierarchy.

With relatively recent versions of ClearCase, there have been two options for the license server. Or rather, an additional option of FLEXlm licenses has been added to the traditional ClearCase offering. The latter one being stable, and mostly satisfactory, has been kept. The main rationale for FLEXlm is that this is the implementation of choice for other IBM/Rational products, starting with ClearQuest. We won't go into ClearQuest (although we'll mention it again in *Chapter 13*, *The Recent Years' Development*), because its integration with ClearCase has not been very convincing. While both request tracking and change management are reasonable concerns, electing ticket life cycle as the main driver of development has not proven to be compelling. It had a large share of the dramatic development of UCM and led in our experience to counterproductive results. Another possible rationale for FLEXlm is that it is a widely used platform, relatively feature rich, and flexible. It does offer an answer to areas in which the standard ClearCase licensing required ad-hoc solutions: redundancy and high-availability in various scenarios (network or host failure) and catastrophe recovery, and monitoring tools. The advantages are however not compelling. The solutions to offer similar benefits with the old framework were neither complex nor expensive. They relied on the acquisition of a second set of license keys, a commercial/political negotiation, disconnected from our technical point of view. We cannot comment much on the FLEXlm offering, because it goes beyond our experience, limited to early evaluations that were not at the time fully convincing.

One piece of advice worth mentioning is to make large license pools (within world "regions", which is the only restriction traditionally set by ClearCase to limit the use of the same licenses around the clock, successively in different time zones), to benefit from a maximal flexibility. Licenses are granted for a period tunable, but typically of 30 minutes, and their acquisition is not significantly impacted by latency.

The license server is at best a stable and dedicated host, seldom to be powered down for maintenance. Its ClearCase version need not be upgraded often, not even for all major releases, as traditional licensing uses only the **albd** protocol. FLEXlm ClearCase licenses do not even require that.

License monitoring (non FLEXlm) relies upon parsing the output of the *clearlicense* tool. One limitation of this method is that it reaches to only the *local* license server (the one specified in the `config/license_host` file on the current host, lacking a `-host` option such as in the `ct getlog` or `hostinfo` commands). But this is easily worked around on a dedicated low-end PC (with no users), by changing the contents of the `config/license_host` file before running the command in order to collect the data from multiple license servers. Another strategy to monitor license usage could be to add the string `-audit`, to the `license.db` file (on the license server, in the `/var/adm/rational/clearcase` directory), and to use `getlog albd`. This is however not fully reliable.

MultiSite licenses are distinct (for historical shortsighted commercial reasons). A user needs one if she uses a replicated vob.

The **registry** collectively designates a few flat files in the `rgy` directory under `/var/adm/rational/clearcase` on the registry server (which may be backed up to a second host, allowing for a `rgy_switch` procedure for maintenance or failure recovery purposes). There are two main pairs of files: for vobs (`vob_object` and `vob_tag`) and views (`view_object` and `view_tag`); each pair consists of an *object* and a *tag* registry. They are meant to be updated with the `register/unregister` (for object registries), and `mktag/rmtag` (for tag ones) commands. The `rgy_check` command allows to detect inconsistencies (such as duplicate or missing tags or the presence of tags for missing objects, among other things).

One cannot in general recommend to edit these files, but it may be useful to read them, in order, for example, to compare entries produced by different commands. This is often faster than parsing the output of `ct lsvob` or `lsview`, for example, to find entries sharing a certain storage or erroneously set to use the file server (where ClearCase is not installed) as vob or view server. Note that it is not necessary to edit the registry files to access objects created with non-intended names such as a `-foo` tag for a view. Cleartool commands understand where necessary the "`--`" last flag convention:

```
$ ct lsview -foo 2>&1 | head -2
cleartool: Error: Unrecognized option "-foo"
Usage: lsview [-short | -long] [-host hostname [-quick]]
$ ct lsview -- -foo
* -foo                 /cc/views/-foo.vws
$ ct rmview -tag -foo
$
```

Of the other files in the registry, let's mention a few:

- `regions`: Accessible via the commands `ct mkregion`, `rmregion`, and `lsregion`. The region to which a ClearCase client belongs (it belongs to one, and only one at a time) is set in the `config/rgy_region.conf` file, in the ClearCase data area. Vobs and views are *registered* once per registry, but *tagged* per region. Regions are usually used for Windows/UNIX interoperability, so that two regions are defined, and vobs are tagged in both (with tags obeying the respective operating system constraints). More exotic setups can be met, with only part of the vobs tagged in certain regions. Views may also be tagged to multiple regions—a practice subject to controversies.
- `vob_tag.sec`: It is the encrypted registry password required to create public vobs. This is produced by the *rgy_passwd* utility, requiring *root* privileges, and vob owner accounts have no special access to it.
- `storage_path`: Accessible via `ct mkstgloc`, `rmstgloc`, and `lsstgloc`. Storage locations are useful and handy for creating vobs and views, by simplifying the options to the `ct mkvob` and `mkview` commands. This is especially true when using the `-stg -auto` combination. One way for the administrator to affect it in presence of several storage locations is for her to use the `-ngpath` flag (no global path) for `mkstgloc` command to disqualify the locations not intended for "automatic" usage.
- `site_config`: Accessible via `ct setsite` and `lssite`. This holds some default values concerning ClearCase behavior, as well as performance tuning (view cache size, accessible on a per view basis with `ct setcache` and `getcache`).

Synchronization between regions

Synchronizing views tags and vobs tags, and between the different ClearCase regions (for example, UNIX and Windows) is just a matter of:

- Having the view and vob storage locations available in both environments
- Having the view and vob servers accessible in both environments
- Creating the view and vob tags with `cleartool mktag` command in the target region

For example, consider the following two regions:

```
$ ct lsreg
unix_reg
win_reg
```

If we have a view created in the `unix_reg` region as:

```
$ ct lsview -l joe_view
Tag: joe_view
 Global path: /filer01/viewstg/joe_view.vws
 Server host: viewserver.domain.com
 Region: unix_reg
 Active: YES
View on host: viewserver.domain.com
View server access path: /filer01/viewstg/joe_view.vws
View owner: joe
```

We can synchronize it with the Windows region by using the following command:

```
W:\>cleartool mktag -view -tag joe_view -reg win_reg \
 -hos viewserver.domain.com \
 -gpa \\filer01\viewstg\joe_view.vws \\filer01\viewstg\joe_view.vws
```

And similarly for the vob:

```
$ ct lsvob -l /vob/foo
Tag: /vob/testi2
 Global path: /filer01/vobstg/foo.vbs
 Server host: vobserver.domain.com
 Access: public
 Mount options:
 Region: unix_reg
Vob on host: vobserver.domain.com
Vob server access path: /filer01/vobstg/foo.vbs

W:\>cleartool mktag -vob -tag \foo -reg win_reg \
 -hos vobserver.domain.com
 -gpa \\filer01\vobstg\foo.vbs \\filer01\vobstg\foo.vbs
```

One persistent problem in interoperable contexts concerns the creation, from Windows, of views with storage on a filer, using a UNIX view server. This would be required for views intended to be accessed from both UNIX and Windows. The error is as follows:

```
cleartool: Error: Failed to record hostname <UNIX view server>
```

This is because the client is responsible for setting the view storage directory permissions and the view ACL. Unfortunately, these are Windows-specific constructs. The views must be created on UNIX and tagged to Windows later.

Monitoring client activity

ClearCase keeps track of the clients of the registry, and of the license database, in `client_list.db`, in the data area (`/var/adm/rational/clearcase`). This file is accessible remotely with the `ct lsclients` utility.

Note that the license database is different from the registry, and here is how one can distinguish between the two (concerning the client access):

```
$ ct lsclients -host beyond.lookingglass.uk -l -type registry
Client: beside.wonderland.uk
 Product: ClearCase 7.0.1.3
 Operating system: Linux 2.6.9-78.0.22.EL #1 Fri Apr 24 12:35:12 EDT 2009
 Hardware type: i686
 Registry host: beyond.lookingglass.uk
 Registry region: alice
 License host: license.wonderland.uk
 Last registry access: 2010-09-05T20:01:23+03:00

$ ct lsclients -host beyond.lookingglass.uk -type license -l
Host "beyond.lookingglass.uk" has no license clients.
```

So, the `beyond.lookingglass.uk` acts as the registry server for the client host `beside.wonderland.uk`, but not as the license server.

The license server configured for this client is `license.wonderland.uk`:

```
$ ct lsclients -host license.wonderland.uk -type license -l
Client: beside.wonderland.uk
 Product: ClearCase 7.0.1.3
 Operating system: Linux 2.6.9-78.0.22.EL #1 Fri Apr 24 12:35:12 EDT 2009
 Hardware type: i686
 Registry host: beyond.lookingglass.uk
 Registry region: alice
 License host: license.wonderland.uk
 Last registry access: 2010-09-05T20:01:23+03:00
```

The file format is line oriented, but the output of the utility, with the `-long` option, is not, and therefore often requires in practice some (Perl) post-processing to be useful. One typical reason for such a need comes from the use of DHCP addresses: client hosts are identified in the client database with their IP addresses, and if these are granted by DHCP servers, they will (typically) be leased for a limited time. This suffices to make the record unreliable, the same hosts being possibly recorded multiple times. The database gets flushed only at restart time (we do not know of any other way).

In the example below, the uniquely sorted list of clients is about 2.5 times shorter than the original one:

```
$ ct lsclients -host licsrv -type license | wc -l
 813
$ ct lsclients -host licsrv -type license | cut -d: -f1 | sort -u | wc -l
 331
```

Using the long output, we may build a hash, and store the recorded access time. When meeting the same name again, we update the hash record only if the new time is more recent than the one stored.

```
$ ct lsclients -l -host licsrv -type license | perl -n00e \
 'chomp; ($n,$a) = (/Client: ([^\n]+).*?(?:access: (.*)|$)/s);
 if (!$h{$n}{a} or $h{$n}{a} lt $a) { $h{$n}{a} = $a?$a:q() }
 END{ for(sort {$h{$a}{a} cmp $h{$b}{a}} keys %h)
  { print "$_: $h{$_}{a}\n" } }' | tail -3
mars: 2010-09-25T19:15:00+01
venus: 2010-09-25T19:15:02+01
jupiter: 2010-09-25T19:15:16+01
```

We are again (refer to *Chapter 9*, *Secondary Metadata*) using the paragraph-oriented input mode, stripping the newline, and matching the items with the `s` modifier (so that the "`.`" wildcard may match newlines). We take advantage of the fact that the standard format of time stamps makes them string-wise comparable.

Note that some records do not have a last access time (for reasons we can only try to guess). In order for them to nevertheless match the pattern with which we extract the fields we are interested in, we enclose the second part in a "`(?: ... )`" bracket pair (group, but do not record), as an alternative to an end of line, and to make the previous indiscriminate pattern *non-greedy*: "`.*?`"; We then store an empty value `q()` instead of the undefined one; we use an empty value so that it compares "older" than any recorded access time. Our output format is line oriented, so as to be greppable (or "tailable" as in the example above). Producing it in an `END` block allows us to sort it, here by the last access time.

Other fields may of course be recorded and printed (see the earlier example) such as `Product`, `Operating system`, `Hardware type`, and so on.

Location broker

Clients can be accessed remotely via the **albd service**, listening to port 371, for example, with the following utilities: `/opt/rational/clearcase/etc/utils/albd_list` (`c:\Program Files\Rational\ClearCase\etc\utils\albd_list.exe` on Windows), `ct getlog`, and so on.

```
$ albd_list beside.wonderland.uk
albd_server addr = 100.175.111.249, port= 371
PID 16530:
 syncmgr_server, tcp socket 33101: version 1; BUSY
  Storage path syncmgr_server
Albd_list complete
```

This "pings" the ClearCase client and lists the currently running albd processes (if any). On a ClearCase server, the output may be quite long.

```
$ ct getlog -host Win12345 albd
==============================================================
Log Name: albd Hostname: Win12345 Date: 2010-09-24T12:36:32+03:00
Selection: Last 10 lines of log displayed
--------------------------------------------------------------
2010-09-24T04:43:51+03:00 albd_server(24860): Job 6 #####################
                                "Daily VOB Space" (166208) Completed: OK.
2010-09-24T04:43:50+03:00 albd_server(24860): Job 6 #####################
                                     "Daily VOB Space"  (166208) Started.
2010-09-24T04:43:49+03:00 albd_server(24860): Job 5 #####################
                               "Daily View Space" (169308) Completed: OK.
2010-09-24T04:30:12+03:00 albd_server(24860): Job 5 #####################
                                     "Daily View Space" (169308) Started.
2010-09-24T04:30:12+03:00 albd_server(24860): Job 3 #####################
                          "Daily Registry Backup" (169276) Completed: OK.
2010-09-24T04:30:12+03:00 albd_server(24860): Job 3 #####################
                                "Daily Registry Backup" (169276) Started.
2010-09-24T04:30:12+03:00 albd_server(24860): Job 2 #####################
                            "Daily VOB Snapshots" (167520) Completed: OK.
2010-09-24T04:30:07+03:00 albd_server(24860): Job 2 #####################
                                  "Daily VOB Snapshots" (167520) Started.
2010-09-24T04:30:07+03:00 albd_server(24860): Job 1 #####################
                       "Daily VOB Pool Scrubbing" (166296) Completed: OK.
2010-09-24T04:30:04+03:00 albd_server(24860): Job 1 #####################
                             "Daily VOB Pool Scrubbing" (166296) Started.
==============================================================
```

We'll look a bit deeper into logs and the scheduler in the further sections.
The **albd port** (actually, both tcp and upd ports) is the only well-known port ClearCase uses, and the only one on which a failure to connect will actually tell of a problem. One can also test it, for example, with `telnet client_host 371`, even if this admin idiom offers no advantage over using `albd_list`: as the albd protocol is not text based, the telnet session will display nothing and have to be interrupted. Sessionless communication resulting in small amounts of data may be carried on this port. For anything beyond that, the albd server will negotiate a connection backwards to the requester, using a high port, and continue on this port. Only for shipping purposes is there a way to restrict this high port within a range (more on this in next chapter). Understanding this is essential for whoever considers configuring a **firewall**: opening port(s) 371 is not enough; all the high ports must be opened, and the negotiation is initiated by the remote end. Let's make clear that using firewalls between ClearCase hosts can thus only really be considered in a few cases:

- For shipping
- For license acquisition, and even then, it will impact the ability to run *clearlicense*
- For web access, including via a remote client (and then the communication is limited and channeled via the http, https, ssl, ...ports)

Surprisingly to old ClearCase users, the presence of a firewall between a client and its registry will not stop ClearCase from working (as it used to do, on timeouts, in previous releases of ClearCase). It will however affect it, and impact even *other* users from other clients: delays between database lock acquisition and release will significantly increase, due to the latency resulting from firewall processing, adding up to the round-trip times.

Firewalls may add specific error cases such as by dynamically blocking ports as a measure against *Denial of Service* attacks, if a single command results in *excessive* output. This example is a typical default firewall configuration problem:

```
$ cleartool lsview
albd_rgy_get_entry call failed: RPC: Timed out
cleartool: Error: Trouble contacting registry on host "regsrv": #########
             timed out trying to communicate with ClearCase remote server
```

Remote monitoring infrastructure

We saw that the ClearCase infrastructure is typically distributed among multiple hosts. Problem investigation thus requires access to distributed information, first and foremost logs of various kinds, but also the state of the background tasks.

This is well supported by the ClearCase architecture, through the `ct getlog` and `schedule` commands. This architecture is well documented, and actually open to extension, and administrators would be well inspired in using this option, instead of resorting to the UNIX tools *crontabs* and *syslog*.

There is an exception to the consistency with which ClearCase uses its own architecture, and it is with shipping server installation, but we'll treat this in the next chapter.

We have already treated the output of both functions. There could be one deficiency to `getlog`, which is worth noting: it is not always obvious which log to inspect (between `error`, `albd`, `vobrpc`, `view`, and so on), and it is not trivial to make up one's mind. The best is often to log in to the remote host, and to tail a list of the logs sorted by their timestamps. Of course, this defeats the advantage we just praised.

An alternative is to ask for the last line of every log, and to sort them by their timestamps:

```
$ ct getlog -last 1 -all -host beyond 2>/dev/null | \
 perl -nle 'if(/^Log Name: (\w+)/) { $n=$1; next }
 $h{$n}=$_ if /^\d+/;
 END {print "$_: $h{$_}" for sort {$h{$a} gt $h{$b}} keys %h}'
view: 2010-09-24T11:01:26+01 view_server(12404): Db closed
mvfs: 2010-09-24T19:34:47+01 global(0): fs: Error: view=joe #############
         vob=/vob/test - Lock on VOB database prevents write transactions
albd: 2010-09-26T04:33:00+01 albd_server(744): ##########################
                            Job 6 "Daily VOB Space" (26669) Completed: OK.
```

This gives a clue of what may be worth reading next: we retained the log name, and the last line only if there was one, in a hash indexed by the name.

In an end block, we print both together on one line, sorting the logs by the contents of the records, taking advantage from the fact that these start with the timestamps, and as we noted earlier already, the timestamps are string-wise comparable. Note however that this only works for the line formatted class of logs, as it assumes a timestamp. This unfortunately excludes the `error` log, for instance.

Scheduler

We'll turn now to the question of adding one's own custom tasks and logs.

Every *job* in the scheduler actually runs a *task*. All the tasks are numbered and listed in a flat file: `/var/adm/rational/clearcase/scheduler/tasks/task_registry`. The tasks themselves should also be located either in the same directory `/var/adm/rational/clearcase/scheduler/tasks` (usually the customized ones) or in `/opt/rational/clearcase/config/scheduler/tasks` (usually the standard ones). Our first move (after writing the script we intend to use as a scheduler job) is thus to add one task description to the end of file.

The example we take here is one on which we'll come back in the next chapter, for monitoring the status of the MultiSite replication:

```
Task.Begin
Task.Id: 103
Task.Name: "Moving received repmon logs"
Task.Pathname: "repmon_mv.sh"
Task.End
```

Here `repmon_mv.sh` is the name of our script. It moves replica monitoring logs received from other hosts to a certain location: `/tmp` dir in our example. As we placed it into the `/var/adm/rational/clearcase/scheduler/tasks` directory, we do not need to specify its full path.

We need next to create a new job in the scheduler, using this task. We can use the interactive command (which will start the editor specified in the environment, by default `vi`):

```
$ cleartool sched -edit
```

And with it, we add something like the following:

```
Job.Begin
Job.Name: "Daily Repmon logs moving"
Job.Description.Begin:
Move all received replica monitoring packets to /tmp
Job.Description.End:
Job.Schedule.Daily.Frequency: 1
Job.Schedule.FirstStartTime: 00:00:00
Job.Schedule.StartTimeRestartFrequency: 04:00:00
Job.DeleteWhenCompleted: FALSE
Job.Task: 103
Job.Args:
Job.NotifyInfo.OnEvents: JobEndFail
Job.NotifyInfo.Using: email
Job.NotifyInfo.Recipients: root
Job.End
```

Administrators may want to keep a backup of the schedule and possibly to version their changes.

This is not conveniently supported by `ct schedule` because the output combines commands and outputs.

Let's focus on one single job (the one we just added), instead of on the whole schedule (`ct sched -get -sched`):

```
$ ct sched -get -job "Daily Repmon logs moving"
Job.Begin
 Job.Id: 20
 Job.Name: "Daily Repmon logs moving"
 Job.Description.Begin:
Move all received replica monitoring packets to /tmp
 Job.Description.End:
 Job.Schedule.Daily.Frequency: 1
 Job.Schedule.StartDate: 2010-09-26
 Job.Schedule.FirstStartTime: 00:00:00
 Job.Schedule.StartTimeRestartFrequency: 04:00:00
 Job.DeleteWhenCompleted: FALSE
 Job.Task: 103
 # Job.Task: "Moving received repmon logs"
 Job.Args:
 Job.NotifyInfo.OnEvents: JobEndFail
 Job.NotifyInfo.Using: email
 Job.NotifyInfo.Recipients: root
 Job.Created: 2010-09-26T14:07:35+01 by anis/root@beyond
 Job.LastModified: 2010-09-26T14:07:35+01 by anis/root@beyond
 Job.NextRunTime: 2010-09-26T20:00:00+01
 Job.LastCompletionInfo.ProcessId: 2894
 Job.LastCompletionInfo.Started: 2010-09-26T16:00:00+01
 Job.LastCompletionInfo.Ended: 2010-09-26T16:00:02+01
 Job.LastCompletionInfo.ExitStatus: 0x0
Job.End
```

Actually, even commands get reinterpreted: job ids are produced automatically and task names are kept only as comments.

But the most obvious is that timing information is added and updated, as well as completion info, possibly including multiline excerpts of the standard error:

```
Job.LastCompletionInfo.Messages.Begin:
mv: cannot stat `/usr/atria/shipping/ms_ship/incoming/*.repmon*': #######
                                                No such file or directory
Job.LastCompletionInfo.Messages.End:
```

This results in the fact that some processing of this output is needed if one intends to use it as input, for example, to restore a previous state. Some more work for Perl.

Now, we want to access the result of our customized task (the replica monitoring logs in the `/tmp` directory) via the `cleartool getlog` command, so we add a new log.

The ClearCase logs are also defined in a flat file, in the *read-only* hierarchy this time: `/opt/rational/clearcase/config/services/log_classes.db`.

This is where we have to add one line to gain access to the received and moved replica monitoring logs (still our same example) via *cleartool getlog*.

This is not documented in ClearCase:

```
   -class=repmon;-form=imsg_file;-sources=/tmp/*.repmon;- ##############
                     description=ClearCase VOB replica monitoring log;
```

The modified `log_classes.db` file does need to be maintained manually through the ClearCase upgrades installations (which may overwrite it).

One may replace it with a symlink to under the `/var/adm` hierarchy (which will not get overwritten by upgrades), and keep the backup there.

There are two main log formats: *ascii* and *imsg_file* (plus *mvfs_file*, where the timestamps are encoded, requiring the `time2date` utility in the `etc/utils` directory, to decode them).

Logs in the `log` directory get rotated, so that the *sources* typically mention two files: the current and the old one.

We chose not to use this feature for our log.

We may now check that our log is known to the `ct getlog -inquire` command, and fetch it as any other:

```
$ ct getlog repmon
```

All the `-host`, `-last`, `-since`, and `-around getlog` options are available.

```
$ ct getlog -host beyond -last 100 repmon
$ ct getlog -host beyond -since today repmon
```

Storage and backup

Storage administration was greatly simplified with the certification of **NetApp filers**, followed by some other vendors.

Filers may be configured for use as **Network Attached Storage** (**NAS**), that is, file servers, serving through NFS and/or CIFS, or **Storage Area Network** (**SAN**). The latter potentially offers greater performance, at a lower level and with less flexibility. We admit having seen them used effectively with ClearCase. Nevertheless, it is the NAS configuration (the filer proper), which seems most attractive to us, and with which we have the best experience. We'll restrict our comments to this case.

Filers were first used to store the pools (which were then known as *remote*) via symbolic links. This was temporary until they could be certified, so that whole vobs, including the database, were stored on the filer. Let us drop this outdated setup as well.

The filer relieves the vob server from some noticeable load, part of which is that of running the NFS and CIFS (*samba*) servers.

It also makes it trivial to switch a vob from one server to another (of the same architecture); the data may stay untouched. This is especially valuable in upgrade scenarios.

The most important advantage of filer storage, from an end user perspective, is the **snapshot** functionality and their use for backup.

Filers do not overwrite the data they store; new versions of files are stored in newly allocated disk clusters. This makes it possible to freeze the state of the storage at any given time by storing an image of the directory objects (and keeping the clusters referenced from there from being reverted to the free pool).

This function is termed taking a *snapshot*. From the perspective of a database, the only concern is to ensure that the database caches are flushed before this happens, so that the disk image is consistent.

This function is performed by a side-effect of *locking* the vobs: once the vob is locked, it may be snapped, which takes a few seconds, and then it may be unlocked. The backup (for example, to tape or to any suitable archive) may thus take place on the snapshot (and not on live data).

Restoring from there is easier and incomparably faster than from tape. We saw in the last chapter how it may offer opportunities to restore individual versions. This implies that the snapshots may be mounted to user accessible mount points.

In recovery scenarios, one typically needs to synchronize the views with the recovered vob.

We'll deal later in this chapter with issues of vob protection, including the storage area, pools, and containers.

Vob size

The `ct space -vob` command allows to understand what composes the size of a given vob:

```
$ ct space -vob /vob/foo
Use(Mb) %Use Directory
 13.7 0% VOB database /vobstg/foo.vbs/db
  0.1 0% administration data /vobstg/foo.vbs/admin
  0.3 0% cleartext pool /vobstg/foo.vbs/c/cdft
  0.0 0% derived object pool /vobstg/foo.vbs/d/ddft
146.9 0% source pool /vobstg/foo.vbs/s/sdft
---------------------------------------------------------
161.1 0% Subtotal
1078199.0 92% Filesystem fstore:/vol/vol01/vobstg (capacity 1177804.8 Mb)

Total usage 2010-09-26T06:47:19+01 for vob "/vob/foo" is 161.1 Mb
```

Note that without the `-update` option, the `ct space -vob` command works on the last results obtained from the *Daily VOB Space* built-in job, and will thus not be affected by any changes on the spot.

One may of course force a run of the job (as with any job):

```
$ ct sched -run "Daily VOB Space"
```

But this runs the job for all local vobs, and may thus take a long time.

The cleartext and the DO pools contain only transient data, which may be reproduced. Cleartext containers only cache versions of elements accessed recently. We saw in the previous chapter in the case of text files how the file manager constructed them. Another typical case is that of expanding versions stored in compressed format. The size of these two pools is affected by the scrubber, which is typically run by the predefined scheduler job: *Daily VOB Pool Scrubbing*.

```
$ ct sched -get -job "Daily VOB Pool Scrubbing" | \
 egrep 'Schedule.*Freq|#.*Task'
    Job.Schedule.Daily.Frequency: 1
    # Job.Task: "VOB Pool Scrubber"
$ ct sched -get -tasks | perl -n00e 'print if /: 3$/ms'
Task.Begin
    Task.Id: 3
    Task.Name: "VOB Pool Scrubber"
    Task.Pathname: "scrubber.sh"
Task.End
```

This task uses a script (by default: `/opt/rational/clearcase/config/scheduler/tasks/scrubber.sh`) to drive the `/opt/rational/clearcase/etc/scrubber` tool. The default options examine all the pools in all the vobs. The result is logged in a way accessible to `getlog`, in unformatted mode:

```
$ ct getlog -host alice -full scrubber | \
 perl -ne 'print if m%^Started.*/foo.*Feb 12%...m%^Fin.*/foo.*Feb 12%'
Started VOB alice:/vobstg/foo.vbs at Sat Feb 12 12:26:42 2011
Start scrub VOB alice:/vobstg/foo.vbs Pool ddft
Stats for VOB alice:/vobstg/foo.vbs Pool ddft:

Get cntr tm   0.060150
Setup tm      0.015319
Scrub tm      0.000035
Total tm      0.075504
Start size 1  Deleted   0  Limit size        0
Start files       2  Deleted          0  Subdir dels       0
Statistics for scrub of DO Pool ddft:
DO's              0  Scrubs            0  Strands          0
Lost refs         0  No DO's           2
Start scrub VOB alice:/vobstg/foo.vbs Pool cdft
Stats for VOB alice:/vobstg/foo.vbs Pool cdft:

Get cntr tm   2.527032
Setup tm      0.000019
Scrub tm      0.000417
Total tm      2.527468
Start size 1514  Deleted   1  Limit size        0
Start files     182  Deleted          1  Subdir dels       8
Finished VOB alice:/vobstg/atcctest.vbs at Sat Feb 12 12:26:45 2011
```

This time we used a range line-oriented matching as we had a clear way to specify the start and end of the area of interest. The output shows various timings and deletion statistics for both pools, DO and cleartext. In this run, only one cleartext container was deleted.

Let's note that the scrubber may be run manually, with parameters involving size, age, and heuristics to preserve useful data, along with options allowing us to select the vobs and pools.

If the source pool grows abnormally, one must suspect the process of importing or creating new versions and look for redundant data: either evil twins or identical versions.

The cure is then to remove the versions, branches, or elements created by mistake (respectively with `rmver`, `rmbranch`, and `rmelem`).

However, this will not affect (at least not decrease) the size of the database. The only thing which may affect it is `vob_scrubber`.
In particular, scrubbing the oplogs may be the solution. We touched the topic in the last chapter.

The tool to give an account of the space consumption within the database is *countdb*. Details on the size of the various objects is found in a technote: *About the ClearCase database utility countdb* (***#1126456***).

Authentication

ClearCase has an identity management mechanism, which is called *credmap*. It is switched off by default and is rarely used in practice, as it introduces one more level of authentication, most often unnecessary in the case of a UNIX vob server with NetApp storage for vobs and views (and the appropriate NetApp access rights) and CIFS for Windows users mapping. This is why IBM mostly ***recommends*** it in case one uses a Windows vob server and needs to introduce a verification mechanism for its UNIX users.

In some rare cases, though, one might want to enable credmap authentication on the UNIX vob server as well, for the purpose of enforcing that Windows users belong to a certain Windows network domain.

This can be done by creating a `/var/adm/rational/clearcase/config/credmap.conf` file, putting the desired Windows domain name there:

```
$ cat /var/adm/rational/clearcase/config/credmap.conf
AllowedDomain=RATIONAL
```

Or in ClearCase 7.0 and up, there is an option to allow Windows users to belong to any network domain (but no local users would be allowed then):

```
$ cat /var/adm/rational/clearcase/config/credmap.conf
AllowedDomain=[-]
```

Importing files to ClearCase

Now we will take a look at a few issues related to importing data to ClearCase or relocating the data between vobs using both standard tools (*clearfsimport*, *clearimport*/*clearexport*, *ct relocate*) and *synctree*.

A priori, these tools are very different:

- ct relocate is targeted at *moving* elements, with all their history, from one vob to another
- clearimport is targeted at *copying* elements with their history into a vob
- clearfsimport and synctree are useful for *copying* only versions, from a source to a destination that may be within the same vob

All these tools are applicable only for base ClearCase vobs. First, we will also take a brief look at UCM to see how data is supposed to be imported and moved there.

Even UCM has to use Perl

One interesting observation concerning UCM (more of it later in Chapter 13) is that it is rather difficult to get there (for example, import the initial set of elements), and rather difficult to change it (relocate something from one vob to another). Like doing arithmetic using Latin numbers: it was just not designed for that! And the general recommendation from IBM is ...not to do it. It is also not possible to convert a UCM component vob to a base ClearCase vob.

The already mentioned *synctree* tool (there is more on it a bit later in this chapter) can also be used to fetch data from a UCM vob via a UCM view and import it to a base ClearCase vob.

The rescue scenario would be the following: choose a UCM baseline (for example, `BL1`), create a development stream based on it along with a view (for example, `BL1_view`), set a base ClearCase view (`view1` in the example that will follow), and then perform the data import via the view-extended path of the UCM view from the UCM vob (`/vob/ucmvob`) into the base ClearCase vob (`/vob/tools`):

```
$ ct setview view1
$ synctree -sb /view/BL1_view/vob/ucmvob/foo -db /vob/tools/foo -ci -yes
```

One can, of course, fetch several baselines in the same way from the UCM vob, by creating a new development stream along with a view for each baseline. There, the `-reuse` and `-vreuse` options (as well as `-rm`) become also useful.

A similar need (the ability to move the data around a bit even within a UCM environment) was also recognized by IBM, and in order to be able to relocate some data from a UCM vob to another something that originally was completely out of the question), one can (without any guarantee of course) find some aid from a custom Perl script `mkelem_cpver.pl`, provided in the ClearCase installation.

Relocating

When it comes to relocating (part of) the vob to a different vob, it is recommendable to always use the `cleartool relocate` command instead of `clearexport`/`clearimport` pair. The latter has a hardcoded behavior, which is worth knowing in advance; it is able to handle the whole version trees of the *file* elements (including all its branches) but has a restriction concerning these of the *directory* elements:

```
$ ct setview myview
$ cd /vob/foo
$ clearexport_ccase -r -o /tmp/foo_export .
$ clearimport -d /vob/newfoo /tmp/foo_export
...
$ ct lsvtree /vob/newfoo
/vob/newfoo@@/main
/vob/newfoo@@/main/0
/vob/newfoo@@/main/1 (FOO, FOO_3.00, RREL_0.80, ZOO, ...)

$ ct ls /vob/newfoo
/vob/newfoo/mdir@@/main/1 Rule: /main/LATEST [-mkbranch br1]
/vob/newfoo/bdir@@/main/1 Rule: /main/LATEST [-mkbranch br1]
/vob/newfoo/zdir@@/main/1 Rule: /main/LATEST [-mkbranch br1]

$ ct ls /vob/foo
/vob/newfoo/mdir@@/main/45 Rule: /main/LATEST [-mkbranch br1]
/vob/newfoo/bdir@@/main/4  Rule: /main/LATEST [-mkbranch br1]
/vob/newfoo/zdir@@/main/2  Rule: /main/LATEST [-mkbranch br1]
```

As can be seen on this example, `clearexport` exports only the current version of the directory element selected by the view, and `clearimport` always imports it as the `/main/1` version in the new vob. Tuning the views config specs does not help to fetch the rest of the directory elements version trees.

However, the file elements' version trees are fully imported:

```
$ ct lsvtree /vob/newfoo/java/1/makefile
/vob/newfoo/java/1/makefile@@/main
/vob/newfoo/java/1/makefile@@/main/8 (LBL_1.03, LBL_1.02, LBL_1.00)
/vob/newfoo/java/1/makefile@@/main/jb
/vob/newfoo/java/1/makefile@@/main/jb/2

$ ct lsvtree /vob/foo/java/1/makefile
/vob/foo/java/1/makefile@@/main
/vob/foo/java/1/makefile@@/main/8 (LBL_1.03, LBL_1.02, LBL_1.00)
/vob/foo/java/1/makefile@@/main/jb
/vob/foo/java/1/makefile@@/main/jb/2
```

`cleartool relocate` lifts these restrictions and is able to relocate both files and directories elements along with all their branches and versions; however, it is not designed to relocate a whole vob but only a part of it (which has to be specified as a pathname in the vob):

```
$ cleartool relocate -f -update /vob/foo/makedir /vob/newfoo
```

The `makedir` directory element and all its entries will be relocated correctly to the `/vob/newfoo` vob.

The closest to relocating the whole vob is to copy/move it under another one as a subdirectory:

```
$ ct relocate -update /vob/foo /vob/newfoo
$ ll /vob/newfoo
total 4
drwxrwxr-x 6 vobadm cc 294 May 6 13:05 foo
drwxr-xr-x 2 joe jgroup 0 May 6 13:03 lost+found
```

So, to relocate the whole vob, one would need to use other tools (see *Copying a vob* section below).

Note that here we used relocate in an *update* mode, which amounts to a *copy*, even if it was originally intended as a transient phase in a move (`ct relocate` without `-update` option). In the update mode, the original data is kept untouched.

Note that the oid of the elements is preserved:

```
foo> ct des -fmt "%On\n" /vob/foo/makedir@@ /vob/newfoo/makedir@@ | \
 sort -u
8a3fd1d6.14a511df.8283.00:01:37:45:23:13
```

Relocate still depends on the current config spec for creating the version of the root directory of the import. In its default mode (move), it is destructive for the information at the original site. Even if it builds symbolic links (and one hyperlink) to trace the relocation, it does break the config records of any derived objects depending on any relocated version.
Note that it may create in the target vob events older than the vob itself.

It looks as if a tool such as *synctree* (see next section) for importing files to ClearCase would desperately be needed for vob relocation purposes.

Importing with synctree

We already mentioned synctree at large in *Chapter 8*, *Tools Maintenance*.

Importing large amounts of files from external directories, for example, third-party tools, may lead to problems mentioned earlier such as importing the same files multiple times.

This may happen when importing several releases out of order in succession, using the same branch.

Let's consider one file, `foo`, which changes between release 1 and release 2; let's suppose we import in succession releases 1, then 2, and then 1.1.
This will yield three versions of `foo`, 1 and 3 being identical.

A worst case yet is when `foo` is deleted in release 2 (with the `-rmname` option either in *clearfsimport* or *synctree*): in that case, importing 1.1 over 2 with `clearfsimport` will yield an evil twin, whereas the `-reuse` option of the *synctree* helps to avoid that. In fact, the `-rm` option is only safe once one knows one can rely on a `-reuse` one to avoid creating evil twins later.

Another scenario is this of releases for different platforms, imported to platform-specific branches: platform independent files (for example, header files) will be needlessly imported into every branch.

These scenarios are taken into consideration by special options of synctree:

```
$ synctree -sb source -db destination -reuse -vreuse -label FOO -/ipc=1
[...]
```

- `-reuse` will avoid the evil twin creation, by resurrecting a previously deleted element from the version tree (note that it requires `-label` or `-lb_mods` to resurrect existing versions without creating duplicates).
- `-vreuse`, in conjunction with `-label`, will compare a file not matching the selected version, to versions of the same size in the version tree. Finding a suitable version, it will skip importing it anew and will label it instead.

- `-/ipc=1` will use *the ipc mode* of the underlying ***ClearCase::Argv*** module, thereby invoking one single instance of cleartool, and ensuring that the extra calls needed to support the above functionality do not incur a prohibitive cost. It also makes the `-cr` option reasonable. This one aims at preserving the config records of derived objects, but as it uses the `-from` option of `checkin`, it requires a distinct invocation for every element, which makes it prohibitive without the ipc mode. This functionality is unique to synctree, with no equivalent in clearfsimport.

Synctree also fixes some clearfsimport hiccups, such as infinite cycles in case the data you want to import is already in place as view-private:

```
$ ct ls -d /vob/test/dir1
/vob/test/dir1
$ clearfsimport -r /vob/test/dir1 /vob/test/dir1
Creating directory "/vob/test/dir1".
Private version of "/vob/test/dir1" saved in "/vob/test/dir1.keep".
Creating directory "/vob/test/dir1".
Created branch "br1" from "/vob/test/dir1" version "/main/0".
Validating directory "/vob/test/dir1".
clearfsimport: Warning: Using existing checkout of directory ############
                                                        "/vob/test/dir1".
Creating directory "/vob/test/dir1/dir1".
Created branch "br1" from "/vob/test/dir1/dir1" version "/main/0".
Creating directory "/vob/test/dir1/dir1/dir1".
Created branch "br1" from "/vob/test/dir1/dir1/dir1" version "/main/0".
```

In synctree, the same is worked around:

```
$ synctree -sb /vob/test/dir1 -db /vob/test/dir1
/usr/bin/synctree: Error: 6 view-private files exist under ##############
                                                   /view/v1/vob/test/dir1:
```

Synctree also provides a number of useful features such as:

- Branch from root (non-cascading) branch support
- Support for regular expressions (`-Narrow [!]<re>` option) to limit files to be imported to those matching `/re/`
- Hash mapping of source set of files to the destination via `-map` option
- Possibility (`-rellinks` option) for turning absolute symlinks within source base into relative ones

ClearCase::Wrapper

For certain fine-tunings related to importing files to ClearCase, one would find a great help in ***ClearCase::Wrapper***, which provides such useful functionality as, for example, recursive `mkelem` and `checkin`.

Copying a vob

We'll consider here moving a whole vob, for instance migrating it to a new server, and why it is not suitable for duplication. Vob duplication is requested in case the company's development environment has been split or in similar situations.

The sad news is that there is no shortcut from exporting and re-importing the relevant data.

Moving vob storage

One can re-register the vob by moving its storage to a new directory and making a new vob tag:

```
$ ct lock vob:/vob/foo
$ ct umount /vob/foo
$ ct unreg -vob /vobstg/foo.vbs
$ ct rmtag -vob /vob/foo
$ pkill -f foo.vbs

$ cd /vobstg/foo.vbs
$ find . -depth | cpio -pdmu /vobstg/bar.vbs

$ ct reg -vob /vobstg/bar.vbs
$ ct mktag -vob -tag /vob/bar -public -host vobserver \
 -gpath /vobstg/bar.vbs /vobstg/bar.vbs
$ ct mount /vob/bar
$ ct unlock vob:/vob/bar
```

Note however, that this method is not suitable to duplicate the vob, in other words, to keep both `foo` and `bar` vobs independently. The problem is that the vob oids of the two copies are identical, which constitutes a mine field: this consanguinity of the two copies will survive their divergence and lead to corruption even years later at yet unknown sites, if a packet originating from the first hierarchy reaches a replica of the second. Let's note that using MultiSite to produce a new replica does nothing to solve the problem. Forcibly modifying the oid in one copy hits "internal error" problems when creating hyperlinks.

Copying vob by replication

The easiest way to copy a vob to a different host is often to replicate it there.

```
[host1]$ multitool mkreplica -exp -work /tmp/foo -nc -fship \
 host2:rep2@/vob/foo
```

If the actual replication is not desired for some reason, one can consider reciprocal replica removal at both sites:

```
[host2]$ ct lsrep -fmt "%n %[replica_host]p\n" -invob /vob/foo
rep1 host1
rep2 host2
[host2]$ mt chmaster -all -obsolete_replica rep1@/vob/foo rep2@/vob/foo
Chmaster -obsolete_replica will transfer mastership of all objects
 currently mastered by rep1@/vob/foo to rep2@/vob/foo. Do not proceed
with this
operation unless the VOB for rep1@/vob/foo has been deleted. Do you want
to proceed? [no] yes
Changed mastership of all objects
You must now complete the removal of rep1@/vob/foo by entering
the following command at the master site for rep1@/vob/foo:
multitool rmreplica rep1@/vob/foo
[host2]$ multitool rmreplica rep1@/vob/foo
The last remaining replica has been deleted; ###########################
                                            disabling replication in VOB.
Deleted replica "rep1".
[host2]$ ct lsrep -fmt "%n %[replica_host]p\n" -invob /vob/foo
rep2 host2


[host1]$ ct lsrep -fmt "%n %[replica_host]p\n" -invob /vob/foo
rep1 host1
rep2 host2
[host1]$ mt chmaster -all -obsolete_replica rep2@/vob/foo rep1@/vob/foo
Chmaster -obsolete_replica will transfer mastership of all objects
currently mastered by rep2@/vob/foo to rep1@/vob/foo. ###################
                                                 Do not proceed with this
operation unless the VOB for rep2@/vob/foo has been deleted. ############
                                         Do you want to proceed? [no] yes
Changed mastership of all objects
```

```
You must now complete the removal of rep2@/vob/foo by entering
the following command at the master site for rep2@/vob/foo:
multitool rmreplica rep2@/vob/foo
[host1]$ multitool rmreplica rep2@/vob/foo
The last remaining replica has been deleted; ############################
                                            disabling replication in VOB.
Deleted replica "rep2".
[host1]$ ct lsrep -fmt "%n %[replica_host]p\n" -invob /vob/foo
rep1 host1
```

Re-registering a replica

This happens when a vob has to be re-registered with a new vob server host name (for example, in case the vob server domain name has changed), or when the vob has been migrated to a new vob server:

```
$ ct reg -vob -host newvobsrv.domain.com -rep \
 -hpath /vobstg/foo.vbs /vobstg/foo.vbs
```

The hostname must also be updated explicitly for the vob's replica (even if the vob is not actually replicated!):

```
$ ct lsvob -l /vob/foo | grep host
 Server host: newvobsrv
Vob on host: newvobsrv

$ ct lsreplica -fmt "%n %[replica_host]p\n" -invob /vob/foo
rep1 oldvobsrv
```

As it turns out that even if the vob is registered with a new vob server, its replica stays configured with the old one.

Here is how to fix it:

```
$ multitool chreplica -host newvobsrv rep1@/vob/foo
Updated replica information for "rep1@/vob/foo".
```

Views cleanup

Having views on a dedicated view server is a big relief for the administrator, as all the "abandoned" views (and other) problems can be handled in place. Troubleshooting the problems with local (end-user workstation located) views is trickier.

ClearCase administrators are used to the following scenario: a local view has been removed incorrectly or is not accessible as the workstation has crashed (has been replaced, and so on). Typically there would be some checkouts in such an abandoned local view.

In order to clean it up, one needs to find out the view uuid, and perform the `ct rmview -uuid` command on some host belonging to the appropriate region (the same one as where the view was registered).

To find out the view tag in question and its uuid, the administrator would sometimes need to check it using the `ct des -l vob:` command:

```
$ ct des -l vob:/vob/foo
versioned object base "/vob/foo"
...
 VOB holds objects from the following views:
 Win123456:c:\cc\winview1.vws ###########################################
                          [uuid 03f6851c.a49f11db.8a12.00:16:35:7f:04:48]
```

The view in question should be unregistered and its tag removed (so that for example, a view with the same tag could be re-created):

```
$ ct unreg -view -uuid 03f6851c.a49f11db.8a12.00:16:35:7f:04:48
$ ct rmtag -view winview1
```

The last step (and the longer) is to clean up the vob references to this view (for all the vobs in the following example):

```
$ ct rmview -uuid 03f6851c.a49f11db.8a12.00:16:35:7f:04:48 -all
```

Note that this operation leaves the view inconsistent, which is why it is better to unregister first: it is not supposed to be usable anymore.

ClearCase and Apache integration

ClearCase integration with a web server (Apache) could be beneficial for both parties:

- **ClearCase**: Providing public and easy (read-only) access to ClearCase vobs via http
- **Apache**: Maintaining the web server pages under ClearCase to be able to manage them better

This can be configured on any ClearCase client host, having Apache installed there as well.

The main idea is to use a dedicated view, let's call it `webview`, with a config spec preventing checkout.

The following is an example config spec:

```
$ ct catcs -tag webview
element * TOOLS -nocheckout
element * .../br1/LATEST -nocheckout
element * .../m/LATEST -nocheckout
element /vob/foo/... .../z/LATEST -nocheckout
element * /main/LATEST -nocheckout
```

Then for the Apache side configuration, set the following in `/etc/httpd/conf/httpd.conf`:

```
Alias /vob/ "/view/webview/vob/"

<Directory "/view/webview/vob">

Options Indexes FollowSymlinks MultiViews
AllowOverride None
Order allow,deny
Allow from all
EnableSendFile off
```

Note the `EnableSendFile off` setting turning Apache optimization off: using the *sendfile* system call, which is efficient but not supported by the MVFS filesystem. The symptom resulting of missing this step is that every file just looks empty. Note that this is necessary only if one wants to use a dynamic view (we do).

You may also set `DocumentRoot` to some location in a ClearCase vob:

```
DocumentRoot "/view/webview/vob/web"

<Directory "/view/webview/vob/web">
```

That's basically all concerning the configuration. Now, the `webview` should be first started, followed by restarting of the web server:

```
$ ct startview webview
$ apachectl -k start
```

Now you can read your ClearCase vobs via http (where `ccserver` is the name of the host where you did this integration):

```
http://ccserver/vob/foo
```

Versions that are not selected by the current `webview` view configuration can be accessible via the ClearCase version extended path (as usual) even in the URL path, for example:

```
http://ccserver/vob/foo/bar@@/main/br1/1
```

This provides the most lightweight type of access to a ClearCase vob: instant and easy (no client installation or setup, or any bulky data download), but of course, it is read-only and non-configurable. This could be ideal for accessing some stable documentation or codebase stored in a vob for reference purposes.

It would be recommendable to add a new UNIX service for managing the `webview` and ensuring it starts up automatically in case of the host reboot:

```
$ cat /etc/init.d/webview

#! /bin/bash
#
# webview Start/Stop the ClearCase webview view
# Source function library.
. /etc/init.d/functions

RETVAL=0

start() {
echo -n $"Starting the webview view: "
/opt/rational/clearcase/bin/cleartool startview webview
RETVAL=$?
echo
[ $RETVAL -eq 0 ]
return $RETVAL
}

stop() {
echo -n $"Stopping the webview view: "
/opt/rational/clearcase/bin/cleartool endview webview
RETVAL=$?
echo
[ $RETVAL -eq 0 ]
return $RETVAL
}

case "$1" in
start)
start
;;
stop)
stop
;;
```

```
*)
echo $"Usage: $0 {start|stop}"
exit 1
esac

exit $?

$ chkconfig --add webview
```

Installation tricks

Sometimes installing ClearCase on a non-supported platform (such as Fedora) is just a matter of one configuration file fine-tuning.

A default installation attempt would produce an error: `platform is not supported`. The applicable workaround is just a matter of creating `/etc/redhat-release` file, such as:

```
$ cat /etc/redhat-release
Red Hat Enterprise Linux AS release 4
```

After this the installation goes fine and ClearCase is fully functional (but it won't be supported by IBM).

Bottom-up

Here we will analyze a few error cases encountered by the end users, and will discuss possible ways of resolving them.

ALBD account problems

Working with a Windows ClearCase client, one might encounter the following problem: the Albd service does not start up due to the ALBD account login failure (note that this is a Windows ClearCase client-specific problem; no such configuration is applicable for the UNIX client). ClearCase becomes then unusable: no views can be created or started, etc. (Let's note though that some ClearCase operations could still be possible, provided that originally ALBD account was set up correctly, and got misconfigured later: for example, dynamic views would not be usable, but certain operations with snapshot views, including the view update and even checkouts and checkins could succeed).

Sometimes one would see errors of this kind when working in a snapshot view with the misconfigured (wrong username/password) ALBD account:

```
Info 10 May, 2009 10:46:31 view_server 1884 Using view ##############
                                    c:\joe\view.stg, on host: Win123456
Warning 10 May, 2008 10:07:12 view_server 2740 albd_contact call #####
                                         failed: RPC: Unable to receive;
                            errno = [WINSOCK] Connection reset by peer
Error 10 May, 2009 10:07:12 view_server 4336 view_server.exe(4336): ##
              Error: Unable to contact albd_server on host 'Win123456'
Error 10 May, 2009 09:11:57 view_server 2740 view_server.exe(2740): ##
    Error: Unable to get cleartext for vob: 1204d946.d7b011db.8730.00:
                                        16:35:7f:04:52 ver 0x112ea.
Error 10 May, 2009 09:11:57 view_server 2740 view_server.exe(2740): ##
      Error: Unable to construct cleartext for object "0x112EA" in VOB
"vobserver.domain.com:/vobstg/foo.vbs": error detected by ClearCase ##
                                                             subsystem
Error 10 May, 2009 09:11:57 view_server 2740 view_server.exe(2740): ##
       Error: Type manager "text_file_delta" failed construct_version
                                                            operation.
Warning 05 May, 2009 09:11:57 view_server 2740 text_file_delta: Error:
    Unable to open file "\\vobserver.domain.com\vobstg\foo.vbs\s/sdft\
        2a/20/0-ade01e85026c4e8da772460af72cbc72-rb": Invalid argument
```

Usually the ALBD password is not available for the end users in clear text form because of security reasons. The ALBD password is stored in the `sitedefs.dat` file in the ClearCase Windows release area, in encrypted format. But it does not help if one needs to reset the ALBD password manually (in **Services | Atria Location Broker | Logon**).

The way to do it is either to reinstall the ClearCase client from the available Windows release area (which is really overkill, at least from the end user's point of view), or to use the ***ALBD account reset utility*** provided by IBM (not officially supported though).

Using this utility would require that one has access to the Windows ClearCase release area, and would specify its location as a parameter. Then the encrypted password is read from the `sitedefs.dat` and is updated in the appropriate way in the Windows Registry. This is quite a light way of working around the ALBD problem, and is usually much appreciated by the end users.

Changing the type manager

Sometimes it turns out that certain files do not "qualify" for their default type manager:

```
$ ct ci -nc /vob/tools/jdk/THIRDPARTYLICENSEREADME.txt
text_file_delta: Error: "/vob/tools/jdk/THIRDPARTYLICENSEREADME.txt" ####
           is not a 'text file': it contains a line exceeding 8000 bytes.
Use a different type manager (such as compressed file).
```

Indeed, the text file type manager cannot handle such long lines. One must obey and change the element type in order to use a different type manager:

```
$ ct chtype file /vob/tools/jdk/THIRDPARTYLICENSEREADME.txt
Change version manager and reconstruct all versions for #################
                    "/vob/tools/jdk/THIRDPARTYLICENSEREADME.txt"? [no] yes
Changed type of element "/vob/tools/jdk/THIRDPARTYLICENSEREADME.txt" ####
                                                              to "file".
```

After that the checkin succeeds:

```
$ ct ci -nc /vob/tools/jdk/THIRDPARTYLICENSEREADME.txt
Checked in "/vob/tools/jdk/THIRDPARTYLICENSEREADME.txt" version #########
                                                     "/main/mybranch/1".
```

dbid and the Raima database

This case starts with a build error:

```
   fetch cleartext view=joe vob=/vob/work dbid=0x101d - #################
                                                     No permission match
```

Similar mentions of `dbid` may be found in other situations, for example, in system logs. They refer to the underlying database, used for both views and vobs. The first question is: what object is this about?

A first way, which may work, uses cleartool from the shell.

In case of any problems, one may use perl and the ***ClearCase::Argv*** module:

```
$ ct setview joe
$ cd /vob/work
$ ct dump $(ct find -a -ver '!created_since(now)' -print) | \
 egrep '^oid=.*dbid=.*\(0x101d\)'
bash: /usr/atria/bin/cleartool: Arg list too long
$ perl -MClearCase::Argv -e \
 '$c = new ClearCase::Argv({autochomp=>1,ipc=>1,stderr=>0});
 for($c->find([qw(-a -ver !created_since(now) -print)])->qx) {
 @v = grep/^oid=.*dbid=.*\(0x101d\)/, $c->dump($_)->qx;
 print "@v\n" if @v }'
oid=15009808.3bcd11de.965e.00:30:6e:4b:55:70 dbid=4125 (0x101d)
```

The `dbid` field is not directly accessible: it is only the identifier for the underlying Raima database.

There is one such record for every version of every element (and in fact for other objects as well, but we are looking for a cleartext container, thus for an element). In order to get all the existing versions in the vob, we need a pass-all query. *Not created since now* plays this role.

There are two problems with this:

- It will typically generate a huge output (and take long)
- Part of the versions (for example, the ones currently checked out) will not be accessible

To tackle these, one can use the following:

- ***ClearCase::Argv***: Perl deals better than a shell with lots of output, and the module presents it one item at a time to cleartool.
- The *ipc* mode: This uses one single cleartool invocation, thus optimizing performance.
- `stderr=>0`: Ignore the errors... The other option would be to run the `dump` commands in dedicated views, but this would be extremely slow. It might be considered in last resort.

The value printed by `ct dump` as `dbid` is given in decimal, with the value in parentheses being until 2003.06 the hexadecimal transcription.

This can be checked for a file element on an example. First, with 2003.06:

```
$ ct dump foo.txt | egrep '^oid'
oid=28003dd9.226a11dd.9bab.00:01:84:1e:63:a9 dbid=147591 (0x24087)
$ printf "%d\n" 0x24087 147591
```

Then with 7.0.1 (same version):

```
$ ct dump foo.txt | egrep '^oid='
oid=28003dd9.226a11dd.9bab.00:01:84:1e:63:a9 dbid=147591 (0xfe8b71e8)
```

With 7.0.1, the hexadecimal value is shared between all the versions, branches, and the element itself:

```
$ ct lsvtree -s apps | tail +2 | sed -e 's/\(.*\)/dump \1/' | \
 cleartool | egrep '^oid=.*dbid='
oid=520147ad.912211dc.8834.00:01:83:0a:15:19 dbid=145669 (0xfe8371e8)
oid=520147b1.912211dc.8834.00:01:83:0a:15:19 dbid=145670 (0xfe8371e8)
oid=520147b5.912211dc.8834.00:01:83:0a:15:19 dbid=145671 (0xfe8371e8)
oid=529147b9.912211dc.8834.00:01:83:0a:15:19 dbid=145672 (0xfe8371e8)
oid=533147c1.912211dc.8834.00:01:83:0a:15:19 dbid=145674 (0xfe8371e8)
oid=5310001f.a40411dc.8f90.00:01:83:0a:15:19 dbid=148344 (0xfe8371e8)
oid=cba8673c.74cb4bd3.a5ff.a8:81:b3:4c:f7:1a dbid=211770 (0xfe8371e8)
$ ct dump apps@@ | egrep '^oid=.*dbid='
oid=520147a5.912211dc.8834.00:01:83:0a:15:19 dbid=145667 (0xfe8371e8)
```

The hexadecimal value is common to all the elements in the vob!
We leave the topic of *dbids* on these findings, without a satisfactory solution for v7.0.

Protecting vobs: protectvob, vob_sidwalk, fix_prot

Here our target is to restrict the access to `/vob/secret` to one single group (sec); thus we first reprotect this vob from the previous group (`grp`) to the new one. To re-protect the vob (change owner, group, add/remove additional access groups), the `cleartool protectvob` command can be used:

```
$ ct protectvob -chown secadm -chgrp sec /filer01/vobstg/secret.vbs
This command affects the protection on your versioned object base.
While this command is running, access to the VOB will be limited.
If you have remote pools, you will have to run this command remotely.
```

In case of a vob having remote pools (for example, located on a NetApp storage, although this kind of setup is now outdated as seen earlier), the remote pools have to be re-protected separately.

For this purpose the *chown_pool* utility (from `/opt/rational/clearcase/etc`) can be useful:

```
   $ chown_pool secadm.sec /filer01/vobstg/secret.vbs/s/sdft
   $ chown_pool secadm.sec /filer01/vobstg/secret.vbs/d/ddft
   $ chown_pool secadm.sec /filer01/vobstg/scret.vbs/c/cdft
```

The `protectvob` command requires *root* access:

```
$ cd /vob/secret
$ newgrp sec
$ id uid=31415(secadm) gid=117(sec)
$ ct find -a -ele '!created_since(now)' -print | \
 perl -MClearCase::Argv -ne \
 'BEGIN{$ct=ClearCase::Argv->new({autochomp=>1,ipc=>1})}
 chomp;$ct->protect([qw(-chgrp sec)],$_)->system
  if $ct->des([qw(-fmt %[group]p)],$_)->qx eq q(grp)'
```

Not bad, especially as it could be done by the vob owner (not root). However, this affects only elements.

Then on the vob server, check what was left:

```
~> /opt/rational/clearcase/etc/utils/vob_siddump /vob/secret /tmp/sec.map
VOB Tag: /vob/secret (vobhost:/vobstg/secret.vbs)
Meta-type "directory element" ... 208 object(s)
Meta-type "directory version" ... 746 object(s)
Meta-type "tree element" ... 0 object(s)
Meta-type "element type" ... 13 object(s)
Meta-type "file element" ... 2622 object(s)
Meta-type "derived object" ... 0 object(s)
Meta-type "derived object version" ... 0 object(s)
Meta-type "version" ... 5334 object(s)
Meta-type "symbolic link" ... 0 object(s)
Meta-type "hyperlink" ... 0 object(s)
Meta-type "branch" ... 2830 object(s)
Meta-type "pool" ... 3 object(s)
Meta-type "branch type" ... 1 object(s)
Meta-type "attribute type" ... 9 object(s)
Meta-type "hyperlink type" ... 9 object(s)
Meta-type "trigger type" ... 0 object(s)
Meta-type "replica type" ... 1 object(s)
Meta-type "label type" ... 40 object(s)
Meta-type "replica" ... 2 object(s)
Meta-type "activity type" ... 0 object(s)
Meta-type "activity" ... 0 object(s)
Meta-type "state type" ... 0 object(s)
Meta-type "state" ... 0 object(s)
Meta-type "role" ... 0 object(s)
Meta-type "user" ... 0 object(s)
Meta-type "baseline" ... 0 object(s)
Meta-type "domain" ... 0 object(s)
Total number of objects found: 11818

Successfully processed VOB "/vob/secret".
~> ll /tmp/sec.map
-rw-r--r-- 1 secadm sec 521 Aug 5 17:05 /tmp/sec.map
~> cat /tmp/sec.map
anis/secadm,USER,UNIX:UID-31415,IGNORE,,,43
anis/joe,USER,UNIX:UID-79521,IGNORE,,,209
anis/bill,USER,UNIX:UID-58228,IGNORE,,,61
anis/jane,USER,UNIX:UID-80911,IGNORE,,,2490
```

```
anis/jill,USER,UNIX:UID-79707,IGNORE,,,35
anis/jack,USER,UNIX:UID-7080,IGNORE,,,62
anis/sam,USER,UNIX:UID-79706,IGNORE,,,4
anis/joan,USER,UNIX:UID-87669,IGNORE,,,9
anis/mark,USER,UNIX:UID-18543,IGNORE,,,1
anis/sec,GROUP,UNIX:GID-117,IGNORE,,,2830
anis/grp,GROUP,UNIX:GID-100,IGNORE,,,84
~> echo "anis/grp,GROUP,UNIX:GID-100,anis/sec,GROUP,UNIX:GID-117" \
 > /tmp/newmap
~> /opt/rational/clearcase/etc/utils/vob_sidwalk -execute \
 -map /tmp/newmap /vob/secret /tmp/map2
vob_sidwalk: Error: No permission to perform operation "modify vob".
vob_sidwalk: Error: Must be one of: root
vob_sidwalk: Error: Insufficient permission to fix protection: ##########
                                   error detected by ClearCase subsystem
```

Despite what the man page says (which was actually a ***documentation bug***), you have to be root to run `vob_sidwalk`, at least with the `-execute` flag!

We had protected so far (using `find/protect`) all the elements. There remained 84 objects. What about types?

```
$ for k in attype brtype eltype hltype lbtype trtype; \
 do ct lstype -fmt "protect -chgrp epc %Km:%n\n" -kind $k; done | \
  cleartool
```

This solved the problems for all the non-locked types.
A next use of `vob_sidwalk` showed a rest of 29 objects, 17 of which were locked types. Of those, seven could be files in the vob storage, and the remainder was unknown.

The `vob_sidwalk` command (with the same map) was run as root, and the result showed (`sam` lost?):

```
~> cat /tmp/epc.map3
anis/secadm,USER,UNIX:UID-31415,IGNORE,,,43
anis/joe,USER,UNIX:UID-79521,IGNORE,,,209
anis/bill,USER,UNIX:UID-58228,IGNORE,,,61
anis/jane,USER,UNIX:UID-80911,IGNORE,,,2490
anis/jill,USER,UNIX:UID-79707,IGNORE,,,35
anis/jack,USER,UNIX:UID-7080,IGNORE,,,62
Account Unknown,USER,UNIX:UID-79706,IGNORE,,,4
anis/joan,USER,UNIX:UID-87669,IGNORE,,,9
anis/mark,USER,UNIX:UID-18543,IGNORE,,,1

anis/sec,GROUP,UNIX:GID-117,IGNORE,,,2914
```

But the `tmp` source containers (left over from failing to create a new version: these are the new containers, which should have been renamed after the previous one was removed) were not protected, nor any of the files in the vob storage directory:

```
~> ll /vobstg/secret.vbs/s/sdft/13/10/tmp_8436.2
-rw-rw-rw- 1 secadm grp 18493016 Dec 15 2009 ###########################
                                /vobstg/secret.vbs/s/sdft/13/10/tmp_8436.2
~> find /vobstg/secret.vbs -group grp | wc -l
7
~> find /vobstg/secret.vbs -group eei-atusers
/vobstg/secret.vbs/.hostname
/vobstg/secret.vbs/s/sdft/pool_id
/vobstg/secret.vbs/s/sdft/13/10/tmp_8436.2
/vobstg/secret.vbs/s/sdft/13/10/tmp_2240.1
/vobstg/secret.vbs/s/sdft/0/3d/tmp_10020.2
/vobstg/secret.vbs/d/ddft/pool_id
/vobstg/secret.vbs/c/cdft/pool_id
```

The next attempt is to use `fix_prot` (as *root*):

```
secret.vbs# fix_prot -force -r -root -chown secadm \
 -chgrp sec /vobstg/secret.vbs
CAUTION! This program reprotects every file and directory in a
storage directory tree and should be used only when the protection
has been damaged (e.g., through the process of copying the tree or by
direct manipulation through a tool like the File Manager/Explorer).
Protecting "/vobstg/secret.vbs/.identity"...
Protecting "/vobstg/secret.vbs/s/sdft/0/0"...
...
```

Note that the `-root` option tells the tool that the argument is the root of a vob storage. Since we do not use a `-chmod` option, we do not need to run the command in two phases.

This does reprotect all the remaining files, but also affects the `.identity` directory (in the vob storage), in a way inconsistent with the output of the `ct des` command:

```
$ ct des vob:/vob/secret
versioned object base "/vobs/swdi/tools"
...
 VOB ownership:
  owner anis/secadm
  group anis/sec
 Additional groups:
  group anis/root
$ ls -l /vobstg/secret.vbs/.identity
total 0
-r----s--- 1 secadm sec 0 Aug 13 19:08 gid
-r-S------ 1 secadm sec 0 Aug 13 19:08 uid
```

One must restore (or remove, depending on the need, but to achieve consistency) the additional group:

```
$ ct protectvob -add root /vob/secret
$ ls -l /vobstg/secret.vbs/.identity
total 0
-r----s--- 1 secadm sec  0 Aug 13 19:08 gid
-r----s--- 1 secadm root 0 Aug 16 16:31 group.0
-r-S------ 1 secadm sec  0 Aug 13 19:08 uid
```

Alternatively, the changes to the `.identity` directory can be done manually (just using `touch`, `chown` and `chmod` command), and this also works out fine.

As a conclusion of this section, let's mention the technote about the ***summary of the protection commands and utilities***.

Cleaning lost+found

Let's first remind what the role of the `lost+found` directory in every vob is:

```
$ ct rmelem -f d1
cleartool: Warning: Object "bar" no longer referenced.
cleartool: Warning: Moving object to vob lost+found directory as ########
                                  "bar.1f89b0fe6e5511df8b600001842becee".
Removed element "d1".
```

A directory was removed: this resulted in the objects it contained losing their last reference in the file system.

The objects were kept and made accessible via the `lost+found` directory. In order to avoid name clashes with other objects, their name was made unique, by appending their oid.

Note that some other scenarios might lead to the same result as destroying a directory element, for example, unchecking a directory right after creating an element inside it.

Preserving file objects in `lost+found` makes it possible to restore them to any directory found suitable—either an existing one or created for this purpose. The directory must of course be checked out preliminarily, and checked in afterwards.

```
$ ct mv /vob/foo/lost+found/bar.1f89b0fe6e5511df8b600001842becee bar
```

Can we detect an event such as the one produced above?

```
$ ct lshis -minor -since 21:42 -all
--09-26T21:47 marc  destroy element in versioned object base "/vob/foo"
 "Destroyed element "/vob/foo/d1"."
--09-26T21:47 marc  remove name from directory version #################
                                                    "/vob/foo@@/main/mg/3"
 "Uncataloged directory element "d1"."
```

The `rmelem` events are kept forever (by default) in `vob_scrubber_params`.
The `rmname` ones are scrubbed at the latest after 14 days.

In order to be able to restore valuable data from `lost+found`, one needs to detect it. This implies that `lost+found` is not cluttered with lots of needless files, and thus sets a rationale for cleaning it up: keep a simple background for cases in which complexity will emerge anyway.

The main tool for the `lost+found` cleanup is `rmelem`; however, the procedure grows complex if it is not practiced regularly.

Do not attempt to remove the contents of `lost+found` recursively! The fact that a directory is not accessible from anywhere anymore doesn't mean that its contents are not: you might remove something valuable and still reference it from elsewhere in you vob.

Also, you may remove only what you master locally.

Some commands for the vob owner:

First, `uncheckout` the checked out versions. You have to do it using the view in which the objects are checked out, assuming this view is *dynamic*, still healthy, and reachable (both host and storage).

However, it is possible that the view cannot access the version anymore, or the `lost+found` directory itself.

Therefore, use the *oid* and the vob tag:

```
$ ct lsvtree * 2>/dev/null | perl -MClearCase::Argv -ne \
 'BEGIN{$ct=new ClearCase::Argv({autochomp=>1,ipc=>1,stderr=>0});
   $tag=$ct->des([qw(-s)],"vob:.")->qx}
 if(m%^(.*)\@\@/.*/CHECKEDOUT view "(.*)"$%){
 $o=$ct->des([qw(-fmt %On)], $1)->qx;
 $ct->argv(qw(setview -exec), "cleartool unco -rm oid:$o\@$tag",
  $2)->system}'
```

This will as such not affect names starting with a dot. Treat them with care, as `".*"` would match the `".."` parent directory!

It will also skip checked out versions bearing a label. You need to remove the label first anyway, before using `rmelem`.

Then, use (non recursive!):

```
$ ct rmelem -f *
```

Process there as well names starting with a dot specifically (like,`".a*"`)
You may repeat either step as many times as necessary (as long as there is something to remove).

Summary

Strict role assignment leads to incompetence. Administrators are not experts, and will not grow such if they only deal with routine administration and never look out of their box. All the same, users will break things if they don't attempt to understand what they do, and do not feel responsible for the environment (somebody else's job!) They will break fragile things: robustness is a quality one acquires by evolutionary mechanisms. Optimizing the processes will only take place if the various actors understand each other enough to criticize what others do and suggest meaningful alternatives; if all respect each other enough to listen to such critique and suggestions.

What this chapter reviewed is a long list of practical spots where users often face their administrators. A basic knowledge, but moreover the confidence that the information is there to be found if needed, are necessary to make this encounter a fruitful collaboration and not a frustrating confrontation. As we showed, this doesn't go without work, and is therefore subject to tradeoffs. This works both ways: users might want to avoid spending too much time and effort on tasks which, they feel fall beyond their attributions, but on the other hand, they have a different perspective, and thus different priorities than administrators.